29th International Conference on Concurrency Theory

CONCUR 2018, September 4–7, 2018, Beijing, China

Edited by

Sven Schewe

Lijun Zhang

Editors

Sven Schewe
Department of Computer Science
University of Liverpool
Liverpool, UK
sven.schewe@liverpool.ac.uk

Lijun Zhang
State Key Laboratory of Computer Science
Institute of Software Chinese Academy of Sciences
Beijing, China
zhanglj@ios.ac.cn

ACM Classification 2012
Theory of Computation

ISBN 978-3-95977-087-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-087-3.

Publication date
August, 2018

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

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 29th Conference on Concurrency Theory, which was held in Beijing, China, on September 4–7, 2018. CONCUR 2018 was organised by the Institute of Software, Chinese Academy of Sciences.

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. 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 also be found at CONCUR.

This edition of the conference attracted 101 full paper submissions, and 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 talks by Moshe Vardi, Yuxin Deng, Rob van Glabbeek, and Bow-Yaw Wang, as well as the tutorial delivered by Ichiro Hasuo. We thank the speakers for having accepted our invitation and their excellent presentations.

This year, the conference was jointly organised with the 16th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS), the 15th International Conference on Quantitative Evaluation of SysTems (QEST), and the fourth Symposium on Dependable Software Engineering (SETTA) in an overarching event, CONFESTA, organised by the Institute of Software, Chinese Academy of Sciences.

CONFESTA included four more satellite events: the combined 25th International Workshop on Expressiveness in Concurrency and 15th Workshop on Structural Operational Semantics (EXPRESS/SOS), the 3rd International workshop on TIming Performance engineering for Safety critical systems (TIPS'18), the 7th IFIP WG 1.8 Workshop on Trends in Concurrency Theory (TRENDS), and the 8th Young Researchers Workshop on Concurrency

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Theory (YR-CONCUR). as well as a number of tutorials. CONFESTA was preceeded by two further associated events, a Summer School on Formal Methods and a CAP Project Workshop.

The CONCUR proceedings are available for open access via LIPIcs, and we thank the staff from Schloss Dagstuhl, in particular Michael Wagner, for helping us with the preparation. Last, but not least, we thank the authors and the participants for making this year's CONCUR a successful and inspiring event.

Sven Schewe (University of Liverpool)
Lijun Zhang (Institute of Software, Chinese Academy of Sciences)

Committees

Programme Committee

Parosh Abdulla
Uppsala University (Sweden)

Christel Baier
TU Dresden (Germany)

Roderick Bloem
Graz University of Technology (Austria)

Ahmed Bouajjani
IRIF, University Paris Diderot (France)

Taolue Chen
Birkbeck, University of London (UK)

Yu-Fang Chen
Academia Sinica (Taiwan)

Alessandro Cimatti
Fondazione Bruno Kessler (Italy)

Pedro R. D'Argenio
Universidad Nacional de Córdoba - CONICET (Argentina)

Josée Desharnais
Université Laval (Canada)

Wan Fokkink
Vrije Universiteit Amsterdam (The Netherlands)

Erich Grädel
RWTH Aachen University (Germany)

Ichiro Hasuo
National Institute of Informatics (Japan)

Fei He
Tsinghua University (China)

Anna Ingólfsdóttir
Reykjavík University (Iceland)

Stefan Kiefer
University of Oxford (UK)

Shankara Narayanan Krishna
IIT Bombay (India)

Antonín Kučera
Masaryk University (Czech Republic)

Salvatore La Torre
Università degli Studi di Salerno (Italy)

Jérôme Leroux
CNRS (France)

Parthasarathy Madhusudan
University of Illinois at Urbana-Champaign (USA)

Rupak Majumdar
MPI-SWS (Germany)

Radu Mardare
Aalborg University (Denmark)

Roland Meyer
TU Braunschweig (Germany)

Angelo Montanari
University of Udine (Italy)

Sriram Sankaranarayanan
University of Colorado, Boulder (USA)

Alexandra Silva
University College London (UK)

Ana Sokolova
University of Salzburg (Austria)

Mariëlle Stoelinga
University of Twente (The Netherlands)

Franck van Breugel
York University (Canada)

Verena Wolf
Saarland University (Germany)

Co-Chairs

Sven Schewe
University of Liverpool (UK)

Lijun Zhang
Institute of Software, Chinese Academy of Sciences (China)

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Steering Committee

Jos Baeten
Centrum Wiskunde & Informatica (CWI) (The Netherlands)

Pedro R. D'Argenio
Universidad Nacional de Córdoba (Argentina)

Javier Esparza
Technische Universität München (Germany)

Joost-Pieter Katoen
RWTH Aachen (Germany)

Kim G. Larsen
Aalborg University (Denmark)

Ugo Montanari
Università di Pisa (Italy)

Catuscia Palamidessi
INRIA and LIX, École Polytechnique (France)

Local Organisers

Teng Fei
Institute of Software, Chinese Academy of Sciences (China)

David N. Jansen
Institute of Software, Chinese Academy of Sciences (China)

Yongjian Li
Institute of Software, Chinese Academy of Sciences (China)

Yi Lv
Institute of Software, Chinese Academy of Sciences (China)

Andrea Turrini
Institute of Software, Chinese Academy of Sciences (China)

Shuling Wang
Institute of Software, Chinese Academy of Sciences (China)

Peng Wu
Institute of Software, Chinese Academy of Sciences (China)

Bai Xue
Institute of Software, Chinese Academy of Sciences (China)

Rongjie Yan
Institute of Software, Chinese Academy of Sciences (China)

Li Zhang
Institute of Software, Chinese Academy of Sciences (China)

Xueyang Zhu
Institute of Software, Chinese Academy of Sciences (China)

Local Organisation Chair

Zhilin Wu
Institute of Software, Chinese Academy of Sciences (China)

Publicity Co-Chairs

Ernst Moritz Hahn
University of Liverpool (UK)

Meng Sun
Peking University (China)

List of External Reviewers

Luca Aceto
Dan Alistarh
Baskar Anguraj
Stavros Aronis
S. Arun-Kumar
Mohamed Faouzi Atig
Giorgio Bacci
Giovanni Bacci
Michael Backenköhler
Eric Badouel
Nikhil Balaji
Borja Balle
Francesco Belardinelli
Dietmar Berwanger
František Blahoudek
Laura Bocchi
Marco Bozzano
Laura Bozzelli
Tomas Brazdil
Simon Castellan
Ilaria Castellani
Pablo Castro
Didier Caucal
Mariano Ceccato
Rohit Chadha
Liqian Chen
Xin Chen
Peter Chini
Corina Cirstea
Emanuele D'Osualdo
Vrunda Dave
David de Frutos Escrig
Giorgio Delzanno
Stéphane Demri
Catalin Dima
Brijesh Dongol
Cezara Dragoi
Clemens Dubslaff
Jérémy Dubut
Constantin Enea
Gidon Ernst
Marco Faella
Uli Fahrenberg
Nathanaël Fijalkow
Brendan Fong
Ignacio Fábregas
Pierre Ganty
Paul Gastin
Simon Gay
Sergey Goncharov
Alexander Graf-Brill
Alberto Griggio
Gerrit Grossmann
Stefan Göller
Vojtěch Havlena
Frédéric Herbreteau
Lukas Holik
Hung-Wei Hsu
Omar Inverso
Ahmed Irfan
Rinat Iusupov
Petr Jancar

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Nils Jansen
Claude Jard
Peter Gjøl Jensen
Thomas Kahl
Benjamin Lucien Kaminski
Anja Karl
Joachim Klein
Bettina Koenighofer
Clemens Kupke
Marcel Kyas
Charalampos Kyriakopoulos
Rom Langerak
Kung-Kiu Lau
Marijana Lazic
Ondřej Lengál
Christoph Lenzen
Hsin-Hung Lin
Alexander Lück
Khushraj Madnani
Konstantinos Mamouras
Richard Mayr
Filip Mazowiecki
Alberto Molinari
J. Garrett Morris
Mohammad Mousavi
Sergio Mover
Sebastian Muskalla
Elisabeth Neumann
Jan Obdrzalek
Oded Padon
Vincent Penelle
Adriano Peron
Kirstin Peters
Gustavo Petri
Thomas Place
Gabriele Puppis
David Pym
Jorge A. Pérez
Hadi Ravanbakhsh
Vojtech Rehak
Antoine Rollet
Jurriaan Rot
Marco Roveri
Enno Ruijters
Prakash Saivasan
Pietro Sala
Tetsuya Sato
Sylvain Schmitz
Lutz Schröder
Roberto Segala
Ilya Sergey
Mahsa Shirmohammadi
David Sprunger
Daniel Stan
Caleb Stanford
Ivan Stojic
Eijiro Sumii
Grégoire Sutre
Toru Takisaka
Qiyi Tang
Peter Thiemann
Chun Tian
Simone Tini
Stefano Tonetta
Tigran Tonoyan
Andrea Turrini

Henning Urbat

Jaco van de Pol

Rob van Glabbeek

Dominik Velan

Walter Vogler

Masaki Waga

Hengfeng Wei

Tim Willemse

Sebastian Wolff

Nicolás Wolovick

James Worrell

Bo Wu

Zhilin Wu

Sascha Wunderlich

Akihisa Yamada

Shaofa Yang

Fabio Zanasi

Georg Zetzsche

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang

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

List of Authors

Parosh Aziz Abdulla
Uppsala University
Sweden
parosh@it.uu.se

Luca Aceto
Reykjavik University
Iceland
luca@ru.is

Shaull Almagor
Oxford University, UK
United Kingdom
shaull.almagor@mail.huji.ac.il

Mohamed Faouzi Atig
Uppsala University
Sweden
mohamed_faouzi.atig@it.uu.se

Stephanie Balzer
Carnegie Mellon University
United States
balzers@cs.cmu.edu

Massimo Bartoletti
Università degli Studi di Cagliari
Italy
bart@unica.it

Michael Blondin
Technical University of Munich
Germany
blondin@in.tum.de

Laura Bocchi
University of Kent
United Kingdom
L.Bocchi@kent.ac.uk

Benedikt Bollig
LSV, ENS Cachan, CNRS
France
bollig@lsv.ens-cachan.fr

Filippo Bonchi
University of Pisa
Italy
filippo.bonchi@ens-lyon.fr

Ahmed Bouajjani
IRIF, University Paris Diderot
France
abou@irif.fr

Tomasz Brengos
Warsaw University of Technology
Poland
t.brengos@mini.pw.edu.pl

Véronique Bruyère
University of Mons
Belgium
veronique.bruyere@umons.ac.be

Benjamin Cabrera
University of Duisburg-Essen
Germany
benjamin.cabrera@uni-due.de

Ian Cassar
University of Malta & Reykjavik University
Malta & Iceland
ian.cassar.10@um.edu.mt

Brynmor Chapman
MIT CSAIL & EECS, Cambridge, MA
United States
brynmor@mit.edu

Krishnendu Chatterjee
Institute of Science and Technology (IST)
Austria
krish.chat@gmail.com

Radu Ciobanu
The University of Edinburgh
United Kingdom
R.Ciobanu@sms.ed.ac.uk

Rodica Condurache
Universite Paris Est, Creteil, and
Universite Libre de Bruxelles
France & Belgium
rodica.bozianu@gmail.com

Wojciech Czerwinski
University of Warsaw
Poland
wczerwin@mimuw.edu.pl

Yuxin Deng
East China Normal University
China
yxdeng@sei.ecnu.edu.cn

Alastair Donaldson
Imperial College London
United Kingdom
alastair.donaldson@imperial.ac.uk

Amina Doumane
CNRS - ENS Lyon
France
Amina.Doumane@ens-lyon.fr

Javier Esparza
Technical University of Munich
Germany
esparza@in.tum.de

Hugues Evrard
Imperial College London
United Kingdom
h.evrard@imperial.ac.uk

Marie Fortin
LSV, ENS Paris-Saclay,
CNRS, Université Paris-Saclay
France
marie.fortin@lsv.fr

Paulin Fournier
LS2N, Université de Nantes
France
paulin.fournier@gmail.com

Adrian Francalanza
University of Malta
Malta
adrian.francalanza@um.edu.mt

Pierre Ganty
IMDEA Software Institute
Spain
pierre.ganty@imdea.org

Paul Gastin
LSV, ENS Paris-Saclay,
CNRS, Université Paris-Saclay
France
gastin@lsv.fr

Hugo Gimbert
CNRS, LABRI, Bordeaux
France
hugo.gimbert@labri.fr

Amir Kafshdar Goharshady
IST Austria
Austria
goharshady@ist.ac.at

Sergey Goncharov
FAU Erlangen-Nürnberg
Germany
Sergey.Goncharov@fau.de

Andrey Gorlin
Stony Brook University
United States
agorlin@cs.stonybrook.edu

Radu Grigore
University of Kent
United Kingdom
radugrigore@gmail.com

Christoph Haase
University of Oxford
United Kingdom
Christoph.Haase@cs.ox.ac.uk

Ichiro Hasuo
National Institute of Informatics
Japan
i.hasuo@acm.org

Quentin Hautem
UMONS
Belgium
quentin.hautem@umons.ac.be

Reiko Heckel
University of Leicester
United Kingdom
reiko@mcs.le.ac.uk

Tobias Heindel
DIKU, University of Copenhagen
Denmark
tobias.heindel@googlemail.com

Thomas A. Henzinger
IST Austria
Austria
tah@ist.ac.at

Piotr Hofman
University of Warsaw
Poland
piotrek.hofman@gmail.com

Mehran Hosseini
University of Oxford
United Kingdom
mehran.hosseini@cs.ox.ac.uk

Rasmus Ibsen-Jensen
IST Austria
Austria
ribsen@ist.ac.at

Anna Ingolfsdottir
Reykjavik University
Iceland
annai@ru.is

Suresh Jagannathan
Purdue University
United States
suresh@cs.purdue.edu

Julian Jakob
FAU Erlangen-Nürnberg
Germany
Julian.Jakob@fau.de

Kartik Nagar
Purdue University
United States
nagark@purdue.edu

Stefan Kiefer
University of Oxford
United Kingdom
stefan.kiefer@cs.ox.ac.uk

Barbara König
Universität Duisburg-Essen
Germany
barbara_koenig@uni-due.de

Igor Konnov
INRIA Nancy (LORIA)
France
igor.konnov@inria.fr

Bernhard Kragl
IST Austria
Austria
bkragl@ist.ac.at

Jan Křetínský
Technical University of Munich
Germany
jan.kretinsky@gmail.com

Antonin Kučera
Masaryk University
Czechia
tony@fi.muni.cz

Jure Kukovec
Vienna University of Technology
Austria
jkukovec@forsyte.at

K Narayan Kumar
Chennai Mathematical Institute
India
kumar@cmi.ac.in

Sławomir Lasota
University of Warsaw
Poland
sl@mimuw.edu.pl

Khushraj Madnani
IIT Bombay
India
khushraj@cse.iitb.ac.in

Rupak Majumdar
Max Planck Institute for Software Systems
Germany
rupak@mpi-sws.org

Richard Mayr
The University of Edinburgh
United Kingdom
rmayr@staffmail.ed.ac.uk

Filip Mazowiecki
LaBRI, Université de Bordeaux
France
filip.mazowiecki@u-bordeaux.fr

Roland Meyer
TU Braunschweig
Germany
roland.meyer@tu-bs.de

Jakub Michaliszyn
University of Wroclaw
Poland
jakub.michaliszyn@gmail.com

Christina Mika-Michalski
University Duisburg-Essen
Germany
christine.mika@uni-due.de

Sayan Mukherjee
Chennai Mathematical Institute
India
sayanm@cmi.ac.in

Maurizio Murgia
University of Kent
United Kingdom
M.Murgia@kent.ac.uk

Sebastian Muskalla
TU Braunschweig
Germany
s.muskalla@tu-bs.de

Renato Neves
INESC TEC (HASLab) and
University of Minho
Portugal
nevrenato@di.uminho.pt

Jan Otop
University of Wrocław
Poland
jotop@cs.uni.wroc.pl

Joel Ouaknine
Max Planck Institute for Software Systems
Germany
joel@mpi-sws.org

Youssouf Oualhadj
Université Paris Est Créteil
France
youssouf.oualhadj@lacl.fr

Paritosh Pandya
TIFR
India
pandya@tifr.res.in

Guillermo Perez
Université libre de Bruxelles
Belgium
gperezme@ulb.ac.be

Daniela Petrisan
Université Paris Diderot - Paris 7
France
daniela.petrisan@gmail.com

Frank Pfenning
Carnegie Mellon University
United States
fp@cs.cmu.edu

Andy Polyakov
The OpenSSL project
Sweden
appro@openssl.org

Damien Pous
CNRS - ENS Lyon
France
Damien.Pous@ens-lyon.fr

Pavithra Prabhakar
Kansas State University
United States
pprabhakar@ksu.edu

Shaz Qadeer
Microsoft
United States
qadeer@microsoft.com

C. R. Ramakrishnan
Stony Brook University
United States
cram@cs.stonybrook.edu

Jean-Francois Raskin
Université Libre de Bruxelles
Belgium
jraskin@ulb.ac.be

Nima Roohi
University of Pennsylvania
United States
roohi2@cis.upenn.edu

Christian Rosenke
University of Rostock
Germany
christian.rosenke@uni-rostock.de

Alexej Rotar
Technical University of Munich
Germany
alexejrotar@gmail.com

Krishna S
IIT Bombay
India
krishnas@cse.iitb.ac.in

Prakash Saivasan
TU Braunschweig
Germany
p.saivasan@tu-bs.de

Tyler Sorensen
Imperial College London
United Kingdom
t.sorensen15@imperial.ac.uk

B Srivathsan
Chennai Mathematical Institute
India
sri@cmi.ac.in

Qiyi Tang
York University, Toronto
Canada
qiyitang@eecs.yorku.ca

Bernardo Toninho
Universidade NOVA de Lisboa
Portugal
btoninho@gmail.com

Patrick Totzke
University of Edinburgh
United Kingdom
p.totzke@ed.ac.uk

Ronny Tredup
University of Rostock
Germany
ronny.tredup2@uni-rostock.de

Nicolas Troquard
Free University of Bozen
Italy
nicolas.troquard@unibz.it

Ming-Hsien Tsai
Academia Sinica
Taiwan
mhtsai208@gmail.com

Franck van Breugel
York University, Toronto
Canada
franck@eecs.yorku.ca

Rob van Glabbeek
CSIRO
Australia
rvg@cs.stanford.edu

Moshe Y. Vardi
Rice University
USA
vardi@cs.rice.edu

Yaron Velner
Tel Aviv University
Israel
yaron172@yahoo.com

Mahesh Viswanathan
University of Illinois at Urbana-Champaign
United States
vmahesh@illinois.edu

Bow-Yaw Wang
Academia Sinica
Taiwan
bywang@iis.sinica.edu.tw

Josef Widder
Vienna University of Technology
Austria
widder@forsyte.at

James Worrell
University of Oxford
United Kingdom
jbw@cs.ox.ac.uk

Bo-Yin Yang
Academia Sinica
Taiwan
by@crypto.tw

Georg Zetzsche
IRIF, CNRS & Université Paris-Diderot
France
zetzsche@irif.fr

The Siren Song of Temporal Synthesis

Moshe Y. Vardi[1]
Department of Computer Science, Rice University, Houston, TX, USA
vardi@cs.rice.edu

Abstract

One of the most significant developments in the area of design verification over the last three decade is the development of algorithmic methods for verifying temporal specification of finite-state designs. A frequent criticism against this approach, however, is that verification is done after significant resources have already been invested in the development of the design. Since designs invariably contains errors, verification simply becomes part of the debugging process. The critics argue that the desired goal is to use temporal specification in the design development process in order to guarantee the development of correct designs. This is called temporal synthesis. In this talk I will review 60 years of research on the temporal synthesis problem, describe the automata-theoretic approach developed to solve this problem, and describe both successes and failures of this research program [1, 2].

2012 ACM Subject Classification Software and its engineering

Keywords and phrases Formal Methods, Temporal Synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.1

Category Invited Talk

References

1 Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. A symbolic approach to safety LTL synthesis. In *Proc. 13th Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing*, volume 10629 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2017. doi:10.1007/978-3-319-70389-3_10.

2 Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTLf synthesis. In *Proc. 26th Int'l Joint Conf. on Artificial Intelligence*, pages 1362–1369. ijcai.org, 2017. doi:10.24963/ijcai.2017/189.

[1] Work supported in part by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering".

Bisimulations for Probabilistic and Quantum Processes

Yuxin Deng[1]
Shanghai Key Laboratory of Trustworthy Computing,
MOE International Joint Lab of Trustworthy Software,
and International Research Center of Trustworthy Software,
East China Normal University, Shanghai, China
yxdeng@sei.ecnu.edu.cn
https://orcid.org/0000-0003-0753-418X

Abstract

Bisimulation is a fundamental concept in the classical concurrency theory for comparing the behaviour of nondeterministic processes. It admits elegant characterisations from various perspectives such as fixed point theory, modal logics, game theory, coalgebras etc. In this paper, we review some key ideas used in the formulations and characterisations of reasonable notions of bisimulations for both probabilistic and quantum processes. To some extent the transition from probabilistic to quantum concurrency theory is smooth and natural. However, new ideas need also to be introduced. We have not yet reached the stage of formally verifying quantum communication protocols and quantum algorithms using bisimulations implemented by automatic tools. We discuss some recent efforts in this direction.

2012 ACM Subject Classification Theory of computation → Process calculi, Theory of computation → Operational semantics, Theory of computation → Modal and temporal logics

Keywords and phrases Bisimulations, probabilistic processes, quantum processes

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.2

Category Invited Paper

1 Introduction

Bisimulation [39, 37] is a fundamental concept in the classical concurrency theory as it admits beautiful characterisations in terms of fixed points, modal logics, co-algebras, pseudometrics, games, decision algorithms, etc. Its generalisation in the probabilistic setting is initiated by Larsen and Skou in [36] and has subsequently been widely investigated in probabilistic concurrency theory. One of the main contributions of [36] is the introduction of a lifting operation that converts a relation between states to a relation between distributions over states. Later on, the lifting operation is shown to be closely related to some prominent concepts in mathematics such as the Kantorovich metric [33, 45] and the maximum network flow problem [1]; the latter is crucial for designing algorithms to check if two states are bisimilar.

The probabilistic bisimulation nicely defined in [36] has natural characterisations by probabilistic extensions of Hennessy-Milner logic [28]; see e.g. [36, 14, 15, 40, 10, 30, 26, 12, 4]. Most characterisations employ some modalities indexed with numbers. A typical modal

[1] Supported by the National Natural Science Foundation of China (61672229) and Shanghai Municipal Natural Science Foundation (16ZR1409100).

formula, dated back to [36], is $\langle a\rangle_p\phi$, where p is a probability value. A state s satisfies this formula if the probability that s can make an a-labelled transition to the set of states satisfying ϕ exceeds p. In [44] van Breugel et al. generalise the characterisation of [36] to labelled Markov processes, i.e. reactive probabilistic processes [36, 46] with continuous state spaces, and surprisingly, without using any modality indexed with numbers. Usually, the simpler the logical characterisation, the more difficult its completeness proof, since constructing distinguishing formulae for non-bisimilar states with fewer modalities is more challenging. Van Breugel et al. prove such an elegant result by using some advanced machinery such as the Lawson topology on probabilistic powerdomains [31] and Banach algebras. However, if we confine ourselves to discrete rather than continuous state spaces, as in e.g. [36], the characterisation result given in [44] has a very elementary proof [7].

Since probabilistic behaviour is prevalent in quantum computation, it is natural to investigate how a quantum concurrency theory can be built upon the probabilistic concurrency theory. Notice that the operational semantics of many quantum systems can be defined in terms of probabilistic labelled transition systems, which allows us to define quantum bisimulations in a very intuitive way by extending probabilstic bisimulations with a requirement on demanding equal environments when comparing two quantum processes. However, to check quantum bisimulations, we need to appeal to the instantiation of quantum variables by quantum systems. What's worse, to check whether or not two quantum processes are bisimilar, we need to consider arbitrarily chosen quantum states, which appears infeasible in practice because quantum states constitute a continuum. Fortunately, it is possible to overcome this difficulty by introducing a symbolic semantics and its associated symbolic quantum bisimulations [20] that are equivalent to the usual concrete bisimulations. This opens the door to design effective algorithms to check quantum bisimulations.

A distinctive feature of quantum computation is entailed by the no-cloning theorem in quantum mechanics. Namely, quantum resources are linear from a type-theoretic point of view. It is then particularly meaningful to study *linear contextual equivalence*, which is a special form of contextual equivalence as the behaviours of programs are observed by executing them only once. In [8], it is shown that for higher-order quantum programs, linear contextual equivalence can be precisely captured by a distribution-based bisimilarity, which is weaker than the usual state-based bisimilarity. Of course, distribution-based bisimulations can also be defined for probabilistic processes, but in the quantum setting they become a more important coinductive proof technique.

The rest of the paper is structured as follows. In Section 2, we review the formal model of probabilistic labelled transition systems, the lifting operation, some of its equivalent formulations, state-based and distribution-based bisimulations. In Section 3 we introduce a quantum process algebra, discuss state-based and distribution-based quantum bisimulations, and symbolic bisimulations. Finally, we conclude in Section 4.

2 Probabilistic Bisimulation

In this section, we introduce the model of probabilistic labelled transition systems, the key concept of lifting operation, the state-based and distribution-based bisimulations.

2.1 Probabilistic Labelled Transition Systems

Let S be a countable set. A *(discrete) probability (sub)distribution* over set S is a function $\Delta : S \to [0,1]$ with *size* $|\Delta| = \sum_{s\in S}\Delta(s) \leq 1$. It is a *(full) distribution* if $|\Delta| = 1$. Its *support*, written $\lceil\Delta\rceil$, is the set $\{s \in S \mid \Delta(s) > 0\}$. Let $\mathcal{D}_{sub}(S)$ and $\mathcal{D}(S)$ denote the set of

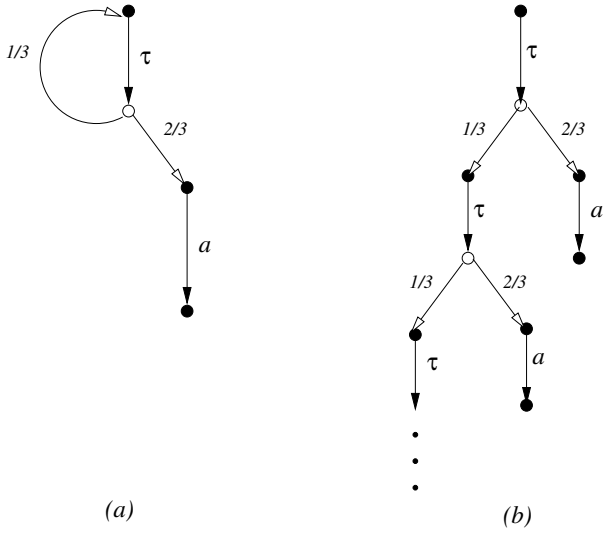


Figure 1 Example pLTSs.

all subdistributions and distributions over S, respectively. We use ε to stand for the empty subdistribution, that is $\varepsilon(s) = 0$ for any $s \in S$. We write $\overline{s}$ for the point distribution for state s, satisfying $\overline{s}(t) = 1$ if $t = s$, and 0 otherwise. If $p_i \geq 0$ and Δ_i is a distribution for each i in some finite index set I, then $\sum_{i\in I} p_i \cdot \Delta_i$ is given by

$$(\sum_{i\in I} p_i \cdot \Delta_i)(s) \quad = \quad \sum_{i\in I} p_i \cdot \Delta_i(s) \ .$$

If $\sum_{i\in I} p_i = 1$ then this is easily seen to be a distribution in $\mathcal{D}(S)$.

▶ **Definition 1.** A *probabilistic labelled transition system* (pLTS) is defined as a triple $\langle S, A, \rightarrow\rangle$, where S is a set of states, A is a set of actions, and the transition relation $\rightarrow$ is a subset of $S \times A \times \mathcal{D}(S)$.

A non-probabilistic labelled transition system (LTS) may be viewed as a degenerate pLTS – one in which only point distributions are used. We often write $s \xrightarrow{\alpha} \Delta$ in place of $(s, \alpha, \Delta) \in \rightarrow$.

In order to visualise pLTSs, we often draw them as directed graphs. Given that in a pLTS transitions go from states to distributions, we need to introduce additional edges to connect distributions back to states, thereby obtaining a bipartite graph. States are therefore represented by nodes of the form $\bullet$ and distributions by nodes of the form $\circ$. For any state s and distribution Δ with $s \xrightarrow{\alpha} \Delta$ we draw an edge from s to Δ, labelled with α. Consequently, the edges leaving a $\bullet$-node are all labelled with actions from A. For any distribution Δ and state s in $\lceil\Delta\rceil$, the support of Δ, we draw an edge from Δ to s, labelled with $\Delta(s)$. Consequently, the edges leaving a $\circ$-node are labelled with positive real numbers that sum to 1. Sometimes we partially unfold this graph by drawing the same nodes multiple times; in doing so, all outgoing edges of a given instance of a node are always drawn, but not necessarily all incoming edges. Edges labelled by probability 1 occur so frequently that it makes sense to omit them, together with the associated nodes $\circ$ representing point distributions.

Two example pLTSs are described this way in Figure 1, where diagram (b) depicts the initial part of the pLTS obtained by unfolding the one in diagram (a).

For each state s, the outgoing transition $s \xrightarrow{\alpha} \Delta$ represents the nondeterministic alternatives available in the state s. The nondeterministic choices provided by s are supposed to be resolved by the environment, which is often formalised by a *scheduler* or an *adversary*. On the other hand, the probabilistic choices in the underlying distribution Δ are made by the system itself. Therefore, for each state s, the environment chooses some outgoing transition $s \xrightarrow{\alpha} \Delta$. Then the action α is performed, the system resolves the probabilistic choice, and subsequently with probability $\Delta(s')$ the system reaches state s'.

If we impose the constraint that for any state s and action α at most one outgoing transition from s is labelled α, then we obtain the special class of pLTSs called *reactive* (or *deterministic*) pLTSs that are the probabilistic counterpart to deterministic LTSs. Formally, a pLTS is reactive if for each $s \in S, \alpha \in A$ we have that $s \xrightarrow{\alpha} \Delta$ and $s \xrightarrow{\alpha} \Delta'$ imply $\Delta = \Delta'$.

2.2 Lifting Relations

In the probabilistic setting, formal systems are usually modelled as distributions over states. To compare two systems involves the comparison of two distributions. So we need a way of lifting relations on states to relations on distributions. This is used, for example, to define a notion of probabilistic bisimulation as we shall see soon. A few approaches of lifting relations have appeared in the literature. We will take the one from [11], and show its coincidence with two other approaches.

▶ **Definition 2.** Given two sets S and T and a binary relation $\mathcal{R} \subseteq S \times T$, the lifted relation $\mathcal{R}^\dagger \subseteq \mathcal{D}(S) \times \mathcal{D}(T)$ is the smallest relation that satisfies:

(1) $s \mathcal{R} t$ implies $\overline{s} \mathcal{R}^\dagger \overline{t}$

(2) (Linearity) $\Delta_i \mathcal{R}^\dagger \Theta_i$ for all $i \in I$ implies $(\sum_{i\in I} p_i \cdot \Delta_i) \mathcal{R}^\dagger (\sum_{i\in I} p_i \cdot \Theta_i)$, where I is a finite index set and $\sum_{i\in I} p_i = 1$.

There are alternative presentations of Definition 2. One example is given below.

▶ **Proposition 3.** *Let Δ and Θ be two distributions over S and T, respectively, and $\mathcal{R} \subseteq S \times T$. Then $\Delta \mathcal{R}^\dagger \Theta$ if and only if there are two collections of states, $\{s_i\}_{i\in I}$ and $\{t_i\}_{i\in I}$, and a collection of probabilities $\{p_i\}_{i\in I}$, for some finite index set I, such that $\sum_{i\in I} p_i = 1$ and Δ, Θ can be decomposed as follows:*

(1) $\Delta = \sum_{i\in I} p_i \cdot \overline{s_i}$

(2) $\Theta = \sum_{i\in I} p_i \cdot \overline{t_i}$

(3) *For each $i \in I$ we have $s_i \mathcal{R} t_i$.*

From Definition 2, the next two propositions follow. In fact, they are sometimes used in the literature as definitions of lifting relations instead of being properties (see e.g. [43, 36, 13, 41]).

▶ **Proposition 4.**

(1) *Let Δ and Θ be distributions over S and T, respectively. Then $\Delta \mathcal{R}^\dagger \Theta$ if and only if there is a probability distribution on $S \times T$, with support a subset of $\mathcal{R}$, such that Δ and Θ are its marginal distributions. In other words, there exists a weight function $w : S \times T \to [0, 1]$ such that*

- **a.** $\forall s \in S : \sum_{t\in T} w(s, t) = \Delta(s)$
- **b.** $\forall t \in T : \sum_{s\in S} w(s, t) = \Theta(t)$
- **c.** $\forall (s, t) \in S \times T : w(s, t) > 0 \Rightarrow s \mathcal{R} t$.

(2) *Let Δ and Θ be distributions over S and $\mathcal{R}$ be an equivalence relation. Then $\Delta \mathcal{R}^\dagger \Theta$ if and only if $\Delta(C) = \Theta(C)$ for all equivalence classes $C \in S/\mathcal{R}$, where $\Delta(C)$ stands for the accumulation probability $\sum_{s\in C} \Delta(s)$.*

Given a binary relation $\mathcal{R} \subseteq S \times T$ and a set $S' \subseteq S$, we write $\mathcal{R}(S')$ for the set $\{t \in T \mid \exists s \in S' : s \mathcal{R} t\}$. A set S' is $\mathcal{R}$-closed if $\mathcal{R}(S') \subseteq S'$.

▶ **Proposition 5.** *Let Δ and Θ be distributions over finite sets S and T, respectively.*

(1) *$\Delta \mathcal{R}^\dagger \Theta$ if and only if $\Delta(S') \leq \Theta(\mathcal{R}(S'))$ for all $S' \subseteq S$.*

(2) *If $\mathcal{R}$ is a preorder, then $\Delta \mathcal{R}^\dagger \Theta$ if and only if $\Delta(S') \leq \Theta(S')$ for each $\mathcal{R}$-closed set $S' \subseteq S$.*

Besides the above interesting properties, the lifting operation has an intrinsic connection with some important concepts in mathematics, notably *the Kantorovich metric* [33]. For example, it turns out that our lifting of binary relations from states to distributions nicely corresponds to the lifting of metrics from states to distributions by using the Kantorovich metric. In addition, the lifting operation is closely related to *the maximum flow problem* in optimisation theory. This observation initially made by Baier *et al.* is crucial for designing decision algorithms for probabilistic bisimulations and simulations [1, 48].

2.3 Probabilistic Bisimulation

With a solid base of the lifting operation, we can proceed to define a probabilistic version of bisimulation. Let s and t be two states in a pLTS. We say t can simulate the behaviour of s if whenever the latter can exhibit some action, say a, and lead to distribution Δ then the former can also perform a and lead to a distribution, say Θ, which then in turn can mimic Δ in successor states. We are interested in defining a relation between two states, but it is expressed by invoking a relation between two distributions. To formalise the mimicking of one distribution by the other, we make use of the lifting operation investigated in Section 2.2.

▶ **Definition 6.** A relation $\mathcal{R} \subseteq S \times S$ is a *probabilistic simulation* if $s \ \mathcal{R} \ t$ implies

- if $s \xrightarrow{a} \Delta$ then there exists some Θ such that $t \xrightarrow{a} \Theta$ and $\Delta \mathcal{R}^\dagger \Theta$.

If both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are probabilistic simulations, then $\mathcal{R}$ is a *probabilistic bisimulation*. The largest probabilistic bisimulation, denoted by $\sim_s$, is called *(state-based) probabilistic bisimilarity*.

Let's look at the two pLTSs in Figure 1. It is easy to check that the top node in diagram (a) and that in diagram (b) are related by $\sim_s$.

Various characterisations of probabilistic bisimilarity by probabilistic versions of Hennessy-Milner logic [28] have appeared in the literature. In particular, if we confine ourselves to reactive pLTSs, then there are neat logical characterisations even without negation. For example, Desharnais et al. [14] uses a logic with the following grammar

$$\varphi ::= \top \mid \varphi \wedge \varphi \mid \langle a \rangle_q \varphi$$

where q is any rational number in the unit interval $[0, 1]$ and a ranges over the fixed set of labels of a given reactive pLTS. The formula $\top$ can always be satisfied. The formula $\varphi \wedge \varphi$ stands for the usual conjunction. The formula $\langle a \rangle_q \varphi$ is satisfied by state s if the probability that s can make an a-labelled transition to the set of states satisfying φ exceeds p. The characterisation result of [14] holds for reactive pLTSs with continuous state spaces. For reactive pLTSs with countable state spaces, a simpler proof of that result is given in [12]. Most other characterisations also employ modalities indexed with numbers. This fits in our intuition: if two states are not bisimilar, then they may satisfy a property with different probabilities, so by fiddling with the numbers we can construct a formula that can tell apart the two states. The only exception is the one given in [44], which shows that, for reactive probabilistic processes, probabilistic bisimilarity can be characterised by a surprisingly simple logic.

Let $\mathcal{L}$ be the set of formulae defined by the grammar

$$\phi \ ::= \ \top \mid \langle \phi, \phi \rangle \mid \langle a \rangle \phi$$

where a ranges over the set of labels of a reactive pLTS. A state s satisfies a formula ϕ with certain probability, given by $Pr(s, \phi)$ defined as follows:

$$\begin{array}{rcl} Pr(s, \top) & = & 1 \\ Pr(s, \langle \phi_1, \phi_2 \rangle) & = & Pr(s, \phi_1) \cdot Pr(s, \phi_2) \\ Pr(s, \langle a \rangle \phi) & = & \begin{cases} \sum_{s' \in S} \Delta(s') \cdot Pr(s', \phi) & \text{if } s \xrightarrow{a} \Delta \\ 0 & \text{otherwise.} \end{cases} \end{array}$$

We call $\langle \phi_1, \phi_2 \rangle$ a *conjunction* of two formulae ϕ_1 and ϕ_2, which models the copying capacity of probabilistic testing originally considered in [36]. Note that conjunction is given the arithmetic interpretation as multiplication, which differs from many other logical characterisations of probabilistic bisimilarity. The formula $\langle a \rangle \phi$ measures the probability that a state performs action a and then its successor states sastisfy ϕ.

The logic $\mathcal{L}$ induces a natural logical equivalence, written $=_{\mathcal{L}}$, by letting $s_1 =_{\mathcal{L}} s_2$ if $Pr(s_1, \phi) = Pr(s_2, \phi)$ for any $\phi \in \mathcal{L}$ and states s_1 and s_2. In [44] van Breugel et al. consider labelled Markov processes with continuous state spaces and they show that probabilistic bisimilarity coincides with the above notion of logical equivalence. Their proof involves advanced machinery such as the Lawson topology on probabilistic powerdomains [31] and Banach algebras. If we confine ourselves to finite-state reactive pLTSs, it is possible to avoid all the advanced machinery and give an elementary proof of the coincidence of $\sim_s$ with $=_{\mathcal{L}}$, as recently demonstrated in [7].

2.4 Distribution-Based Bisimulation

In Definition 6 we compare the behaviour of two states, and then resort to the lifting operation when talking about the simulation of one distribution by another. Alternatively, it is possible to consider subdistributions as first-class citizens and directly define a relation that compares subdistributions. In order to do so, we first define a transition relation between subdistributions.

▶ **Definition 7.** With a slight abuse of notation, we also use the notation $\xrightarrow{a}$ to stand for the transition relation between subdistributions, which is the smallest relation satisfying the following three rules:

(1) if $s \xrightarrow{a} \Delta$ then $\overline{s} \xrightarrow{a} \Delta$;

(2) if $s \not\xrightarrow{a}$ then $\overline{s} \xrightarrow{a} \varepsilon$;

(3) if $\Delta_i \xrightarrow{a} \Theta_i$ for all $i \in I$ then $(\sum_{i \in I} p_i \cdot \Delta_i) \xrightarrow{a} (\sum_{i \in I} p_i \cdot \Theta_i)$, where I is a finite index set and $\sum_{i \in I} p_i \leq 1$.

Note that if $\Delta \xrightarrow{a} \Delta'$ then some (not necessarily all) states in the support of Δ can perform action a. Those states that cannot enable action a contribute nothing for Δ'.

▶ **Definition 8.** Let $\sim_d \subseteq \mathcal{D}_{sub}(S) \times \mathcal{D}_{sub}(S)$ be the largest symmetric relation such that if $\Delta \sim_d \Theta$ then $|\Delta| = |\Theta|$ and $\Delta \xrightarrow{a} \Delta'$ implies the existence of some Θ' such that $\Theta \xrightarrow{a} \Theta'$ and $\Delta' \sim_d \Theta'$.

The distribution-based bisimilarity $\sim_d$ is shown in [6] as a sound and complete coinductive proof technique for linear contextual equivalence, a natural extensional behavioural equivalence for functional programs. In the literature there are several proposals of distribution-based bisimilarities [23, 26, 9, 17, 29], and some typical ones are compared in [16].

3 Quantum Bisimulation

In this section, we will see that quantum bisimulations can be obtained by extending probabilistic bisimulations in a smooth way.

As is well known, it is very difficult to guarantee the correctness of classical communication protocols at the design stage, and some simple protocols were eventually found to have fundamental flaws. One expects that the design of complex quantum protocols is at least as error-prone, if not more, than in the classical case. Bisimulation and its associated coinduction proof technique have also been explored in quantum concurrency theory.

Due to the presence of measurements, quantum processes exhibit probabilistic behaviour. It is then natural to define the operational semantics of a quantum process in turns of a pLTS, on which the probabilistic bisimulations we discussed before, with some modifications, may play a role in providing a coinduction proof technique for quantum processes. Note that in the quantum setting, bisimulations are defined to be relations over configurations that are pairs of a quantum process and a density operator describing the state of environment quantum systems. Below we illustrate this idea in the framework of a quantum process algebra.

3.1 Quantum Bisimulation for qCCS

We first briefly review the syntax and semantics of a quantum extension of value-passing CCS [37, 25], called qCCS, studied in [18, 47, 19, 21], and the definition of open bisimulation between qCCS processes presented in [5]; the idea can be applied in other quantum process algebras such as CQP [24] and QPAlg [32].

We assume three types of data in qCCS: Bool for booleans, real numbers Real for classical data, and qubits Qbt for quantum data. Let $cVar$, ranged over by $x, y, \ldots$, be the set of classical variables, and $qVar$, ranged over by $q, r, \ldots$, the set of quantum variables. It is assumed that $cVar$ and $qVar$ are both countably infinite. We assume a set Exp of classical data expressions over `Real`, which includes $cVar$ as a subset and is ranged over by $e, e', \ldots$, and a set of boolean-valued expressions $BExp$, ranged over by $b, b', \ldots$. We further assume that only classical variables can occur free in both data expressions and boolean expressions. Let $cChan$ be the set of classical channel names, ranged over by $c, d, \ldots$, and $qChan$ the set of quantum channel names, ranged over by $\mathsf{c}, \mathsf{d}, \ldots$. We often abbreviate a sequence of distinct variables $\{q_1, \ldots, q_n\}$ into $\widetilde{q}$.

Based on these notations, the syntax of qCCS terms can be given by the Backus-Naur form

$$\begin{array}{lcl} U & ::= & \mathbf{nil} \mid K(\widetilde{e}, \widetilde{q}) \mid \alpha.U \mid U + U \mid U \| U \mid \mathbf{if}\ b\ \mathbf{then}\ U \\ \alpha & ::= & \tau \mid c?x \mid c!e \mid \mathsf{c}?q \mid \mathsf{c}!q \mid \mathcal{E}[\widetilde{q}] \mid M[\widetilde{q}; x] \end{array}$$

where $c \in cChan$, $x \in cVar$, $\mathsf{c} \in qChan$, $q \in qVar$, $\widetilde{q} \subseteq qVar$, $e \in Exp$, $\widetilde{e} \subseteq Exp$, τ is the silent action, $b \in BExp$, $K(\widetilde{x}, \widetilde{q})$ is a process constant with a defining equation $K(\widetilde{x}, \widetilde{q}) \stackrel{def}{=} U$, and $\mathcal{E}$ and M are respectively a trace-preserving super-operator and a non-degenerate projective measurement applying on the Hilbert space associated with the systems $\widetilde{q}$. In this paper, we assume all super-operators are completely positive.

The notion of free classical variables in quantum processes, denoted by $fv(\cdot)$, can be defined in the usual way with the only modification that the quantum measurement prefix $M[\widetilde{q}; x]$ has binding power on x. A quantum process term U is closed if $fv(U) = \emptyset$. We let $\mathcal{U}$, ranged over by $U, V, \cdots$, be the set of all qCCS terms, and $\mathcal{P}$, ranged over by $P, Q, \cdots$, the set of closed terms.

The process constructs we give here are quite similar to those in classical CCS, and they also have similar intuitive meanings: **nil** stands for a process which does not perform any action; $c?x$ and $c!e$ are respectively classical input and classical output, while $\mathsf{c}?q$ and $\mathsf{c}!q$ are their quantum counterparts. $\mathcal{E}[\widetilde{q}]$ denotes the action of performing the super-operator $\mathcal{E}$ on the qubits $\widetilde{q}$ while $M[\widetilde{q};x]$ measures the qubits $\widetilde{q}$ according to M and the measurement outcome is substituted for the classical variable x. The binary sum $+$ models nondeterministic choice: $U+V$ behaves like either U or V depending on the choice of the environment. $\|$ denotes the usual parallel composition. Finally, **if** b **then** U is the standard conditional choice where U can be executed only if b is $\mathtt{tt}$.

We now turn to the operational semantics of qCCS. For each quantum variable $q \in qVar$, we assume a 2-dimensional Hilbert space $\mathcal{H}_q$ to be the state space of the q-system. For any $S \subseteq qVar$, we denote $\mathcal{H}_S = \bigotimes_{q\in S}\mathcal{H}_q$. In particular, $\mathcal{H} = \mathcal{H}_{qVar}$ is the state space of the whole environment consisting of all the quantum variables. Note that $\mathcal{H}$ is a countably-infinite dimensional Hilbert space.

Suppose P is a closed quantum process. A pair of the form $\langle P,\rho\rangle$ is called a *configuration*, where $\rho \in \mathcal{D}(\mathcal{H})$ is a density operator on $\mathcal{H}$ (As $\mathcal{H}$ is infinite dimensional, ρ should be understood as a density operator on some finite dimensional subspace of $\mathcal{H}$ which contains $\mathcal{H}_{qv(P)}$). The set of configurations is denoted by Con, and ranged over by $\mathcal{C},\mathcal{D},\cdots$. Let

$$Act \quad = \quad \{\tau\} \cup \{c?v, c!v \mid c \in cChan, v \in \mathsf{Real}\} \cup \{\mathsf{c}?r, \mathsf{c}!r \mid \mathsf{c} \in qChan, r \in qVar\}.$$

Let $\mathcal{D}(Con)$, ranged over by $\Delta,\Theta,\cdots$, be the set of all finite-supported probabilistic distributions over Con. Then the operational semantics of qCCS can be given by the pLTS $\langle Con, Act, \longrightarrow\rangle$, where $\longrightarrow \subseteq Con \times Act \times \mathcal{D}(Con)$ is the smallest relation satisfying some inference rules. Here we select two rules related to super-operator application and quantum measurements; the others can be found in [5].

$$\textit{(Oper)}\quad \langle \mathcal{E}[\tilde{q}].P,\rho\rangle \xrightarrow{\tau} \langle P, \mathcal{E}_{\tilde{q}}(\rho)\rangle \qquad\qquad \textit{(Meas)}\quad \frac{M=\sum_{i\in I}\lambda_i E^i \qquad p_i = tr(E^i_{\tilde{q}}\rho)}{\langle M[\widetilde{q};x].P,\rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i\langle P[\lambda_i/x], E^i_{\widetilde{q}}\rho E^{i\dagger}_{\widetilde{q}}/p_i\rangle}$$

In rule *(Meas)*, $E^i_{\tilde{q}}$ denotes the operator E^i acting on the quantum systems $\tilde{q}$ and $tr(E^i_{\tilde{q}}\rho)$ stands for the trace of $E^i_{\tilde{q}}\rho$. This rule tells us that a measurement on the quantum system $\tilde{q}$ entails a probabilistic transition; each candidate configuration $\langle P[\lambda_i/x], E^i_{\tilde{q}}\rho E^{i\dagger}_{\tilde{q}}/p_i\rangle$ occurs with probability $tr(E^i_{\tilde{q}}\rho)$.

Let $\mathcal{C} = \langle P,\rho\rangle$. We use the notation $qv(\mathcal{C}) := qv(P)$ for free quantum variables and $\mathrm{env}(\mathcal{C}) := \mathrm{tr}_{qv(P)}(\rho)$ for partial traces. Let $\Delta = \sum_{i\in I} p_i \cdot \overline{\langle P_i,\rho_i\rangle}$. We write $\mathcal{E}(\Delta)$ for the distribution $\sum_{i\in I} p_i \cdot \overline{\langle P_i, \mathcal{E}(\rho_i)\rangle}$. In addition, we let $qv(\Delta) := \bigcup_{i\in I} qv(P_i)$ and $\mathrm{env}(\Delta) := \sum_{i\in I} p_i \cdot \mathrm{tr}_{qv(P_i)}(\rho_i)$.

▶ **Definition 9.** A symmetric relation $\mathcal{R} \subseteq Con \times Con$ is called an open bisimulation if for any $\mathcal{C},\mathcal{D} \in Con$, $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies that

(1) $qv(\mathcal{C}) = qv(\mathcal{D})$, and $\mathrm{env}(\mathcal{C}) = \mathrm{env}(\mathcal{D})$,

(2) for any trace-preserving super-operator $\mathcal{E}$ acting on $\mathcal{H}_{\overline{qv(\mathcal{C})}}$ (Again, $\mathcal{E}$ should be understood as a super-operator on some finite dimensional subspace of $\mathcal{H}_{\overline{qv(\mathcal{C})}}$), whenever $\mathcal{E}(\mathcal{C}) \xrightarrow{\alpha} \Delta$, there exists Θ such that $\mathcal{E}(\mathcal{D}) \xrightarrow{\alpha} \Theta$ and $\Delta\ \mathcal{R}^\dagger\ \Theta$.

Two quantum configurations $\mathcal{C}$ and $\mathcal{D}$ are open bisimilar, denoted by $\mathcal{C} \sim_o \mathcal{D}$, if there exists an open bisimulation $\mathcal{R}$ such that $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$.

Here we are using exactly the same lifting operation as that in the probabilistic case (cf. Definition 2). The above definition is inspired by the work of Sangiorgi [42], where a

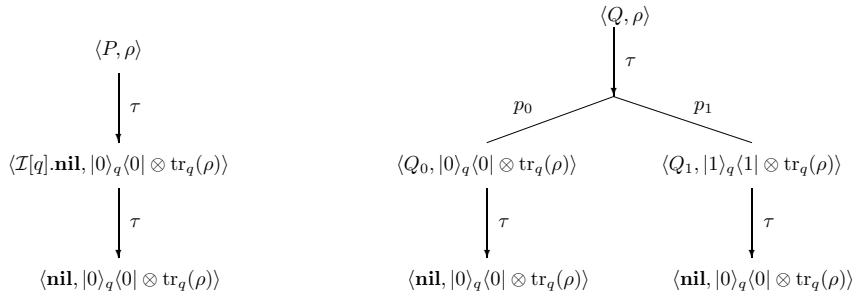


Figure 2 pLTSs for the two ways of setting a quantum system to $|0\rangle$.

notion of bisimulation is defined for the π-calculus [38, 42] by treating name instantiation in an "open" style (name instantiation happens before any transition). Here we deal with super-operator application in an "open" style, but the instantiation of variables can be in an "early" style (variables are instantiated when input actions are performed). For example, the operational semantics given in [5] is essentially an early semantics.

To illustrate the operational semantics and open bisimulation presented in this section, we give a simple example.

▶ **Example 10.** This example shows two alternative ways of setting a quantum system to the pure state $|0\rangle$. Let $P \stackrel{def}{=} Set^0[q].\mathcal{I}[q].\mathbf{nil}$ and

$$Q \stackrel{def}{=} M_{0,1}[q;x].(\mathbf{if}\ x=0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ x=1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}),$$

where $Set^0 = \{|0\rangle\langle 0|, |0\rangle\langle 1|\}$, $M_{0,1}$ is the 1-qubit measurement according to the computational basis $\{|0\rangle, |1\rangle\}$, $\mathcal{I}$ is the identity super-operator, and $\mathcal{X}$ is the Pauli-X super-operator. For any $\rho \in \mathcal{D}(\mathcal{H})$, the pLTSs rooted by $\langle P, \rho\rangle$ and $\langle Q, \rho\rangle$ respectively are depicted in Figure 2 where

$$\begin{aligned} Q_0 &\stackrel{def}{=} \mathbf{if}\ 0=0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ 0=1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}, \\ Q_1 &\stackrel{def}{=} \mathbf{if}\ 1=0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ 1=1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}, \end{aligned}$$

and $p_i = \mathrm{tr}(|i\rangle\langle i|_q \cdot \rho)$. Note that both P and Q are free of quantum input. We can show $P \sim_o Q$ easily by verifying that the relation $\mathcal{R} \cup \mathcal{R}^{-1}$, where

$$\begin{aligned} \mathcal{R} = \{&(\langle P, \rho\rangle, \langle Q, \rho\rangle), (\langle \mathcal{I}[q].\mathbf{nil}, \rho_0\rangle, \langle Q_0, \rho_0\rangle), \\ &(\langle \mathcal{I}[q].\mathbf{nil}, \rho_0\rangle, \langle Q_1, \rho_1\rangle), (\langle \mathbf{nil}, \rho_0\rangle, \langle \mathbf{nil}, \rho_0\rangle) : \rho \in \mathcal{D}(\mathcal{H})\} \end{aligned}$$

and $\rho_i = |i\rangle\langle i|_q \otimes \mathrm{tr}_q\rho$, is an open bisimulation.

3.2 A Useful Proof Technique

In Definition 9 super-operator application and transitions are considered at the same time. In fact, we can separate the two issues and approach the concept of open bisimulation in an incremental way, which turns out to be very useful when proving that two configurations are bisimilar.

▶ **Definition 11.** A relation $\mathcal{R} \subseteq Con \times Con$ is closed under super-operator application if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies $\mathcal{E}(\mathcal{C})\ \mathcal{R}\ \mathcal{E}(\mathcal{D})$ for any trace-preserving super-operator $\mathcal{E}$ acting on $\mathcal{H}_{\overline{qv(\mathcal{C})}}$.

▶ **Definition 12.** A relation $\mathcal{R} \subseteq Con \times Con$ is a *ground simulation* if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies that $qv(\mathcal{C}) = qv(\mathcal{D})$, $\mathrm{env}(\mathcal{C}) = \mathrm{env}(\mathcal{D})$, and

- whenever $\mathcal{C} \xrightarrow{\alpha} \Delta$, there is some distribution Θ with $\mathcal{D} \xrightarrow{\alpha} \Theta$ and $\Delta\ \mathcal{R}^{\dagger}\ \Theta$.

A relation $\mathcal{R}$ is a *ground bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are ground simulations.

The following property is shown in [5].

▶ **Proposition 13.** *$\sim_o$ is the largest ground bisimulation that is closed under all super-operator applications.*

Proposition 13 provides us with a useful proof technique: in order to show that two configurations $\mathcal{C}$ and $\mathcal{D}$ are open bisimilar, it suffices to exhibit a binary relation including the pair $(\mathcal{C}, \mathcal{D})$, and then to check that the relation is a ground bisimulation and is closed under all super-operator application. This is analogous to a proof technique of open bisimulation for the π-calculus [42], where name instantiation is playing the same role as super-operator application here.

3.3 Distribution-Based Quantum Bisimulation

The distribution-based bisimulation defined in Section 2.4 can also be extended to the quantum setting.

▶ **Definition 14.** A relation $\mathcal{R} \subseteq \mathcal{D}(Con) \times \mathcal{D}(Con)$ is a *distribution-based ground simulation* if $\Delta\ \mathcal{R}\ \Theta$ implies that $qv(\Delta) = qv(\Theta)$, $\mathrm{env}(\Delta) = \mathrm{env}(\Theta)$, and

- whenever $\Delta \xrightarrow{\alpha} \Delta'$, there is some subdistribution Θ' with $\Theta \xrightarrow{\alpha} \Theta'$ and $\Delta'\ \mathcal{R}\ \Theta'$.

A relation $\mathcal{R}$ is a *distribution-based ground bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are distribution-based ground simulations.

A relation $\mathcal{R}$ is a distribution-based bisimulation if it is a distribution-based ground bisimulation, and is closed under super-operator applications.

Note that the distribution-based bisimulation given in Definition 14 is slightly coarser than that considered in [22], for the same reason as the comparison of the corresponding probabilistic bisimulations [16].

In quantum mechanics, a fundamental principle is the no-cloning theorem of quantum resources. From a type-theoretic point of view, quantum resources are linear and can be described by linear types in quantum programming languages. How to define appropriate program equivalences for this kind of languages is an interesting problem. In [8] a linear contextual equivalence is introduced to compare the behaviour of quantum programs. Two notions of bisimilarity, a state-based and a distribution-based are introduced as proof techniques for reasoning about higher-order quantum programs. Both notions of bisimilarity are sound with respect to the linear contextual equivalence, but only the distribution-based one turns out to be complete.

3.4 Symbolic Bisimulations

The quantum bisimulations introduced so far, either state-based or distribution-based, are generalised from the corresponding probabilistic bisimulations naturally and smoothly. A major problem with them is that they all resort to the instantiation of quantum variables by quantum states. As a result, to check whether or not two processes are bisimilar, we have to accompany them with arbitrarily chosen quantum states, and check if the resultant configurations are bisimilar. Note that all quantum states constitute a continuum. Therefore, it seems that the verification of quantum bisimulations is infeasible from an algorithmic point of view.

Recall that for classical process algebras, Hennessy and Lin [27] introduced a notion of symbolic bisimulation to deal with possibly infinite classical data sets. As a quantum extension of value-passing CCS, the quantum process algebra qCCS has both (possibly infinite) classical data domain and (doomed-to-be infinite) quantum data domain. To overcome the additional difficulty caused by the infinity of all quantum states, we can make use of super-operator valued distributions, which allow us to fold the operational semantics of qCCS into a symbolic version and thus provide us with a notion of symbolic bisimulation. To check the symbolic bisimilarity of two quantum processes, only a finite number of process-superoperator pairs need to be considered, without appealing to quantum states. This idea has been successful in developing an algorithm to check the state-based ground bisimulation for quantum processes [20]. It would be interesting to purse this line of research so as to develop algorithms of checking the symbolic versions of other quantum bisimulations.

4 Concluding Remarks

We have briefly reviewed a few ingredients for formulating reasonable notions of probabilistic and quantum bisimulations.

(1) The lifting operation is the key of defining state-based probabilistic and quantum bisimualtions. It is mathematically interesting in itself because of the close connection with the Kantorovich metric and the maximum network flow problem.

(2) Distribution-based bisimulation is more relevant to quantum processes because it offers a coinductive proof technique for linear contextual equivalene, and linear resources are prominent in quantum computation.

(3) The symbolic approach is promising to yield feasible algorithms of checking quantum bisimulations.

There is a huge amount of literature on probabilistic bisimulations, and the current paper is by no means a complete survey. A more detailed account of probabilistic bisimulations is given in [4, Chapter3]. For quantum processes, a branching bisimulation is firstly proposed in [35]. However, it is not a congruence because it is not preserved by parallel composition. Quantum bisimulations that are congruence relations are given in [19, 20] and independently in [3]. Both of them are defined for concrete quantum transition systems, and are difficult to check with algorithms, which motivated the introduction of symbolic bisimulations for quantum processes [20].

In [34] a semi-automated tool is developed to verify security proofs based on a weak bisimulation similar to that given in Definition 9 for a finite fragment of qCCS. In that tool, security parameters and quantum states are represented as symbols, and some user-defined equations are used as rewriting rules for simplification. This differs from the symbolic semantics discussed in Section 3.4 as the latter is more in line with the idea investigated in [27] for value-passing CCS.

In the future, we believe that distribution-based symbolic bisimulations would be promising to be used in software tools in support of verifying quantum communication protocols. Some efforts are made in [22], which considers distribution-based bisimulations and the proofs are manual when reasoning about the behavioural equivalence of quantum processes. In order to deal with advanced protocols such as the quantum key distribution protocol BB84 [2], it would be helpful to have some tool support, for which symbolic semantics could play a role.

References

1 Christel Baier, Bettina Engelen, and Mila E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.

2 C.H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, 1984.

3 T.A.S. Davidson. *Formal verification techniques using quantum process calculus*. PhD thesis, University of Warwick, 2011.

4 Yuxin Deng. *Semantics of Probabilistic Processes: An Operational Approach*. Springer, 2015.

5 Yuxin Deng and Yuan Feng. Open bisimulation for quantum processes. In *Proceedings of the 7th IFIP International Conference on Theoretical Computer Science*, volume 7604 of *LNCS*, pages 119–133. Springer, 2012.

6 Yuxin Deng and Yuan Feng. Bisimulations for probabilistic linear lambda calculi. In *Proceedings of the 11th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 1–8. IEEE Computer Society, 2017.

7 Yuxin Deng and Yuan Feng. Probabilistic bisimilarity as testing equivalence. *Information and Computation*, 257:58–64, 2017.

8 Yuxin Deng, Yuan Feng, and Ugo Dal Lago. On coinduction and quantum lambda calculi. In *Proceedings of the 26th International Conference on Concurrency Theory*, volume 42 of *LIPIcs*, pages 427–440. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

9 Yuxin Deng and Matthew Hennessy. On the semantics of Markov automata. *Information and Computation*, 222:139–168, 2013.

10 Yuxin Deng and Rob van Glabbeek. Characterising probabilistic processes logically. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 6397 of *LNCS*, pages 278–293. Springer, 2010.

11 Yuxin Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Testing finitary probabilistic processes (extended abstract). In *Proceedings of the 20th International Conference on Concurrency Theory*, volume 5710 of *LNCS*, pages 274–288. Springer, 2009.

12 Yuxin Deng and Hengyang Wu. Modal characterisations of probabilistic and fuzzy bisimulations. In *Proceedings of the 16th International Conference on Formal Engineering Methods*, volume 8829 of *LNCS*, pages 123–138. Springer, 2014.

13 Josée Desharnais. *LabelledMarkovProcesses*. PhD thesis, McGillUniversity, 1999.

14 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, 2002.

15 Josée Desharnais, V. Gupta, R. Jagadeesan, and Prakash Panangaden. Approximating labelled Markov processes. *Information and Computation*, 184(1):160–200, 2003.

16 Wenjie Du, Yuxin Deng, and Daniel Gebler. Behavioural pseudometrics for nondeterministic probabilistic systems. In *Proceedings of the the 2nd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, volume 9984 of *LNCS*, pages 67–84. Springer, 2016.

17 Christian Eisentraut, Jens Chr. Godskesen, Holger Hermanns, Lei Song, and Lijun Zhang. Probabilistic bisimulation for realistic schedulers. In *Proceedings of the 20th International Symposium on Formal Methods*, volume 9109 of *LNCS*, pages 248–264. Springer, 2015.

18 Y Feng, R Duan, Z Ji, and M Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.

19 Y Feng, R Duan, and M Ying. Bisimulations for quantum processes. In Mooly Sagiv, editor, *Proceedings of the 38th ACM Symposium on Principles of Programming Languages*, pages 523–534, Austin, Texas, USA, 2011.

20 Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):1–32, 2014.

21 Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for Quantum Processes. *ACM Transactions on Programming Languages and Systems*, 34(4):1–43, 2012.

22 Yuan Feng and Mingsheng Ying. Toward automatic verification of quantum cryptographic protocols. In *26th International Conference on Concurrency Theory*, volume 42 of *LIPIcs*, pages 441–455. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

23 Yuan Feng and Lijun Zhang. When equivalence and bisimulation join forces in probabilistic automata. In *Proceedings of the 19th International Symposium on Formal Methods*, volume 8442 of *LNCS*, pages 247–262. Springer, 2014.

24 Simon J. Gay and Rajagopal Nagarajan. Communicating quantum processes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 145–157. ACM, 2005.

25 M Hennessy and A. Ingólfsdóttir. A theory of communicating processes value-passing. *Information and Computation*, 107(2):202–236, 1993.

26 Matthew Hennessy. Exploring probabilistic bisimulations, part I. *Formal Aspects of Computing*, 24(4-6):749–768, 2012.

27 Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

28 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

29 Holger Hermanns, Jan Krcál, and Jan Kretínský. Probabilistic bisimulation: Naturally on distributions. In *Proceedings of the 25th International Conference on Concurrency Theory*, volume 8704 of *LNCS*, pages 249–265. Springer, 2014.

30 Holger Hermanns, Augusto Parma, Roberto Segala, Björn Wachter, and Lijun Zhang. Probabilistic logical characterization. *Information and Computation*, 209(2):154–172, 2011.

31 C. Jones. *Probabilistic nondeterminism*. PhD thesis, University of Edinburgh, 1990.

32 Philippe Jorrand and Marie Lalire. Toward a quantum process algebra. In *Proceedings of the First Conference on Computing Frontiers*, pages 111–119. ACM, 2004.

33 L. Kantorovich. On the transfer of masses (in Russian). *Doklady Akademii Nauk*, 37(2):227–229, 1942.

34 Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, and Hideki Sakurada. Semi-automated verification of security proofs of quantum cryptographic protocols. *Journal of Symbolic Computation*, 73:192–220, 2016.

35 Marie Lalire. Relations among quantum processes: bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006.

36 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

37 Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

38 Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

39 David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

40 Augusto Parma and Roberto Segala. Logical characterizations of bisimulations for discrete probabilistic systems. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures*, volume 4423 of *LNCS*, pages 287–301. Springer, 2007.

41 J. Sack and Lijun Zhang. A general framework for probabilistic characterizing formulae. In *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *LNCS*, pages 396–411. Springer, 2012.

42 Davide Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Informatica*, 33(1):69–97, 1996.

43 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings of the 5th International Conference on Concurrency Theory*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.

44 Franck van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theoretical Computer Science*, 333(1-2):171–197, 2005.

45 Franck van Breugel and James Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 336–350. Springer, 2001.

46 Rob J. van Glabbeek, Scott A. Smolka, Bernhard Steffen, and Chris M. N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, pages 130–141. IEEE Computer Society, 1990.

47 M Ying, Y Feng, R Duan, and Z Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic*, 10(3):1–36, 2009.

48 Lijun Zhang, Holger Hermanns, Friedrich Eisenbrand, and David N. Jansen. Flow faster: Efficient decision algorithms for probabilistic simulations. *Logical Methods in Computer Science*, 4(4):1–43, 2008.

Is Speed-Independent Mutual Exclusion Implementable?

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

Abstract

A mutual exclusion algorithm is called speed independent if its correctness does not depend on the relative speed of the components. Famous mutual exclusion protocols such as Dekker's, Peterson's and Lamport's bakery are meant to be speed independent.

In this talk I argue that speed-independent mutual exclusion may not be implementable on standard hardware, depending on how we believe reading and writing to a memory location is really carried out. It can be implemented on electrical circuits, however.

This builds on previous work showing that mutual exclusion cannot be accurately modelled in standard process algebras.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Mutual exclusion, speed independence, concurrent reading and writing, liveness, justness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.3

Category Invited Talk

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 3; pp. 3:1–3:1

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

Verifying Arithmetic Assembly Programs in Cryptographic Primitives

Andy Polyakov
The OpenSSL project
appro@openssl.org

Ming-Hsien Tsai[1]
Academia Sinica, Taiwan
mhtsai208@gmail.com

Bow-Yaw Wang[2]
Academia Sinica, Taiwan
bywang@iis.sinica.edu.tw

Bo-Yin Yang[3]
Academia Sinica, Taiwan
by@crypto.tw

Abstract

Arithmetic over large finite fields is indispensable in modern cryptography. For efficienty, these operations are often implemented in manually optimized assembly programs. Since these arithmetic assembly programs necessarily perform lots of non-linear computation, checking their correctness is a challenging verification problem. We develop techniques to verify such programs automatically in this paper. Using our techniques, we have successfully verified a number of assembly programs in OpenSSL. Moreover, our tool verifies the boringSSL Montgomery Ladderstep (about 1400 assembly instructions) in 1 hour. This is by far the fastest verification technique for such programs.

2012 ACM Subject Classification Software and its engineering → Formal software verification

Keywords and phrases Formal verification, Cryptography, Assembly Programs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.4

Category Invited Talk

1 Introduction

Cryptographic primitives are the building blocks of computer security. They are indispensable in various encryption, authentication, and key exchange protocols. Underneath these critical primitives, arithmetic over large finite fields is necessitated by modern cryptography. Efficiency of arithmetic operations such as addition and multiplication is crucial to practical applicability of cryptographic primitives due to their wide usage. Consequently, these operations are implemented by low-level languages (such as C or assembly). Indeed, more than forty arithmetic assembly subroutines for different architectures are found in the OpenSSL NIST

[1] partially supported by the Academia Sinica Project AS-106-TP-A06
[2] partially supported by the Academia Sinica Project AS-106-TP-A06
[3] partially supported by the MOST Project 105-2221-E-001-014-MY3

P-256 elliptic curve cryptographic library. Each is manually optimized to attain the best performance.

Since cryptographic primitives rely on arithmetic operations over large finite fields, correct implementations of such operations are essential to computer security. For implementations written in C, source codes are compiled into executable binary codes. Certified compilation (for instance, the CompCert project) is necessary to ensure correctness after C codes are verified. Even so, hand-optimized assembly implementations are still significantly more efficient than C implementations (up to two times faster for the OpenSSL NIST P-256 Intel Broadwell microarchitecture). Manually optimized assembly implementations for arithmetic operations are typical in practical cryptography. We verify such low-level codes for cryptographic primitives in this paper.

Several obstacles must be overcome. Different architectures have different instruction sets. Even for the same architecture (x86_64), different microarchitectures may have different instruction sets (Broadwell versus its predecessors). Since one would like to develop verification techniques across different instruction sets, a unified framework is preferred. Code sizes also vary from operations significantly. From a dozen of instructions (multiplication by two) to more than a thousand (group operation in elliptic curves), these assembly implementations must be verified with reasonable resources. Last but not least, several assembly programs realize non-linear multiplication over large finite fields. Such algebraic properties are hard to verify by bit blasting. Existing SMT solvers for the bit-vector theory cannot verify the multiplication of two 256-bit numbers. New techniques have to be developed for assembly codes in this special domain.

We propose a domain specific language CryptoLine for modeling assembly programs across different architectures. The language contains instructions used in implementations of arithmetic operations. Different from assembly languages, operands and flags are explicit in CryptoLine for clarity. CryptoLine moreover allows users to specify program properties with assertions, pre- and post-conditions. Using CryptoLine, assembly programs of different arithmetic operations from the OpenSSL cryptographic library are modeled and specified. We feel CryptoLine is a suitable abstraction for assembly programs in cryptographic primitives.

Rewriting assembly codes in CryptoLine can be a daunting task. Moreover, assembly programs can be written in various ways. For example, assembly code fragments can be put in the `asm` statement in GNU C compilers; the OpenSSL library allows programmers to write portable assembly codes with its Perl scripts. Instead of writing parsers for various assembly development environments, we extract assembly codes from execution traces. Since CryptoLine is designed to model assembly programs, each assembly instruction can be translated to CryptoLine statements straightforwardly. Using a simple Python script, extracted assembly codes are converted to CryptoLine programs automatically. Users can transform their assembly programs to CryptoLine programs for verification via simple scripts. Usability of our work is greatly improved.

For verification, we consider conjunctions of range and algebraic predicates. Range predicates specify program variable ranges with the unsigned bit-vector theory; they are resolved by SMT solvers rather straightforwardly. Algebraic predicates specify algebraic properties among program variables; they are reduced to instances of the ideal membership problem through a series of transformations. Instances of the ideal membership problem are sent to and solved by computer algebra systems. Our hybrid technique decomposes verification problems and takes advantages of recent developments in SMT solving and computer algebra. It opens new opportunities in verifying non-linear arithmetic computation in cryptographic programs.

We report case studies in OpenSSL, boringSSL, and mbedTLS. For OpenSSL, we verify eight assembly subroutines converted from execution traces in the OpenSSL NIST P-256 cryptographic library. These subroutines perform arithmetic computation including but not limited to square and multiplication over the Galois field $GF(2^{256}-2^{224}+2^{192}+2^{96}-1)$. The Montegomery multiplication subroutine over arbitrary 256-bit Montgomery primes is also verified. For boringSSL, the multiplication and square subroutines over $GF(2^{255}-19)$ are verified, as well as the Montgomery Ladderstep subroutine for X25519 ($\approx$ 1400 instructions). For mbedTLS, the multiplication subroutine accepts any size of inputs and involves both assembly and C code. The execution traces are extracted and verified.

Related Work. A domain-specific language bvCryptoLine has been proposed to model low-level mathematical constructs in cryptographic programs [16]. Programs in bvCryptoLine can be verified automatically by a certified approach. Both the certified approach and ours reduce the verification problem to SMT problems and ideal membership problems. However, we introduce new instructions in our domain-specific language so that more low-level arithmetic programs can be verified. Unlike the certified approach which verifies programs in bvCryptoLine, we target on real industrial programs and provide scripts for the extraction of assembly codes from execution traces and for the translation from assembly codes to our domain-specific language. Although our approach is not certified, it verifies programs much faster than the certified approach does. The tool `gfverif` [6] has been used to automatically verify a C implementation of the Montgomery Ladderstep. In `gfverif`, range properties and algebraic properties are verified separatedly by a specialized range analysis and by the Sage computer-algebra system. A drawback of `gfverif` is that programs not written in the constructs provided by `gfverif` cannot be verified.

A hand-optimized assembly implementation of the Montgomery Ladderstep has been verified by a semi-automatic approach [8] with SMT solvers. As non-linear arithmetic operations are hard for SMT solvers to verify, the semi-automatic approach requires manual program annotation to reduce verification problems to smaller ones and Coq proofs for some theorems about modulo operation. Similarly, several mathematical constructs have been re-implemented in F* [18] and Vale [7] and to be verified using a combination of SMT solving and manual proofs.

Fiat-Crypto can synthesize correct-by-construction assembly codes for mathematical constructs but the synthesized codes are not as efficient as hand-optimized assembly implementations [9]. Various implementations of mathematical constructs, hash functions, and random number generators have been formalized and manually verified in proof assistants [1, 3, 2, 14, 13, 4, 5, 17]. Cryptol/SAW can automatically verify several cryptographic implementations in C and Java against their reference implementations but the correctness of the reference implementations is not proven [15].

After preliminaries (Section 2), the domain specific language CryptoLine for cryptographic assembly programs is presented in Section 3. Our verification algorithm is given in Section 4. Case studies are reported in Section 5. Section 6 concludes the presentation.

2 Preliminaries

Let $\mathbb{N}$ be the set of non-negative integers and $\mathbb{Z}$ the set of integers. $[n] = \{0, 1, \ldots, n\}$ for $n \in \mathbb{N}$. Fix a set $\vec{v} = \{x, y, z, \ldots\}$ of variables and a set $\vec{c} = \{a, b, c, \ldots\}$ of carry flags such that $\vec{v} \cap \vec{c} = \emptyset$. Let $\vec{x} = \vec{v} \cup \vec{c}$. $\mathbb{Z}[\vec{x}]$ denotes the set of polynomials over $\vec{x}$ with coefficients in $\mathbb{Z}$. A set $I \subseteq \mathbb{Z}[\vec{x}]$ is an *ideal* if $f + g \in I$ for every $f, g \in I$; and $h \times f \in I$ for every $h \in \mathbb{Z}[\vec{x}]$ and $f \in I$. Given $G \subseteq \mathbb{Z}[\vec{x}]$, $\langle G \rangle$ is the minimal ideal containing G; G are the *generators* of $\langle G \rangle$. The *ideal membership* problem is to decide if $f \in I$ for a given ideal I and $f \in \mathbb{Z}[\vec{x}]$.

$$
\begin{array}{rcl}
\mathit{Num} & ::= & 0 \mid 1 \mid 2 \mid \cdots \\
\mathit{Var} & ::= & \vec{v} \\
\mathit{Flag} & ::= & \vec{c} \\
\mathit{Expr} & ::= & \mathit{Num} \mid \mathit{Var} \mid \mathit{Flag} \mid \\
 & & \mathit{Expr}+\mathit{Expr} \mid \mathit{Expr}-\mathit{Expr} \mid \\
 & & \mathit{Expr}\times\mathit{Expr}
\end{array}
$$

(a) Expressions.

$$
\begin{array}{rcl}
\mathit{APred} & ::= & \top \mid \mathit{APred} \wedge \mathit{APred} \mid \\
 & & \mathit{Expr} = \mathit{Expr} \mid \mathit{Expr} \equiv \mathit{Expr} \ \mathsf{mod}\ \mathit{Expr} \mid \\
\mathit{RPred} & ::= & \top \mid \mathit{RPred} \wedge \mathit{RPred} \mid \\
 & & \mathit{Expr} <_w \mathit{Expr} \mid \mathit{Expr} \leq_w \mathit{Expr}
\end{array}
$$

(b) Predicates.

Figure 1 Syntax of Expressions and Predicates.

3 Domain Specific Language – CryptoLine

CryptoLine is designed to model and specify arithmetic assembly programs in cryptograhpic primitives. Arithmetic over large finite fields is essential to modern cryptography. In practice, it is necessary to perform arithmetic computation with numbers in hundreds of bits lest security may be compromised due to cryptoanalysis. We analyze real arithmetic assembly programs, identify a small subset of assembly instructions, and formalize the subset in our domain specific language CryptoLine. In order to specify properties about programs, the language is enriched with statements like Assert and Assume. We detail the design of CryptoLine in this section.

3.1 Syntax

As in low-level cryptographic programs, numbers are non-negative in CryptoLine. The language also allows variables and binary flags. Expressions are only used in property specifications. They admit arithmetic operators $+$, $-$, and $\times$ (Figure 1a). Property specifications are divided into two classes: algebraic and range predicates. An algebraic predicate is a conjunction of equalities or modulo equalities. A range predicate is a conjunction of finite-width comparisons[4]. For instance, $\mathit{Expr} <_w \mathit{Expr}$ means the w-bit less-than relation between two expressions in Figure 1b.

CryptoLine statements contain assembly instructions used in arithmetic computation[5]. They even have a similar syntax: mnemonic, destination variables, and source arguments in order (Figure 2a). Set is the assignment statement. The Cmov statement is the conditional assignment. Mul is the half multiplication whereas Mulf is the full multiplication. Add! is the addition statement without setting the carry flag. Add is the addition statement with setting the carry flag. Adc is the addition with carry statement. Similarly, Sub! is the subtraction statement without setting the borrow flag, Sub is the subtraction with setting the borrow flag, and Sbb is the subtraction with borrow statement. A predicate consists of an algebraic and a range predicate separated by $\|$ (Figure 2b). The Assert statement asserts a predicate. The Assume statement assumes a predicate. A program is a sequence of statements. Finally, a specification contains a program with two predicates as the pre- and post-conditions.

[4] Range predicates such as negation, equality, and disjunction are also allowed in our implementation. Expressions allowed in a range predicate also include signed/unsigned remainder, bit-wise and, bit-wise or, and bit-wise xor.

[5] Instructions such as setting a binary flag, clearing a binary flag, and instructions in [16] are also allowed in our implementation.

$$
\begin{array}{lll}
\mathit{Stmt} ::= \mathsf{Set}\,\mathit{Var}\,\mathit{Arg} & | & \mathsf{Cmov}\,\mathit{Var}\,\mathit{Flag}\,\mathit{Arg}\,\mathit{Arg} \\
\quad \mathsf{Add!}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} & | & \mathsf{Add}\mathit{Flag}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} \\
\quad \mathsf{Sub!}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} & | & \mathsf{Sub}\mathit{Flag}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} \\
\quad \mathsf{Adc}\mathit{Flag}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg}\,\mathit{Flag} & | & \mathsf{Mul}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} \\
\quad \mathsf{Sbb}\mathit{Flag}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg}\,\mathit{Flag} & | & \mathsf{Mulf}\,\mathit{Var}\,\mathit{Var}\,\mathit{Arg}\,\mathit{Arg} \\
\quad \mathsf{Assert}\mathit{Pred} & | & \mathsf{Assume}\mathit{Pred}
\end{array}
$$

(a) Statements.

$$
\begin{array}{lcl}
\mathit{Arg} & ::= & \mathit{Num} \mid \mathit{Var} \\
\mathit{Prog} & ::= & \mathit{Stmt};\ \mid \mathit{Stmt};\,\mathit{Prog} \\
\mathit{Pred} & ::= & \mathit{RPred} \| \mathit{APred} \\
\mathit{Spec} & ::= & (\!|\mathit{Pred}|\!)\,\mathit{Prog}\,(\!|\mathit{Pred}|\!)
\end{array}
$$

(b) Programs and Specifications.

Figure 2 Syntax of Programs and Specifications.

```
1:  Set r0 x0;                  6:  Add! r0 r0 4503599627370458;   11:  Sub! r0 r0 y0;
2:  Set r1 x1;                  7:  Add! r1 r1 4503599627370494;   12:  Sub! r1 r1 y1;
3:  Set r2 x2;                  8:  Add! r2 r2 4503599627370494;   13:  Sub! r2 r2 y2;
4:  Set r3 x3;                  9:  Add! r3 r3 4503599627370494;   14:  Sub! r3 r3 y3;
5:  Set r5 x4;                  10: Add! r4 r4 4503599627370494;   15:  Sub! r4 r4 y4;
```

Figure 3 Subtraction *sub*.

Example. Consider the CryptoLine program for the subtraction over $GF(2^{255}-19)$ in X25519. In Figure 3, each element in the finite field is represented by five 51-bit numbers. Each 51-bit number is a *limb* of the representation. The program first assigns the minuend (x_i's) to the result (r_i's). It then adds $4503599627370458 = 2^{52} - 38$ to the lowest limb and $4503599627370494 = 2^{52} - 2$ to other limbs of the result. Finally, the program subtracts the subtrahend (y_i's) from the result.

In order to specify the subtraction program, let $\mathit{radix51}(\ell_4, \ell_3, \ell_2, \ell_1, \ell_0)$ denote

$$\ell_4 \times 2^{51\times 4} + \ell_3 \times 2^{51\times 3} + \ell_2 \times 2^{51\times 2} + \ell_1 \times 2^{51\times 1} + \ell_0.$$

That is, $\mathit{radix51}(\ell_4, \ell_3, \ell_2, \ell_1, \ell_0)$ is the element represented by ℓ_i's. We have the following specification for the program in Figure 3:

$$
\begin{array}{c}
(\!|\ \bigwedge_{i=0}^{4} x_i \leq_{64} 2^{51} + 2^{15} \wedge \bigwedge_{i=0}^{4} y_i \leq_{64} 2^{51} + 2^{15}\ \|\top|\!) \\
\mathit{sub} \\
(\!|\ \bigwedge_{i=0}^{4} r_i <_{64} 2^{54}\ \|\ \begin{array}{c} \mathit{radix51}(x_4, x_3, x_2, x_1, x_0) - \mathit{radix51}(y_4, y_3, y_2, y_1, y_0) \\ \equiv \mathit{radix51}(r_4, r_3, r_2, r_1, r_0) \bmod 2^{255} - 19 \end{array}\ |\!).
\end{array}
$$

Given each limb of the minuend and subtrahend slightly greater than 2^{51}, the specification says the subtraction program computes the difference over $GF(2^{255}-19)$ with the result less than 2^{54}. To see why the subtraction program satisfies the algebraic specification, observe that $\mathit{radix51}(2^{51}-1, 2^{51}-1, 2^{51}-1, 2^{51}-1, 2^{51}-19) = 2^{255}-19$. Hence $\mathit{radix51}(2^{52}-2, 2^{52}-2, 2^{52}-2, 2^{52}-2, 2^{52}-38) = 2\times(2^{255}-19)$. That is, the subtraction program adds $2\times(2^{255}-19)$ to the minuend and then subtracts the subtrahend. Since the algebraic specification only requires modulo equality, the program is indeed correct. Adding $2 \times (2^{255} - 19)$ does not induce the propagation of carry flags during addition. It moreover prevents borrow flag propagation during subtraction.

3.2 Semantics

Let W be a architecture-dependent parameter: $W = 2^{64}$ for 64-bit architectures; $W = 2^{32}$ for 32-bit architectures. A *state* is a mapping from *Var* to $[W-1]$ and from *Flag* to $\{0, 1\}$,

and $\bot$ is the designated *error* state. The error state does not satisfy any predicate. Figure 4 gives the operational semantics of each statement.

The semantics for CryptoLine is standard for unsigned bounded arithmetic. The Set statement updates the value of a variable in a state. Cmov updates a variable by the given flag. Add! updates a variable by the sum of arguments if the sum is smaller than W, and otherwise results in the error state (overflows in this case). Add and Adc update a flag and a variable by the sum of arguments. Sub! updates a variable by the difference of arguments if the difference is non-negative, and otherwise results in the error state (underflows in this case). Sub and Sbb update a flag and a variable by the difference of arguments. For the half multiplication Mul, it is an error if the higher bits of the product is non-zero (overflows in this case). The full multiplication Mulf always terminates successfully if the two updated variables are different. Assert results in the error state if the current state does not satisfy the given predicate. Assume ensures the new state satisfying the given predicate. Finally, the error state always propagates.

Let σ and τ be states. The semantics of a program is inductively defined as follows. $\sigma \xrightarrow{stmt} \tau$ is defined in Figure 4. $\sigma \xrightarrow{stmt;prog} \tau$ if there is a state λ such that $\sigma \xrightarrow{stmt} \lambda$ and $\lambda \xrightarrow{prog} \tau$. We call a program *prog safe* with respect to a predicate P if for every σ and τ, $\sigma \models P$ and $\sigma \xrightarrow{prog} \tau$ imply $\tau \neq \bot$. We write $\models (\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$ if for every σ, τ with $\sigma \models P \wedge Q$ and $\sigma \xrightarrow{prog} \tau$, we have $\tau \models P' \wedge Q'$. Note that $\not\models (\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$ if there is an assertion failure during execution.

Example (continued). The subtraction program will never reach the error state. For example, consider the variable r_0. The statement at line 1 assigns r_0 the value of x_0, which is smaller than $2^{51} + 2^{15}$ by the precondition. The Add! statement at line 6 assigns r_0 a value between $4503599627370458 = 2^{52} - 38$ and $2^{51} + 2^{15} + 4503599627370458 = 2^{51} + 2^{15} + 2^{52} - 38 < 2^{64}$. Thus the Add! statement never goes to the error state. Finally, observe $y_0 \leq 2^{51} + 2^{15} \leq 2^{52} - 38 \leq r_0$ after line 6. The Sub! statement at line 11 therefore never goes to the error state.

4 Verifying CryptoLine Programs

We want to check if $\models (\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$ for a given specification $(\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$ with range predicates P, P' and algebraic predicates Q, Q'. Firstly, we transform the specification to static single assignments where variables are indexed such that no input variables are assigned and every variable is assigned at most once. As CryptoLine programs are straight-line, the transformation can be done easily so that the validity of specification is preserved. In this section, we will assume the specification $(\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$ is in static single assignments and write $x^{(i)}$ to explicitly indicate a variable x with index i when needed.

Secondly, we need to ensure that all assertions in *prog* are valid. Consider the first assertion $\mathsf{Assert}P''\|Q''$ in *prog* and let $prog = prog_1; \mathsf{Assert}P''\|Q''; prog_2$. We verify the validity of this assertion by checking if $\models (\!|P\|Q|\!)prog_1(\!|P''\|Q''|\!)$ is valid. Once the assertion is valid, it is safe to be removed when verifying *prog*. Therefore all the assertion checking can be reduced to specification checking of programs without assertions. In this section, we will assume there is no assertion in *prog*.

Finally, observe $\models (\!|P\|\top|\!)prog(\!|P'\|\top|\!)$ and $\models (\!|P\|Q|\!)prog(\!|\top\|Q'|\!)$ imply $\models (\!|P\|Q|\!)prog(\!|P'\|Q'|\!)$. The specification $(\!|P\|\top|\!)prog(\!|P'\|\top|\!)$ involves only range predicates; and the specification $(\!|P\|Q|\!)prog(\!|\top\|Q'|\!)$ concerns only algebraic properties. We therefore divide the verfication task into two parts: range and algebraic properties.

$$[\![n]\!]_\sigma = n \qquad [\![x]\!]_\sigma = \sigma(x), \text{ for } x \in \vec{v} \qquad [\![c]\!]_\sigma = \sigma(c), \text{ for } c \in \vec{c}$$

σ	$\xrightarrow{\mathsf{Set}\ x\ u}$	$\sigma[x \mapsto [\![u]\!]_\sigma]$	
σ	$\xrightarrow{\mathsf{Cmov}\ x\ b\ u\ v}$	$\sigma[x \mapsto R]$	where $R = [\![u]\!]_\sigma$ if $[\![b]\!]_\sigma = 1; [\![v]\!]_\sigma$ if $[\![b]\!]_\sigma = 0$
σ	$\xrightarrow{\mathsf{Add!}\ x\ u\ v}$	σ'	where $R = [\![u]\!]_\sigma + [\![v]\!]_\sigma$ and $\sigma' = \sigma[x \mapsto R]$ if $R/W = 0; \bot$ otherwise
σ	$\xrightarrow{\mathsf{Add}\ b\ x\ u\ v}$	$\sigma[b, x \mapsto R/W, R \bmod W]$	where $R = [\![u]\!]_\sigma + [\![v]\!]_\sigma$
σ	$\xrightarrow{\mathsf{Adc}\ b\ x\ u\ v\ c}$	$\sigma[b, x \mapsto R/W, R \bmod W]$	where $R = [\![u]\!]_\sigma + [\![v]\!]_\sigma + [\![c]\!]_\sigma$
σ	$\xrightarrow{\mathsf{Sub!}\ x\ u\ v}$	σ'	where $R = [\![u]\!]_\sigma - [\![v]\!]_\sigma$ and $\sigma' = \sigma[x \mapsto R]$ if $[\![u]\!]_\sigma \geq [\![v]\!]_\sigma; \bot$ otherwise
σ	$\xrightarrow{\mathsf{Sub}\ b\ x\ u\ v}$	$\sigma[b, x \mapsto B, R \bmod W]$	where $R = [\![u]\!]_\sigma - [\![v]\!]_\sigma + W$ and $B = 0$ if $[\![u]\!]_\sigma \geq [\![v]\!]_\sigma; 1$ otherwise
σ	$\xrightarrow{\mathsf{Sbb}\ b\ x\ u\ v\ c}$	$\sigma[b, x \mapsto B, R \bmod W]$	where $R = [\![u]\!]_\sigma - [\![v]\!]_\sigma - [\![c]\!]_\sigma + W$ and $B = 0$ if $[\![u]\!]_\sigma \geq [\![v]\!]_\sigma + [\![c]\!]_\sigma; 1$ otherwise
σ	$\xrightarrow{\mathsf{Mul}\ x\ u\ v}$	σ'	where $R = [\![u]\!]_\sigma \times [\![v]\!]_\sigma$ and $\sigma' = \sigma[x \mapsto R]$ if $R/W = 0; \bot$ otherwise
σ	$\xrightarrow{\mathsf{Mulf}\ x\ y\ u\ v}$	$\sigma[x, y \mapsto R/W, R \bmod W]$	if $x \neq y$ and $R = [\![u]\!]_\sigma \times [\![v]\!]_\sigma$
σ	$\xrightarrow{\mathsf{Assert}P \| Q}$	σ'	where $\sigma' = \sigma$ if $\sigma \models P \wedge Q; \bot$ otherwise
σ	$\xrightarrow{\mathsf{Assume}P \| Q}$	σ	if $\sigma \models P \wedge Q$
$\bot$	$\xrightarrow{stmt}$	$\bot$	where $stmt \in Stmt$

Figure 4 Semantics.

4.1 Range Properties

Range properties are amenable to analysis by bit blasting. We therefore reduce the problem of deciding $\models (\!|P\|\top|\!)prog(\!|P'\|\top|\!)$ to SMT solving. More specifically, we construct a formula Ψ in the bit vector theory such that Ψ is satisfiable if and only if $\not\models (\!|P\|\top|\!)prog(\!|P'\|\top|\!)$. An SMT solver is then employed to check range properties. Since the reduction is standard, details are omitted here (see, for instance, [12]). Note that $\not\models (\!|P\|\top|\!)prog(\!|P'\|\top|\!)$ may be caused by the violation of the post-condition or by the violation of *program safety* (that is, overflow/underflow of some program statement). Therefore, safety check of *prog* is also encoded in Ψ by the same way as in [16]) and $\models (\!|P\|\top|\!)prog(\!|P'\|\top|\!)$ actually indicates that *prog* is safe with respect to P.

4.2 Algebraic Properties

Algebraic properties, especially those involving non-linear multiplication, are not suitable for SMT solving. In order to effectively verify algebraic properties, we propose modular polynomial abstraction. In modular polynomial abstraction, program behaviors are modeled by solutions to systems of modular polynomial equations. Checking algebraic properties is reduced to modular polynomial equation entailments. The modular polynomial equation entailment problem in turn is reduced to the ideal membership problem. The ideal membership problem is widely studied in commutative algebra. Decision procedures for the ideal membership problem are available in computer algebra systems. We hence employ computer algebra systems to verify algebraic properties.

Set $x\ u$	$\hookrightarrow$	$x - u = 0$
Cmov $x\ b\ u\ v$	$\hookrightarrow$	$x - (b \times u + (1 - b) \times v) = 0$
Add! $x\ u\ v$	$\hookrightarrow$	$x - (u + v) = 0$
Add $b\ x\ u\ v$	$\hookrightarrow$	$(x + b \times W) - (u + v) = 0 \wedge b \times (1 - b) = 0$
Adc $b\ x\ u\ v\ c$	$\hookrightarrow$	$(x + b \times W) - (u + v + c) = 0 \wedge b \times (1 - b) = 0$
Sub! $x\ u\ v$	$\hookrightarrow$	$x - (u - v) = 0$
Sub $b\ x\ u\ v$	$\hookrightarrow$	$(x - b \times W) - (u - v) = 0 \wedge b \times (1 - b) = 0$
Sbb $b\ x\ u\ v\ c$	$\hookrightarrow$	$(x - b \times W) - (u - v - c) = 0 \wedge b \times (1 - b) = 0$
Mul $x\ u\ v$	$\hookrightarrow$	$x - (u \times v) = 0$
Mulf $x\ y\ u\ v$	$\hookrightarrow$	$(x \times W + y) - (u \times v) = 0$
Assert $P \Vert Q$	$\hookrightarrow$	$\top$
Assume $P \Vert Q$	$\hookrightarrow$	$poly(Q)$

Figure 5 Modular Polynomial Equations.

In the following, the three transformations from the verification problem of algebraic properties on CryptoLine programs to the ideal membership problem are explained. We first show how to transform program behaviors to systems of modular polynomial equations. The algebraic property verification problem is then reduced to the modular polynomial equation entailment problem. Finally, we explicate how to solve the entailment problem by the ideal membership problem in commutative algebra.

4.2.1 Modular Polynomial Equations

Let $f(\vec{x}), g(\vec{x}) \in \mathbb{Z}[\vec{x}]$. A *modular polynomial equation* is of the form $f(\vec{x}) = 0$ or $f(\vec{x}) \equiv 0 \bmod g(\vec{x})$. A *system* of modular polynomial equations is denoted by $\bigwedge_{i=1}^{k} f_i(\vec{x}) = 0 \wedge \bigwedge_{i=1}^{l} g_i(\vec{x}) \equiv 0 \bmod h_i(\vec{x})$ where $f_i(\vec{x}), g_i(\vec{x}), h_i(\vec{x}) \in \mathbb{Z}[\vec{x}]$ for all i. A state σ is a *solution* to a modular polynomial equation (written $\sigma \models f(\vec{x}) = 0$ or $\sigma \models f(\vec{x}) \equiv 0 \bmod g(\vec{x})$) if the equation holds under the valuation σ. A state σ is a solution to a system of modular polynomial equations if it is a solution to every equations in the system.

Our first task is to describe program behaviors. Consider the transformation from CryptoLine statements to systems of modular polynomial equations (Figure 5). Set is transformed to the equation stating that the updated variable is equal to the argument. For Cmov, the argument b can be either 0 or 1. Hence its corresponding equation identifying the updated variable to u or v by the value of b. Add! and Sub! are transformed to equations respectively mimicing the addition and the subtraction. In addition to the equations for the updated variable and flag, Add, Adc, Sub, Sbb also have equations restricting the values of flags to be 0 or 1. Mul and Mulf are transformed to equations mimicing the multiplication. Assert is verified as another specification and is ignored. Finally, Assume$P \Vert Q$ is transformed to $poly(Q)$. Given $Q \in \mathbb{Z}[\vec{x}]$, $poly(Q)$ is Q with $e_1 = e_2$ replaced by $e_1 - e_2 = 0$ and $e_1 \equiv e_2 \bmod e_3$ replaced by $e_1 - e_2 \equiv 0 \bmod e_3$.

We have the following theorem for the transformation from CryptoLine to a system of modular polynomial equations.

▶ **Theorem 1.** *For each stmt* $\hookrightarrow \Phi$ *in Figure 5 and non-error states* σ *and* τ*, we have* $\sigma \xrightarrow{stmt} \tau$ *implies* $\tau \models \Phi$*.*

1:	Set $r_0^{(0)}$ $x_0^{(0)}$;	6:	Add! $r_0^{(1)}$ $r_0^{(0)}$ 4503599627370458;	11:	Sub! $r_0^{(2)}$ $r_0^{(1)}$ $y_0^{(0)}$;
2:	Set $r_1^{(0)}$ $x_1^{(0)}$;	7:	Add! $r_1^{(1)}$ $r_1^{(0)}$ 4503599627370494;	12:	Sub! $r_1^{(2)}$ $r_1^{(1)}$ $y_1^{(0)}$;
3:	Set $r_2^{(0)}$ $x_2^{(0)}$;	8:	Add! $r_2^{(1)}$ $r_2^{(0)}$ 4503599627370494;	13:	Sub! $r_2^{(2)}$ $r_2^{(1)}$ $y_2^{(0)}$;
4:	Set $r_3^{(0)}$ $x_3^{(0)}$;	9:	Add! $r_3^{(1)}$ $r_3^{(0)}$ 4503599627370494;	14:	Sub! $r_3^{(2)}$ $r_3^{(1)}$ $y_3^{(0)}$;
5:	Set $r_5^{(0)}$ $x_4^{(0)}$;	10:	Add! $r_4^{(1)}$ $r_4^{(0)}$ 4503599627370494;	15:	Sub! $r_4^{(2)}$ $r_4^{(1)}$ $y_4^{(0)}$;

Figure 6 *sub* in static single assignments.

Sketch. We only show the proof for two statements here. Suppose $\sigma \xrightarrow{\mathsf{Add}\ b\ x\ u\ v} \tau$. If $[\![u]\!]_\sigma + [\![v]\!]_\sigma < W$, $[\![b]\!]_\tau = 0$ and $[\![x]\!]_\tau = [\![u]\!]_\sigma + [\![v]\!]_\sigma$. Otherwise, we have $[\![b]\!]_\tau = 1$ and $[\![x]\!]_\tau = [\![u]\!]_\sigma + [\![v]\!]_\sigma - W$. As statements are in static single assignments, $[\![u]\!]_\tau = [\![u]\!]_\sigma$ and $[\![v]\!]_\tau = [\![v]\!]_\sigma$. Hence $\tau \models (x + b \times W) - (u + v) = 0$ and $\tau \models b \times (1 - b) = 0$ in both cases.

Now suppose $\sigma \xrightarrow{\mathsf{Cmov}\ x\ b\ u\ v} \tau$. If $[\![b]\!]_\sigma = 1$, $[\![x]\!]_\tau = [\![u]\!]_\sigma = [\![b]\!]_\sigma \times [\![u]\!]_\sigma$. Otherwise $[\![b]\!]_\sigma = 0$ and $[\![x]\!]_\tau = [\![v]\!]_\sigma = (1-[\![b]\!]_\sigma)\times[\![v]\!]_\sigma$. As statements are in static single assignments, $[\![b]\!]_\tau = [\![b]\!]_\sigma$, $[\![u]\!]_\tau = [\![u]\!]_\sigma$ and $[\![v]\!]_\tau = [\![v]\!]_\sigma$. In both cases, we have $\tau \models x - (b \times u + (1 - b) \times v) = 0$. ◀

By the assumption that programs are in static single assignments and Theorem 1, a system of modular polynomial equations whose solutions are program execution traces is constructed. More formally, we have the following corollary:

▶ **Corollary 2.** *Suppose $stmt_1; stmt_2; \cdots; stmt_n$ is in static single assignments. Let σ and τ be non-error states with $\sigma \xrightarrow{stmt_1; stmt_2; \cdots; stmt_n} \tau$. Suppose $stmt_i \hookrightarrow \Phi_i$ (Figure 5) for every $1 \le i \le n$. Then $\tau \models \bigwedge_{i=1}^{n} \Phi_i$.*

Example (continued). Let us compute the system of modular polynomial equations for the subtraction program in Figure 3. We rewrite the program in static single assignments (Figure 6). The specification concerning algebraic properties, denoted by $aspec_{sub}$, is rewritten as well:

$$(\!|\ \bigwedge_{i=0}^{4} x_i^{(0)} \le_{64} 2^{51} + 2^{15} \wedge \bigwedge_{i=0}^{4} y_i^{(0)} \le_{64} 2^{51} + 2^{15}\ \|\top|\!)$$
$$sub$$
$$(\!|\top\|\ \ radix51(x_4^{(0)}, x_3^{(0)}, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}) - radix51(y_4^{(0)}, y_3^{(0)}, y_2^{(0)}, y_1^{(0)}, y_0^{(0)})$$
$$\equiv radix51(r_4^{(2)}, r_3^{(2)}, r_2^{(2)}, r_1^{(2)}, r_0^{(2)}) \bmod 2^{255} - 19\ \ |\!)$$

For each statement in the static single assignments, we apply the transformation in Figure 5. Figure 7 shows the corresponding modular polynomial equations.

4.2.2 Modular Polynomial Equation Entailment

Let Φ and Φ' be systems of modular polynomial equations with variables over $\vec{x}$. A *modular polynomial entailment* is a formula of the form $\forall \vec{x}(\Phi \implies \Phi')$. Given a modular polynomial entailment, the *modular polynomial entailment problem* is to decide whether the entailment holds in the theory of integers. Given systems of modular polynomial equations describing program behaviors and intended algebraic properties, it is standard to transform the verification problem to an instance of the modular polynomial entailment problem.

$$
\begin{array}{ll}
1: & r_0^{(0)} - x_0^{(0)} = 0 \wedge \\
2: & r_1^{(0)} - x_1^{(0)} = 0 \wedge \\
3: & r_2^{(0)} - x_2^{(0)} = 0 \wedge \\
4: & r_3^{(0)} - x_3^{(0)} = 0 \wedge \\
5: & r_4^{(0)} - x_4^{(0)} = 0 \wedge \\
6: & r_0^{(1)} - (r_0^{(0)} + 4503599627370458) = 0 \wedge \\
7: & r_1^{(1)} - (r_1^{(0)} + 4503599627370494) = 0 \wedge \\
8: & r_2^{(1)} - (r_2^{(0)} + 4503599627370494) = 0 \wedge \\
9: & r_3^{(1)} - (r_3^{(0)} + 4503599627370494) = 0 \wedge \\
10: & r_4^{(1)} - (r_4^{(0)} + 4503599627370494) = 0 \wedge \\
11: & r_0^{(2)} - (r_0^{(1)} - y_0^{(0)}) = 0 \wedge \\
12: & r_1^{(2)} - (r_1^{(1)} - y_1^{(0)}) = 0 \wedge \\
13: & r_2^{(2)} - (r_2^{(1)} - y_2^{(0)}) = 0 \wedge \\
14: & r_3^{(2)} - (r_3^{(1)} - y_3^{(0)}) = 0 \wedge \\
15: & r_4^{(2)} - (r_4^{(1)} - y_4^{(0)}) = 0
\end{array}
$$

Figure 7 System of Modular Polynomial Equations Φ_{sub} for *sub*.

▶ **Theorem 3.** *Given a specification* $(\!|P\|Q|\!)prog(\!|\top\|Q'|\!)$ *in static single assignments and the corresponding system of modular polynomial equations* Φ_{prog} *for prog where there is no* Assert *statement, if prog is safe with respect to* P *and* $\forall\vec{x}(poly(Q) \wedge \Phi_{prog} \implies poly(Q'))$ *holds, then* $\models (\!|P\|Q|\!)prog(\!|\top\|Q'|\!)$.

Example (continued). We apply Theorem 3 to verify algebraic properties on the subtraction program in Figure 3. Recall the system of modular polynomial equations Φ_{sub} for the program in Figure 7. If we can show

$$
\forall\vec{x}\left[\top \wedge \Phi_{sub} \implies \begin{array}{r} radix51(x_4^{(0)}, x_3^{(0)}, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}) - radix51(y_4^{(0)}, y_3^{(0)}, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}) \\ -radix51(r_4^{(2)}, r_3^{(2)}, r_2^{(2)}, r_1^{(2)}, r_0^{(2)}) \equiv 0 \bmod 2^{255} - 19 \end{array}\right],
$$

denoted by ent_{sub}, then $\models aspec_{sub}$.

Our next task is to solve the modular polynomial entailment problem. It is known how to replace modular polynomial equations with polynomial equations and hence simplify the modular polynomial entailment problem [11]. In the following, we review the simplication for the sake of completeness.

Let $e_i(\vec{x}), f_j(\vec{x}), n_j(\vec{x})g_k(\vec{x}), h_l(\vec{x}), m_l(\vec{x}) \in \mathbb{Z}[\vec{x}]$ for $i \in [I]$, $j \in [J]$, $k \in [K]$, and $l \in [L]$. Consider the instance of the modular polynomial entailment problem:

$$
\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x}) = 0 \wedge \bigwedge_{j\in[J]} f_j(\vec{x}) \equiv 0 \bmod n_j(\vec{x}) \implies \bigwedge_{k\in[K]} g_k(\vec{x}) = 0 \wedge \bigwedge_{l\in[L]} h_l(\vec{x}) \equiv 0 \bmod m_l(\vec{x})]
$$

Its consequent has $K + L + 2$ modular polynomial equations. We decompose the problem into $K + L + 2$ instances of the modular polynomial entailment problem. Each instance is of the form

$$
\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x}) = 0 \wedge \bigwedge_{j\in[J]} f_j(\vec{x}) \equiv 0 \bmod n_j(\vec{x}) \implies g(\vec{x}) = 0]; \text{ or}
$$

$$
\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x}) = 0 \wedge \bigwedge_{j\in[J]} f_j(\vec{x}) \equiv 0 \bmod n_j(\vec{x}) \implies h(\vec{x}) \equiv 0 \bmod m(\vec{x})].
$$

For the first form, we expand the modular polynomial equations and obtain

$$
\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x}) = 0 \wedge \bigwedge_{j\in[J]} [\exists d_j . f_j(\vec{x}) - d_j \cdot n_j(\vec{x}) = 0] \implies g(\vec{x}) = 0],
$$

$$\left\langle \begin{array}{c} 2^{255}-19, r_0^{(0)}-x_0^{(0)}, r_1^{(0)}-x_1^{(0)}, r_2^{(0)}-x_2^{(0)}, r_3^{(0)}-x_3^{(0)}, r_4^{(0)}-x_4^{(0)}, \\ r_0^{(1)}-r_0^{(0)}-4503599627370458, r_1^{(1)}-r_1^{(0)}-4503599627370494, \\ r_2^{(1)}-r_2^{(0)}-4503599627370494, r_3^{(1)}-r_3^{(0)}-4503599627370494, \\ r_4^{(1)}-r_4^{(0)}-4503599627370494, \\ r_0^{(2)}-r_0^{(1)}+y_0^{(0)}, r_1^{(2)}-r_1^{(1)}+y_1^{(0)}, r_2^{(2)}-r_2^{(1)}+y_2^{(0)}, r_3^{(2)}-r_3^{(1)}+y_3^{(0)}, \\ r_4^{(2)}-r_4^{(1)}+y_4^{(0)}, \end{array} \right\rangle$$

Figure 8 Ideal for Checking Algebraic Property on Subtraction.

which in turn is equivalent to

$$\forall\vec{x}\forall\vec{d}[\bigwedge_{i\in[I]} e_i(\vec{x})=0 \wedge \bigwedge_{j\in[J]} f_j(\vec{x})-d_j\cdot n_j(\vec{x})=0 \implies g(\vec{x})=0].$$

Similarly, we obtain the following entailment for the second form:

$$\forall\vec{x}\forall\vec{d}[\bigwedge_{i\in[I]} e_i(\vec{x})=0 \wedge \bigwedge_{j\in[J]} f_j(\vec{x})-d_j\cdot n_j(\vec{x})=0 \implies h(\vec{x})\equiv 0 \bmod m(\vec{x})].$$

Note that antecedents in both forms are but polynomial equations. In order to solve the modular polynomial entailment problem, it suffices to solve the following forms:

$$\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x})=0 \implies g(\vec{x})=0] \tag{1}$$

$$\forall\vec{x}[\bigwedge_{i\in[I]} e_i(\vec{x})=0 \implies h(\vec{x})\equiv 0 \bmod m(\vec{x})] \tag{2}$$

4.2.3 Solving Modular Polynomial Equation Entailment Problem

There is an interesting connection between solving the formula (2) and the ideal membership problem. Suppose $h(\vec{x}) \in \langle e_i(\vec{x}), m(\vec{x})\rangle_{i\in[I]}$. By the definition of ideal, $h(\vec{x})$ is a linear combination of $e_i(\vec{x})$ and $m(\vec{x})$. Hence, there are $c(\vec{x}), c_i(\vec{x}) \in \mathbb{Z}[\vec{x}]$ such that

$$h(\vec{x}) = c(\vec{x})\cdot m(\vec{x}) + \sum_{i\in[I]} c_i(\vec{x})\cdot e_i(\vec{x}).$$

When $e_i(\vec{x})=0$ for every $i\in[I]$, we have $h(\vec{x})=c(\vec{x})\cdot m(\vec{x})$, that is, $h(\vec{x})\equiv 0 \bmod m(\vec{x})$. The following theorem summarizes the connection:

▶ **Theorem 4** ([11]). *Let $e_i(\vec{x}), g(\vec{x}), h(\vec{x}), m(\vec{x}) \in \mathbb{Z}[\vec{x}]$ for $i\in[I]$.*
1. *The formula (1) holds if $g(\vec{x})$ is in the ideal $\langle e_i(\vec{x})\rangle_{i\in[I]}$;*
2. *The formula (2) holds if $h(\vec{x})$ is in the ideal $\langle e_i(\vec{x}), m(\vec{x})\rangle_{i\in[I]}$.*

Example (continued). Recall that we would like to establish ent_{sub}. By Theorem 4, it suffices to show $radix51(x_4^{(0)}, x_3^{(0)}, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}) - radix51(y_4^{(0)}, y_3^{(0)}, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}) - radix51(r_4^{(2)}, r_3^{(2)}, r_2^{(2)}, r_1^{(2)}, r_0^{(2)})$ is in the ideal from Figure 8.

4.2.4 Completeness

The reduction from the algebraic verification problem to the ideal membership problem is sound but incomplete. There are instances of (1) or (2) whose corresponding ideal membership problems do not hold.

4.2.5 Optimization

The ideal membership problem typically becomes harder when the number of polynomials grows [10]. To reduce the number of polynomials, we apply variable substitution same as in [16]. For the instruction Set x u and its corresponding modular polynomial equation $x - u = 0$, we remove the equation and replace x with u in all other modular polynomial equations. Other instructions that update only one variable are processed similarly. Consider Mulf x y u v and its corresponding modular polynomial equation $(x \times W + y) - (u \times v) = 0$ for an example of instructions that update two variables. Since $(x \times W + y) - (u \times v) = 0$ implies $y = u \times v - x \times W$, we remove the equation and replace y with $u \times v - x \times W$ in all other modular polynomial equations. For an Assume$P \| Q$ statement with an equation $e_1 - e_2 = 0$ in $poly(Q)$, we identify a variable x and an expression e such that $e_1 - e_2 = 0$ if and only if $x = e$. Then $e_1 - e_2 = 0$ is removed after replacing x with e.

Example (continued). By the optimization, it suffices to show

$$\begin{aligned}
&\mathit{radix51}(x_4^{(0)}, x_3^{(0)}, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}) - \mathit{radix51}(y_4^{(0)}, y_3^{(0)}, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}) - \\
&\mathit{radix51}(x_4^{(0)} + 4503599627370494 - y_4^{(0)}, x_3^{(0)} + 4503599627370494 - y_3^{(0)}, \\
&\qquad x_2^{(0)} + 4503599627370494 - y_2^{(0)}, x_1^{(0)} + 4503599627370494 - y_1^{(0)}, \\
&\qquad x_0^{(0)} + 4503599627370458 - y_0^{(0)})
\end{aligned}$$

is in the ideal of $\langle 2^{255} - 19 \rangle$.

5 Evaluation

We have implemented our approach in OCaml. Followed by a case study on Montgomery multiplication in this section, the verification of arithmetic assembly programs in cryptographic libraries is reported.

5.1 Montgomery Multiplication

Consider modulo arithmetic computation over $\mathbb{Z}_m$. Since results must be in $\mathbb{Z}_m$, a modulo operation is necessary. For modulo addition or subtraction, it is relatively easy since results are obtained by subtracting or adding m respectively. For modulo multiplication, the naïve algorithm requires division. This is inefficient. Consider, for instance, modulo arithmetic computation over $\mathbb{Z}_{93}$. Let $a = 79$ and $b = 39$. We have $a + b = 118$ and $118 - 93 = 25$. Hence $(a + b) \bmod 93 = 25$. But $a \times b = 79 \times 39 = 3081$. Since $3081 = 93 \times 33 + 12$. We obtain $(79 \times 39) \bmod 93 = 12$ by division.

To avoid inefficient division, cryptographic programs perform modulo arithmetic computation over $\mathbb{Z}_m$ in Montgomery forms. Two numbers R and m' with $\gcd(R, m) = 1$ and $mm' \equiv -1 \bmod R$ are chosen by programmers. For any $a \in \mathbb{Z}_m$, its *Montgomery representation* is $aR \bmod m$. For modulo addition and subtraction, it is still easy to compute in Montgomery forms since $(a \pm b)R \equiv aR \pm bR \bmod m$. For modulo multiplication, one would like to compute the Montgomery representation $abR \bmod m$ from the representations $aR \bmod m$ and $bR \bmod m$ of a and b respectively. Yet $aR \cdot bR \equiv abR^2 \bmod m$. It appears that one has to multiply the inverse of R and then modulo m to obtain the result.

Surprisingly, Montgomery proposed a reduction algorithm which multiplies the inverse of R and modulo m *simultaneously*. The following computation performs the Montgomery

reduction on a number T in Montgomery representation:

$$\begin{aligned} n &= ((T \bmod R) \times m') \bmod R \\ t &= (T + n \times m)/R \\ r &= \text{if } t \geq m \text{ then } t - m \text{ else } t \end{aligned}$$

We will illustrate by an example. Choose $(m, R, m') = (93, 100, 43)$. We have $\gcd(100, 93) = 1$ and $93 \times 43 \equiv -1 \bmod 100$. The numbers a and b in Montgomery representation are $7900 \bmod 93 = 88$ and $3900 \bmod 93 = 87$ respectively (two hard divisions). To perform the Montegomery reduction on $T = 88 \times 87 = 7656$, we compute $n = ((7656 \bmod 100) \times 43) \bmod 100 = (56 \times 43) \bmod 100 = 2408 \bmod 100 = 8$. $t = (7656 + 8 \times 93)/100 = 8400/100 = 84$. Since $84 < m = 93$, we obtain the product 84 in Montgomery representation. To compute $ab \bmod m$, we perform Montgomery reduction again on $T = 84$. Thus $n = ((84 \bmod 100) \times 43) \bmod 100 = (84 \times 43) \bmod 100 = 3612 \bmod 100 = 12$ and $t = (84 + 12 \times 93)/100 = 1200/100 = 12$. Since $12 < 93$, we have $(79 \times 39) \bmod 93 = 12$ as before. Observe that only two hard divisions are necessary for computing Montgomery representations of 79 and 39. Modulo 100 and dividing multiples of 100 by 100 are trivial. If a number of arithmetic operations are required (as in the case for cryptographic primitives), computation in Montgomery representation is much more efficient than textbook algorithms. Cryptographic libraries subsequently implement arithmetic in Montgomery representation.

Assembly subroutines in OpenSSL go even further than that. Previously, we compute multiplication 88×87 followed by reduction to perform one Montgomery multiplication. In practice, multiplication and reduction are performed simultaneously in Montgomery multiplication. OpenSSL moreover uses the multi-limb Montegomery multiplication algorithm where R can be a large power of 2. Consider the four-limb Montgomery multiplication with 64-bit limbs. m is hence a 256-bit number. Choose $R = 2^{256}$. Define $\mathit{radix64}(\ell_3, \ell_2, \ell_1, \ell_0)$ to be the expression

$$\ell_3 \times 2^{64\times 3} + \ell_2 \times 2^{64\times 2} + \ell_1 \times 2^{64\times 1} + \ell_0.$$

Let $m = \mathit{radix64}(m_3, m_2, m_1, m_0)$, $x = \mathit{radix64}(x_3, x_2, x_1, x_0)$, $y = \mathit{radix64}(\ y_3, y_2, y_1, y_0)$ and $m' \in [2^{64} - 1]$ be inputs and $r = \mathit{radix64}(r_3, r_2, r_1, r_0)$ the output. The four-limb Montgomery multiplication subroutine bn_mul_mont_4 in OpenSSL has the following specification:

$$\begin{aligned} &(\!|\top \| m_0 \equiv 1 \bmod 2 \wedge m' \times m + 1 \equiv 0 \bmod 2^{64}|\!) \\ &\quad r = \text{bn\_mul\_mont\_4}(x, y, m, m') \\ &\quad\quad (\!|\top \| x \times y \equiv r \times 2^{256} \bmod m|\!) \end{aligned}$$

The precondition $m_0 \equiv 1 \bmod 2$ is equivalent to $\gcd(m, 2^{64}) = 1$. On input numbers $x = aR \bmod m$ and $y = bR \bmod m$ in Montgomery representation, the output r satisfies $xy \equiv abR^2 \equiv rR \bmod m$. That is, $r \equiv abR \bmod m$. The output r is the product of a and b in Montgomery representation.

The Montgomery multiplication subroutine for x86_64 is invoked by the C fragment:

```
bn_mul_mont(r, x, y, m, m', n_limbs);
```

where `x`, `y`, `m`, `m'` are arrays of 64-bit unsigned integer and `n_limbs` is the number of limbs. We compile the C code and link it with the OpenSSL cryptographic library. The program execution trace is then extracted by `gdb` along with effective addresses automatically. For 4-limb bn_mul_mont_4 (`n_limbs` = 4), there are about 350 assembly instructions. As an illustration, the following three instructions load the value of `m'`, `y[0]`, and `x[0]` to the registers `r8`, `rbx`, and `rax` respectively.

```
mov        (%r8),%r8                                  #!  EA = L0x6060e0
mov        (%r12),%rbx                                #!  EA = L0x6060a0
mov        (%rsi),%rax                                #!  EA = L0x606080
```

For each instruction, we write a Python script to translate it to a CryptoLine statement. Here are the corresponding CryptoLine code for the three instructions:

Set $r8$ $L0x6060e0$;
Set rbx $L0x6060a0$;
Set rax $L0x606080$;

In our automatic translation, each memory cell is a variable identified by its address. Each register is also a variable with the same name. Since effective addresses are obtained from `gdb`, indirect memory operands (such as `-0x10(%rsp,%r15,8)`) are translated to variables corresponding to their effective addresses. It remains to initialize memory cells for inputs.

1:	Set $L0x606080$ x_0;	5:	Set $L0x6060a0$ y_0;	9:	Set $L0x6060c0$ m_0;
2:	Set $L0x606088$ x_1;	6:	Set $L0x6060a8$ y_1;	10:	Set $L0x6060c8$ m_1;
3:	Set $L0x606090$ x_2;	7:	Set $L0x6060b0$ y_2;	11:	Set $L0x6060d0$ m_2;
4:	Set $L0x606098$ x_3;	8:	Set $L0x6060b8$ y_3;	12:	Set $L0x6060d8$ m_3;
				13:	Set $L0x6060e0$ m';

The Assert and Assume statements are indispensable in verifying bn_mul_mont. Consider the following fragment extracted from the assembly subroutine:

1 :	Set rbx y_0;	4 :	Mulf rdx rax rbx rax;	7 :	Mulf *unused* rbp $r10$ rbp;
2 :	Set rax x_0;	5 :	Set $r10$ rax;	8 :	Mulf rdx rax rbp rax;
3 :	Set rbp m';	6 :	Set rax m_0;	9 :	Add! *carry* $r10$ rax $r10$;

At line 5, we have $r10 \equiv y_0 \times x_0 \bmod 2^{64}$. At line 7, $rbp \equiv y_0 \times x_0 \times m' \bmod 2^{64}$. At line 8, $rax \equiv y_0 \times x_0 \times m' \times m_0 \bmod 2^{64}$. Finally at line 9, we have $r10 \equiv (y_0 \times x_0 \times m' \times m_0) + (y_0 \times x_0) \equiv (y_0 \times x_0) \times (m' \times m_0 + 1) \bmod 2^{64}$. From the precondition $m' \times m + 1 \equiv 0 \bmod 2^{64}$, we have $m' \times \mathit{radix64}(m_3, m_2, m_1, m_0) + 1 \equiv 0 \bmod 2^{64}$ and $m' \times m_0 + 1 \equiv 0 \bmod 2^{64}$. Hence $r10 \equiv (y_0 \times x_0) \times (m' \times m_0 + 1) \equiv 0 \bmod 2^{64}$. Now $r10$ is a 64-bit register. $r10 \equiv 0 \bmod 2^{64}$ implies $r10 = 0$ on x86_64. Its value can be safely discarded. The equality is essential to the proof of correctness in the Montgomery multiplication program.

Although $r10 \equiv 0 \bmod 2^{64}$ can be verified by modular polynomial equation entailment, the algebraic technique fails to prove $r10 = 0$. In order to verify bn_mul_mont_, we add two instructions following the code fragment: Assert$\top \| r10 \equiv 0 \bmod 2^{64}$ and Assume$\top \| r10 = 0$. The Assert statement is automatically verified; the Assume statement is safe because $r10 \equiv 0 \bmod 2^{64}$ implies $r10 = 0$ when $r10$ is a 64-bit register.

5.2 Arithmetic in Cryptographic Libraries

We have successfully verified assembly codes extracted from the arithmetic programs in cryptographic libraries OpenSSL, boringSSL, and mbedTLS. The extracted traces are not affected by the inputs in all programs except mbedTLS. The big integer multiplication in mbedTLS contains a loop with undetermined iterations in C for propagating carry chains. For this multiplication, we extracted and verified assembly codes for different cases of carry chains but only report two cases with longest carry chains. All the verification tasks are performed on a Linux machine with a 3.47GHz CPU and 128GB memory. We use Boolector

Table 1 Experimental Results.

library	program	ln	assert	range	alg	total
OpenSSL	ecp_nistz256_add	89	0.44	4.17	0.03	4.63
	ecp_nistz256_sub	88	-	18.54	~0	18.55
	ecp_nistz256_from_mont	82	-	0.41	0.02	0.45
	ecp_nistz256_mul_mont	192	-	21.49	0.03	21.53
	ecp_nistz256_mul_mont$^+$	153	-	15.43	0.03	15.47
	ecp_nistz256_mul_by_2	49	-	0.05	0.02	0.08
	ecp_nistz256_sqr_mont	148	-	16.43	0.03	16.47
	ecp_nistz256_sqr_mont$^+$	131	-	22.50	0.03	22.54
	x86_64_mont_2	228	832.60	13.41	0.03	846.05
	x86_64_mont_4	490	8279.87	523.27	0.91	8804.06
boringSSL	x25519_x86_64_mul	226	-	28.73	0.03	28.78
	x25519_x86_64_sqr	171	-	6.14	0.03	6.18
	x25519_x86_64_ladderstep	1459	-	2921.82	107.93	3029.78
mbedTLS	mbedtls_mpi_mul_mpi_2	76	0.46	0.42	0.03	0.92
	mbedtls_mpi_mul_mpi_4	249	12.85	9.27	0.02	22.16

2.4.0 for SMT solving and use Singular 4.1.0 for ideal membership solving. Table 1 shows the verification results. For each arithmetic program, we report the number of lines (ln) in CryptoLine (including the specification), the time in seconds for assertion checking (assert), range checking (range), algebraic checking (alg), and overall verification (total). A "-" in the assertion column indicates that the program contains no assertion. In OpenSSL, two versions of ecp_nistz256_mul_mont and ecp_nistz256_sqr_mont are availble: one for typical x86_64 microarchitectures, the other for Broadwell microarchitecture (annotated by "$^+$" in the table). For multiplication, we verified 2- and 4-limb versions in OpenSSL (x86_64_mont_*) and mbedTLS (mbedtls_mpi_mul_mpi_*).

To the best of our knowledge, our work is the first on automatically verifying assembly codes extracted from low-level arithmetic implementations of industrial cryptographic libraries. Most of the other works verified re-implementations of arithmetic operations written in high-level languages. In the most related work [16], an implementation of the Montgomery Ladderstep in bvCryptoLine was verified in days. With our approach, the implementation of the Montgomery Ladderstep in boringSSL can be verified in 1 hour.

6 Conclusion

We have described a domain-specific language CryptoLine for modeling arithmetic assembly programs in cryptographic primitives across different instruction sets. Scripts have been developed to extract execution traces from programs as assembly codes and to translate assembly codes to CryptoLine. A specification for a program in CryptoLine is divided into range predicates and algebraic predicates. While range predicates are verified by SMT solving, algebraic predicates are verified via transformation to ideal membership problems solved by computer algebra systems. We have implemented our verification approach to successfully verified several arithmetic programs in cryptographic libraries OpenSSL, boringSSL, and mbedTLS.

We are working on a certified translator to accurately generate CryptoLine codes from different assembly. The case studies in mbedTLS expose limitations of verifying cryptographic codes with CryptoLine. We would like to extend our techniques to such programs.

References

1 Reynald Affeldt. On construction of a library of formally verified low-level arithmetic functions. *Innovations in Systems and Software Engineering*, 9(2):59–77, 2013.

2 Reynald Affeldt and Nicolas Marti. An approach to formal verification of arithmetic functions in assembly. In Mitsu Okada and Ichiro Satoh, editors, *Advances in Computer Science*, volume 4435 of *LNCS*, pages 346–360. Springer, 2007.

3 Reynald Affeldt, David Nowak, and Kiyoshi Yamada. Certifying assembly with formal security proofs: The case of BBS. *Science of Computer Programming*, 77(10–11):1058–1074, 2012.

4 Andrew W. Appel. Verification of a cryptographic primitive: SHA-256. *ACM Transactions on Programming Languages and Systems*, 37(2):7:1–7:31, 2015. doi:10.1145/2701415.

5 Lennart Beringer, Adam Petcher, Katherine Q. Ye, and Andrew W. Appel. Verified correctness and security of openssl HMAC. In *USENIX Security Symposium 2015*, pages 207–221. USENIX Association, 2015.

6 Daniel J. Bernstein and Peter Schwabe. gfverif: Fast and easy verification of finite-field arithmetic, 2016. URL: http://gfverif.cryptojedi.org.

7 B. Bond, C. Hawblitzel, M. Kapritsos, K. R. M. Leino, J. R. Lorch, B. Parno, A. Rane, S. Setty, and L. Thompson. Vale: Verifying high-performance cryptographic assembly code. In *USENIX Security Symposium 2017*, pages 917–934. USENIX Association, 2017.

8 Yu-Fang Chen, Chang-Hong Hsu, Hsin-Hung Lin, Peter Schwabe, Ming-Hsien Tsai, Bow-Yaw Wang, Bo-Yin Yang, and Shang-Yi Yang. Verifying curve25519 software. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *CCS*, pages 299–309. ACM, 2014.

9 Fiat-crypto. https://github.com/mit-plv/fiat-crypto, 2015. Accessed: 2017-05-19.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and company, 1979.

11 John Harrison. Automating elementary number-theoretic proofs using Gröbner bases. In Frank Pfenning, editor, *CADE*, volume 4603 of *LNCS*, pages 51–66. Springer, 2007.

12 D. Kroening and O. Strichman. *Decision Procedures - an algorithmic point of view*. EATCS. Springer, 2008.

13 Magnus O. Myreen and Gregorio Curello. Proof pearl: A verified bignum implementation in x86-64 machine code. In *Certified Programs and Proofs*, volume 8307 of *LNCS*, pages 66–81. Springer, 2013. doi:10.1007/978-3-319-03545-1_5.

14 Magnus O. Myreen and Michael J. C. Gordon. Hoare logic for realistically modelled machine code. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *LNCS*, pages 568–582. Springer, 2007.

15 Aaron Tomb. Automated verification of real-world cryptographic implementations. *IEEE Security & Privacy*, 14(6):26–33, 2016. doi:10.1109/MSP.2016.125.

16 Ming-Hsien Tsai, Bow-Yaw Wang, and Bo-Yin Yang. Certified verification of algebraic properties on low-level mathematical constructs in cryptographic programs. In David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS*. ACM, 2017.

17 Katherine Q. Ye, Matthew Green, Naphat Sanguansin, Lennart Beringer, Adam Petcher, and Andrew W. Appel. Verified correctness and security of mbedtls HMAC-DRBG. In *CCS*, pages 2007–2020. ACM, 2017. doi:10.1145/3133956.3133974.

18 Jean Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. HACL*: A verified modern cryptographic library. In *CCS*, pages 1789–1806. ACM, 2017. doi:10.1145/3133956.3134043.

Coalgebraic Theory of Büchi and Parity Automata: Fixed-Point Specifications, Categorically

Ichiro Hasuo
National Institute of Informatics, Japan
i.hasuo@acm.org
https://orcid.org/0000-0002-8300-4650

Abstract

Coalgebra is a categorical modeling of state-based dynamics. Final coalgebras – as categorical greatest fixed points – play a central role in the theory; somewhat analogously, most coalgebraic proof techniques have been devoted to *greatest* fixed-point properties such as safety and bisimilarity. In this tutorial, I introduce our recent coalgebraic framework that accommodates those fixed-point specifications which are not necessarily the greatest. It does so specifically by characterizing the accepted languages of *Büchi* and *parity* automata in categorical terms. We present two characterizations of accepted languages. The proof for their coincidence offers a unique categorical perspective of the correspondence between (logical) fixed-point specifications and the (combinatorial) parity acceptance condition.

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

Keywords and phrases Coalgebra, category theory, fixed-point logic, automata, Büchi automata, parity automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.5

Category Invited Tutorial

Funding Supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST; Grants-in-Aid No. 15KT0012 & 15K11984, JSPS; and the JSPS-INRIA Bilateral Joint Research Project "CRECOGI."

Studies of automata, and state-based transition systems in general, have been shed a fresh categorical light in the 1990s by the theory of *coalgebra* [7, 5]. In the theory, a state-based dynamics is modeled by a coalgebra, that is, an arrow $c\colon X \to FX$ in a category $\mathbb{C}$; and this simple modeling has produced numerous results that capture mathematical essences and provide general techniques.

Final coalgebras as "categorical greatest fixed points" play a central role in the theory of coalgebra. Somewhat analogously, most coalgebraic proof methods have focused on greatest fixed-point properties – a notable example being a span-based categorical characterization of *bisimilarity*.

In this tutorial, I will outline our recent results [10, 8] about how we can accommodate, in the theory of coalgebra, those fixed-point properties which are not necessarily the greatest. This takes the concrete form of characterizing the accepted languages of *Büchi* and *parity* automata in the language of category theory. Our framework, based on the so-called *Kleisli* approach to coalgebraic trace semantics [6, 4, 2, 1], is generic and covers both automata with nondeterministic and probabilistic branching. It covers both word and tree automata, too.

We present two characterizations of the accepted languages of Büchi and parity automata. The first one is called *logical* fixed points; it is formulated in terms of the order-enriched structure of the underlying Kleisli category (where the monad in question models branching type) [10]. The second one, called *categorical* fixed points, utilizes nested datatypes specified by a functor. The latter resembles repeated application of (co)free (co)monads. We exhibit a proof for the coincidence of the two characterizations. What arises through it is a categorical perspective of one of the key observations that underpin the recent developments in computer science – namely the fact that the *combinatorial* notion of parity acceptance condition represents *logical* specifications given by nested and alternating fixed points.

The tutorial is based on the speaker's joint works with Corina Cîrstea, Bart Jacobs, Shunsuke Shimizu, Ana Sokolova, and Natsuki Urabe [2, 3, 8, 10]. A detailed account of the technical material of the tutorial will be given in a forthcoming paper [9].

References

1 Corina Cîrstea. Canonical coalgebraic linear time logics. In Lawrence S. Moss and Pawel Sobocinski, editors, *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands*, volume 35 of *LIPIcs*, pages 66–85. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CALCO.2015.66.

2 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Comp. Sci.*, 3(4:11), 2007.

3 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. Lattice-theoretic progress measures and coalgebraic model checking. In Rastislav Bodik 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 718–732. ACM, 2016. doi:10.1145/2837614.2837673.

4 B. Jacobs. Trace semantics for coalgebras. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, volume 106 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 2004.

5 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

6 J. Power and D. Turi. A coalgebraic foundation for linear time semantics. In *Category Theory and Computer Science*, volume 29 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 1999.

7 J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249:3–80, 2000.

8 Natsuki Urabe and Ichiro Hasuo. Categorical Büchi and parity conditions via alternating fixed points of functors. In Corina Cîrstea, editor, *Proc. Coalgebraic Methods in Computer Science - 14th IFIP WG 1.3 International Workshop, CMCS 2018*, Lect. Notes Comp. Sci., 2018. to appear, preprint available at arxiv.org/abs/1803.06811.

9 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic theory of Büchi and parity automata: Fixed-point specifications, categorically (tentative). forthcoming.

10 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic trace semantics for buechi and parity automata. 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 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

Universal Safety for Timed Petri Nets is PSPACE-complete

Parosh Aziz Abdulla
Uppsala University, Sweden

Mohamed Faouzi Atig
Uppsala University, Sweden

Radu Ciobanu
University of Edinburgh, UK

Richard Mayr
University of Edinburgh, UK

Patrick Totzke
University of Edinburgh, UK
https://orcid.org/0000-0001-5274-8190

Abstract

A timed network consists of an arbitrary number of initially identical 1-clock timed automata, interacting via hand-shake communication. In this setting there is no unique central controller, since all automata are initially identical. We consider the universal safety problem for such controller-less timed networks, i.e., verifying that a bad event (enabling some given transition) is impossible regardless of the size of the network.

This universal safety problem is dual to the existential coverability problem for timed-arc Petri nets, i.e., does there exist a number m of tokens, such that starting with m tokens in a given place, and none in the other places, some given transition is eventually enabled.

We show that these problems are PSPACE-complete.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases timed networks, safety checking, Petri nets, coverability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.6

Funding This work was supported by the EPSRC, grant EP/M027651/1.

1 Introduction

Background. Timed-arc Petri nets (TPN) [4, 16, 3, 8, 13] are an extension of Petri nets where each token carries one real-valued clock and transitions are guarded by inequality constraints where the clock values are compared to integer bounds (via strict or non-strict inequalities). The known models differ slightly in what clock values newly created tokens can have, i.e., whether newly created tokens can inherit the clock value of some input token of the transition, or whether newly created tokens always have clock value zero. We consider the former, more general, case.

Decision problems associated with the reachability analysis of (extended) Petri nets include *Reachability* (can a given marking reach another given marking?) and *Coverability* (can a given marking ultimately enable a given transition?).

While Reachability is undecidable for all these TPN models [15], Coverability is decidable using the well-quasi ordering approach of [1, 10] and complete for the hyper-Ackermannian

complexity class F_{ω^ω} [12]. With respect to Coverability, TPN are equivalent [7] to (linearly ordered) data nets [14].

The *Existential Coverability* problem for TPN asks, for a given place p and transition t, whether there exists a number m such that the marking $M(m) \stackrel{\text{def}}{=} m \cdot \{(p, \mathbf{0})\}$ ultimately enables t. Here, $M(m)$ contains exactly m tokens on place p with all clocks set to zero and *no other tokens*. This problem corresponds to checking safety properties in distributed networks of arbitrarily many (namely m) initially identical timed processes that communicate by handshake. A negative answer certifies that the "bad event" of transition t can never happen regardless of the number m of processes, i.e., the network is safe for any size. Thus by checking existential coverability, one solves the dual problem of *Universal Safety*. (Note that the m timed tokens/processes are only initially identical. They can develop differently due to non-determinacy in the transitions.)

The corresponding problem for timed networks studied in [2] does not allow the dynamic creation of new timed processes (unlike the TPN model which can increase the number of timed tokens), but considers multiple clocks per process (unlike our TPN with one clock per token).

The TPN model above corresponds to a distributed network without a central controller, since initially there are no tokens on other places that could be used to simulate one. Adding a central controller would make *Existential Coverability* polynomially inter-reducible with normal *Coverability* and thus complete for F_{ω^ω} [12] (and even undecidable for > 1 clocks per token [2]).

Aminof et. al. [6] study the model checking problem of ω-regular properties for the controller-less model and in particular claim an EXPSPACE upper bound for checking universal safety. However, their result only holds for discrete time (integer-valued clocks) and they do not provide a matching lower bound.

Our contribution. We show that *Existential Coverability* (and thus universal safety) is decidable and PSPACE-complete. This positively resolves an open question from [2] regarding the decidability of universal safety in the controller-less networks. Moreover, a symbolic representation of the set of coverable configurations can be computed (using exponential space).

The PSPACE lower bound is shown by a reduction from the iterated monotone Boolean circuit problem. (It does not follow directly from the PSPACE-completeness of the reachability problem in timed automata of [5], due to the lack of a central controller.)

The main ideas for the PSPACE upper bound are as follows. First we provide a logspace reduction of the Existential Coverability problem for TPN to the corresponding problem for a syntactic subclass, non-consuming TPN. Then we perform an abstraction of the real-valued clocks, similar to the one used in [3]. Clock values are split into integer parts and fractional parts. The integer parts of the clocks can be abstracted into a finite domain, since the transition guards cannot distinguish between values above the maximal constant that appears in the system. The fractional parts of the clock values that occur in a marking are ordered sequentially. Then every marking can be abstracted into a string where all the tokens with the i-th fractional clock value are encoded in the i-th symbol in the string. Since token multiplicities do not matter for existential coverability, the alphabet from which these strings are built is finite. The primary difficulty is that the length of these strings can grow dynamically as the system evolves, i.e., the space of these strings is still infinite for a given TPN. We perform a forward exploration of the space of reachable strings. By using an acceleration technique, we can effectively construct a symbolic representation of the set of reachable strings in terms of finitely many regular expressions. Finally, we can check existential coverability by using this symbolic representation.

2 Timed Petri Nets

We use $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ to denote the sets of nonnegative integers and reals, respectively. For $n \in \mathbb{N}$ we write $[n]$ for the set $\{0, \ldots, n\}$.

For a set A, we use A^* to denote the set of words, i.e. finite sequences, over A, and write ε for the empty word. If R is a regular expression over A then $\mathcal{L}(R) \subseteq A^*$ denotes its language.

A *multiset* over a set X is a function $M : X \to \mathbb{N}$. The set $X^{\oplus}$ of all (finitely supported) multisets over X is partially ordered pointwise (by $\leq$). The multiset union of $M, M' \in X^{\oplus}$ is $(M \oplus M') \in X^{\oplus}$ with $(M \oplus M')(\alpha) \stackrel{\text{def}}{=} M(\alpha) + M'(\alpha)$ for all $\alpha \in X$. If $M \geq M'$ then the multiset difference $(M \ominus M')$ is the unique $M'' \in X^{\oplus}$ with $M = M' \oplus M''$. We will use a monomial representation and write for example $(\alpha + \beta^3)$ for the multiset $(\alpha \mapsto 1, \beta \mapsto 3)$. For a multiset M and a number $m \in \mathbb{N}$ we let $m \cdot M$ denote the m-fold multiset sum of M. We further lift this to sets of numbers and multisets on the obvious fashion, so that in particular $\mathbb{N} \cdot S \stackrel{\text{def}}{=} \{n \cdot M \mid n \in \mathbb{N}, M \in S\}$.

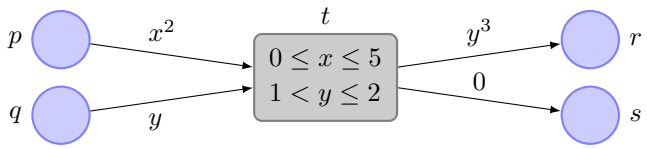
Timed Petri nets are place/transition nets where each token carries a real value, sometimes called its *clock value* or *age*. Transition firing depends on there being sufficiently many tokens whose value is in a specified interval. All tokens produced by a transition either have age 0, or inherit the age of an input-token of the transition. To model time passing, all token ages can advance simultaneously by the same (real-valued) amount.

▶ **Definition 1** (TPN). A *timed Petri net* (TPN) $\mathcal{N} = (P, T, Var, G, Pre, Post)$ consists of finite sets of *places* P, *transitions* T and *variables* Var, as well as functions $G, Pre, Post$ defining transition *guards*, *pre–* and *postconditions*, as follows.

For every transition $t \in T$, the guard $G(t)$ maps variables to (open, half-open or closed) intervals with endpoints in $\mathbb{N} \cup \{\infty\}$, restricting which values variables may take. All numbers are encoded in unary. The precondition $Pre(t)$ is a finite multiset over $(P \times Var)$. Let $Var(t) \subseteq Var$ be the subset of variables appearing positively in $Pre(t)$. The postcondition $Post(t)$ is then a finite multiset over $(P \times (\{0\} \cup Var(t)))$, specifying the locations and clock values of produced tokens. Here, the symbolic clock value is either 0 (demanding a reset to age 0), or a variable that appeared already in the precondition.

A *marking* is a finite multiset over $P \times \mathbb{R}_{\geq 0}$.

▶ **Example 2.** The picture below shows a place/transition representation of an TPN with four places and one transition. $Var(t) = \{x, y\}$, $Pre(t) = (p, x)^2 + (q, y)$, $G(t)(x) = [0, 5]$, $G(t)(y) =]1, 2]$ and $Post(t) = (r, y)^3 + (s, 0)$.



The transition t consumes two tokens from place p, both of which have the same clock value x (where $0 \leq x \leq 5$) and one token from place q with clock value y (where $1 < y \leq 2$). It produces three tokens on place r who all have the same clock value y (where y comes from the clock value of the token read from q), and another token with value 0 on place s.

There are two different binary step relations on markings: *discrete* steps $\longrightarrow_t$ which fire a transition t as specified by the relations G, Pre, and $Post$, and *time passing* steps $\longrightarrow_d$ for durations $d \in \mathbb{R}_{\geq 0}$, which simply increment all clocks by d.

▶ **Definition 3** (Discrete Steps). For a transition $t \in T$ and a variable evaluation $\pi : \mathit{Var} \to \mathbb{R}_{\geq 0}$, we say that π *satisfies* $G(t)$ if $\pi(x) \in G(t)(x)$ holds for all $x \in \mathit{Var}$. By lifting π to multisets over $(P \times \mathit{Var})$ (respectively, to multisets over $(P \times (\{0\} \cup \mathit{Var}))$ with $\pi(0) = 0$) in the canonical way, such an evaluation translates preconditions $\mathit{Pre}(t)$ and $\mathit{Post}(t)$ into markings $\pi(\mathit{Pre}(t))$ and $\pi(\mathit{Post}(t))$, where for all $p \in P$ and $c \in \mathbb{R}_{\geq 0}$,

$$\pi(\mathit{Pre}(t))(p,c) \stackrel{\text{def}}{=} \sum_{\pi(v)=c} \mathit{Pre}(t)(p,v) \qquad \text{and} \qquad \pi(\mathit{Post}(t))(p,c) \stackrel{\text{def}}{=} \sum_{\pi(v)=c} \mathit{Post}(t)(p,v).$$

A transition $t \in T$ is called *enabled* in marking M, if there exists an evaluation π that satisfies $G(t)$ and such that $\pi(\mathit{Pre}(t)) \leq M$. In this case, there is a discrete step $M \longrightarrow_t M'$ from marking M to M', defined as $M' = M \ominus \pi(\mathit{Pre}(t)) \oplus \pi(\mathit{Post}(t))$.

▶ **Definition 4** (Time Steps). Let M be a marking and $d \in \mathbb{R}_{\geq 0}$. There is a time step $M \longrightarrow_d M'$ to the marking M' with $M'(p,c) \stackrel{\text{def}}{=} M(p, c-d)$ for $c \geq d$, and $M'(p,c) \stackrel{\text{def}}{=} 0$, otherwise. We also refer to M' as $(M + d)$.

We write $\to_{\mathit{Time}}$ for the union of all timed steps, $\to_{\mathit{Disc}}$ for the union of all discrete steps and simply $\to$ for $\to_{\mathit{Disc}} \cup \to_{\mathit{Time}}$. The transitive and reflexive closure of $\to$ is $\stackrel{*}{\to}$. $\mathit{Cover}(M)$ denotes the set of markings M' for which there is an $M'' \geq M'$ with $M \stackrel{*}{\to} M''$.

We are interested in the *existential coverability problem* (∃COVER for short), as follows.

Input:	A TPN, an initial place p and a transition t.
Question:	Does there exist $M \in \mathit{Cover}(\mathbb{N} \cdot \{(p,0)\})$ that enables t?

We show that this problem is PSPACE-complete. Both lower and upper bound will be shown (w.l.o.g., see Lemma 8) for the syntactic subclass of *non-consuming* TPN, defined as follows.

▶ **Definition 5.** A *timed Petri net* $(P, T, \mathit{Var}, G, \mathit{Pre}, \mathit{Post})$ is *non-consuming* if for all $t \in T$, $p \in P$ and $x \in \mathit{Var}$ it holds that both 1) $\mathit{Pre}(t)(p,x) \leq 1$, and 2) $\mathit{Pre}(t) \leq \mathit{Post}(t)$.

In a non-consuming TPN, token multiplicities are irrelevant for discrete transitions. Intuitively, having one token (p,c) is equivalent to having an inexhaustible supply of such tokens.

The first condition is merely syntactic convenience. It asks that each transition takes at most one token from each place. The second condition in Definition 5 implies that for each discrete step $M \longrightarrow_t M'$ we have $M' \geq M$. Therefore, once a token (p,c) is present on a place p, it will stay there unchanged (unless time passes), and it will enable transitions with (p,c) in their precondition.

Wherever possible, we will from now on therefore allow ourselves to use the set notation for markings, that is simply treat markings $M \in (P \times \mathbb{R}_{\geq 0})^{\oplus}$ as sets $M \subseteq (P \times \mathbb{R}_{\geq 0})$.

3 Lower Bound

PSPACE-hardness of ∃COVER does not follow directly from the PSPACE-completeness of the reachability problem in timed automata of [5]. The non-consuming property of our TPN makes it impossible to fully implement the control-state of a timed automaton. Instead our proof uses multiple timed tokens and a reduction from the iterated monotone Boolean circuit problem [11].

A depth-1 monotone Boolean circuit is a function $F : \{0,1\}^n \to \{0,1\}^n$ represented by n constraints: For every $0 \leq i < n$ there is a constraint of the form $i' = j \otimes k$, where $0 \leq j, k < n$ and $\otimes \in \{\wedge, \vee\}$, which expresses how the next value of bit i depends on the

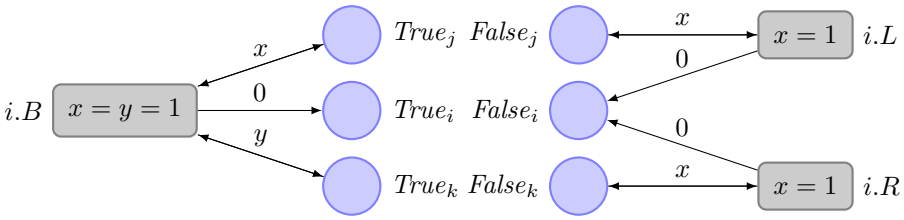


Figure 1 The transitions $i.B, i.R$ and $i.L$ that simulate the update of bit i according to constraint $i' = j \wedge k$. All transitions demand that incoming tokens are of age exactly 1 and only tokens of age 0 are produced.

current values of bits j and k. For every bitvector $\boldsymbol{v} \in \{0,1\}^n$, the function F then satisfies $F(\boldsymbol{v})[i] \stackrel{\text{def}}{=} \boldsymbol{v}[j] \otimes \boldsymbol{v}[k]$. It is PSPACE-complete to check whether for a given vector $\boldsymbol{v} \in \{0,1\}^n$ there exists a number $m \in \mathbb{N}$ such that $F^m(\boldsymbol{v})[0] = 1$.

Towards a lower bound for ∃COVER (Theorem 7) we construct a non-consuming TPN as follows, for a given circuit. The main idea is to simulate circuit constraints by transitions that reset tokens of age 1 (encoding $\boldsymbol{v}$) to fresh ones of age 0 (encoding $F(\boldsymbol{v})$), and let time pass by one unit to enter the next round.

For every bit $0 \leq i < n$, the net contains two places $True_i$ and $False_i$. A marking $M_{\boldsymbol{v}} \leq P \times \mathbb{R}_{\geq 0}$ is an *encoding* of a vector $\boldsymbol{v} \in \{0,1\}^n$ if for every $0 \leq i < n$ the following hold.

1. $(True_i, 0) \in M_{\boldsymbol{v}} \iff \boldsymbol{v}[i] = 1$.
2. $(False_i, 0) \in M_{\boldsymbol{v}} \iff \boldsymbol{v}[i] = 0$.
3. If $(p, c) \in M_{\boldsymbol{v}}$ then $c = 0$ or $c \geq 1$.

Note that in particular one cannot have both $(True_i, 0)$ and $(False_i, 0)$ in $M_{\boldsymbol{v}}$. For every constraint $i' = j \wedge k$ we introduce three transitions, $i.L, i.R$, and $i.B$, where

$$Pre(i.B) \stackrel{\text{def}}{=} (True_j, x) + (True_k, y) \qquad Post(i.B) \stackrel{\text{def}}{=} Pre(i.B) + (True_i, 0)$$
$$Pre(i.L) \stackrel{\text{def}}{=} (False_j, x) \qquad Post(i.L) \stackrel{\text{def}}{=} Pre(i.L) + (False_i, 0)$$
$$Pre(i.R) \stackrel{\text{def}}{=} (False_k, x) \qquad Post(i.R) \stackrel{\text{def}}{=} Pre(i.R) + (False_i, 0)$$

and the guard for all transitions is $G(x) = G(y) = 1$. See Figure 1 for an illustration. For disjunctions $i' = j \vee k$ the transitions are defined analogously, with *True* and *False* inverted. The correctness proof of our construction rests on the following simple observation.

▶ **Lemma 6.** *If $F(\boldsymbol{v}) = \boldsymbol{v}'$ then for every encoding $M_{\boldsymbol{v}}$ of $\boldsymbol{v}$, there exists an encoding $M_{\boldsymbol{v}'}$ of $\boldsymbol{v}'$ such that $M_{\boldsymbol{v}} \longrightarrow_1 \stackrel{*}{\rightarrow}_{Disc} M_{\boldsymbol{v}'}$. Conversely, if $M_{\boldsymbol{v}} \longrightarrow_1 \stackrel{*}{\rightarrow}_{Disc} M_{\boldsymbol{v}'}$ for encodings $M_{\boldsymbol{v}}$ and $M_{\boldsymbol{v}'}$ of $\boldsymbol{v}$ and $\boldsymbol{v}'$ respectively, then $F(\boldsymbol{v}) = \boldsymbol{v}'$.*

Proof. For the first part, we construct a sequence $M_0 \rightarrow_{Disc} M_1 \rightarrow_{Disc} \ldots \rightarrow_{Disc} M_{n-1}$ where $M_0 \stackrel{\text{def}}{=} (M_{\boldsymbol{v}} + 1)$ and every step $M_{i-1} \rightarrow_{Disc} M_i$ adds tokens simulating the ith constraint of F. Since the TPN is non-consuming, we will have that $M_i \geq (M_{\boldsymbol{v}} + 1)$, for all $i < n$. Consider now constraint i', and assume w.l.o.g. that $i' = j \wedge k$ (the other case is analogous). There are two cases depending on $\boldsymbol{v}'[i]$.

1. Case $\boldsymbol{v}'[i] = 1$. By our assumption that $F(\boldsymbol{v}) = \boldsymbol{v}'$ we know that $\boldsymbol{v}[j] = 1$ and $\boldsymbol{v}[k] = 1$. So $(True_j, 1) \in (M_{\boldsymbol{v}} + 1) \leq M_{i-1}$ and $(True_k, 1) \in (M_{\boldsymbol{v}} + 1) \leq M_{i-1}$. By construction of the net, there is a transition $i.B$ with $Pre(i.B) = (True_j, 1) + (True_k, 1)$ and $Post(i.B) = Pre(i.B) + (True_i, 0)$. This justifies step $M_{i-1} \longrightarrow_{i.B} M_i$ and therefore that $(True_i, 0) \in$

$M_i \leq M_{n-1}$. Also notice that no marking reachable from M_0 using only discrete steps can contain the token $(False_i, 0)$. This is because these can only be produced by transitions requiring either $(False_j, 1)$ or $(False_k, 1)$, which are not contained in M_0 by assumption that $M_{\boldsymbol{v}}$ encodes $\boldsymbol{v}$. Therefore $(False_i, 0) \notin M_{n-1}$.

2. Case $\boldsymbol{v'}[i] = 0$. W.l.o.g., $\boldsymbol{v}[j] = 0$. Therefore, $(False_j, 1) \in (M_{\boldsymbol{v}} + 1) \leq M_{i-1}$. By construction of the net, there exists transition $i.L$ with $Pre(i.L) = (False_j, 1)$ and $Post(i.L) = Pre(i.L) + (False_i, 0)$. This justifies the step $M_{i-1} \longrightarrow_{i.L} M_i$, with $(False_i, 0) \in M_i \leq M_{n-1}$. Notice again that no marking reachable from M_0 using only discrete steps can contain the token $(True_i, 0)$. This is because these can only be produced by transitions $i.B$, requiring both $(True_j, 1), (True_k, 1) \in M_0$, contradicting our assumptions. Hence, $(True_i, 0) \notin M_{n-1}$.

We conclude that the constructed marking M_{n-1} is an encoding of $\boldsymbol{v'}$.

For the other part of the claim, assume that there exist markings $M_{\boldsymbol{v}}$ and $M_{\boldsymbol{v'}}$ which are encodings of vectors $\boldsymbol{v}$ and $\boldsymbol{v'}$, respectively, with $M_{\boldsymbol{v}} \longrightarrow_1 \xrightarrow{*}_{Disc} M_{\boldsymbol{v'}}$. We will show that $F(\boldsymbol{v}) = \boldsymbol{v}'$. Recall that $F(\boldsymbol{v})[i] \stackrel{\text{def}}{=} \boldsymbol{v}[j] \otimes \boldsymbol{v}[k]$, where $0 \leq j, k < n$ and $\otimes \in \{\wedge, \vee\}$. We will show for each $i < n$ that $\boldsymbol{v}'[i] = \boldsymbol{v}[j] \otimes \boldsymbol{v}[k]$. Again, consider the constraint i', and assume w.l.o.g. that $i' = j \wedge k$ (the other case is analogous). There are two cases.

1. Case $\boldsymbol{v'}[i] = 1$. By definition of a marking encoding, we have that $(True_i, 0) \in M_{\boldsymbol{v}}$. By construction, there is a transition $i.B$ with $Pre(i.B) = (True_j, 1) + (True_k, 1)$ and $Post(i.B) = Pre(i.B) + (True_i, 0)$. By assumption, it holds that $(M_{\boldsymbol{v}} + 1) \xrightarrow{*}_{Disc} M'_{\boldsymbol{v}}$, where $M_{\boldsymbol{v}} \longrightarrow_1 (M_{\boldsymbol{v}} + 1)$. Note that $(True_j, 1) \in (M_{\boldsymbol{v}} + 1)$ and $(True_k, 1) \in (M_{\boldsymbol{v}} + 1)$. Hence, we have that $\boldsymbol{v}[j] = 1$ and $\boldsymbol{v}[k] = 1$, and therefore that $F(\boldsymbol{v})[i] = \boldsymbol{v'}[i] = \boldsymbol{v}[j] \wedge \boldsymbol{v}[k]$.
2. Case $\boldsymbol{v'}[i] = 0$. Then $(False_i, 0) \in M_{\boldsymbol{v}}$ and, since this token can only be produced by transitions $i.L$ or $i.R$, either $(False_j, 1) \in (M_{\boldsymbol{v}} + 1)$ or $(False_k, 1) \in (M_{\boldsymbol{v}} + 1)$.
Therefore $(False_j, 0) \in (M_{\boldsymbol{v}})$ or $(False_k, 0) \in (M_{\boldsymbol{v}})$ and because $M_{\boldsymbol{v}}$ is an encoding of $\boldsymbol{v}$, this means that either $\boldsymbol{v}[j] = 0$ or $\boldsymbol{v}[k] = 0$. Therefore, $F(\boldsymbol{v'})[i] = \boldsymbol{v}[j] \wedge \boldsymbol{v}[k] = 0$. ◀

▶ **Theorem 7.** *$\exists COVER$ is* PSPACE*-hard for non-consuming TPN.*

Proof. For a given monotone Boolean circuit, define a non-consuming TPN as above. By induction on $m \in \mathbb{N}$ using Lemma 6, we derive that there exists $m \in \mathbb{N}$ with $F^m(\boldsymbol{v}) = \boldsymbol{v'}$ and $\boldsymbol{v}'[0] = 1$ if, and only if, there exists encodings $M_{\boldsymbol{v}}$ of $\boldsymbol{v}$ and $M_{\boldsymbol{v'}}$ of $\boldsymbol{v'}$, with $M_{\boldsymbol{v}} \xrightarrow{*} M_{\boldsymbol{v}'}$. Moreover, if there is a marking M such that $M_{\boldsymbol{v}} \xrightarrow{*} M$ and $0 \in frac(M)$, where M contains a token of age 0, then $M \leq M_{\boldsymbol{v'}}$ for some encoding $M_{\boldsymbol{v}'}$ of a vector $\boldsymbol{v'} = F^m(\boldsymbol{v})$. This means that it suffices to add one transition t with $Pre(t) = (True_0, 0)$ whose enabledness witnesses the existence of a reachable encoding $M_{\boldsymbol{v}'}$ containing a token $(True_0, 0)$. By the properties above, there exists $m \in \mathbb{N}$ with $F^m(\boldsymbol{v}) = \boldsymbol{v'}$ and $\boldsymbol{v}'[0] = 1$ iff $M_{\boldsymbol{v}} \xrightarrow{*} M_{\boldsymbol{v}'} \xrightarrow{t}$. ◀

This lower bound holds even for discrete time TPN, e.g. [9], because the proof uses only timed steps with duration $d = 1$.

4 Upper Bound

We start by observing that we can restrict ourselves, without loss of generality, to non-consuming TPN (Definition 5) for showing the upper bound. Intuitively, since we start with an arbitrarily high number of tokens anyway, it does not matter how many of them are consumed by transitions during the computation, since some always remain.

▶ **Lemma 8.** *The ∃COVER problem for TPN logspace-reduces to the ∃COVER problem for non-consuming TPN. That is, for every TPN $\mathcal{N}$ and for every place p and transition t of $\mathcal{N}$, one can construct, using logarithmic space, a non-consumimg TPN $\mathcal{N}'$ together with a place p' and transition t' of $\mathcal{N}'$, so that there exists $M \in Cover_{\mathcal{N}}(\mathbb{N} \cdot \{(p,0)\})$ enabling t in $\mathcal{N}$ if and only if there exists $M' \in Cover_{\mathcal{N}'}(\mathbb{N} \cdot \{(p',0)\})$ that enables t' in $\mathcal{N}'$.*

4.1 Region Abstraction

We recall a constraint system called regions defined for timed automata [5]. The version for TPN used here is similar to the one in [3].

Consider a fixed, nonconsuming TPN $\mathcal{N} = (P, T, Var, G, Pre, Post)$. Let c_{max} be the largest finite value appearing in transition guards G. Since different tokens with age $> c_{max}$ cannot be distinguished by transition guards, we consider only token ages below or equal to c_{max} and treat the integer parts of older tokens as equal to $c_{max}+1$. Let $int(c) \stackrel{\text{def}}{=} \min\{c_{max}+1, \lfloor c \rfloor\}$ and $frac(c) \stackrel{\text{def}}{=} c - \lfloor c \rfloor$ for a real value $c \in \mathbb{R}_{\geq 0}$. We will work with an abstraction of TPN markings as words over the alphabet $\Sigma \stackrel{\text{def}}{=} 2^{P \times [c_{max}+1]}$. Each symbol $X \in \Sigma$ represents the places and integer ages of tokens for a particular fractional value.

▶ **Definition 9.** Let $M \subseteq P \times \mathbb{R}_{\geq 0}$ be a marking and let $frac(M) \stackrel{\text{def}}{=} \{frac(c) \mid (p,c) \in M\}$ be the set of fractional clock values that appear in M.

Let $S \subset [0,1[$ be a finite set of real numbers with $0 \in S$ and $frac(M) \subseteq S$ and let $f_0, f_1, \ldots, f_n$, be an enumeration of S so that $f_{i-1} < f_i$ for all $i \leq n$. The *S-abstraction* of M is

$$abs_S(M) \stackrel{\text{def}}{=} x_0 x_1 \ldots x_n \in \Sigma^*$$

where $x_i \stackrel{\text{def}}{=} \{(p, int(c)) \mid (p,c) \in M \wedge frac(c) = f_i\}$ for all $i \leq n$. We simply write $abs(M)$ for the shortest abstraction, i.e. with respect to $S = \{0\} \cup frac(M)$.

▶ **Example 10.** The abstraction of marking $M = \{(p,2.1),(q,2.2),(p,5.1),(q,5.1)\}$ is $abs(M) = \emptyset\ \{(p,2),(p,5),(q,5)\}\ \{(q,2)\}$. The first symbol is $\emptyset$, because M contains no token with an integer age (i.e., no token whose age has fractional part 0). The second and third symbols represent sets of tokens with fractional values 0.1 and 0.2, respectively.

Clocks with integer values play a special role in the behavior of TPN, because the constants in the transition guards are integers. Thus we always include the fractional part 0 in the set S in Definition 9.

We use a special kind of regular expressions over Σ to represent coverable sets of TPN markings as follows.

▶ **Definition 11.** A regular expression E over Σ represents the downward-closed set of TPN markings covered by one that has an abstraction in the language of E:

$$[\![E]\!] \stackrel{\text{def}}{=} \{N \mid \exists M \exists S.\ M \geq N \wedge abs_S(M) \in \mathcal{L}(E)\}.$$

An expression is *simple* if it is of the form $E = x_0 x_1 \ldots x_k$ where for all $i \leq k$ either $x_i \in \Sigma$ or $x_i = y_i^*$ for some $y_i \in \Sigma$. In the latter case we say that x_i *carries a star*. That is, a simple expression is free of Boolean combinators and uses only concatenation and Kleene star. We will write $\hat{x}_i$ to denote the symbol in Σ at position i: it is x_i if $x_i \in \Sigma$ and y_i otherwise.

▶ Remark 12. Notice that for all simple expressions α, β so that $|\alpha| > 0$, we have that $[\![\alpha\emptyset\beta]\!] = [\![\alpha\beta]\!]$. However, unless α has length 0 or is of the form $\alpha = \emptyset\alpha'$, we have $[\![\emptyset\alpha]\!] \neq [\![\alpha]\!]$. This is because a marking M that contains a token (p, c) with $frac(c) = 0$ has the property that all abstractions $abs_S(M) = x_0 \dots x_k$ of M have $x_0 \neq \emptyset$.

The following lemmas express the effect of TPN transitions at the level of the region abstraction. Lemmas 13 and 15 state that maximally firing of discrete transitions (the relation $\xrightarrow{*}_{Disc}$) is computable and monotone. Lemmas 16 and 17 state how to represent timed-step successor markings.

▶ **Lemma 13.** *For every non-consuming TPN $\mathcal{N}$ there are polynomial time computable functions $f : \Sigma \times \Sigma \times \Sigma \to \Sigma$ and $g : \Sigma \times \Sigma \times \Sigma \to \Sigma$ with the following properties.*

1. *f and g are monotone (w.r.t. subset ordering) in each argument.*
2. *$f(\alpha, \beta, x) \supseteq x$ and $g(\alpha, \beta, x) \supseteq x$ for all $\alpha, \beta, x \in \Sigma$.*
3. *Suppose that $E = x_0 x_1 \dots x_k$ is a simple expression, $\alpha \stackrel{def}{=} x_0$ and $\beta \stackrel{def}{=} \bigcup_{i>0} \hat{x}_i$, and $E' = x'_0 x'_1 \dots x'_k$ is the derived expression defined by conditions:*
   a. *$x'_0 \stackrel{def}{=} f(\alpha, \beta, x_0)$,*
   b. *$x'_i \stackrel{def}{=} g(\alpha, \beta, \hat{x}_i)^*$ for $i > 0$,*
   c. *x'_i carries a star iff x_i does.*

   Then $[\![E']\!] = \{M'' \mid \exists M \in [\![E]\!] \wedge M \xrightarrow{*}_{Disc} M' \geq M''\}$.

▶ **Definition 14.** We will write $SAT(E) \stackrel{\text{def}}{=} E'$ for the successor expression E' of E guaranteed by Lemma 13. I.e., $SAT(E)$ is the saturation of E by maximally firing discrete transitions.

Notice that by definition it holds that $[\![E]\!] \subseteq [\![SAT(E)]\!] \subseteq Cover([\![E]\!])$, and consequently also that $Cover([\![SAT(E)]\!]) = Cover([\![E]\!])$.

▶ **Lemma 15.** *Suppose that $X = x_0 x_1 \dots x_k$ is a simple expression of length $k+1$ with $SAT(X) = x'_0 x'_1 \dots x'_k$ and $x_0, x'_0 \in \Sigma$. Let $Y = y_0 \alpha_1 y_1 \alpha_2 \dots \alpha_k y_k$ be a simple expression with $SAT(Y) = y'_0 \alpha'_1 y'_1 \alpha'_2 \dots \alpha'_k y'_k$ and $y_0, y'_0 \in \Sigma$.*

If $\hat{x}_i \subseteq \hat{y}_i$ for all $i \leq k$ then $\hat{x}'_i \subseteq \hat{y}'_i$ for all $i \leq k$.

Proof. The assumption of the lemma provides that $\alpha_x \stackrel{\text{def}}{=} x_0 \subseteq \alpha_y \stackrel{\text{def}}{=} y_0$ and $\beta_x \stackrel{\text{def}}{=} \bigcup_{k \geq i > 0} \hat{x}_i \subseteq \beta_y \stackrel{\text{def}}{=} \bigcup_{k \geq i > 0} \hat{y}_i$. Therefore, by Item 1 of Lemma 13, we get that

$$x'_0 = f(\alpha_x, \beta_x, x_0) \quad \subseteq \quad f(\alpha_y, \beta_y, y_0) = y'_0$$

and similarly, for all $k \geq i \geq 0$, that $\hat{x}'_i = g(\alpha_x, \beta_x, \hat{x}_i) \ \subseteq \ g(\alpha_y, \beta_y, \hat{y}_i) = \hat{y}'_i$. ◀

For $x \in \Sigma$ we write $(x+1) \stackrel{\text{def}}{=} \{(p, int(n+1)) \mid (p, n) \in x\}$ for the symbol where token ages are incremented by 1.

▶ **Lemma 16.** $[\![\emptyset E]\!] = \{M' \mid \exists M \in [\![E]\!] \wedge M \longrightarrow_d M' \wedge d < 1 - \max(frac(M))\}$.

▶ **Lemma 17.** *Let αz be a simple expression where $\hat{z} = z \in \Sigma$ (the rightmost symbol is not starred). Then, $[\![(z+1)\alpha]\!]$ contains a marking N if, and only if, there exists markings $N' \geq N$ and M, and a set $S \subseteq [0, 1[$ so that*

1. *$|S| = |\alpha z|$*
2. *$abs_S(M) \in \mathcal{L}(\alpha z)$*
3. *$M \longrightarrow_d N'$ for $d = 1 - \max(S)$.*

Proof. Suppose markings N, N', M, a set $S \subseteq [0,1[$ and $d \in \mathbb{R}_{\geq 0}$ so that the conditions 1 to 3 are satisfied. Let $S' \stackrel{\text{def}}{=} \{0\} \cup \{s + d \mid s \in S \setminus \{d\}\}$. Then, $|S'| = |S|$ and $abs_{S'}(N') \in \mathcal{L}((z+1)\alpha)$, which witnesses that $N \in [\![(z+1)\alpha]\!]$.

Conversely, let $N \in [\![(z+1)\alpha]\!]$ be a non-empty marking. If $|\alpha| = 0$, then $N \in [\![(z+1)]\!]$ and so $abs_S(N) \in \mathcal{L}((z+1))$ for $S \stackrel{\text{def}}{=} frac(N) = \{0\}$. This means that $M \longrightarrow_1 N = (M+1)$ for a marking M with $abs_S(M) \in \mathcal{L}(z) = \mathcal{L}(\alpha z)$.

If $|\alpha| > 0$, pick some marking $N' \geq N$ and set S' so that $abs_{S'}(N') = (z+1)w$, for some word $w \in \mathcal{L}(\alpha)$. Then we must have that $|S'| = |(z+1)\alpha| > 1$ and so $d \stackrel{\text{def}}{=} \min(S' \setminus \{0\})$ exists. Let $S \stackrel{\text{def}}{=} \{s - d \mid s \in S'\} \cup \{1 - d\}$ and M be the unique marking with $M \longrightarrow_d N'$. Notice that $1 - d = \max(S)$. It follows that $abs_S(M) = wz \in \mathcal{L}(\alpha z)$. ◀

We will often use the following simple fact, which is a direct consequence of Lemma 17.

▶ **Corollary 18.** $[\![(z+1)\alpha]\!] \subseteq Cover([\![\alpha z]\!])$.

Finally, the following lemma will be the basis for our exploration algorithm.

▶ **Lemma 19.** *Let αx_0^* be a simple expression with $SAT(\alpha x_0^*) = \alpha x_0^*$. Then $Cover([\![\alpha x_0^*]\!]) = [\![\alpha x_0^*]\!] \cup Cover([\![(x_0+1)\alpha x_0^*]\!])$.*

Proof. For the right to left inclusion notice that $[\![\alpha x_0^*]\!] \subseteq Cover([\![\alpha x_0^*]\!])$ trivially holds. For the rest, we have $[\![(x_0+1)\alpha x_0^*]\!] \subseteq Cover([\![\alpha x_0^*]\!])$ by Corollary 18, and therefore $Cover([\![(x_0+1)\alpha x_0^*]\!]) \subseteq Cover(Cover([\![\alpha x_0^*]\!])) = Cover([\![\alpha x_0^*]\!])$. For the left to right inclusion, we equivalently show that

$$Cover([\![\alpha x_0^*]\!]) \setminus [\![\alpha x_0^*]\!] \subseteq Cover([\![(x_0+1)\alpha x_0^*]\!]) \tag{1}$$

Using the assumption that $SAT(\alpha x_0^*) = \alpha x_0^*$, the set on the left contains everything coverable from $[\![\alpha x_0^*]\!]$ by a sequence that starts with a (short) time step. It can therefore be written as

$$Cover(\{N_1 \mid \exists N_0 \in [\![\alpha x_0^*]\!] \wedge N_0 \longrightarrow_d N_1 \wedge 0 < d < 1 - \max(frac(N_0))\}).$$

By Lemma 16 and because $[\![\emptyset\alpha]\!] \subseteq [\![X\alpha]\!]$ for all $X \in \Sigma$ and $\alpha \in \Sigma^*$, we conclude that indeed, $Cover([\![\alpha x_0^*]\!]) \setminus [\![\alpha x_0^*]\!] \subseteq Cover([\![\emptyset\alpha x_0^*]\!]) \subseteq Cover([\![(x_0+1)\alpha x_0^*]\!])$. ◀

4.2 Acceleration

We propose an acceleration procedure based on unfolding expressions according to Lemma 19 (interleaved with saturation steps to guarantee its premise) and introducing new Kleene stars to keep the length of intermediate expressions bounded. This procedure (depicted in Algorithm 1), is used to characterize an initial subset of the coverability set.

Given a length-2 simple expression S_0 where the rightmost symbol is starred, the algorithm will first saturate (Definition 14, in line 1), and then alternatingly rotate a copy of the rightmost symbol (Lemma 17), and saturate the result (see lines 2, 3, 6). Since each such round extends the length of the expression by one, we additionally collapse them (in line 7) by adding an extra Kleene star to the symbol at the second position. The crucial observation for the correctness of this procedure is that the subsumption step in line 7 does not change the cover sets of the respective expressions.

Observe that Algorithm 1 is well defined because the $SAT(S_i)$ are computable by Lemma 13. Termination is guaranteed by the following simple observation.

Algorithm 1 Accelerate.

Input: a simple expression $S_0 = x_1 x_0^*$ (of length 2 and with last symbol starred)

Output: simple expressions S_1, S_i and R, of lengths 2, 4, and 2, respectively.

1: $S_1 \stackrel{\text{def}}{=} x_1^1 (x_0^1)^* = SAT(x_1 x_0^*)$
2: $S_2 \stackrel{\text{def}}{=} x_2^2 x_1^2 (x_0^2)^* = SAT((x_0^1 + 1) S_1)$
3: $S_3 \stackrel{\text{def}}{=} x_3^3 x_2^3 x_1^3 (x_0^3)^* = SAT((x_0^2 + 1) S_2)$
4: $i \leftarrow 3$
5: **repeat**
6: $\quad x_{i+1}^{i+1} x_i^{i+1} x_{i-1}^{i+1} x_1^{i+1} (x_0^{i+1})^* \stackrel{\text{def}}{=} SAT((x_0^i + 1) S_i)$
7: $\quad S_{i+1} \stackrel{\text{def}}{=} x_{i+1}^{i+1} (x_i^{i+1})^* x_1^{i+1} (x_0^{i+1})^*$
8: $\quad i \leftarrow i + 1$
9: **until** $S_i = S_{i-1}$
10: $R \stackrel{\text{def}}{=} (x_1^i + 1)(x_{i-1}^i)^*$
11: **return** S_1, S_i, R

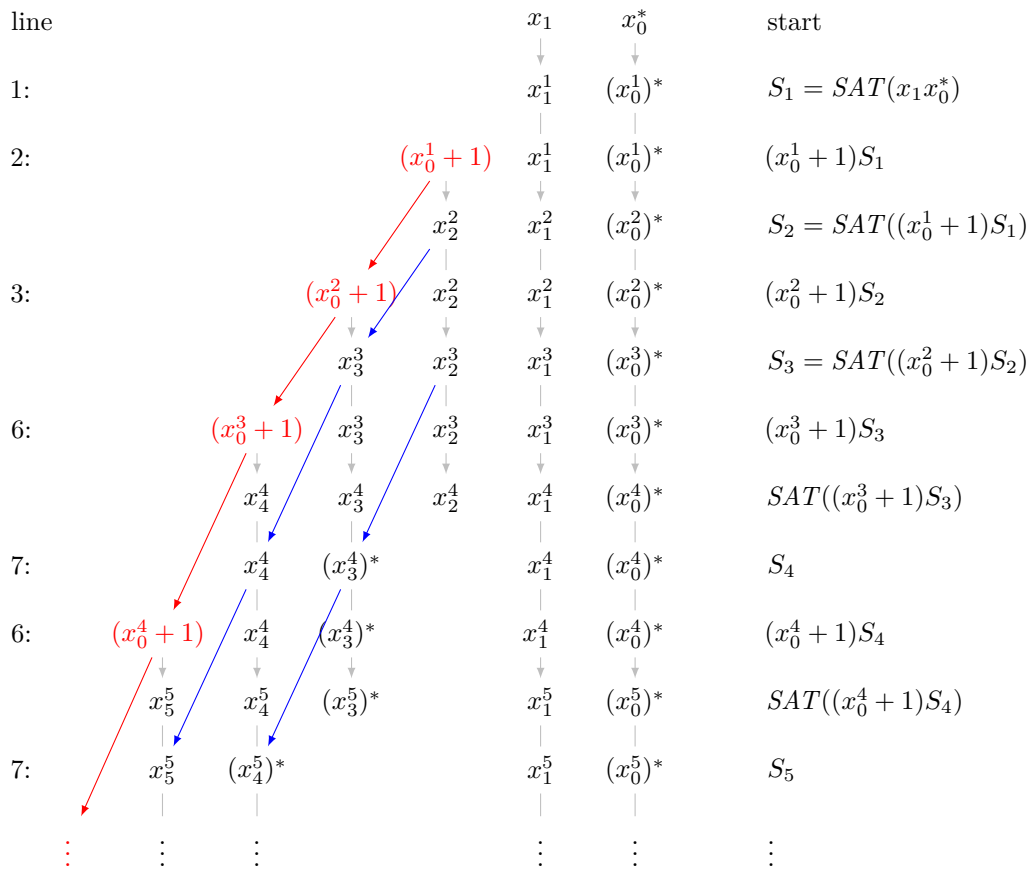


Figure 2 A Run of Algorithm 1 (initial steps). The column on the left indicates the line of code, the middle depicts the current expression and the column on the right recalls its origin. Gray bars indicate that the respective symbols are equal. Arrows denote (set) inclusion between symbols. The gray vertical arrows indicate inclusions due to saturation (Lemma 13), as claimed in item 1 of Lemma 20. Red and blue arrows indicate derived inclusions (as stated in Lemma 20).

▶ **Lemma 20.** *Let $x^i_j \in \Sigma$ be the symbols computed by Algorithm 1. Then*
1. *$x^{i+1}_j \supseteq x^i_j$, for all $i > j \geq 0$.*
2. *$x^i_i \supseteq x^{i-1}_{i-1}$ and $x^{i+1}_i \supseteq x^i_{i-1}$, for all $i \geq 3$.*

Proof. The first item is guaranteed by Point 2 of Lemma 13. In particular this means that $x^{i+1}_0 \supseteq x^i_0$ and therefore that $(x^{i+1}_0 + 1) \supseteq (x^i_0 + 1)$ for all $i \geq 0$ (indicated as red arrows in Figure 2). The second item now follows from this observation by Lemma 15. ◀

▶ **Lemma 21** (Termination)**.** *Algorithm 1 terminates with $i \leq 4 \cdot |P| \cdot (c_{max} + 1)$.*

Proof. From Lemma 20 we deduce that for all $i \geq 2$, the expression S_{i+1} is point-wise larger than or equal to S_i with respect to the subset ordering on symbols. The claim now follows from the observation that all expressions $S_{i\geq 3}$ have length 4 and that every symbol $x_i \in \Sigma$ can only increase at most $|P| \cdot (c_{max} + 1)$ times. ◀

▶ **Lemma 22** (Correctness)**.** *Suppose that S_1, S_ℓ, R be the expressions computed by Algorithm 1 applied to the simple expression $x_1 x_0^*$. Then $Cover([\![x_1 x_0^*]\!]) = [\![S_1]\!] \cup [\![S_\ell]\!] \cup Cover([\![R]\!])$.*

Proof. Let $S_1, \dots S_\ell$ denote the expressions defined in lines 1,2,3, and 7 of the algorithm. That is, ℓ is the least index i such that $S_{i+1} = S_i$. We define a sequence E_i of expressions inductively, starting with $E_1 \stackrel{\text{def}}{=} S_1$ and if $E_i = e^i_i e^i_{i-1} \dots e^i_0$, we let $E_{i+1} \stackrel{\text{def}}{=} e^{i+1}_{i+1} e^{i+1}_i e^{i+1}_{i-1} \dots e^{i+1}_0 \stackrel{\text{def}}{=} SAT((\hat{e}^i_0 + 1)E_i)$. Here, the superscript indicates the position of a symbol and not iteration. This is the sequence of expressions resulting from unfolding Lemma 19, interleaved with saturation steps, just in line 6 of the algorithm. That is, the expressions E_i are *not* collapsed (line 7) and instead grow in length with i. Still, $E_1 = S_1$, $E_2 = S_2$ and $E_2 = S_3$, but $E_4 \neq S_4$, because the latter is the result of applying the subsumption step of line 7 in our algorithm. Notice that $Cover([\![x_1 x_0^*]\!]) = \left(\bigcup_{k-1 \geq i \geq 1} [\![E_i]\!]\right) \cup Cover([\![E_k]\!])$ holds for all $k \in \mathbb{N}$. We will use that

$$\bigcup_{i \geq 2} [\![E_i]\!] = \bigcup_{i \geq 2} [\![S_i]\!] = [\![S_\ell]\!]. \tag{2}$$

We start by observing that for all $i, j \in \mathbb{N}$ it holds that $e^i_j = x^i_j$. For $i \leq 3$ this holds trivially by definition of $E_i = S_i$. For larger i, this can be seen by induction using Lemma 13. Towards the first equality in Equation (2), let S^j_i be the expression resulting from $S_i = x^i_i (x^i_{i-1})^* x^i_1 (x^i_0)^*$ by unfolding the first star j times. That is, $S^j_i \stackrel{\text{def}}{=} x^i_i (x^i_{i-1})^{(j)} x^i_1 (x^i_0)^*$, where the superscript (j) denotes j-fold concatenation. Clearly, $[\![S_i]\!] = \bigcup_{j \geq 0} [\![S^j_i]\!]$ and so the $\supseteq$-direction of the first equality in Equation (2) follows by

$$\begin{aligned}
[\![S^j_i]\!] = [\![x^i_i (x^i_{i-1})^{(j)} x^i_1 (x^i_0)^*]\!] &\subseteq [\![x^{i+j}_{i+j} \left(x^{i+j}_{i+j-1} x^{i+j}_{i+j-2} \dots x^{i+j}_i\right) x^{i+1}_1 (x^{i+1}_0)^*]\!] \\
&\subseteq [\![x^{i+j}_{i+j} \left(x^{i+j}_{i+j-1} x^{i+j}_{i+j-2} \dots x^{i+j}_i\right) \left(x^{i+j}_{i-1} \dots x^{i+j}_2\right) x^{i+1}_1 (x^{i+j}_0)^*]\!] \\
&= [\![E_{i+j}]\!],
\end{aligned}$$

where the first inclusion is due to Lemma 20. The same helps for the other direction:

$$[\![E_i]\!] = [\![x^i_i x^i_{i-1} x^i_{i-2} \dots x^i_2 x^i_1 x^i_0]\!] \subseteq [\![x^i_i (x^i_{i-1})^{(i-2)} x^i_1 x^i_0]\!] = [\![S^{i-2}_i]\!] = [\![S_i]\!], \tag{3}$$

which completes the proof of the first equality in Equation (2). The second equality holds because $[\![S_i]\!] \subseteq [\![S_{i+1}]\!]$ for all $i \geq 2$, by Lemma 20, and by definition of $S_\ell = S_{\ell+1}$. As a next step we show that

$$Cover([\![S_\ell]\!]) = [\![S_\ell]\!] \cup Cover([\![R]\!]) \tag{4}$$

First observe that $[\![R]\!] = [\![(x_1^\ell + 1)(x_{\ell-1}^\ell)^*]\!] = [\![(x_1^\ell + 1)x_\ell^\ell(x_{\ell-1}^\ell)^*]\!]$ and consequently,

$$\begin{aligned} \mathit{Cover}([\![R]\!]) &= \mathit{Cover}\Big([\![(x_1^\ell + 1)x_\ell^\ell(x_{\ell-1}^\ell)^*]\!]\Big) \\ &\subseteq \mathit{Cover}\Big([\![x_\ell^\ell(x_{\ell-1}^\ell)^* x_1^\ell]\!]\Big) \\ &\subseteq \mathit{Cover}\Big([\![x_\ell^\ell(x_{\ell-1}^\ell)^* x_1^\ell(x_0^\ell)^*]\!]\Big) = \mathit{Cover}([\![S_\ell]\!]) \end{aligned}$$

where the first equation follows by Corollary 18 and the second because $\mathcal{L}\Big(x_\ell^\ell(x_{\ell-1}^\ell)^* x_1^\ell\Big) \subseteq \mathcal{L}\Big(x_\ell^\ell(x_{\ell-1}^\ell)^* x_1^\ell(x_0^\ell)^*\Big)$. For the left to right inclusion in Equation (4), consider a marking $M \in \mathit{Cover}([\![S_\ell]\!]) \setminus [\![S_\ell]\!]$. We show that $M \in \mathit{Cover}([\![R]\!])$. Recall that $\mathit{Cover}([\![S_\ell]\!])$ consists of all those markings M so that there exists a finite path

$$M_0 \xrightarrow{*}_{Disc} M_0' \xrightarrow{d_1}_{Time} M_1 \xrightarrow{*}_{Disc} M_1' \xrightarrow{d_2}_{Time} M_2 \dots M_{k-1}' \xrightarrow{*}_{Disc} M_k$$

alternating between timed and (sequences of) discrete transition steps, with $M_0 \in [\![S_\ell]\!]$, $M_k \geq M$ and all $d_i \leq \max(\mathit{frac}(M_i'))$.

By our choice of M, there must be a first expression in the sequence which is not a member of $[\![S_\ell]\!]$. Since $[\![SAT(S_\ell)]\!] = [\![S_\ell]\!]$, we can assume an index $i > 0$ so that $M_i \notin [\![S_\ell]\!]$ but $M_{i-1}' \in [\![S_\ell]\!]$ that is, the step that takes us out of $[\![S_\ell]\!]$ is a timed step.

Because $[\![S_\ell]\!] = \bigcup_{i \geq 2} [\![S_i]\!]$, it must hold that $M_{i-1}' \in [\![S_j]\!] = [\![x_j^j(x_{j-1}^j)^* x_1^j(x_0^j)^*]\!]$ for some index $j \geq 2$. We claim that it already holds that

$$M_{i-1}' \in [\![x_j^j(x_{j-1}^j)^* x_1^j]\!]. \tag{5}$$

Suppose not. If $d_i < \max(\mathit{frac}(M_{i-1}'))$ then $M_i \in [\![\emptyset S_j]\!] \subseteq [\![S_j]\!]$ by Lemma 16, contradiction. Otherwise, if $d_i = \max(\mathit{frac}(M_{i-1}'))$, notice that every abstraction $\mathit{abs}_S(M_{i-1}') \in \mathcal{L}(S_j)$ must have $|S| = 4$. So by Lemma 17, $M_i \in [\![(x_0^j + 1)S_j]\!]$. But then again

$$[\![(x_0^j + 1)S_j]\!] \subseteq [\![SAT((x_0^j + 1)S_j)]\!] \subseteq [\![S_{j+1}]\!], \tag{6}$$

contradicting our assumption that $M_i \notin [\![S_\ell]\!]$. Therefore Equation (5) holds. By Lemma 17 we derive that $M_i \in [\![(x_1^j + 1)x_j^j(x_{j-1}^j)^*]\!] = [\![(x_1^j + 1)(x_{j-1}^j)^*]\!] \subseteq [\![(x_1^\ell + 1)(x_{\ell-1}^\ell)^*]\!] = [\![R]\!]$. This concludes the proof of Equation (4).

Notice that by Lemma 19 we have that

$$\mathit{Cover}([\![x_1 x_0^*]\!]) = [\![SAT(x_1 x_0^*)]\!] \cup \mathit{Cover}([\![SAT(x_1 x_0^*)]\!]) = [\![S_1]\!] \cup \mathit{Cover}([\![S_1]\!]). \tag{7}$$

Analogously, we get for every $i \geq 1$ that

$$\mathit{Cover}([\![E_i]\!]) = [\![SAT(E_i)]\!] \cup \mathit{Cover}\big([\![SAT((x_0^i + 1)E_i)]\!]\big) = [\![E_i]\!] \cup \mathit{Cover}([\![E_{i+1}]\!]) \tag{8}$$

This used Lemma 19 and the fact that $SAT(E_i) = E_i$ by construction. Using Equation (8) and that $[\![E_i]\!] \subseteq [\![E_{i+1}]\!]$ for $i \geq 2$, we deduce

$$\mathit{Cover}([\![S_1]\!]) = \mathit{Cover}([\![E_1]\!]) = [\![E_1]\!] \cup \left(\bigcup_{i \geq 2} \mathit{Cover}([\![E_i]\!]) \right). \tag{9}$$

Finally we can conclude the desired result as follows.

$$\begin{aligned} \mathit{Cover}(\llbracket x_1 x_0^* \rrbracket) &\overset{(7)}{=} \llbracket S_1 \rrbracket \cup \mathit{Cover}(\llbracket S_1 \rrbracket) \overset{(9)}{=} \llbracket S_1 \rrbracket \cup \mathit{Cover}\left(\bigcup_{i \geq 2} \llbracket E_i \rrbracket\right) \\ &\overset{(2)}{=} \llbracket S_1 \rrbracket \cup \mathit{Cover}(\llbracket S_\ell \rrbracket) \\ &\overset{(4)}{=} \llbracket S_1 \rrbracket \cup \llbracket S_\ell \rrbracket \cup \mathit{Cover}(\llbracket R \rrbracket) \end{aligned}$$ ◀

4.3 Main Result

The following theorem summarizes our main claims regarding the ∃COVER problem.

▶ **Theorem 23.** *Consider an instance of ∃COVER with $\mathcal{N} = (P, T, Var, G, Pre, Post)$ a non-consuming TPN where c_{max} is the largest constant appearing in the transition guards G encoded in unary, and let p be an initial place and t be a transition.*

1. *The number of different simple expressions of length m is $B(m) \overset{def}{=} 2^{(|P|\cdot(c_{max}+2)\cdot m)+m}$.*
2. *It is possible to compute a symbolic representation of the set of markings coverable from some marking in the initial set $\mathbb{N} \cdot \{(p,0)\}$, as a finite set of simple expressions. I.e., one can compute simple expressions $S_1, \ldots, S_\ell$ s.t. $\bigcup_{1 \leq i \leq \ell} \llbracket S_i \rrbracket = \mathit{Cover}(\mathbb{N} \cdot \{(p,0)\})$ and where $\ell \leq 3 \cdot B(2)$. Each of the S_i has length either 2 or 4.*
3. *Checking if there exists $M \in \mathit{Cover}(\mathbb{N} \cdot \{(p,0)\})$ with $M \longrightarrow_t$ can be done in $\mathcal{O}(|P| \cdot c_{max})$ deterministic space.*

Proof. For Item 1 note that a simple expression is described by a word where some symbols have a Kleene star. There are $|\Sigma|^m$ different words of length m and 2^m possibilities to attach stars to symbols. Since the alphabet is $\Sigma \overset{\text{def}}{=} 2^{P \times [c_{max}+1]}$ and $|[c_{max}+1]| = c_{max}+2$, the result follows.

Towards Item 2, we can assume w.l.o.g. that our TPN is non-consuming by Lemma 8, and thus the region abstraction introduced in Section 4.1 applies. In particular, the initial set of markings $\mathbb{N} \cdot \{(p,0)\}$ is represented exactly by the expression $S_0 \overset{\text{def}}{=} \{(p,0)\}\emptyset^*$ where $\emptyset \in \Sigma$ is the symbol corresponding to the empty set. That is, we have $\llbracket S_0 \rrbracket = \mathbb{N} \cdot \{(p,0)\}$ and thus $\mathit{Cover}(\llbracket S_0 \rrbracket) = \mathit{Cover}(\mathbb{N} \cdot \{(p,0)\})$.

The claimed expressions S_i are the result of iterating Algorithm 1 until a previously seen expression is revisited. Starting at $i = 0$ and $S_0 \overset{\text{def}}{=} \{(p,0)\}\emptyset^*$, each round will set S_{i+1}, S_{i+2} and S_{i+3} to the result of applying Algorithm 1 to S_i, and increment i to $i+3$.

Notice that then all S_i are simple expressions of length 2 or 4 and that in particular, all expressions with index divisible by 3 are of the form ab^* for $a, b \in \Sigma$. Therefore after at most $B(2)$ iterations, an expression S_ℓ is revisited (with $\ell \leq 3B(2)$). Finally, an induction using Lemma 22 provides that $\bigcup_{1 \leq i \leq \ell} \llbracket S_i \rrbracket = \mathit{Cover}(\mathbb{N} \cdot \{(p,0)\})$.

Towards Item 3, we modify the above algorithm for the ∃COVER problem with the sliding window technique. The algorithm is the same as above where instead of recording all the expressions $S_1, \ldots, S_\ell$, we only store the most recent ones and uses them to decide whether the transition t is enabled. If the index i reaches the maximal value of $3 \cdot B(2)$ we return unsuccessfully.

The bounded index counter uses $\mathcal{O}(\log(B(2)))$ space; Algorithm 1 uses space $\mathcal{O}(\log(B(5)))$ because it stores only simple expressions of length ≤ 5. The space required to store the three expressions resulting from each application of Algorithm 1 is $\mathcal{O}(3 \cdot \log(B(4)))$. For every encountered simple expression we can check in logarithmic space whether the transition t is enabled by some marking in its denotation. Altogether the space used by our new algorithm is bounded by $\mathcal{O}(\log(B(5)))$. By Item 1, this is $\mathcal{O}(|P| \cdot (c_{max}+2)) = \mathcal{O}(|P| \cdot c_{max})$. ◀

▶ **Corollary 24.** *The ∃COVER problem for TPN is* PSPACE*-complete.*

Proof. The PSPACE lower bound was shown in Theorem 7. The upper bound follows from Lemma 8 and Item 3 of Theorem 23. ◀

5 Conclusion and Future Work

We have shown that *Existential Coverability* (and its dual of universal safety) is PSPACE-complete for TPN with one real-valued clock per token. This implies the same complexity for checking safety of arbitrarily large timed networks without a central controller. The absence of a central controller makes a big difference, since the corresponding problem *with* a central controller is complete for $F_{\omega^{\omega^\omega}}$ [12].

It remains an open question whether these positive results for the controller-less case can be generalized to multiple real-valued clocks per token. In the case *with* a controller, safety becomes undecidable already for two clocks per token [2].

Another question is whether our results can be extended to more general versions of timed Petri nets. In our version, clock values are either inherited, advanced as time passes, or reset to zero. However, other versions of TPN allow the creation of output-tokens with new non-deterministically chosen non-zero clock values, e.g., the timed Petri nets of [3, 4] and the read-arc timed Petri nets of [8].

References

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

2 Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 345–354, 2004.

3 Parosh Aziz Abdulla, Pritha Mahata, and Richard Mayr. Dense-timed Petri nets: Checking Zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1), 2007.

4 Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs. In *International Conference on Application and Theory of Petri Nets (ICATPN)*, volume 2075 of *LNCS*, pages 53–70. Springer, 2001.

5 R. Alur and D. L. Dill. A theory of timed automata. *tcs*, 126(2):183–235, 1994.

6 Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 9135 of *LNCS*, 2015.

7 Rémi Bonnet, Alain Finkel, Serge Haddad, and Fernando Rosa-Velardo. Comparing Petri data nets and timed Petri nets. Technical Report LSV-10-23, LSV Cachan, 2010.

8 Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 420–431. Springer, 2006.

9 David de Frutos Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *International Conference on Application and Theory of Petri Nets (ICATPN)*, volume 1825 of *LNCS*, pages 187–206. Springer, 2000.

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

11 Eric Goles, Pedro Montealegre, Ville Salo, and Ilkka Törmä. PSPACE-completeness of majority automata networks. *Theor. Comput. Sci.*, 609(1):118–128, 2016.

12 Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The ordinal recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 355–364, 2012.

13 Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jiří Srba. Verification of timed-arc Petri nets. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 6543 of *LNCS*, pages 46–72, 2011.

14 Ranko Lazić, Tom Newcomb, Joël Ouaknine, A.W. Roscoe, and James Worrell. Nets with tokens which carry data. *Fundamenta Informaticae*, 88(3):251–274, 2008.

15 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*. IEEE Computer Society, 1999.

16 Jiří Srba. Timed-arc Petri nets vs. networks of timed automata. In *International Conference on Application and Theory of Petri Nets (ICATPN)*, volume 3536 of *LNCS*, pages 385–402. Springer, 2005.

It Is Easy to Be Wise After the Event: Communicating Finite-State Machines Capture First-Order Logic with "Happened Before"

Benedikt Bollig
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, Cachan, France
bollig@lsv.fr

Marie Fortin
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, Cachan, France
fortin@lsv.fr

Paul Gastin
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, Cachan, France
gastin@lsv.fr

Abstract

Message sequence charts (MSCs) naturally arise as executions of communicating finite-state machines (CFMs), in which finite-state processes exchange messages through unbounded FIFO channels. We study the first-order logic of MSCs, featuring Lamport's happened-before relation. We introduce a star-free version of propositional dynamic logic (PDL) with loop and converse. Our main results state that (i) every first-order sentence can be transformed into an equivalent star-free PDL sentence (and conversely), and (ii) every star-free PDL sentence can be translated into an equivalent CFM. This answers an open question and settles the exact relation between CFMs and fragments of monadic second-order logic. As a byproduct, we show that first-order logic over MSCs has the three-variable property.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases communicating finite-state machines, first-order logic, happened-before relation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.7

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1804.10076`.

Funding Partly supported by ANR FREDDA (ANR-17-CE40-0013) and UMI RELAX.

1 Introduction

First-order (FO) logic can be considered, in many ways, a reference specification language. It plays a key role in automated theorem proving and formal verification. In particular, FO logic over finite or infinite words is central in the verification of reactive systems. When a word is understood as a total order that reflects a chronological succession of events, it represents an execution of a sequential system. Apart from being a natural concept in itself, FO logic over words enjoys manifold characterizations. It defines exactly the star-free languages and coincides with recognizability by aperiodic monoids or natural subclasses of finite (Büchi, respectively) automata (cf. [8, 31] for overviews). Moreover, linear-time temporal logics are

usually measured against their expressive power with respect to FO logic. For example, LTL is considered the yardstick temporal logic not least due to Kamp's famous theorem, stating that LTL and FO logic are expressively equivalent [21].

While FO logic on words is well understood, a lot remains to be said once concurrency enters into the picture. When several processes communicate through, say, unbounded first-in first-out (FIFO) channels, events are only partially ordered and a behavior, which is referred to as a *message sequence chart (MSC)*, reflects Lamport's happened-before relation: an event e happens before an event f if, and only if, there is a "message flow" path from e to f [23]. *Communicating finite-state machines* (CFMs) [5] are to MSCs what finite automata are to words: a canonical model of finite-state processes that communicate through unbounded FIFO channels. Therefore, the FO logic of MSCs can be considered a canonical specification language for such systems. Unfortunately, its study turned out to be difficult, since algebraic and automata-theoretic approaches that work for words, trees, or Mazurkiewicz traces do not carry over. In particular, until now, the following central problem remained open:

> *Can every first-order sentence be transformed into an equivalent communicating finite-state machine, without any channel bounds?*

Partial answers were given for CFMs with bounded channel capacity [14, 20, 22] and for fragments of FO that restrict the logic to bounded-degree predicates [4] or to two variables [1].

In this paper, we answer the general question positively. To do so, we make a detour through a variant of propositional dynamic logic (PDL) with loop and converse [11, 29]. Actually, we introduce *star-free* PDL, which serves as an interface between FO logic and CFMs. That is, there are two main tasks to accomplish:

(i) Translate every FO sentence into a star-free PDL sentence.

(ii) Translate every star-free PDL sentence into a CFM.

Both parts constitute results of own interest. In particular, step (i) implies that, over MSCs, FO logic has the three-variable property, i.e., every FO sentence over MSCs can be rewritten into one that uses only three different variable names. Note that this is already interesting in the special case of words, where it follows from Kamp's theorem [21]. It is also noteworthy that star-free PDL is a *two-dimensional* temporal logic in the sense of Gabbay et al. [12,13]. Since every star-free PDL sentence is equivalent to some FO sentence, we actually provide a (higher-dimensional) temporal logic over MSCs that is expressively complete for FO logic.[1] While step (i) is based on purely logical considerations, step (ii) builds on new automata constructions that allow us to cope with the loop operator of PDL.

Combining (i) and (ii) yields the translation from FO logic to CFMs. It follows that CFMs are expressively equivalent to *existential* MSO logic. Moreover, we can derive self-contained proofs of several results on channel-bounded CFMs whose original proofs refer to involved constructions for Mazurkiewicz traces (cf. Section 5).

Related Work. Let us give a brief account of what was already known on the relation between logic and CFMs. In the 60s, Büchi, Elgot, and Trakhtenbrot proved that finite automata over words are expressively equivalent to monadic second-order logic [6, 10, 32]. Note that finite automata correspond to the special case of CFMs with a single process.

This classical result has been generalized to CFMs with bounded channels: Over *universally* bounded MSCs (where all possible linear extensions meet a given channel bound), deterministic CFMs are expressively equivalent to MSO logic [20, 22]. Over *existentially*

[1] It is open whether there is an equivalent one-dimensional one.

bounded MSCs (some linear extension meets the channel bound), CFMs are still expressively equivalent to MSO logic [14], but inherently nondeterministic [15]. The proofs of these characterizations reduce message-passing systems to finite-state shared-memory systems so that deep results from Mazurkiewicz trace theory [9] can be applied.

This generic approach is no longer applicable when the restriction on the channel capacity is dropped. Actually, in general, CFMs do not capture MSO logic [4]. On the other hand, they are expressively equivalent to existential MSO logic when we discard the happened-before relation [4] or when restricting to two first-order variables [1]. Both results rely on normal forms of FO logic, due to Hanf [19] and Scott [17], respectively. However, MSCs with the happened-before relation are structures of *unbounded* degree (while Hanf's normal form requires structures of bounded degree), and we consider FO logic with *arbitrarily* many variables (while Scott's normal form only applies to two-variable logic). That is, neither approach is applicable in our case.

Finally, there exists a translation of a loop-free PDL into CFMs [3]. As our star-free PDL has a loop operator, we cannot exploit [3] either.

Outline. In Section 2, we recall basic notions such as MSCs, FO logic, and CFMs. Moreover, we state one of our main results: the translation of FO formulas into CFMs. Section 3 presents star-free PDL and proves that it captures FO logic. In Section 4, we establish the translation of star-free PDL into CFMs. We conclude in Section 5 mentioning applications of our results. Some proof details can be found in the long version [2].

2 Preliminaries

We consider message-passing systems in which processes communicate through unbounded FIFO channels. We fix a nonempty finite set of *processes* P and a nonempty finite set of *labels* Σ. For all $p, q \in P$ such that $p \neq q$, there is a channel (p, q) that allows p to send messages to q. The set of channels is denoted Ch.

In the following, we define message sequence charts, which represent executions of a message-passing system, and logics to reason about them. Then, we recall the definition of communicating finite-state machines and state one of our main results.

2.1 Message Sequence Charts

A *message sequence chart (MSC)* (over P and Σ) is a graph $M = (E, \rightarrow, \lhd, loc, \lambda)$ with nonempty finite set of nodes E, edge relations $\rightarrow, \lhd \subseteq E \times E$, and node-labeling functions $loc\colon E \to P$ and $\lambda\colon E \to \Sigma$. An example MSC is depicted in Figure 1. A node $e \in E$ is an *event* that is executed by process $loc(e) \in P$. In particular, $E_p := \{e \in E \mid loc(e) = p\}$ is the set of events located on p. The label $\lambda(e) \in \Sigma$ may provide more information about e such as the message that is sent/received at e or "enter critical section" or "output some value".

Edges describe causal dependencies between events:

- The relation $\rightarrow$ contains *process edges*. They connect successive events executed by the same process. That is, we actually have $\rightarrow \subseteq \bigcup_{p \in P}(E_p \times E_p)$. Every process p is sequential so that $\rightarrow \cap (E_p \times E_p)$ must be the direct-successor relation of some total order on E_p. We let $\leq_{\mathsf{proc}} := \rightarrow^*$ and $<_{\mathsf{proc}} := \rightarrow^+$.
- The relation $\lhd$ contains *message edges*. If $e \lhd f$, then e is a *send event* and f is the corresponding *receive event*. In particular, $(loc(e), loc(f)) \in Ch$. Each event is part of at most one message edge. An event that is neither a send nor a receive event is called *internal*. Moreover, for all $(p, q) \in Ch$ and $(e, f), (e', f') \in \lhd \cap (E_p \times E_q)$, we have $e \leq_{\mathsf{proc}} e'$ iff $f \leq_{\mathsf{proc}} f'$ (which guarantees a FIFO behavior).

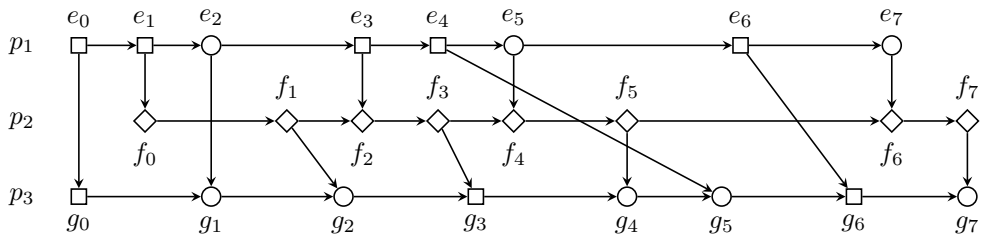


Figure 1 A message sequence chart (MSC).

We require that $\to \cup \lhd$ be acyclic (intuitively, messages cannot travel backwards in time). The associated partial order is denoted $\leq := (\to \cup \lhd)^*$ with strict part $< = (\to \cup \lhd)^+$. We do not distinguish isomorphic MSCs. Let $\mathbb{MSC}(P, \Sigma)$ denote the set of MSCs over P and Σ.

Actually, MSCs are very similar to the space-time diagrams from Lamport's seminal paper [23], and $\leq$ is commonly referred to as the *happened-before relation*.

It is worth noting that, when P is a singleton, an MSC with events $e_1 \to e_2 \to \ldots \to e_n$ can be identified with the word $\lambda(e_1)\lambda(e_2)\ldots\lambda(e_n) \in \Sigma^*$.

▶ **Example 1.** Consider the MSC from Figure 1 over $P = \{p_1, p_2, p_3\}$ and $\Sigma = \{\square, \bigcirc, \diamond\}$. We have, for instance, $E_{p_1} = \{e_0, \ldots, e_7\}$. The process relation is given by $e_i \to e_{i+1}$, $f_i \to f_{i+1}$, and $g_i \to g_{i+1}$ for all $i \in \{0, \ldots, 6\}$. Concerning the message relation, we have $e_1 \lhd f_0$, $e_4 \lhd g_5$, etc. Moreover, $e_2 \leq f_3$, but neither $e_2 \leq f_1$ nor $f_1 \leq e_2$.

2.2 MSO Logic and Its Fragments

Next, we give an account of *monadic second-order* (MSO) logic and its fragments. Note that we restrict our attention to MSO logic interpreted *over MSCs*. We fix an infinite supply $\mathcal{V}_{\mathsf{event}} = \{x, y, \ldots\}$ of first-order variables, which range over events of an MSC, and an infinite supply $\mathcal{V}_{\mathsf{set}} = \{X, Y, \ldots\}$ of second-order variables, ranging over sets of events. The syntax of MSO (we consider that P and Σ are fixed) is given as follows:

$$\Phi \ ::= \ p(x) \mid a(x) \mid x = y \mid x \to y \mid x \lhd y \mid x \leq y \mid x \in X \mid \Phi \vee \Phi \mid \neg\Phi \mid \exists x.\Phi \mid \exists X.\Phi$$

where $p \in P$, $a \in \Sigma$, $x, y \in \mathcal{V}_{\mathsf{event}}$, and $X \in \mathcal{V}_{\mathsf{set}}$. We use the usual abbreviations to also include implication $\Longrightarrow$, conjunction $\wedge$, and universal quantification $\forall$. Moreover, the relation $x \leq_{\mathsf{proc}} y$ can be defined by $x \leq y \wedge \bigvee_{p \in P} p(x) \wedge p(y)$. We write $\mathsf{Free}(\Phi)$ the set of free variables of Φ.

Let $M = (E, \to, \lhd, loc, \lambda)$ be an MSC. An *interpretation* (for M) is a mapping $\nu \colon \mathcal{V}_{\mathsf{event}} \cup \mathcal{V}_{\mathsf{set}} \to E \cup 2^E$ assigning to each $x \in \mathcal{V}_{\mathsf{event}}$ an event $\nu(x) \in E$, and to each $X \in \mathcal{V}_{\mathsf{set}}$ a set of events $\nu(X) \subseteq E$. We write $M, \nu \models \Phi$ if M satisfies Φ when the free variables of Φ are interpreted according to ν. Hereby, satisfaction is defined in the usual manner. In fact, whether $M, \nu \models \Phi$ holds or not only depends on the interpretation of variables that occur free in Φ. Thus, we may restrict ν to any set of variables that contains at least all free variables. For example, for $\Phi(x, y) = (x \lhd y)$, we have $M, [x \mapsto e, y \mapsto f] \models \Phi(x, y)$ iff $e \lhd f$. For a *sentence* $\Phi \in \mathrm{MSO}$ (without free variables), we define $L(\Phi) := \{M \in \mathbb{MSC}(P, \Sigma) \mid M \models \Phi\}$.

We say that two formulas Φ and Φ' are *equivalent*, written $\Phi \equiv \Phi'$, if, for all MSCs $M = (E, \to, \lhd, loc, \lambda)$ and interpretations $\nu \colon \mathcal{V}_{\mathsf{event}} \cup \mathcal{V}_{\mathsf{set}} \to E \cup 2^E$, we have $M, \nu \models \Phi$ iff $M, \nu \models \Phi'$.

Let us identify two important fragments of MSO logic: *First-order* (FO) formulas do not make use of second-order quantification (however, they may contain formulas $x \in X$). Moreover, *existential* MSO (EMSO) formulas are of the form $\exists X_1 \dots \exists X_n.\Phi$ with $\Phi \in \mathrm{FO}$.

Let $\mathcal{F}$ be MSO or EMSO or FO and let $R \subseteq \{\rightarrow, \lhd, \leq\}$. We obtain the logic $\mathcal{F}[R]$ by restricting $\mathcal{F}$ to formulas that do not make use of $\{\rightarrow, \lhd, \leq\} \setminus R$. Note that $\mathcal{F} = \mathcal{F}[\rightarrow, \lhd, \leq]$. Moreover, we let $\mathcal{L}(\mathcal{F}[R]) := \{L(\Phi) \mid \Phi \in \mathcal{F}[R] \text{ is a sentence}\}$.

Since the reflexive transitive closure of an MSO-definable binary relation is MSO-definable, MSO and MSO$[\rightarrow, \lhd]$ have the same expressive power: $\mathcal{L}(\mathrm{MSO}[\rightarrow, \lhd, \leq]) = \mathcal{L}(\mathrm{MSO}[\rightarrow, \lhd])$. However, MSO$[\leq]$ (without the message relation) is strictly weaker than MSO [4].

▶ **Example 2.** We give an FO formula that allows us to recover, at some event f, the most recent event e that happened in the past on, say, process p. More precisely, we define the predicate $\mathit{latest}_p(x, y)$ as $x \leq y \wedge p(x) \wedge \forall z\big((z \leq y \wedge p(z)) \implies z \leq x\big)$. The "gossip language" says that process q always maintains the latest information that it can have about p. Thus, it is defined by $\Phi^{\mathsf{gossip}}_{p,q} = \forall x \forall y.\big((\mathit{latest}_p(x, y) \wedge q(y)) \implies \bigvee_{a \in \Sigma}(a(x) \wedge a(y))\big) \in \mathrm{FO}^3[\leq]$. For example, for $P = \{p_1, p_2, p_3\}$ and $\Sigma = \{\Box, \bigcirc, \Diamond\}$, the MSC M from Figure 1 is contained in $L(\Phi^{\mathsf{gossip}}_{p_1,p_3})$. In particular, $M, [x \mapsto e_5, y \mapsto g_5] \models \mathit{latest}_{p_1}(x, y)$ and $\lambda(e_5) = \lambda(g_5) = \bigcirc$.

2.3 Communicating Finite-State Machines

In a communicating finite-state machine, each process $p \in P$ can perform internal actions of the form $\langle a \rangle$, where $a \in \Sigma$, or send/receive messages from a finite set of messages Msg. A send action $\langle a, !_q m \rangle$ of process p writes message $m \in \mathit{Msg}$ to channel (p, q), and performs $a \in \Sigma$. A receive action $\langle a, ?_q m \rangle$ reads message m from channel (q, p). Accordingly, we let $\mathit{Act}_p(\mathit{Msg}) := \{\langle a \rangle \mid a \in \Sigma\} \cup \{\langle a, !_q m \rangle \mid a \in \Sigma,\ m \in \mathit{Msg},\ q \in P \setminus \{p\}\} \cup \{\langle a, ?_q m \rangle \mid a \in \Sigma,\ m \in \mathit{Msg},\ q \in P \setminus \{p\}\}$ denote the set of possible actions of process p.

A *communicating finite-state machine (CFM)* over P and Σ is a tuple $((\mathcal{A}_p)_{p \in P}, \mathit{Msg}, \mathit{Acc})$ consisting of a finite set of messages Msg and a finite-state transition system $\mathcal{A}_p = (S_p, \iota_p, \Delta_p)$ for each process p, with finite set of states S_p, initial state $\iota_p \in S_p$, and transition relation $\Delta_p \subseteq S_p \times \mathit{Act}_p(\mathit{Msg}) \times S_p$. Moreover, we have an acceptance condition $\mathit{Acc} \subseteq \prod_{p \in P} S_p$.

Given a transition $t = (s, \alpha, s') \in \Delta_p$, we let $\mathit{source}(t) = s$ and $\mathit{target}(t) = s'$ denote the source and target states of t. In addition, if $\alpha = \langle a \rangle$, then t is an *internal transition* and we let $\mathit{label}(t) = a$. If $\alpha = \langle a, !_q m \rangle$, then t is a *send transition* and we let $\mathit{label}(t) = a$, $\mathit{msg}(t) = m$, and $\mathit{receiver}(t) = q$. Finally, if $\alpha = \langle a, ?_q m \rangle$, then t is a *receive transition* with $\mathit{label}(t) = a$, $\mathit{msg}(t) = m$, and $\mathit{sender}(t) = q$.

A *run* ρ of $\mathcal{A}$ on an MSC $M = (E, \rightarrow, \lhd, \mathit{loc}, \lambda) \in \mathbb{MSC}(P, \Sigma)$ is a mapping associating with each event $e \in E_p$ a transition $\rho(e) \in \Delta_p$, and satisfying the following conditions:

1. for all events $e \in E$, we have $\mathit{label}(\rho(e)) = \lambda(e)$,
2. for all $\rightarrow$-minimal events $e \in E$, we have $\mathit{source}(\rho(e)) = \iota_p$, where $p = \mathit{loc}(e)$,
3. for all process edges $(e, f) \in \rightarrow$, we have $\mathit{target}(\rho(e)) = \mathit{source}(\rho(f))$,
4. for all internal events $e \in E$, $\rho(e)$ is an internal transition, and
5. for all message edges $e \lhd f$, $\rho(e)$ and $\rho(f)$ are respectively send and receive transitions such that $\mathit{msg}(\rho(e)) = \mathit{msg}(\rho(f))$, $\mathit{receiver}(\rho(e)) = \mathit{loc}(f)$, and $\mathit{sender}(\rho(f)) = \mathit{loc}(e)$.

To determine whether ρ is accepting, we collect the last state s_p of every process p. If $E_p \neq \emptyset$, we let $s_p = \mathit{target}(\rho(e))$, where e is the last event of E_p. Otherwise, $s_p = \iota_p$. We say that ρ is *accepting* if $(s_p)_{p \in P} \in \mathit{Acc}$.

The *language* $L(\mathcal{A})$ of $\mathcal{A}$ is the set of MSCs M such that there exists an accepting run of $\mathcal{A}$ on M. Moreover, $\mathcal{L}(\mathrm{CFM}) := \{L(\mathcal{A}) \mid \mathcal{A} \text{ is a CFM}\}$. Recall that, for these definitions, we have fixed P and Σ.

One of our main results states that CFMs and EMSO logic are expressively equivalent. This solves a problem that was stated as open in [15]:

▶ **Theorem 3.** $\mathcal{L}(\mathrm{EMSO}[\rightarrow, \lhd, \leq]) = \mathcal{L}(\mathrm{CFM})$.

It is standard to prove $\mathcal{L}(\mathrm{CFM}) \subseteq \mathcal{L}(\mathrm{EMSO}[\rightarrow, \lhd])$: The formula guesses an assignment of transitions to events in terms of existentially quantified second-order variables (one for each transition) and then checks, in its first-order kernel, that the assignment is indeed an (accepting) run. As, moreover, the class $\mathcal{L}(\mathrm{CFM})$ is closed under projection, the proof of Theorem 3 comes down to the proposition below (whose proof is spread over Sections 3 and 4). Note that the translation from $\mathrm{FO}[\rightarrow, \lhd, \leq]$ to CFMs is inherently non-elementary, already when $|P| = 1$ [28].

▶ **Proposition 4.** $\mathcal{L}(\mathrm{FO}[\rightarrow, \lhd, \leq]) \subseteq \mathcal{L}(\mathrm{CFM})$.

3 Star-Free Propositional Dynamic Logic

In this section, we introduce a star-free version of propositional dynamic logic and show that it is expressively equivalent to $\mathrm{FO}[\rightarrow, \lhd, \leq]$. This is the second main result of the paper. Then, in Section 4, we show how to translate star-free PDL formulas into CFMs.

3.1 Syntax and Semantics

Originally, propositional dynamic logic (PDL) has been used to reason about program schemas and transition systems [11]. Since then, PDL and its extension with intersection and converse have developed a rich theory with applications in artificial intelligence and verification [7, 16, 18, 24, 25]. It has also been applied in the context of MSCs [3, 27].

Here, we introduce a *star-free* version of PDL, denoted $\mathrm{PDL}_{\mathsf{sf}}$. It will serve as an "interface" between FO logic and CFMs. The syntax of $\mathrm{PDL}_{\mathsf{sf}}$ and its fragment $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ is given by the following grammar:

$\mathrm{PDL}_{\mathsf{sf}} = \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}, \cup, \cap, \mathsf{c}]$

$\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$	$\xi ::= \mathsf{E}\,\varphi \mid \xi \vee \xi \mid \neg\xi$	
	$\varphi ::= p \mid a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\,\varphi \mid \mathsf{Loop}(\pi)$	
	$\pi ::= \rightarrow \mid \leftarrow \mid \lhd_{p,q} \mid \lhd_{p,q}^{-1} \mid \overset{\varphi}{\rightarrow} \mid \overset{\varphi}{\leftarrow} \mid \mathsf{jump}_{p,r} \mid \{\varphi\}? \mid \pi \cdot \pi$	$\pi \cup \pi \mid \pi \cap \pi \mid \pi^{\mathsf{c}}$

where $p, r \in P$, $q \in P \setminus \{p\}$, and $a \in \Sigma$. We refer to ξ as a *sentence*, to φ as an *event formula*, and to π as a *path formula*. We name the logic star-free because we use the operators $(\cup, \cap, \mathsf{c}, \cdot)$ of star-free regular expressions instead of the regular-expression operators $(\cup, \cdot, *)$ of classical PDL. However, the formula $\overset{\varphi}{\rightarrow}$, whose semantics is explained below, can be seen as a restricted use of the construct π^*.

A sentence ξ is evaluated wrt. an MSC $M = (E, \rightarrow, \lhd, loc, \lambda)$. An event formula φ is evaluated wrt. M and an event $e \in E$. Finally, a path formula π is evaluated over *two* events. In other words, it defines a binary relation $[\![\pi]\!]_M \subseteq E \times E$. We often write $M, e, f \models \pi$ to denote $(e, f) \in [\![\pi]\!]_M$. Moreover, for $e \in E$, we let $[\![\pi]\!]_M(e) := \{f \in E \mid (e, f) \in [\![\pi]\!]_M\}$. When M is clear from the context, we may write $[\![\pi]\!]$ instead of $[\![\pi]\!]_M$. The semantics of sentences, event formulas, and path formulas is given in Table 1.

Table 1 The semantics of $\mathrm{PDL}_{\mathsf{sf}}$.

$M \models \mathsf{E}\,\varphi$ if $M, e \models \varphi$ for some event $e \in E$	
$M \models \neg\xi$ if $M \not\models \xi$	$M \models \xi_1 \vee \xi_2$ if $M \models \xi_1$ or $M \models \xi_2$
$M, e \models p$ if $loc(e) = p$	$M, e \models \langle\pi\rangle\,\varphi$ if $\exists f \in [\![\pi]\!]_M(e) : M, f \models \varphi$
$M, e \models a$ if $\lambda(e) = a$	$M, e \models \mathsf{Loop}(\pi)$ if $(e,e) \in [\![\pi]\!]_M$
$M, e \models \neg\varphi$ if $M, e \not\models \varphi$	$M, e \models \varphi_1 \vee \varphi_2$ if $M, e \models \varphi_1$ or $M, e \models \varphi_2$
$[\![\to]\!]_M := \{(e,f) \in E \times E \mid e \to f\}$	$[\![\lhd_{p,q}]\!]_M := \{(e,f) \in E_p \times E_q \mid e \lhd f\}$
$[\![\leftarrow]\!]_M := \{(f,e) \in E \times E \mid e \to f\}$	$[\![\lhd^{-1}_{p,q}]\!]_M := \{(f,e) \in E_q \times E_p \mid e \lhd f\}$
$[\![\mathsf{jump}_{p,r}]\!]_M := E_p \times E_r$	$[\![\{\varphi\}?]\!]_M := \{(e,e) \mid e \in E : M, e \models \varphi\}$
$[\![\xrightarrow{\varphi}]\!]_M := \{(e,f) \in E \times E \mid e <_{\mathsf{proc}} f$ and $\forall g \in E\colon e <_{\mathsf{proc}} g <_{\mathsf{proc}} f \implies M, g \models \varphi\}$	
$[\![\xleftarrow{\varphi}]\!]_M := \{(e,f) \in E \times E \mid f <_{\mathsf{proc}} e$ and $\forall g \in E\colon f <_{\mathsf{proc}} g <_{\mathsf{proc}} e \implies M, g \models \varphi\}$	
$[\![\pi_1 \cdot \pi_2]\!]_M := \{(e,g) \in E \times E \mid \exists f \in E : (e,f) \in [\![\pi_1]\!]_M \wedge (f,g) \in [\![\pi_2]\!]_M\}$	
$[\![\pi_1 \cup \pi_2]\!]_M := [\![\pi_1]\!]_M \cup [\![\pi_2]\!]_M$	$[\![\pi^{\mathsf{c}}]\!]_M := (E \times E) \setminus [\![\pi]\!]_M$
$[\![\pi_1 \cap \pi_2]\!]_M := [\![\pi_1]\!]_M \cap [\![\pi_2]\!]_M$	

▶ **Example 5.** Consider again the MSC M from Figure 1 and the path formula $\pi = \lhd^{-1}_{p_1,p_3} \to \lhd_{p_1,p_2} \to \lhd_{p_2,p_3} \to$. We have $M, g_5 \models \mathsf{Loop}(\pi)$. Moreover, $(e_2, e_5) \in [\![\xrightarrow{\square}]\!]_M$ but $(e_2, e_6) \notin [\![\xrightarrow{\square}]\!]_M$.

We use the usual abbreviations for sentences and event formulas such as implication and conjunction. Moreover, $true := p \vee \neg p$ (for some arbitrary process $p \in P$) and $false := \neg true$. Finally, we define the event formula $\langle\pi\rangle := \langle\pi\rangle\, true$, and the path formulas $\xrightarrow{+} := \xrightarrow{true}$ and $\xrightarrow{*} := \xrightarrow{+} \cup \{true\}?$.

Note that there are some redundancies in the logic. For example (letting $\equiv$ denote logical equivalence), $\to \equiv \xrightarrow{false}$, $\pi_1 \cap \pi_2 \equiv (\pi_1^{\mathsf{c}} \cup \pi_2^{\mathsf{c}})^{\mathsf{c}}$, and $\mathsf{Loop}(\pi) \equiv \langle\{true\}? \cap \pi\rangle$. Some of them are necessary to define certain subclasses of $\mathrm{PDL}_{\mathsf{sf}}$. For every $R \subseteq \{\mathsf{Loop}, \cup, \cap, \mathsf{c}\}$, we let $\mathrm{PDL}_{\mathsf{sf}}[R]$ denote the fragment of $\mathrm{PDL}_{\mathsf{sf}}$ that does not make use of $\{\mathsf{Loop}, \cup, \cap, \mathsf{c}\} \setminus R$. In particular, $\mathrm{PDL}_{\mathsf{sf}} = \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}, \cup, \cap, \mathsf{c}]$. Note that, syntactically, $\xrightarrow{*}$ is not contained in $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ since union is not permitted.

3.2 Main Results

Let $\mathrm{FO}^3[\to, \lhd, \leq]$ be the set of formulas from $\mathrm{FO}[\to, \lhd, \leq]$ that use at most three different first-order variables (however, a variable can be quantified and reused several times in a formula). The main result of this section is that, for formulas with zero or one free variable, the logics $\mathrm{FO}[\to, \lhd, \leq]$, $\mathrm{FO}^3[\to, \lhd, \leq]$, $\mathrm{PDL}_{\mathsf{sf}}$, and $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ are expressively equivalent.

Consider $\mathrm{FO}[\to, \lhd, \leq]$ formulas Φ_0, $\Phi_1(x)$ and $\Phi_2(x, y)$ with respectively zero, one, and two free variables (hence, Φ_0 is a sentence). Consider also some $\mathrm{PDL}_{\mathsf{sf}}$ sentence ξ, event formula φ, and path formula π. The respective formulas are equivalent, written $\Phi_0 \equiv \xi$,

$\Phi_1(x) \equiv \varphi$, and $\Phi_2(x,y) \equiv \pi$, if, for all MSCs M and all events e, f in M, we have

$$\begin{aligned} M &\models \Phi_0 && \text{iff} && M \models \xi \\ M, [x \mapsto e] &\models \Phi_1(x) && \text{iff} && M, e \models \varphi \\ M, [x \mapsto e, y \mapsto f] &\models \Phi_2(x,y) && \text{iff} && M, e, f \models \pi \end{aligned}$$

We start with a simple observation, which can be shown easily by induction:

▶ **Proposition 6.** *Every* $\mathrm{PDL}_{\mathsf{sf}}$ *formula is equivalent to some* $\mathrm{FO}^3[\rightarrow, \lhd, \leq]$ *formula. More precisely, for every* $\mathrm{PDL}_{\mathsf{sf}}$ *sentence* ξ*, event formula* φ*, and path formula* π*, there exist some* $\mathrm{FO}^3[\rightarrow, \lhd, \leq]$ *sentence* $\widetilde{\xi}$*, formula* $\widetilde{\varphi}(x)$ *with one free variable, and formula* $\widetilde{\pi}(x,y)$ *with two free variables, respectively, such that,* $\xi \equiv \widetilde{\xi}$*,* $\varphi \equiv \widetilde{\varphi}(x)$*, and* $\pi \equiv \widetilde{\pi}(x,y)$*.*

The main result is a *strong* converse of Proposition 6:

▶ **Theorem 7.** *Every* $\mathrm{FO}[\rightarrow, \lhd, \leq]$ *formula with at most two free variables is equivalent to some* $\mathrm{PDL}_{\mathsf{sf}}$ *formula. More precisely, for every* $\mathrm{FO}[\rightarrow, \lhd, \leq]$ *sentence* Φ_0*, formula* $\Phi_1(x)$ *with one free variable, and formula* $\Phi_2(x,y)$ *with two free variables, there exist some* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *sentence* ξ*,* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *event formula* φ*, and* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *path formulas* π_{ij}*, respectively, such that,* $\Phi_0 \equiv \xi$*,* $\Phi_1(x) \equiv \varphi$*, and* $\Phi_2(x,y) \equiv \bigcup_i \bigcap_j \pi_{ij}$*.*

From Theorem 7 and Proposition 6, we deduce that FO has the three variable property:

▶ **Corollary 8.** $\mathcal{L}(\mathrm{FO}[\rightarrow, \lhd, \leq]) = \mathcal{L}(\mathrm{FO}^3[\rightarrow, \lhd, \leq])$.

3.3 From FO to $\mathrm{PDL}_{\mathsf{sf}}$

In the remainder of this section, we give the translation from FO to $\mathrm{PDL}_{\mathsf{sf}}$. We start with some basic properties of $\mathrm{PDL}_{\mathsf{sf}}$. First, the converse of a $\mathrm{PDL}_{\mathsf{sf}}$ formula is definable in $\mathrm{PDL}_{\mathsf{sf}}$ (easy induction on π).

▶ **Lemma 9.** *Let* $R \subseteq \{\mathsf{Loop}, \cup, \cap, \mathsf{c}\}$ *and* $\pi \in \mathrm{PDL}_{\mathsf{sf}}[R]$ *be a path formula. There exists* $\pi^{-1} \in \mathrm{PDL}_{\mathsf{sf}}[R]$ *such that, for all MSCs* M*,* $[\![\pi^{-1}]\!]_M = [\![\pi]\!]_M^{-1} = \{(f,e) \mid (e,f) \in [\![\pi]\!]_M\}$*.*

Given a $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ path formula π, we denote by $\mathsf{Comp}(\pi)$ the set of pairs $(p,q) \in P \times P$ such that there may be a π-path from some event on process p to some event on process q. Formally, we let $\mathsf{Comp}(\rightarrow) = \mathsf{Comp}(\leftarrow) = \mathsf{Comp}(\xrightarrow{\varphi}) = \mathsf{Comp}(\xleftarrow{\varphi}) = \mathsf{Comp}(\{\varphi\}?) = \mathsf{id}$, where $\mathsf{id} = \{(p,p) \mid p \in P\}$; $\mathsf{Comp}(\lhd_{p,q}) = \mathsf{Comp}(\lhd_{q,p}^{-1}) = \{(p,q)\}$; $\mathsf{Comp}(\mathsf{jump}_{p,r}) = \{(p,r)\}$; and $\mathsf{Comp}(\pi_1 \cdot \pi_2) = \mathsf{Comp}(\pi_2) \circ \mathsf{Comp}(\pi_1) = \{(p,r) \mid \exists q : (p,q) \in \mathsf{Comp}(\pi_1), (q,r) \in \mathsf{Comp}(\pi_2)\}$.

Notice that, for all path formulas $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$, the relation $\mathsf{Comp}(\pi)$ is either empty or a singleton $\{(p,q)\}$ or the identity id. Moreover, $M, e, f \models \pi$ implies $(loc(e), loc(f)) \in \mathsf{Comp}(\pi)$. Therefore, all events in $[\![\pi]\!](e)$ are on the same process, and if this set is nonempty (i.e., if $M, e \models \langle\pi\rangle$), then $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$ are well-defined.

▶ **Example 10.** Consider the MSC from Figure 1 and $\pi = \xrightarrow{+} \lhd_{p_1,p_2} \rightarrow \lhd_{p_2,p_3} \rightarrow$. We have $\mathsf{Comp}(\pi) = \{(p_1, p_3)\}$. Moreover, $\min[\![\pi]\!](e_2) = g_4$ and $\max[\![\pi]\!](e_2) = g_5$.

We say that $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ is *monotone* if, for all MSCs M and events e, f such that $M, e \models \langle\pi\rangle$, $M, f \models \langle\pi\rangle$, and $e \leq_{\mathsf{proc}} f$, we have $\min[\![\pi]\!](e) \leq_{\mathsf{proc}} \min[\![\pi]\!](f)$ and $\max[\![\pi]\!](e) \leq_{\mathsf{proc}} \max[\![\pi]\!](f)$. Lemmas 11 and 12 are easily shown by simultaneous induction.

▶ **Lemma 11.** *Let* $\pi_1, \pi_2 \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *be path formulas, and* $\pi = \pi_1 \cdot \pi_2$*. For all MSCs* M *and events* e *such that* $M, e \models \langle\pi\rangle$*, we have*

$$\min[\![\pi]\!](e) = \min[\![\pi_2]\!](\min[\![\pi_1 \cdot \{\langle\pi_2\rangle\}?]\!](e)) \text{ and}$$
$$\max[\![\pi]\!](e) = \max[\![\pi_2]\!](\max[\![\pi_1 \cdot \{\langle\pi_2\rangle\}?]\!](e)) .$$

▶ **Lemma 12.** *All* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *path formulas are monotone.*

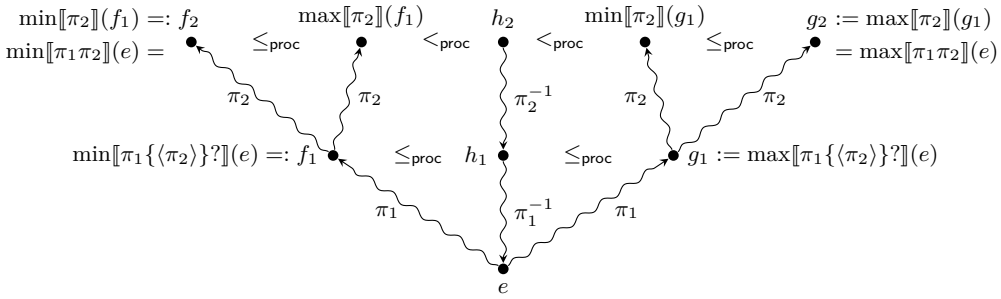
The following crucial lemma states that, for all path formulas $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ and events e in some MSC, $[\![\pi]\!](e)$ contains precisely the events that lie in the interval between $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$ and that satisfy $\langle \pi^{-1} \rangle$.

▶ **Lemma 13.** *Let* π *be a* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *path formula. For all MSCs* M *and events* e *such that* $M, e \models \langle \pi \rangle$*, we have*

$$[\![\pi]\!](e) = \{f \in E \mid \min[\![\pi]\!](e) \leq_{\mathsf{proc}} f \leq_{\mathsf{proc}} \max[\![\pi]\!](e) \wedge M, f \models \langle \pi^{-1} \rangle\}\,.$$

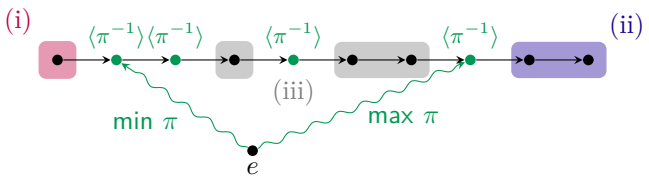
Proof. The left-to-right inclusion is trivial. We prove the right-to-left inclusion by induction on π. The base cases are immediate.

Assume that $\pi = \pi_1 \cdot \pi_2$. For illustration, consider the figure below.



We let $f_1 = \min[\![\pi_1\{\langle\pi_2\rangle\}?]\!](e)$, $f_2 = \min[\![\pi_2]\!](f_1)$, $g_1 = \max[\![\pi_1\{\langle\pi_2\rangle\}?]\!](e)$, and $g_2 = \max[\![\pi_2]\!](g_1)$. By Lemma 11, we have $f_2 = \min[\![\pi_1\pi_2]\!](e)$ and $g_2 = \max[\![\pi_1\pi_2]\!](e)$. Let $h_2 \in E$ such that $f_2 \leq_{\mathsf{proc}} h_2 \leq_{\mathsf{proc}} g_2$ and $M, h_2 \models \langle(\pi_1\pi_2)^{-1}\rangle$. If $h_2 \leq_{\mathsf{proc}} \max[\![\pi_2]\!](f_1)$, then by induction hypothesis, $M, f_1, h_2 \models \pi_2$, and we obtain $M, e, h_2 \models \pi_1\pi_2$. Similarly, if $\min[\![\pi_2]\!](g_1) \leq_{\mathsf{proc}} h_2$, then $M, g_1, h_2 \models \pi_2$ and $M, e, h_2 \models \pi_1\pi_2$. So assume $\max[\![\pi_2]\!](f_1) <_{\mathsf{proc}} h_2 <_{\mathsf{proc}} \min[\![\pi_2]\!](g_1)$. Since $M, h_2 \models \langle\pi_2^{-1}\pi_1^{-1}\rangle$, there exists h_1 such that $M, h_1, h_2 \models \pi_2$ and $M, h_1 \models \langle\pi_1^{-1}\rangle$. Moreover, $\min[\![\pi_2]\!](h_1) \leq_{\mathsf{proc}} h_2 <_{\mathsf{proc}} \min[\![\pi_2]\!](g_1)$, hence $h_1 \leq_{\mathsf{proc}} g_1$ by Lemma 12 (notice that g_1 and h_1 must be on the same process). Similarly, $\max[\![\pi_2]\!](f_1) <_{\mathsf{proc}} h_2 \leq_{\mathsf{proc}} \max[\![\pi_2]\!](h_1)$, hence $f_1 \leq_{\mathsf{proc}} h_1$. We then have $f_1 \leq_{\mathsf{proc}} h_1 \leq_{\mathsf{proc}} g_1$, and $M, h_1 \models \langle\pi_1^{-1}\rangle$. By induction hypothesis, $M, e, h_1 \models \pi_1$. Hence, $M, e, h_2 \models \pi_1\pi_2$. ◀

Using Lemma 13, we can give a characterization of $[\![\pi^{\mathsf{c}}]\!](e)$ (when $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$) that also relies on intervals delimited by $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$. More precisely, $[\![\pi^{\mathsf{c}}]\!](e)$ is the union of the following sets (see figure below): (i) the interval of all events to the left of $\min[\![\pi]\!](e)$, (ii) the interval of all events to the right of $\max[\![\pi]\!](e)$, (iii) the set of events located between $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$ and satisfying $\neg\langle\pi^{-1}\rangle$, (iv) all events located on other processes than $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$.



This description of $[\![\pi^{\mathsf{c}}]\!](e)$ can be used to rewrite π^{c} as a union of $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ formulas. In a first step, we show that, if π is a $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ formula, then the relation $\{(e, \min[\![\pi]\!](e))\}$ can also be expressed in $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ (and similarly for max).

▶ **Lemma 14.** *Let $R = \emptyset$ or $R = \{\mathsf{Loop}\}$. For every path formula $\pi \in \mathrm{PDL}_{\mathsf{sf}}[R]$, there exist $\mathrm{PDL}_{\mathsf{sf}}[R]$ path formulas $\mathsf{min}\ \pi$ and $\mathsf{max}\ \pi$ such that $M, e, f \models \mathsf{min}\ \pi$ iff $f = \min[\![\pi]\!](e)$, and $M, e, f \models \mathsf{max}\ \pi$ iff $f = \max[\![\pi]\!](e)$.*

Proof. We construct, by induction on π, formulas $\mathsf{min}\ (\pi \cdot \{\psi\}?)$ for all $\mathrm{PDL}_{\mathsf{sf}}[R]$ event formulas ψ. For $\pi \in \{\rightarrow, \leftarrow, \lhd_{p,q}, \lhd_{p,q}^{-1}, \{\varphi\}?\}$, we let $\mathsf{min}\ (\pi \cdot \{\psi\}?) = \pi \cdot \{\psi\}?$. Then,

$$\begin{aligned}
\mathsf{min}\ (\xrightarrow{\varphi} \cdot \{\psi\}?) &= \xrightarrow{\varphi \wedge \neg\psi} \cdot \{\psi\}? \\
\mathsf{min}\ (\xleftarrow{\varphi} \cdot \{\psi\}?) &= \xleftarrow{\varphi} \cdot \{\psi \wedge (\neg\varphi \vee \neg \langle \xleftarrow{\varphi} \rangle \psi)\}? \\
\mathsf{min}\ (\mathsf{jump}_{p,q} \cdot \{\psi\}?) &= \mathsf{jump}_{p,q} \cdot \{\psi \wedge \neg \langle \xleftarrow{+} \rangle \psi\}? \\
\mathsf{min}\ (\pi_1 \cdot \pi_2 \cdot \{\psi\}?) &= \mathsf{min}\ (\pi_1 \cdot \{\langle \pi_2 \rangle \psi\}?) \cdot \mathsf{min}\ (\pi_2 \cdot \{\psi\}?)\,.
\end{aligned}$$

The construction of $\mathsf{max}\ \pi$ is similar. ◀

We are now ready to prove that any boolean combination of $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ formulas is equivalent to a positive one, i.e., one that does not use complement.

▶ **Lemma 15.** *For all path formulas $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$, there exist $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ path formulas $(\pi_i)_{1 \leq i \leq |P|^2+3}$ such that $\pi^{\mathsf{c}} \equiv \bigcup_{1 \leq i \leq |P|^2+3} \pi_i$.*

Proof. We show $\pi^{\mathsf{c}} \equiv \sigma$, where

$$\sigma = (\mathsf{min}\ \pi \cdot \xleftarrow{+}) \cup (\mathsf{max}\ \pi \cdot \xrightarrow{+}) \cup (\pi \cdot \xrightarrow{+} \cdot \{\neg \langle \pi^{-1} \rangle\}?) \cup \bigcup_{(p,q) \in P^2} \{\neg \langle \pi \rangle\, q\}? \cdot \mathsf{jump}_{p,q}\,.$$

Let $M = (E, \rightarrow, \lhd, loc, \lambda)$ be an MSC and $e, f \in E$. We write $p = loc(e)$, $q = loc(f)$. Let us show that $M, e, f \models \pi^c$ iff $M, e, f \models \sigma$. If $M, e \models \neg \langle \pi \rangle\, q$, then both $M, e, f \models \pi^{\mathsf{c}}$ and $M, e, f \models \sigma$ hold. In the following, we assume that $M, e \models \langle \pi \rangle\, q$, and thus that $\min[\![\pi]\!](e)$ and $\max[\![\pi]\!](e)$ are well-defined and on process q. Again, if $f <_{\mathsf{proc}} \min[\![\pi]\!](e)$ or $\max[\![\pi]\!](e) <_{\mathsf{proc}} f$, then both $M, e, f \models \pi^{\mathsf{c}}$ and $M, e, f \models \sigma$ hold. And if $\min[\![\pi]\!](e) \leq_{\mathsf{proc}} f \leq_{\mathsf{proc}} \max[\![\pi]\!](e)$, then, by Lemma 13, we have $M, e, f \models \pi^c$ iff $M, f \models \neg \langle \pi^{-1} \rangle$, iff $M, e, f \models \sigma$. ◀

The rest of this section is dedicated to the proof of Theorem 7, stating that every $\mathrm{FO}[\rightarrow, \lhd, \leq]$ formula with at most two free variables can be translated into an equivalent $\mathrm{PDL}_{\mathsf{sf}}$ formula. As we proceed by induction, we actually need a more general statement, which takes into account arbitrarily many free variables:

▶ **Proposition 16.** *Every formula $\Phi \in \mathrm{FO}[\rightarrow, \lhd, \leq]$ with at least one free variable is equivalent to a boolean combination of formulas of the form $\widetilde{\pi}(x, y)$, where $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ and $x, y \in \mathsf{Free}(\Phi)$.*

Proof. In the following, we will simply write $\pi(x, y)$ for $\widetilde{\pi}(x, y)$, where $\widetilde{\pi}(x, y)$ is the FO formula equivalent to π as defined in Proposition 6. The proof is by induction. For convenience, we assume that Φ is in prenex normal form. If Φ is quantifier free, then it is a boolean combination of atomic formulas. For $x, y \in \mathcal{V}_{\mathsf{event}}$, atomic formulas are translated as follows:

$$\begin{array}{lllllllll}
p(x) & \equiv & \{p\}?(x,x) & \qquad x \rightarrow y & \equiv & \rightarrow(x,y) & \qquad x = y & \equiv & \{true\}?(x,y) \\
a(x) & \equiv & \{a\}?(x,x) & \qquad x \lhd y & \equiv & \displaystyle\bigvee_{(p,q) \in Ch} \lhd_{p,q}(x,y) & & &
\end{array}$$

Moreover, $x \leq y$ is equivalent to the disjunction of the formulas $\big(\pi \cdot \lhd_{p_1,p_2} \cdot \xrightarrow{+} \cdot \lhd_{p_2,p_3} \cdots \xrightarrow{+} \cdot \lhd_{p_{m-1},p_m} \cdot \pi'\big)(x, y)$, where $1 \leq m \leq |P|$, $p_1, \ldots, p_m \in P$ are such that $p_i \neq p_{i+1}$ for all $i \in \{1, \ldots, m-1\}$, and $\pi, \pi' \in \{\xrightarrow{+}, \{true\}?\}$.

Universal quantification. We have $\forall x.\Psi \equiv \neg\exists x.\neg\Psi$. Since we allow boolean combinations, dealing with negation is trivial. Hence, this case reduces to existential quantification.

Existential quantification. Suppose that $\Phi = \exists x.\Psi$. If x is not free in Ψ, then $\Phi \equiv \Psi$ and we are done by induction. Otherwise, assume that $\mathsf{Free}(\Psi) = \{x_1, \dots, x_n\}$ with $n > 1$ and that $x = x_n$. By induction, Ψ is equivalent to a boolean combination of formulas of the form $\pi(y, z)$ with $y, z \in \mathsf{Free}(\Psi)$. We transform it into a finite disjunction of formulas of the form $\bigwedge_j \pi_j(y_j, z_j)$, where $y_j = x_{i_1}$ and $z_j = x_{i_2}$ for some $i_1 \le i_2$. To do so, we first eliminate negation using Lemma 15. The resulting positive boolean combination is then brought into disjunctive normal form. Note that this latter step may cause an exponential blow-up so that the overall construction is nonelementary (which is unavoidable [28]). Finally, the variable ordering can be guaranteed by replacing π_j with π_j^{-1} whenever needed.

Now, $\Phi = \exists x_n.\Psi$ is equivalent to a finite disjunction of formulas of the form

$$\bigwedge_{j\in I} \pi_j(y_j, z_j) \;\wedge\; \underbrace{\exists x_n.\Big(\bigwedge_{j\in J} \pi_j(y_j, x_n) \wedge \bigwedge_{j\in J'} \pi_j(x_n, x_n)\Big)}_{=:\,\Upsilon}$$

for three finite, pairwise disjoint index sets I, J, J' such that $y_j \in \{x_1, \dots, x_{n-1}\}$ for all $j \in I \cup J$, and $z_j \in \{x_1, \dots, x_{n-1}\}$ for all $j \in I$. Notice that $\mathsf{Free}(\Upsilon) \subseteq \{x_1, \dots, x_{n-1}\}$. If $J = \emptyset$, then[2]

$$\Upsilon \equiv \bigvee_{p,q\in P} \Big(\mathsf{jump}_{p,q} \cdot \{\bigwedge_{j\in J'} \mathsf{Loop}(\pi_j)\}? \cdot \mathsf{jump}_{q,p}\Big)(x_1, x_1)\,.$$

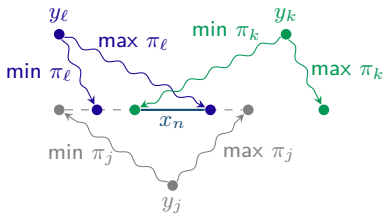
So assume $J \neq \emptyset$. Set

$$\Upsilon' := \bigvee_{k,\ell\in J} \begin{pmatrix} & \bigwedge_{j\in J}((\min\,\pi_j)\cdot \xrightarrow{*} \cdot (\min\,\pi_k)^{-1})(y_j, y_k) \\ \wedge & \bigwedge_{j\in J}((\max\,\pi_\ell)\cdot \xrightarrow{*} \cdot (\max\,\pi_j)^{-1})(y_\ell, y_j) \\ \wedge & (\pi_k \cdot \{\psi\}? \cdot \pi_\ell^{-1})(y_k, y_\ell) \end{pmatrix}$$

where $\psi = \bigwedge_{j\in J} \langle \pi_j^{-1} \rangle \wedge \bigwedge_{j\in J'} \mathsf{Loop}(\pi_j)$. We have $\mathsf{Free}(\Upsilon') = \mathsf{Free}(\Upsilon) \subseteq \{x_1, \dots, x_{n-1}\}$.

▶ **Claim 17.** *We have* $\Upsilon \equiv \Upsilon'$.

Intuitively, by Lemma 13, we know that Υ holds iff the intersection of the intervals $[\min[\![\pi_j]\!](y_j), \max[\![\pi_j]\!](y_j)]$ contains some event satisfying ψ. The formula Υ' identifies some π_k such that $\min[\![\pi_k]\!](y_k)$ is maximal (first line), some π_ℓ such that $\max[\![\pi_\ell]\!](y_\ell)$ is minimal (second line), and tests that there exists an event x_n satisfying ψ between the two (third line). This is illustrated in the figure below.



[2] In this case, Υ is a sentence whereas x_1 is free in the right hand side. Notice that $\equiv$ does not require the two formulas to have the same free variables.

Thus, Υ is equivalent to some positive combination of formulas $\pi(x,y)$ with $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ and $x,y \in \{x_1,\dots,x_{n-1}\} = \mathsf{Free}(\Phi)$, therefore, so is Φ. Note that the two formulas $\big((\mathsf{min}\ \pi_j)\cdot \xrightarrow{*} \cdot (\mathsf{min}\ \pi_k)^{-1}\big)(y_j,y_k)$ and $\big((\mathsf{max}\ \pi_\ell)\cdot \xrightarrow{*} \cdot (\mathsf{max}\ \pi_j)^{-1}\big)(y_\ell,y_j)$ are not $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ formulas (since $\xrightarrow{*}$ is not). However, they are disjunctions of $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ formulas, for instance, $\big((\mathsf{min}\ \pi_j)\cdot \xrightarrow{*} \cdot (\mathsf{min}\ \pi_k)^{-1}\big)(y_j,y_k) \equiv \big((\mathsf{min}\ \pi_j)\cdot (\mathsf{min}\ \pi_k)^{-1}\big)(y_j,y_k) \vee \big((\mathsf{min}\ \pi_j)\cdot \xrightarrow{+} \cdot (\mathsf{min}\ \pi_k)^{-1}\big)(y_j,y_k)$. ◀

We are now able to prove the main result relating $\mathrm{FO}[\rightarrow,\lhd,\leq]$ and $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$.

Proof of Theorem 7. Let $\Phi_2(x_1,x_2)$ be an $\mathrm{FO}[\rightarrow,\lhd,\leq]$ formula with two free variables. We apply Proposition 16 to $\Phi_2(x_1,x_2)$ and obtain a boolean combination of path formulas $\pi(y,z)$ with $y,z \in \{x_1,x_2\}$. First, we bring it into a positive boolean combination using Lemma 15. Next, we replace formulas $\pi(x_1,x_1)$ with $\bigvee_{p,q}(\{\mathsf{Loop}(\pi)\}?\cdot \mathsf{jump}_{p,q})(x_1,x_2)$. Similarly, $\pi(x_2,x_2)$ is replaced with $\bigvee_{p,q}(\mathsf{jump}_{p,q}\cdot\{\mathsf{Loop}(\pi)\}?)(x_1,x_2)$. Also, $\pi(x_2,x_1)$ is replaced with $\pi^{-1}(x_1,x_2)$. Finally, we transform it into disjunctive normal form: we obtain $\Phi_1(x_1,x_2) \equiv \bigvee_i \bigwedge_j \pi_{ij}(x_1,x_2)$, which concludes the proof in the case of two free variables.

Next, let $\Phi_1(x)$ be an $\mathrm{FO}[\rightarrow,\lhd,\leq]$ formula with one free variable. As above, applying Proposition 16 to $\Phi_1(x)$ and then Lemma 15, we obtain $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ path formulas π_{ij} such that $\Phi_1(x) \equiv \bigvee_i \bigwedge_j \pi_{ij}(x,x)$. Now, $M,[x\mapsto e] \models \pi_{ij}(x,x)$ iff $M,e \models \mathsf{Loop}(\pi_{ij})$. Hence, $\Phi(x) \equiv \bigvee_i \bigwedge_j \mathsf{Loop}(\pi_{ij})$.

Finally, an $\mathrm{FO}[\rightarrow,\lhd,\leq]$ sentence Φ_0 is a boolean combination of formulas of the form $\exists x.\Phi_1(x)$. Applying the theorem to $\Phi_1(x)$, we obtain an equivalent $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ event formula φ. Then, we take $\xi = \mathsf{E}\,\varphi$, which is trivially equivalent to $\exists x.\Phi_1(x)$. ◀

4 From $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ to CFMs

Letter-to-letter MSC transducers. For the translation of $\mathrm{FO}[\rightarrow,\lhd,\leq]$ sentences into CFMs, we will need to introduce MSC transducers to handle subformulas with one free variable, or, equivalently, $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ event formulas. More precisely, we will associate with an event formula φ a transducer that evaluates φ at all events, and outputs 1 when the formula holds, and 0 otherwise.

Let Γ be a nonempty finite output alphabet. A *(nondeterministic) letter-to-letter MSC transducer* (or simply, *transducer*) $\mathcal{A}$ over P and from Σ to Γ is a CFM over P and $\Sigma\times\Gamma$. The transducer $\mathcal{A}$ accepts the relation $[\![\mathcal{A}]\!] = \{\big((E,\rightarrow,\lhd,loc,\lambda),(E,\rightarrow,\lhd,loc,\gamma)\big) \mid (E,\rightarrow,\lhd,loc,\lambda\times\gamma)\in L(\mathcal{A})\}$. Transducers are closed under product and composition, using standard constructions:

▶ **Lemma 18.** *Let $\mathcal{A}$ be a transducer from Σ to Γ, and $\mathcal{A}'$ a transducer from Σ to Γ'. There exists a transducer $\mathcal{A}\times\mathcal{A}'$ from Σ to $\Gamma\times\Gamma'$ such that*

$$[\![\mathcal{A}\times\mathcal{A}']\!] = \big\{\big((E,\rightarrow,\lhd,loc,\lambda),(E,\rightarrow,\lhd,loc,\gamma\times\gamma')\big) \mid \big((E,\rightarrow,\lhd,loc,\lambda),(E,\rightarrow,\lhd,loc,\gamma)\big)\in[\![\mathcal{A}]\!],\ \big((E,\rightarrow,\lhd,loc,\lambda),(E,\rightarrow,\lhd,loc,\gamma')\big)\in[\![\mathcal{A}']\!]\big\}.$$

▶ **Lemma 19.** *Let $\mathcal{A}$ be a transducer from Σ to Γ, and $\mathcal{A}'$ a transducer from Γ to Γ'. There exists a transducer $\mathcal{A}'\circ\mathcal{A}$ from Σ to Γ' such that*

$$[\![\mathcal{A}'\circ\mathcal{A}]\!] = [\![\mathcal{A}']\!]\circ[\![\mathcal{A}]\!] = \{(M,M'') \mid \exists M'\in \mathbb{MSC}(P,\Gamma) : (M,M')\in[\![\mathcal{A}]\!],(M',M'')\in[\![\mathcal{A}']\!]\}.$$

Translation of $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ Event Formulas into CFMs. For a $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ event formula φ and an MSC $M = (E, \to, \lhd, loc, \lambda)$ over P and Σ, we define an MSC $M_\varphi = (E, \to, \lhd, loc, \gamma)$ over P and $\{0,1\}$, by setting $\gamma(e) = 1$ if $M, e \models \varphi$, and $\gamma(e) = 0$ otherwise. Our goal is to construct a transducer $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!] = \{(M, M_\varphi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$.

We start with the case of formulas from $\mathrm{PDL}_{\mathsf{sf}}[\emptyset]$, i.e., without Loop. A straightforward induction shows:

▶ **Lemma 20.** *Let φ be a $\mathrm{PDL}_{\mathsf{sf}}[\emptyset]$ event formula. There exists a transducer $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!] = \{(M, M_\varphi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$.*

Next, we look at a single loop where the path $\pi \in \mathrm{PDL}_{\mathsf{sf}}[\emptyset]$ is of the form $\mathsf{min}\ \pi'$ or $\mathsf{max}\ \pi'$. This case will be simpler than general loop formulas, because of the fact that $[\![\mathsf{min}\ \pi']\!](e)$ is always either empty or a singleton. Recall that, in addition, $\mathsf{min}\ \pi'$ is monotone.

▶ **Lemma 21.** *Let π be a $\mathrm{PDL}_{\mathsf{sf}}[\emptyset]$ path formula of the form $\pi = \mathsf{min}\ \pi'$ or $\pi = \mathsf{max}\ \pi'$, and let $\varphi = \mathsf{Loop}(\pi)$. There exists a transducer $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!] = \{(M, M_\varphi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$.*

Proof. We can assume that $\mathsf{Comp}(\pi) \subseteq \mathsf{id}$. We define $\mathcal{A}_\varphi$ as the composition of three transducers that will guess and check the evaluation of φ. More precisely, $\mathcal{A}_\varphi$ will be obtain as an inverse projection α^{-1}, followed by the intersection with an MSC language K, followed by a projection β.

We first enrich the labeling of the MSC with a color from $\Theta = \{\square, \blacksquare, \bigcirc, \bullet\}$. Intuitively, colors $\square$ and $\blacksquare$ will correspond to a guess that the formula φ is satisfied, and colors $\bigcirc$ and $\bullet$ to a guess that the formula is not satisfied. Consider the projection $\alpha\colon \mathbb{MSC}(P, \Sigma \times \Theta) \to \mathbb{MSC}(P, \Sigma)$ which erases the color from the labeling. The inverse projection α^{-1} can be realized with a transducer $\mathcal{A}$, i.e., $[\![\mathcal{A}]\!] = \{(\alpha(M'), M') \mid M' \in \mathbb{MSC}(P, \Sigma \times \Theta)\}$.

Define the projection $\beta\colon \mathbb{MSC}(P, \Sigma \times \Theta) \to \mathbb{MSC}(P, \{0,1\})$ by $\beta((E, \to, \lhd, loc, \lambda \times \theta)) = (E, \to, \lhd, loc, \gamma)$, where $\gamma(e) = 1$ if $\theta(e) \in \{\square, \blacksquare\}$, and $\gamma(e) = 0$ otherwise. The projection β can be realized with a transducer $\mathcal{A}''$: we have $[\![\mathcal{A}'']\!] = \{(M', \beta(M')) \mid M' \in \mathbb{MSC}(P, \Sigma \times \Theta)\}$.

Finally, consider the language $K \subseteq \mathbb{MSC}(P, \Sigma \times \Theta)$ of MSCs $M' = (E, \to, \lhd, loc, \lambda \times \theta)$ satisfying the following two conditions:

1. Colors $\square$ and $\blacksquare$ alternate on each process $p \in P$: if $e_1 < \cdots < e_n$ are the events in $E_p \cap \theta^{-1}(\{\square, \blacksquare\})$, then $\theta(e_i) = \square$ if i is odd, and $\theta(e_i) = \blacksquare$ if i is even.
2. For all $e \in E$, $\theta(e) \in \{\square, \blacksquare\}$ iff there exists $f \in E$ such that $M, e, f \models \pi$ and $\theta(e) = \theta(f)$.

The first property is trivial to check with a CFM. Using Lemma 20, we can easily show that the second property can also be checked with a CFM. We deduce that there is a transducer $\mathcal{A}'$ such that $[\![\mathcal{A}']\!] = \{(M', M') \mid M' \in K\}$. We let $\mathcal{A}_\varphi = \mathcal{A}'' \circ \mathcal{A}' \circ \mathcal{A}$. Notice that $[\![\mathcal{A}_\varphi]\!] = \{(\alpha(M'), \beta(M')) \mid M' \in K\}$. From the following two claims, we deduce immediately that $[\![\mathcal{A}_\varphi]\!] = \{(M, M_\varphi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$.

▶ **Claim 22.** *For all $M \in \mathbb{MSC}(P, \Sigma)$, there exists $M' \in K$ with $\alpha(M') = M$.*

Let $M = (E, \to, \lhd, loc, \lambda) \in \mathbb{MSC}(P, \Sigma)$. Let $E_1 = \{e \in E \mid M, e \models \varphi\}$ and $E_0 = E \setminus E_1$. Consider the graph $G = (E, \{(e, f) \mid M, e, f \models \pi\})$. Since $\pi = \mathsf{min}\ \pi'$ or $\pi = \mathsf{max}\ \pi'$, every vertex has outdegree at most 1, and, by Lemma 12, there are no cycles except for self-loops. So the restriction of G to E_0 is a forest, and there exists a 2-coloring $\chi\colon E_0 \to \{\bigcirc, \bullet\}$ such that, for all $e, f \in E_0$ with $M, e, f \models \pi$, we have $\chi(e) \neq \chi(f)$. There exists $\theta\colon E \to \Theta$ such that $\theta(e) = \chi(e)$ for $e \in E_0$, and $\theta(e) \in \{\square, \blacksquare\}$ for $e \in E_1$ is such that Condition 1 of the definition of K is satisfied. It is easy to see that Condition 2 is also satisfied. Indeed, if $\theta(e) \in \{\square, \blacksquare\}$, then $e \in E_1$ and $M, e, e \models \pi$. Now, if $\theta(e) \notin \{\square, \blacksquare\}$, then $e \in E_0$ and either $M, e \not\models \langle \pi \rangle$ or, by definition of χ, we have $\theta(e) \neq \theta(f)$ for the unique f such that $M, e, f \models \pi$.

▶ **Claim 23.** *For all $M' \in K$, we have $\beta(M') = M_\varphi$, where $M = \alpha(M')$.*

Let $M' = (E, \to, \lhd, loc, \lambda \times \theta) \in K$ and $M = \alpha(M')$. Suppose towards a contradiction that $M_\varphi \neq \beta(M) = (E, \to, \lhd, loc, \gamma)$. By Condition 2, for all $e \in E$ such that $\gamma(e) = 0$, we have $M, e \not\models \varphi$. So there exists $f_0 \in E$ such that $\gamma(f_0) = 1$ and $M, f_0 \not\models \varphi$. Notice that $\theta(f_0) \in \{\square, \blacksquare\}$. For all $i \in \mathbb{N}$, let f_{i+1} be the unique event such that $M, f_i, f_{i+1} \models \pi$. Such an event exists by Condition 2, and is unique since $\pi = \mathsf{min}\ \pi'$ or $\pi = \mathsf{max}\ \pi'$. Note that, for all i, $\theta(f_{i+1}) = \theta(f_i) \in \{\square, \blacksquare\}$. Suppose $f_0 <_{\mathsf{proc}} f_1$ (the case $f_1 <_{\mathsf{proc}} f_0$ is similar). By Condition 1, there exists g_0 such that $f_0 <_{\mathsf{proc}} g_0 <_{\mathsf{proc}} f_1$ and $\{\theta(f_0), \theta(g_0)\} = \{\square, \blacksquare\}$. Again, for all $i \in \mathbb{N}$, let g_{i+1} be the unique event such that $M, g_i, g_{i+1} \models \pi$. Note that all $f_0, f_1, \ldots$ have the same color, in $\{\square, \blacksquare\}$, and all $g_0, g_1, \ldots$ carry the complementary color. Thus, $f_i \neq g_j$ for all $i, j \in \mathbb{N}$. But, by Lemma 12, this implies $f_0 <_{\mathsf{proc}} g_0 <_{\mathsf{proc}} f_1 <_{\mathsf{proc}} g_1 <_{\mathsf{proc}} \cdots$, which contradicts the fact that we deal with finite MSCs. ◀

The general case is more complicated. We first show how to rewrite an arbitrary loop formula using loops on paths of the form $\mathsf{max}\ \pi$ or $(\mathsf{max}\ \pi) \cdot \xleftarrow{+}$. Intuitively, this means that loop formulas will only be used to test, given an event e such that $e' = \max[\![\pi]\!](e)$ is well-defined and on the same process as e, whether $e' <_{\mathsf{proc}} e$, $e' = e$, or $e <_{\mathsf{proc}} e'$. Indeed, we have $M, e \models \mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+})$ iff $e <_{\mathsf{proc}} \max[\![\pi]\!](e)$.

▶ **Lemma 24.** *For all* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *path formulas π,*

$$\mathsf{Loop}(\pi) \equiv \mathsf{Loop}(\mathsf{max}\ \pi) \vee \left(\langle \pi^{-1} \rangle \wedge \mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+}) \wedge \neg \mathsf{Loop}((\mathsf{min}\ \pi) \cdot \xleftarrow{+})\right).$$

Proof. The result follows from Lemma 13. Indeed, if we have $M, e \models \mathsf{Loop}(\pi)$ and $M, e \not\models \mathsf{Loop}(\mathsf{max}\ \pi)$, then $\min[\![\pi]\!](e) \leq_{\mathsf{proc}} e <_{\mathsf{proc}} \max[\![\pi]\!](e)$ and $M, e \models \langle \pi^{-1} \rangle$, hence $M, e \models \langle \pi^{-1} \rangle \wedge \mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+}) \wedge \neg \mathsf{Loop}((\mathsf{min}\ \pi) \cdot \xleftarrow{+})$. Conversely, if $M, e \models \mathsf{Loop}(\mathsf{max}\ \pi)$, then $M, e \models \mathsf{Loop}(\pi)$, and if $M, e \models (\langle \pi^{-1} \rangle \wedge \mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+}) \wedge \neg \mathsf{Loop}((\mathsf{min}\ \pi) \cdot \xleftarrow{+}))$, then $M, e \models \langle \pi^{-1} \rangle$ and $\min[\![\pi]\!](e) \leq_{\mathsf{proc}} e <_{\mathsf{proc}} \max[\![\pi]\!](e)$, hence $M, e, e \models \pi$, i.e., $M, e \models \mathsf{Loop}(\pi)$. ◀

Notice that, since $\mathsf{min}\ \pi \equiv \mathsf{max}\ (\mathsf{min}\ \pi)$, the formula $\mathsf{Loop}((\mathsf{min}\ \pi) \cdot \xleftarrow{+})$ can also be seen as a special case of a $\mathsf{Loop}((\mathsf{max}\ \pi') \cdot \xleftarrow{+})$ formula.

▶ **Theorem 25.** *For all* $\mathrm{PDL}_{\mathsf{sf}}[\mathsf{Loop}]$ *event formulas φ, there exists a transducer $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!] = \{(M, M_\varphi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$.*

Proof. By Lemma 24, we can assume that all loop subformulas in φ are of the form $\mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+})$ or $\mathsf{Loop}(\mathsf{max}\ \pi)$ (notice that $\mathsf{min}\ \pi = \mathsf{max}\ \mathsf{min}\ \pi$). We prove Theorem 25 by induction on the number of loop subformulas in φ. The base case is stated in Lemma 20.

Let $\psi = \mathsf{Loop}(\pi')$ be a subformula of φ such that π' contains no loop subformulas and $\mathsf{Comp}(\pi') \subseteq \mathsf{id}$. Let us show that there exists $\mathcal{A}_\psi$ such that $[\![\mathcal{A}_\psi]\!] = \{(M, M_\psi) \mid M \in \mathbb{MSC}(P, \Sigma)\}$. If $\pi' = \mathsf{max}\ \pi$, then we apply Lemma 21. Otherwise, $\pi' = (\mathsf{max}\ \pi) \cdot \xleftarrow{+}$ for some $\mathrm{PDL}_{\mathsf{sf}}[\emptyset]$ path formula π. So we assume from now on that $\psi = \mathsf{Loop}((\mathsf{max}\ \pi) \cdot \xleftarrow{+})$.

We start with some easy remarks. Let $p \in P$ be some process and $e \in E_p$. A necessary condition for $M, e \models \psi$ is that $M, e \models \langle \pi \rangle \wedge \neg \mathsf{Loop}(\mathsf{max}\ \pi)$. Also, it is easy to see that $M, e \models \mathsf{Loop}(\mathsf{min}\ (\xrightarrow{+} \cdot \pi^{-1}))$ is a sufficient condition for $M, e \models \psi$.

We let E_p^π be the set of events $e \in E_p$ satisfying $\langle \pi \rangle\, p$. For all $e \in E_p^\pi$ we let $e' = [\![\mathsf{max}\ \pi]\!](e) \in E_p$. The transducer $\mathcal{A}_\psi$ will establish, for each $e \in E_p^\pi$, whether $e' <_{\mathsf{proc}} e$, $e' = e$, or $e <_{\mathsf{proc}} e'$, and it will output 1 if $e <_{\mathsf{proc}} e'$, and 0 otherwise. The case $e' = e$ means

$M, e \models \mathsf{Loop}(\max\ \pi)$ and can be checked with the help of Lemma 21. So the difficulty is to distinguish between $e' <_{\mathsf{proc}} e$ and $e <_{\mathsf{proc}} e'$ when $M, e \models \langle \pi \rangle \wedge \neg \mathsf{Loop}(\max\ \pi)$.

The following two claims rely on Lemma 12:

▶ **Claim 26.** *Let f be the minimal event in E_p^π (assuming this set is nonempty). Then, $M, f \models \psi$ iff $M, f \models \mathsf{Loop}(\min\ (\xrightarrow{+} \cdot \pi^{-1}))$.*

▶ **Claim 27.** *Let e, f be consecutive events in E_p^π, i.e., $e, f \in E_p^\pi$ and $M, e, f \models \xrightarrow{\neg \langle \pi \rangle}$.*

1. *If $M, e \not\models \psi$, then $[M, f \models \psi$ iff $M, f \models \mathsf{Loop}(\min\ (\xrightarrow{+} \cdot \pi^{-1}))]$.*
2. *If $M, e \models \psi$, then $[M, f \not\models \psi$ iff $M, f \models \mathsf{Loop}(\max\ \pi) \vee \mathsf{Loop}(\max\ ((\max\ \pi) \cdot \xrightarrow{\neg \langle \pi \rangle}))]$.*

To conclude the proof, let $\varphi_1 = \langle \pi \rangle$, $\varphi_2 = \mathsf{Loop}(\max\ \pi)$, $\varphi_3 = \mathsf{Loop}(\min\ (\xrightarrow{+} \cdot \pi^{-1}))$, and $\varphi_4 = \mathsf{Loop}(\max\ ((\max\ \pi) \cdot \xrightarrow{\neg \langle \pi \rangle}))$. By Lemmas 20 and 21, we already have transducers $\mathcal{A}_{\varphi_i}$ for $i \in \{1, 2, 3, 4\}$. We let $\mathcal{A}_\psi = \mathcal{A} \circ (\mathcal{A}_{\varphi_1} \times \mathcal{A}_{\varphi_2} \times \mathcal{A}_{\varphi_3} \times \mathcal{A}_{\varphi_4})$, where, at an event f labeled (b_1, b_2, b_3, b_4), the transducer $\mathcal{A}$ outputs 1 if $b_3 = 1$ or if $(b_1, b_2, b_3, b_4) = (1, 0, 0, 0)$ and the output was 1 at the last event e on the same process satisfying φ_1 (to do so, each process keeps in its state the output at the last event where b_1 was 1), and 0 otherwise.

Consider the formula φ' over $\Sigma \times \{0, 1\}$ obtained from φ by replacing ψ by $\bigvee_{a \in \Sigma} (a, 1)$, and all event formulas a, with $a \in \Sigma$, by $(a, 0) \vee (a, 1)$. It contains fewer Loop operators than φ, so by induction hypothesis, we have a transducer $\mathcal{A}_{\varphi'}$ for φ'. We then let $\mathcal{A}_\varphi = \mathcal{A}_{\varphi'} \circ (\mathcal{A}_{Id} \times \mathcal{A}_\psi)$, where $\mathcal{A}_{Id}$ is the transducer for the identity relation. ◀

Proof of Proposition 4. By Theorem 7, every FO$[\rightarrow, \lhd, \leq]$ formula $\Phi(x)$ with a single free variable is equivalent to some PDL$_{\mathsf{sf}}[\mathsf{Loop}]$ state formula, for which we obtain a transducer $\mathcal{A}_\Phi$ using Theorem 25. It is easy to build from $\mathcal{A}_\Phi$ CFMs for the sentences $\forall x.\Phi(x)$ and $\exists x.\Phi(x)$. Closure of $\mathcal{L}(\text{CFM})$ under union and intersection takes care of disjunction and conjunction. ◀

5 Discussion

Though the translation of EMSO/FO formulas into CFMs is interesting on its own, it allows us to obtain some difficult results for bounded CFMs as corollaries. We will briefly sketch some of them. For details, we refer to [2].

First, note that, for a given channel bound, the set of existentially bounded MSCs is FO-definable (essentially due to [26]). By Theorem 3, we obtain [14, Proposition 5.14] stating that this set is recognized by some CFM. Second, we obtain [14, Proposition 5.3], a Kleene theorem for existentially bounded MSCs, as a corollary of Theorem 3 in combination with a linearization normal form from [30].

Since (bounded) MSCs can be seen as a special case of Mazurkiewicz traces [9], we also get Zielonka's theorem [33] (though a weaker, nondeterministic version, and without guarantee on the size of the constructed automaton).

We leave open whether there is a one-dimensional temporal logic over MSCs, with a finite set of FO-definable modalities, that is expressively complete for FO$[\rightarrow, \lhd, \leq]$.

References

1 B. Bollig, M. Fortin, and P. Gastin. Communicating finite-state machines and two-variable logic. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics*, pages 17:1–17:14. Leibniz-Zentrum für Informatik, 2018.

2 B. Bollig, M. Fortin, and P. Gastin. It is easy to be wise after the event: Communicating finite-state machines capture first-order logic with "happened before". *CoRR*, abs/1804.10076, 2018. `arXiv:1804.10076`.

3 B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), 2010.

4 B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2-3):150–172, 2006.

5 D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.

6 J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

7 G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 205–212. AAAI Press / The MIT Press, 1994.

8 V. Diekert and P. Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

9 V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

10 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.

11 M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

12 D. M. Gabbay. Expressive functional completeness in tense logic. In Uwe Mönnich, editor, *Aspects of Philosophical Logic: Some Logical Forays into Central Notions of Linguistics and Philosophy*, pages 91–117. Springer Netherlands, Dordrecht, 1981.

13 D. M. Gabbay, I. Hodkinson, and M. A. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects, vol. 1*. Oxford University Press, 1994.

14 B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.

15 B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.

16 S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *Journal of Symbolic Logic*, 74(1):279–314, 2009.

17 E. Grädel and M. Otto. On logics with two variables. *Theoretical Computer Science*, 224(1-2):73–113, 1999.

18 J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artif. Intell.*, 54(2):319–379, 1992.

19 W. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*. North-Holland, Amsterdam, 1965.

20 J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.

21 H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

22 D. Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187:80–109, 2003.

23 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

24 M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2006.

25 M. Lange and C. Lutz. 2-ExpTime lower bounds for Propositional Dynamic Logics with intersection. *Journal of Symbolic Logic*, 70(5):1072–1086, 2005.

26 M. Lohrey and A. Muscholl. Bounded MSC Communication. *Information and Computation*, 189(2):160–181, 2004.

27 R. Mennicke. Propositional dynamic logic with converse and repeat for message-passing systems. *Logical Methods in Computer Science*, 9(2:12):1–35, 2013.

28 L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, MIT, 1974.

29 R. S. Streett. Propositional dynamic logic of looping and converse. In *Proceedings of STOC'81*, pages 375–383. ACM, 1981.

30 P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. *Inf. Comput.*, 179(2):230–249, 2002.

31 W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

32 B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:103–131, 1962. In Russian; English translation in *Amer. Math. Soc. Transl.* 59, 1966, 23–55.

33 W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.

Learning-Based Mean-Payoff Optimization in an Unknown MDP under Omega-Regular Constraints

Jan Křetínský
Technische Universität München, Munich, Germany
jan.kretinsky@in.tum.de
https://orcid.org/0000-0002-8122-2881

Guillermo A. Pérez[1]
Université libre de Bruxelles, Brussels, Belgium
gperezme@ulb.ac.be
https://orcid.org/0000-0002-1200-4952

Jean-François Raskin[2]
Université libre de Bruxelles, Brussels, Belgium
jraskin@ulb.ac.be

Abstract

We formalize the problem of maximizing the mean-payoff value with high probability while satisfying a parity objective in a Markov decision process (MDP) with unknown probabilistic transition function and unknown reward function. Assuming the support of the unknown transition function and a lower bound on the minimal transition probability are known in advance, we show that in MDPs consisting of a single end component, two combinations of guarantees on the parity and mean-payoff objectives can be achieved depending on how much memory one is willing to use. (i) For all ε and γ we can construct an online-learning finite-memory strategy that almost-surely satisfies the parity objective and which achieves an ε-optimal mean payoff with probability at least $1-\gamma$. (ii) Alternatively, for all ε and γ there exists an online-learning infinite-memory strategy that satisfies the parity objective surely and which achieves an ε-optimal mean payoff with probability at least $1-\gamma$. We extend the above results to MDPs consisting of more than one end component in a natural way. Finally, we show that the aforementioned guarantees are tight, i.e. there are MDPs for which stronger combinations of the guarantees cannot be ensured.

2012 ACM Subject Classification Theory of computation → Logic and verification, Theory of computation → Reinforcement learning

Keywords and phrases Markov decision processes, Reinforcement learning, Beyond worst case

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.8

Related Version A full version is available at `https://arxiv.org/abs/1804.08924`.

Funding This research was funded in part by the Czech Science Foundation grant No. 18-11193S, the German Research Foundation (DFG) project 383882557 "Statistical Unbounded Verification", the ERC Starting grant 279499 "inVEST", the ARC (Fédération Wallonie-Bruxelles) project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond", and the EOS (FNRS-FWO) project 30992574 "Verifying Learning Artificial Intelligence Systems".

[1] G. A. Pérez has been supported by an F.R.S.-FNRS Aspirant fellowship.
[2] J.-F. Raskin is Professeur Francqui de Recherche funded by the Francqui foundation.

1 Introduction

Reactive synthesis and online reinforcement learning. Reactive systems are systems that maintain a continuous interaction with the environment in which they operate. When designing such systems, we usually face two partially conflicting objectives. First, to ensure a safe execution, we want some basic and critical properties to be enforced by the system no matter how the environment behaves. Second, we want the reactive system to be as efficient as possible given the actual observed behaviour of the environment in which the system is executed. As an illustration, let us consider a robot that needs to explore an unknown environment as efficiently as possible while avoiding any collision. While operating at low speed makes it easier to avoid collisions, it will impair its ability to explore the environment quickly even if the environment is clear of other objects.

There has been, in the past, a large research effort to define mathematical models and algorithms in order to address the two objectives above, but in isolation only. To synthesize safe control strategies, two-player zero-sum games with omega-regular objectives have been proposed [29, 4]. Reinforcement-learning (RL, for short) algorithms for partially-specified Markov decision processes (MDPs) have been proposed (see e.g. [32, 22, 26, 28]) to learn strategies that reach (near-)optimal performance in the actual environment in which the system is executed. In this paper, we want to answer the following question: *How efficient can online-learning techniques be if only correct executions, i.e. executions that satisfy a specified omega-regular objective, are explored during execution?* So, we want to understand how to combine synthesis and RL to construct systems that are safe, yet, at the same time, can adapt their behaviour according to the actual environment in which they execute.

Problem statement. In order to answer in a precise way the question above, we consider a model halfway between the fully-unknown models considered in RL and the full-known models used in verification. To be precise, we consider as input an MDP with rewards whose transition probabilities are not known and whose rewards are discovered on the fly. That is, the input is the support of the unknown transition function of the MDP. This is natural from the point of view of verification since: we may be working with an underspecified system, its qualitative behaviour may have already been observed, or we may not trust all given probability values. As optimization objective on this MDP, we consider the mean-payoff function, and to capture the sure omega-regular constraint we use a parity objective.

Contributions. Given a lower bound $\pi_{\min}$ on the minimal transition probability, we show that, in partially-specified MDPs consisting of a single end component (EC), two combinations of guarantees on the parity and mean-payoff objectives can be achieved. (i) For all ε and γ, we show how to construct a finite-memory strategy which almost-surely satisfies the parity objective and which achieves an ε-optimal mean payoff with probability at least $1-\gamma$ (Prop. 20). (ii) For all ε and γ, we show how to construct an infinite-memory strategy which satisfies the parity objective surely and which achieves an ε-optimal mean payoff with probability at least $1-\gamma$ (Prop. 14). We also extend our results to MDPs consisting of more than one EC in a natural way (Thms. 21 and 16) and study special cases that allow for improved optimality results as in the case of good ECs (Props. 11 and 17). Finally, we show that there are partially-specified MDPs for which stronger combinations of the guarantees cannot be ensured.

Our usage of $\pi_{\min}$ follows [9, 18] where it is argued that it is necessary for the statistical analysis of unbounded-horizon properties and realistic in many scenarios.

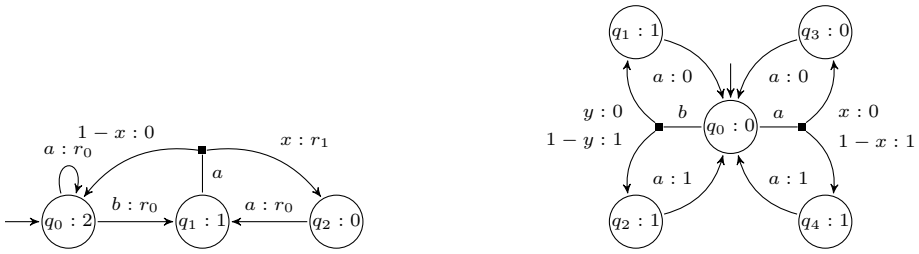


Figure 1 Two automata, representing unknown MDPs, are depicted in the figure. Actions label edges from states (circles) to distributions (squares); a probability-reward pair, edges from distributions to states; an action-reward pair, Dirac transitions; a name-priority pair, states.

Example: almost-sure constraints. Consider the MDP on the right-hand side of Fig. 1 for which we know the support of the transition function but not the probabilities x and y (for simplicity the rewards are assumed to be known). First, note that while there is no surely winning strategy for the parity objective in this MDP, playing action a forever in q_0 guarantees to visit state q_3 infinitely many times with probability one, i.e. this is a strategy that almost-surely wins the parity objective. Clearly, if $x > y$ then it is better to play b for optimizing the mean-payoff, otherwise, it is better to play a. As x and y are unknown, we need to learn estimates $\hat{x}$ and $\hat{y}$ for those values to make a decision. This can be done by playing a and b a number of times from q_0 and by observing how many times we get up and how many times we get down. If $\hat{x} > \hat{y}$, we may choose to play b forever in order to optimize our mean payoff. We then face two difficulties. First, after the learning episode, we may instead observe $\hat{x} < \hat{y}$ while $x \geq y$. This is because we may have been unlucky and observed statistics that differ from the real distribution. Second, playing b always is not an option if we want to satisfy the parity objective with probability 1 (almost surely). In this paper, we give algorithms to overcome these two problems and compute a finite-memory strategy that satisfies the parity objective with probability 1 and is close to the optimal expected mean-payoff value with high probability.

The finite-memory learning strategy produced by our algorithm works as follows in this example. First, it chooses $n \in \mathbb{N}$ large enough so that trying a and b from q_0 as many as n times allows it to learn $\hat{x}$ and $\hat{y}$ such that $|\hat{x} - x| \leq \varepsilon$ and $|\hat{y} - y| \leq \varepsilon$ with probability at least $1 - \gamma$. Then, if $\hat{x} > \hat{y}$ the strategy plays b for K steps and then a once. K is chosen large enough so that the mean payoff of any run will be ε-close to the best obtainable expected mean payoff with probability at least $1 - \gamma$. Furthermore, as a is played infinitely many times, the upper-right state will be visited infinitely many times with probability 1. Hence, the strategy is also almost-surely satisfying the parity objective.

In the sequel we also show that if we allow for learning all along the execution of the strategy then we can get, on this example, the exact optimal value and satisfy the parity objective almost surely. However, to do so, we need infinite memory.

Related works. In [11, 17, 8, 16], we initiated the study of a mathematical model that combines MDPs and two-player zero sum games. With this new model, we provide formal grounds to synthesize strategies that guarantee *both* some minimal performance against any adversary *and* a higher expected performance against a given expected behaviour of the environment, thus essentially combining the two traditional standpoints from games and MDPs. Following this approach, in [1], Almagor et al. study MDPs equipped with a mean-payoff and parity objective. They study the problem of synthesizing a strategy that

ensures an expected mean-payoff value that is as large as possible while satisfying a parity objective surely. In [15], Chatterjee and Doyen study how to enforce almost surely a parity objective together with threshold constraint on the expected mean-payoff. See also [10], where mean-payoff MDPs with energy constraints are studied. In all those works, the transition probability and the reward function are *known* in advance. In contrast, we consider the more complex setting in which the reward function is *discovered on the fly during execution time* and the transition probabilities need to be *learned*.

In [19, 33, 21, 2], RL is combined with safety guarantees. In those works, there is a MDP with a set of unsafe states that must be avoided at all cost. This MDP is then restricted to states and actions that are safe and cannot lead to unsafe states. Thereafter, classical RL is exercised. The problem that is considered there is thus very similar to the problem that we study here with the difference that they only consider *safety constraints*. For safety constraints, the reactive synthesis phase and the RL can be entirely decoupled with a two-phase algorithm. A simple two-phase approach cannot be applied to the more general setting of parity objectives. In our more challenging setting, we need to intertwine the learning with the satisfaction of the parity objective in a non trivial way. It is easy to show that reducing parity to safety, as in [7], could lead to learning strategies that are arbitrary far from the optimal value that our learning strategies achieve. In [34], Topcu and Wen study how to learn in a MDP with a discounted-sum (and not mean-payoff) function and liveness constraints expressed as deterministic Büchi automata that must be enforced *almost surely*. Contrary to our setting, they do not consider general omega-regular specifications expressed as parity objectives nor *sure* satisfaction.

Finally, in [9], we apply RL to MDPs where even the topology is unknown. Only $\pi_{\min}$ and, for convenience, the size of the state space is given. There, we optimize the probability to satisfy an omega-regular property; however, no mean payoff is involved.

Structure of the paper. In Sect. 2, we introduce the necessary preliminaries. In Sect. 3, we study online finite and infinite-memory learning strategies for mean-payoff objectives without omega-regular constraints. In Sect. 4, we study strategies for mean-payoff objectives under a parity constraint that must be enforced surely. In Sect. 5, we study strategies for mean-payoff objectives under a parity constraint that must be enforced almost surely.

2 Preliminaries

Let S be a finite set. We denote by $\mathcal{D}(S)$ the set of all *(rational) probabilistic distributions* on S, i.e. the set of all functions $f : S \to \mathbb{Q}_{\geq 0}$ such that $\sum_{s \in S} f(s) = 1$. For sets A and B and functions $g : A \to \mathcal{D}(S)$ and $h : A \times B \to \mathcal{D}(S)$, we write $g(s|a)$ and $h(s|a,b)$ instead of $g(a)(s)$ and $h(a,b)(s)$ respectively. The *support* of a distribution $f \in \mathcal{D}(S)$ is the set $\operatorname{supp}(f) \stackrel{\text{def}}{=} \{s \in S \mid f(s) > 0\}$. The support of a function $g : A \to \mathcal{D}(S)$ is the relation $R \subseteq A \times S$ such that $(a,s) \in R \stackrel{\text{def}}{\iff} g(s|a) > 0$.

2.1 Markov chains

▶ **Definition 1** (Markov chains). A *Markov chain* $\mathcal{C}$ (MC, for short) is a tuple (Q, δ, p, r) where Q is a (potentially countably infinite) set of states, δ is a (probabilistic) transition function $\delta : Q \to \mathcal{D}(Q)$, $p : Q \to \mathbb{N}$ is a priority function, and $r : \operatorname{supp}(\delta) \to [0,1] \cap \mathbb{Q}$ is an (instantaneous) reward function.

A *run* of an MC is an infinite sequence of states $q_0 q_1 \cdots \in Q^\omega$ such that $\delta(q_{i+1}|q_i) > 0$ for all $0 \leq i$. We denote by $\mathrm{Runs}^{q_0}(\mathcal{C})$ the set of all runs of $\mathcal{C}$ that start with the state q_0.

Consider an initial state q_0. The *probability* of every measurable *event* $\mathcal{A} \subseteq \mathrm{Runs}^{q_0}(\mathcal{C})$ is well-defined [31, 25]. We denote by $\mathbb{P}^{q_0}_{\mathcal{C}}[\mathcal{A}]$ the probability of $\mathcal{A}$; for a measurable function $f : \mathrm{Runs}^{q_0}(\mathcal{C}) \to \mathbb{R}$, we write $\mathbb{E}^{q_0}_{\mathcal{C}}[f]$ for the *expected value* of the function f under the probability measure $\mathbb{P}^{q_0}_{\mathcal{C}}[\cdot]$ (see [23, 25] for a detailed definition of these classical notions).

Parity and mean payoff. Consider a run $\varrho = q_0 q_1 \ldots$ of $\mathcal{C}$. We say ϱ *satisfies the parity objective*, written $\varrho \models \textsc{Parity}$, if the minimal priority of states along the run is even. That is to say $\varrho \models \textsc{Parity} \overset{\text{def}}{\iff} \liminf\{p(q_i) \mid i \in \mathbb{N}\}$ is even. In a slight abuse of notation, we sometimes write Parity to refer to the set of all runs of an MC which satisfy the parity objective $\{\varrho \in \mathrm{Runs}^{q_0}(\mathcal{C}) \mid \varrho \models \textsc{Parity}\}$. The latter set of runs is clearly measurable.

The *mean-payoff function* $\underline{\mathbf{MP}}$ is defined for all runs $\varrho = q_0 q_1 \ldots$ of $\mathcal{C}$ as follows $\underline{\mathbf{MP}}(\varrho) \overset{\text{def}}{=} \liminf_{j \in \mathbb{N}_{>0}} \frac{1}{j} \sum_{i=0}^{j-1} r(q_i, q_{i+1})$. This function is readily seen to be Borel definable [13], thus also measurable.

2.2 Markov decision processes

▶ **Definition 2** (Markov decision processes)**.** A *(finite discrete-time) Markov decision process* $\mathcal{M}$ (MDP, for short) is a tuple $(Q, A, \alpha, \delta, p, r)$ where Q is a finite set of states, A a finite set of actions, $\alpha : Q \to \mathcal{P}(A)$ a function that assigns to q its set of available actions, $\delta : Q \times A \to \mathcal{D}(Q)$ a (partial probabilistic) transition function with $\delta(q, a)$ defined for all $q \in Q$ and all $a \in \alpha(q)$, $p : Q \to \mathbb{N}$ a priority function, and $r : \mathrm{supp}(\delta) \to [0, 1] \cap \mathbb{Q}$ a reward function. We make the assumption that $\alpha(q) \neq \varnothing$ for all $q \in Q$, i.e. there are no deadlocks.

A *history* h in an MDP is a finite state-reward-action sequence that ends in a state and respects α, δ, and r, i.e. if $h = q_0 a_0 x_0 \ldots a_{k-1} x_{k-1} q_k$ then $a_i \in \alpha(q_i)$, $\delta(q_{i+1}|q_i, a_i) > 0$, and $r(q_i, a_i, q_{i+1})$, for all $0 \leq i < k$. We write $\mathrm{last}(h)$ to denote the state q_k. For two histories h, h', we write $h < h'$ if h is a *proper prefix* of h'.

▶ **Definition 3** (Strategies)**.** A *strategy* σ in an MDP $\mathcal{M} = (Q, A, \alpha, \delta, p, r)$ is a function $\sigma : (Q \cdot A \cdot \mathbb{Q})^* Q \to \mathcal{D}(A)$ such that $\sigma(a|h) > 0 \implies a \in \alpha(\mathrm{last}(h))$.

We write that a strategy σ is *memoryless* if $\sigma(h) = \sigma(h')$ whenever $\mathrm{last}(h) = \mathrm{last}(h')$; *deterministic* if for all histories h the distribution $\sigma(h)$ is Dirac.

Throughout this work we will speak of *steps*, *episodes*, and *following strategies*. We write that *σ follows τ (from the history $h = q_0 a_0 x_0 \ldots q_k$) during n steps* if for all $h' = q'_0 a'_0 x'_0 \ldots q'_\ell$, such that $h < h'$ and $\ell \leq k + n$, we have that $\sigma(h') = \tau(h')$. An episode is simply a finite sequence of consecutive steps, i.e. a finite infix of the history, during which one or more strategies may have been sequentially followed.

A *stochastic Mealy machine* $\mathcal{T}$ is a tuple (M, m_0, f_u, f_o) where M is a (potentially countably infinite) set of memory elements, $m_0 \in M$ is the initial memory element, $f_u : M \times Q \times \mathbb{Q} \to M$ is an update function, and $f_o : M \times Q \to \mathcal{D}(A)$ is an output function. The machine $\mathcal{T}$ is said to implement a strategy σ if for all histories $h = q_0 a_0 x_0 \ldots a_{k-1} x_{k-1} q_k$ we have $\sigma(h) = f_o(m_k, q_k)$, where m_k is inductively defined as $m_i = f_u(m_{i-1}, q_{i-1}, x_{i-1})$ for all $i \geq 1$. It is easy to see that any strategy can be implemented by such a machine. A strategy σ is said to have *finite memory* if there exists a stochastic Mealy machine that implements it and such that its set M of memory elements is finite.

A (possibly infinite) state-action sequence $h = q_0 a_0 x_0 q_1 a_1 x_1 \ldots$ is *consistent with strategy* σ if $\sigma(a_i | q_0 a_0 x_0 \ldots a_{i-1} x_{i-1} q_i) > 0$ for all $i \geq 0$.

From MDPs to MCs. The MDP $\mathcal{M}$ and a strategy σ implemented by the stochastic Mealy machine (M, m_0, f_u, f_0) induce the MC $\mathcal{M}^\sigma = (Q', \delta', p', r')$ where $Q' = (Q \times M \times A) \cup (Q \times M)$; $\delta'(\langle q', m', a' \rangle | s) = f_o(a'|m, q) \cdot \delta(q'|q, a')$ for any $s \in \{\langle q, m, a \rangle, \langle q, m \rangle\}$ and $a' \in \alpha(q)$ with $(q, a', q') \in \operatorname{supp}(\delta)$ and $m' = f_u(m, q, r(q, a', q'))$; $p'(\langle q, m, a \rangle) = p'(\langle q, m \rangle) = p(q)$; and $r'(s, \langle q', m', a' \rangle) = r(q, a, q')$ for any $s \in \{\langle q, m, a \rangle, \langle q, m \rangle\}$. For convenience, we write $\mathbb{P}^{q_0}_{\mathcal{M}^\sigma}[\cdot]$ instead of $\mathbb{P}^{\langle q_0, m_0 \rangle}_{\mathcal{M}^\sigma}[\cdot]$.

A strategy σ is said to be *unichain* if $\mathcal{M}^\sigma$ has a single recurrent class, i.e. a single bottom strongly-connected component (BSCC).

End components. Consider a pair (S, β) where $S \subseteq Q$ and $\beta : S \to \mathcal{P}(A)$ gives a subset of actions allowed per state (i.e. $\beta(q) \subseteq \alpha(q)$ for all $q \in S$). Let $\mathcal{G}_{(S,\beta)}$ be the directed graph (S, E) where E is the set of all pairs $(q, q') \in S \times S$ such that $\delta(q'|q, a) > 0$ for some $a \in \beta(q)$. We say (S, β) is an *end component* (EC) if the following hold: if $a \in \beta(s)$, for $(s, a) \in S \times A$, then $\operatorname{supp}(\delta(s, a)) \subseteq S$; and the graph $\mathcal{G}_{(S,\beta)}$ is strongly connected. Furthermore, we say the EC (S, β) is *good (for the parity objective)* (a GEC, for short) if the minimal priority over all states from S is even; *weakly good* if it contains a GEC.

For ECs (S, β) and (S', β'), let us denote by $(S, \beta) \subseteq (S', \beta')$ the fact that $S \subseteq S'$ and $\beta(s) \subseteq \beta'(s)$ for all $s \in S$. We denote by $\mathrm{MEC}_{\mathcal{M}}$ the set of all maximal ECs (MECs) in $\mathcal{M}$ with respect to $\subseteq$. It is easy to see that for all $(S, \cdot), (S', \cdot) \in \mathrm{MEC}_{\mathcal{M}}$ we have that $S \cap S' = \varnothing$, i.e. every state belongs to at most one MEC.

Model learning and robust strategies. In this work we will "approximate" the stochastic dynamics of an unknown EC in an MDP. Below, we formalize what we mean by approximation.

▶ **Definition 4** (Approximating distributions)**.** Let $\mathcal{M} = (Q, A, \alpha, \delta, p, r)$ be an MDP, (S, β) an EC, and $\varepsilon \in (0, 1)$. We say δ' is ε-close to δ in (S, β), denoted $\delta' \sim^{\varepsilon}_{(S,\beta)} \delta$, if $|\delta'(q'|q, a) - \delta(q'|q, a)| \leq \varepsilon$ for all $q, q' \in S$ and all $a \in \beta(q)$. If the inequality holds for all $q, q' \in Q$ and all $a \in \alpha(q)$, then we write $\delta' \sim^{\varepsilon} \delta$.

A strategy σ is said to be *(uniformly) expectation-optimal* if for all $q_0 \in Q$ we have $\mathbb{E}^{q_0}_{\mathcal{M}^\sigma}[\underline{\mathbf{MP}}] = \sup_\tau \mathbb{E}^{q_0}_{\mathcal{M}^\tau}[\underline{\mathbf{MP}}]$. The following result captures the idea that some expectation-optimal strategies for MDPs whose transition function have the same support are "robust". That is, when used to play in another MDP with the same support and close transition functions, they achieve near-optimal expectation.

▶ **Lemma 5** (Follows from [27, Theorem 6] and [14, Theorem 5])**.** *Consider values $\varepsilon, \eta_\varepsilon \in (0, 1)$ such that $\eta_\varepsilon \leq \frac{\varepsilon \cdot \pi_{\min}}{24|Q|}$; a transition function δ' such that $\operatorname{supp}(\delta) = \operatorname{supp}(\delta')$ and $\delta \sim^{\eta_\varepsilon} \delta'$ where $\pi_{\min}$ is the minimal nonzero probability value from δ and δ'; and a reward function r' such that $\max\{|r(q, a, q') - r'(q, a, q')| : (q, a, q') \in \operatorname{supp}(\delta)\} \leq \frac{\varepsilon}{4}$. For all memoryless deterministic expectation-optimal strategies σ in $(Q, A, \alpha, \delta', p, r')$, for all $q_0 \in Q$, it holds that $|\mathbb{E}^{q_0}_{\mathcal{M}^\sigma}[\underline{\mathbf{MP}}] - \sup_\tau \mathbb{E}^{q_0}_{\mathcal{M}^\tau}[\underline{\mathbf{MP}}]| \leq \varepsilon$.*

We say a strategy σ such as the one in the result above is *ε-robust-optimal* (with respect to the expected mean payoff).

2.3 Automata as proto-MDPs

We study MDPs with unknown transition and reward functions. It is therefore convenient to abstract those values and work with *automata*.

▶ **Definition 6** (Automata). A *(finite-state parity) automaton* $\mathcal{A}$ is a tuple (Q, A, T, p) where Q is a finite set of states, A is a finite alphabet of actions, $T \subseteq Q \times A \times Q$ is a transition relation, and $p : Q \to \mathbb{N}$ is a priority function. We make the assumption that for all $q \in Q$ we have $(q, a, q') \in T$ for some $(a, q') \in A \times Q$.

A transition function $\delta : Q \times A \to \mathcal{D}(Q)$ is then said to be *compatible* with $\mathcal{A}$ if $\forall (q, a) \in Q \times A : \mathrm{supp}(\delta(q, a)) = \{q' \mid T(q, a, q')\}$. For a transition function δ compatible with $\mathcal{A}$ and a reward function $r : T \to [0, 1] \cap \mathbb{Q}$, we denote by $\mathcal{A}_{\delta,r}$ the MDP $(Q, A, \alpha_T, \delta, p, r)$ where $a \in \alpha_T(q) \overset{\text{def}}{\iff} \exists (q, a, q') \in T$. It is easy to see that the sets of ECs of MDPs $(Q, A, \alpha_T, \delta, p, r)$ and $(Q, A, \alpha_T, \delta', p, r')$ coincide for all δ' compatible with $\mathcal{A}$ and all reward functions r'. Hence, we will sometimes speak of the ECs of an automaton.

Example: sure-constraints. Consider the (variable-labelled-)automaton on the left-hand side of Fig. 1. Note that playing a forever surely wins the parity objective from everywhere but it may not be optimal for the expected mean payoff. To play optimally, we need to learn about the values r_1, r_2, and x. Assume that we play for n steps a and b uniformly at random when in state q_0. This will probably allows us to reach q_1 and q_2 a number of times, and so to learn r_0 and r_1, and compute an estimation $\hat{x}$ of x. If $\hat{x} \cdot r_1 > r_0$, we may want to conclude that the optimal strategy is to always play b from q_0. But we face here two major difficulties. First, after the learning episode of n steps, we can observe $\hat{x} \cdot r_1 > r_0$ while $x \cdot r_1 \leq r_0$, this is because we may have been unlucky and observed statistics that differ from the real distribution. Second, playing b always is not an option if we want to surely satisfy the parity objective. In this paper, we give algorithms to overcome the two problems. In our example, the strategy constructed by our algorithm will do the following: given $\varepsilon, \gamma \in (0, 1)$, choose $n \in \mathbb{N}$ large enough, learn $\hat{x}$ such that $|\hat{x} - x| \leq \varepsilon$ with probability more than $1 - \gamma$, then if $\hat{x} \cdot r_1 \leq r_0$, play a forever. Otherwise, keep playing b for longer and longer episodes. If during one of these episodes, the state q_2 is not visited (i.e. the parity objective is endangered as the minimal priority seen during the episode is odd) switch to playing a forever.

Transition-probability lower bound. Let $\pi_{\min} \in [0, 1] \cap \mathbb{Q}$ be a *transition-probability lower bound*. We say that δ is *compatible* with $\pi_{\min}$ if for all $(q, a, q') \in Q \times A \times Q$ we have that: either $\delta(q'|q, a) \geq \pi_{\min}$ or $\delta(q'|q, a) = 0$.

3 Learning for MP: the Unconstrained Case

In this section, we focus on the design of optimal learning strategies for the mean-payoff function in the unconstrained single-end-component case. That is, we have an unknown strongly connected MDP with no parity objective.

We consider, in turn, learning strategies that use finite and infinite memory. Whereas classical RL algorithms focus on achieving an optimal expected value (see, e.g., [32]; cf. [6]), we prove here that a stronger result is achievable: one can ensure – using finite memory only – outcomes that are close to the best expected value with high probability. Further, with infinite memory the optimal outcomes can be ensured with probability 1. In both cases, we argue that our results are tight.

For the rest of this section, let us fix an automaton $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is an EC, and some $\pi_{\min} \in (0, 1]$.

Yardstick. Let δ be a transition function compatible with $\mathcal{A}$ and $\pi_{\min}$, and r be a reward function. The optimal expected mean-payoff value that is achievable in the unique EC (Q, α_T) is defined as $\mathbf{Val}(Q, \alpha_T) \stackrel{\text{def}}{=} \sup_\sigma \mathbb{E}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\underline{\mathbf{MP}}]$ for any $q_0 \in Q$. Indeed, it is well known that this value is the same for all states in the same EC.

Note that this value can always be obtained by a memoryless deterministic [20] and unichain [11] expectation-optimal strategy when δ and r are known. We will use this value as a yardstick for measuring the performance of the learning strategies we describe below.

Model learning. Our strategies learn approximate models of δ and r to be able to compute near-optimal strategies. To obtain those models, we use an approach based on ideas from probably approximately correct (PAC) learning. Namely, we will execute a random exploration of the MDP for some number of steps and obtain an empirical estimation of its stochastic dynamics, see e.g. [30]. We say that a memoryless strategy λ is a *(uniform random) exploration strategy* for a function $\beta : Q \to \mathcal{P}(A)$ if $\lambda(a|q) = 1/|\beta(q)|$ for all $q \in Q, a \in \alpha(q)$ such that $a \in \beta(q)$ and $\lambda(a|q) = 0$ otherwise. Each time the random exploration enters a state q and chooses an action a, we say that it performs an experiment on (q, a), and if the state reached is q' then we say that the result of the experiment is q'. Furthermore, the value $r(q, a, q')$ is then known to us. To learn an approximation δ' of the transition function δ, and to learn r, the learning strategy remembers statistics about such experiments. If the random exploration strategy is executed long enough then it collects sufficiently many experiment results to accurately approximate the transition function δ and the exact reward function r with high probability.

The next lemma gives us a bound on the number of $|Q|$-step episodes for which we need to exercise such a strategy to obtain the desired approximation with at least some given probability. It can be proved via a simple application of Hoeffding's inequality.

▶ **Lemma 7.** *For all ECs (S, β) and all $\varepsilon, \gamma \in (0, 1)$ one can compute $n \in \mathbb{N}$ (exponential in $|Q|$ and polynomial in $|A|$, $\pi_{\min}^{-1}$, $\ln(\gamma^{-1})$, and ε^{-1}) such that following an exploration strategy for β during n (potentially non-consecutive) episodes of $|Q|$-steps suffices to collect enough information to be able to compute a transition function δ' such that $\mathbb{P}\left[\delta' \sim^{\varepsilon}_{(S,\beta)} \delta\right] \geq 1 - \gamma$.*

3.1 Finite memory

We now present a family of finite memory strategies σ_{fin} that force, given any $\varepsilon, \gamma \in (0, 1)$, outcomes with a mean payoff that is ε-close to the optimal expected value with probability higher than $1 - \gamma$. The strategy σ_{fin} is defined as follows.

1. First, σ_{fin} follows the model-learning strategy above for L steps, according to Lemma 7, in order to obtain an approximation δ' of δ such that $\delta' \sim^{\eta} \delta$ with probability at least $1 - \gamma$. A reward function r' is also constructed from the observed rewards.
2. Then, σ_{fin} follows a memoryless deterministic expectation-optimal strategy τ for $\mathcal{A}_{\delta',r'}$.

The following result tells us that if the learning phase is sufficiently long, then we can obtain, with σ_{fin}, a near-optimal mean payoff with high probability.

▶ **Proposition 8.** *For all $\varepsilon, \gamma \in (0, 1)$, one can compute $L \in \mathbb{N}$ such that for the resulting finite memory strategy σ_{fin}, for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma_{\text{fin}}}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T) - \varepsilon] \geq 1 - \gamma$.*

Proof. We will make use of Lemma 5. For that purpose, let $\eta = \min\{\pi_{\min}, \eta_\varepsilon\}$ where η_ε is as in the statement of the lemma. Next, we set $L = |Q|n$ where n is as dictated by Lemma 7 using η and γ. By Lemma 7, with probability at least $1 - \gamma$ our approximation δ' is such

that $\delta' \sim^{\eta} \delta$. Since $\eta \leq \pi_{\min}$, it follows that $\mathrm{supp}(\delta) = \mathrm{supp}(\delta')$ and we now have learned r, again with probability $1-\gamma$. Finally, since $\eta \leq \eta_{\varepsilon}$, Lemma 5 implies the desired result. ◀

▶ Remark (Finite-memory implementability). Note that σ_{fin}, as we described it previously, is not immediately seen to be a computable finite stochastic Mealy machine. Let us consider all possible histories of length L. Observe that this set is not finite because of the unknown rewards which can range over arbitrary rational numbers in $[0,1]$. However, we can finitize the set by focusing only on rewards of bounded representation size by imposing an upper-bound on the bitsize of their representation (truncating the rest off observed rewards) while still satisfying the hypotheses of Lemma 5. Now, for all such histories we can compute an approximation δ' of δ and an approximation r' of the observed reward function r. Using that information, the required finite-memory expectation-optimal strategy τ can be computed. We encode these (finitely many) strategies into the machine implementing σ_{fin} so that it only has to choose which one to follow forever after the (finite) learning phase has ended. Hence, one can indeed construct a finite-memory strategy realizing the described strategy.

Optimality. The following tells us that we cannot do better with finite memory strategies.

▶ **Proposition 9.** *Let $\mathcal{A}$ be the single-EC automaton on the right-hand side of Fig. 1 and $\pi_{\min} \in (0,1]$. For all $\varepsilon, \gamma \in (0,1)$, the following two statements hold.*

- *For all finite memory strategies σ, there exist δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and a reward function r, such that $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q,\alpha_T) - \varepsilon] < 1$.*
- *For all finite memory strategies σ, there exist δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and a reward function r such that $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) < \mathbf{Val}(Q,\alpha_T)] \geq \gamma$.*

Proof sketch. With a finite-memory strategy we cannot satisfy a stronger guarantee than being ε-optimal with probability at least $1-\gamma$ in this example. Indeed, as we can only use finite memory, we can only learn imprecise models of δ and r. That is, we will always have a non-zero probability to have approximated x or y arbitrarily far from their actual values. It should then be clear that neither optimality with high probability nor almost-sure ε-optimality can be achieved. ◀

3.2 Infinite memory

While we have shown that probably approximately optimal is the best that can be obtained with finite memory learning strategies, we now establish that with infinite memory, one can guarantee almost sure optimality.

To this end, we define a strategy σ_{∞} which operates in episodes consisting of two phases: learning and optimization. In episode $i \in \mathbb{N}$, the strategy does the following.

1. It first follows an exploration strategy λ for α_T during L_i steps, there exist models δ_i and r_i based on the experiments obtained throughout the $\sum_{j=0}^{i} L_j$ steps during which λ has been followed so far.
2. Then, σ_{∞} follows a unichain memoryless deterministic expectation-optimal strategy $\sigma^{\delta_i}_{\text{MP}}$ for $\mathcal{A}_{\delta_i,r_i}$ during O_i steps.

One can then argue that σ_{∞} can be instantiated so that in every episode the finite average obtained so far gets ever close to $\mathbf{Val}(Q,\alpha_T)$ with ever higher probability. This is achieved by choosing the L_i as an increasing sequence so that the approximations δ_i get ever better with ever higher probability. Then, the O_i are chosen so as to compensate for the past history, for the time before the induced MC reaches its limit distribution, and for the future number of steps that will be spent learning in the next episode. The latter then allows us to use the Borel-Cantelli lemma to show that in the unknown EC we can obtain its value almost surely.

▶ **Proposition 10.** *One can compute a sequence $(L_i, O_i)_{i\in\mathbb{N}}$ such that $L_i \geq |Q|$ for all $i \in \mathbb{N}$; additionally the resulting strategy σ_∞ is such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma_\infty}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T)] = 1$.*

Optimality. Note that σ_∞ is optimal since it obtains with probability 1 the best value that can be obtained when the MDP is fully known, i.e. when δ and r are known in advance.

4 Learning for MP under a Sure Parity Constraint

We show here how to design learning strategies that obtain near-optimal mean-payoff values while ensuring that all runs satisfy a given parity objective with certainty.

First, we note that all such learning strategies must avoid entering states q from which there is no strategy to enforce the parity objective with certainty. Hence, we make the hypothesis that all such states have been removed from the automaton $\mathcal{A}$, and so we assume that for all $q_0 \in Q$ there exists a strategy σ_{par} such that for all functions δ compatible with $\mathcal{A}$, for all reward functions r, and for all $\varrho \in \text{Runs}^{q_0}(\mathcal{A}^{\sigma}_{\delta,r})$, we have $\varrho \models \textsc{Parity}$. It is worth noting that, in fact, there exists a memoryless deterministic strategy such that the condition holds for all $q_0 \in Q$ [4, 3]. Notice the swapping of the quantifiers over the initial states and the strategy, this is why we say it is *uniformly winning (for the parity objective)*. The set of states to be removed, along with a uniformly winning strategy, can be computed in quasi-polynomial time [12]. We say that an automaton with no states from which there is no winning strategy is *surely good*.

We study the design of learning strategies for mean-payoff optimization under *sure* parity constraints for increasingly complex cases.

4.1 The case of a single good EC

Consider a surely-good automaton $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is a GEC, i.e. the minimal priority of a state in the EC is even, and some $\pi_{\min} \in (0, 1]$.

Yardstick. For this case, we use as yardstick the optimal expected mean-payoff value: $\mathbf{Val}(Q, \alpha_T) = \sup_\sigma \mathbb{E}^{q_0}_{\mathcal{A}^{\sigma}_{\delta,r}}[\underline{\mathbf{MP}}]$.

Learning strategy. We show here that it is possible to obtain an optimal mean-payoff with high probability. Note that our solution extends a result given by Almagor et al. [1] for *known* MDPs. The main idea behind our solution is to use the strategy σ_∞ from Proposition 10 in a controlled way: we verify that during all successive learning and optimization episodes, the minimal parity value that is visited is even. If during some episode, this is not the case, then we resort to a strategy σ_{par} that enforces the parity objective with certainty. Such σ_{par} is guaranteed to exist as $\mathcal{A}$ is surely good.

▶ **Proposition 11.** *For all $\gamma \in (0, 1)$, there exists a strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\varrho \models \textsc{Parity}$ for all $\varrho \in \text{Runs}^{q_0}(\mathcal{A}^{\sigma}_{\delta,r})$ and $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T)] \geq 1 - \gamma$.*

Proof sketch. We modify σ_∞ so as to "give up" on optimizing the mean payoff if the minimal even priority has not been seen during a long sequence of episodes. This will guarantee that the measure of runs which give up on the mean-payoff optimization is at most γ.

First, recall that we can instantiate σ_∞ so that $L_i \geq |Q|$ for all $i \in \mathbb{N}$. Hence, with some probability $\zeta > 0$, during every learning phase, we visit a state with even minimal priority. We can then find a sequence $n_1, n_2, \cdots \in \mathbb{N}^\omega$ of natural numbers such that $\prod_{j=i}^{\infty}(1-\zeta^{n_j}) \geq 1-\gamma$, for some $i \in \mathbb{N}$. Given this sequence, we apply the following monitoring. If for $\ell \in \mathbb{N}$ we write $N_\ell \stackrel{\text{def}}{=} \sum_{k=1}^{\ell-1} n_k$, then at the end of the ℓ-th episode we verify that during some learning phase from $L_{N_\ell}, L_{N_\ell+1}, \ldots, L_{N_\ell+n_\ell}$ we have visited a state with minimal even priority, otherwise we switch to a parity-winning strategy forever. ◀

Optimality. The following proposition tells us that the guarantees from Proposition 11 are indeed optimal w.r.t. our chosen yardstick.

▶ **Proposition 12.** *Let $\mathcal{A}$ be the single-GEC automaton on the left-hand side of Fig. 1 and $\pi_{\min} \in (0,1]$. For all parity-winning strategies σ, there exist δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and a reward function r, such that $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T)] < 1$.*

Proof sketch. Consider a reward function such that $r_0 = 0$ and $r_1 = 1$ and an arbitrary δ. It is easy to see that $\mathbf{Val}(Q, \alpha_T) = 1$. However, any strategy that ensures the parity objective is satisfied surely must be such that, with probability $\gamma > 0$, it switches to follow a strategy $q_2 \mapsto (a \mapsto 1)$ forever. Hence, with probability at least γ its mean-payoff is sub-optimal. ◀

4.2 The case of a single EC

We now turn to the case where the surely-good automaton $\mathcal{A} = (Q, A, T, p)$ consists of a unique, not necessarily good, EC (Q, α_T). Let us also fix some $\pi_{\min} \in (0, 1]$.

An important observation regarding single-end-component MDPs that are surely good is that they contain at least one GEC as stated in the following lemma.

▶ **Lemma 13.** *For all surely-good automata $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is an EC there exists $(S, \beta) \subseteq (Q, \alpha_T)$ such that (S, β) is a GEC in $\mathcal{A}_{\delta,r}$ for all δ compatible with $\mathcal{A}$ and all reward functions r, i.e. (Q, α_T) is weakly good.*

Yardstick. Let δ and r be fixed in the single EC, our yardstick for this case is defined as follows: $\mathbf{sVal}(Q, \alpha_T) \stackrel{\text{def}}{=} \max_{q \in Q} \sup\left\{ \mathbb{E}^q_{\mathcal{A}^\sigma_{\delta,r}}[\underline{\mathbf{MP}}] \;\middle|\; \sigma \text{ is a parity-winning strategy} \right\}$. That is $\mathbf{sVal}(Q, \alpha_T)$ is the best MP expectation value that can be obtained from a state $q \in Q$ with a parity-winning strategy. It is remarkable to note that we take the maximal value over all states in Q. As noted by Almagor et al. [1], this value is not always achievable even when δ and r are a priori known, but it can be approached arbitrarily close.

Learning strategy. The following proposition tells us that we can obtain a value close to $\mathbf{sVal}(Q, \alpha_T)$ with arbitrarily high probability while satisfying the parity objective surely.

▶ **Proposition 14.** *For all $\varepsilon, \gamma \in (0, 1)$ there exists a strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\varrho \models$ Parity for all $\varrho \in \mathrm{Runs}^{q_0}(\mathcal{A}^\sigma_{\delta,r})$ and $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{sVal}(Q, \alpha_T) - \varepsilon] \geq 1 - \gamma$.*

Proof sketch. We define a strategy σ as follows. Let $\eta = \min\{\pi_{\min}, \eta_{\varepsilon/2}\}$ for $\eta_{\varepsilon/2}$ as defined for Lemma 5. The strategy σ plays as follows.

1. It first computes δ' such that $\delta' \sim^\eta \delta$ with probability at least $1 - \gamma/4$ and a reward function r' by following an exploration strategy λ for α_T during J_0 steps (see Lemma 7).

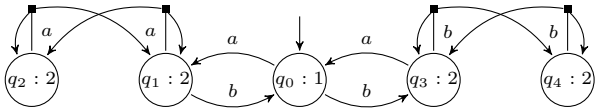


Figure 2 An automaton for which it is impossible to learn to obtain near-optimal mean-payoff almost surely or optimal mean-payoff with high probability, while satisfying the parity objective. For clarity, probability and reward placeholders have been omitted.

2. It then selects a contained maximal good EC (MGEC) with maximal expected mean-payoff value (see Lemma 13) and tries to reach it with probability at least $1 - \gamma/4$ by following λ during J_1 steps.
3. Finally, if the component is reached, it follows a strategy τ as in Proposition 11 with $\gamma/4$ from then onward.

If the learning "fails" or if the component is not reached, the strategy reverts to following a winning strategy forever. (A failed learning phase is one in which the approximated distribution function does not have T as its support.) ◀

Optimality. The following states that we cannot improve on the result of Proposition 14.

▶ **Proposition 15.** *Let $\mathcal{A}$ be the single-EC automaton in Fig. 2 and $\pi_{\min} \in (0, 1]$. For all $\varepsilon, \gamma \in (0, 1)$, the two following statements hold.*

- *For all strategies σ, there exist δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and a reward function r, such that $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{sVal}(Q, \alpha_T) - \varepsilon] < 1$.*
- *For all strategies σ, there exist δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and a reward function r, such that $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{sVal}(Q, \alpha_T)] < 1 - \gamma$.*

Proof sketch. Observe that the MEC is not a good EC. However, it does contain the GECs with states $\{q_1, q_2\}$ and $\{q_3, q_4\}$ respectively. Now, since those two GECs are separated by q_0, whose priority is 1, any winning strategy must at some point stop playing to q_0 and commit to a single GEC. Thus, the learning of the global EC can only last for a finite number of steps. It is then straightforward to argue that near-optimality with high-probability is the best achievable guarantee. ◀

4.3 General surely-good automata

In this section, we generalize our approach from single-EC automata to general automata. We will argue that, under a sure parity constraint, we can achieve a near-optimal mean payoff with high probability in any MEC (S, β) in which we end up with non-zero probability. That is, given that the event $\mathrm{Inf} \subseteq S$, defined as the set of all runs that eventually always stay within S, has non-zero probability measure.

▶ **Theorem 16.** *Consider a surely-good automaton $\mathcal{A} = (Q, A, T, p)$ and some $\pi_{\min} \in (0, 1]$. For all $\varepsilon, \gamma \in (0, 1)$ there exists a strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and all reward functions r, we have*

- *$\varrho \models \textsc{Parity}$ for all $\varrho \in \mathrm{Runs}^{q_0}(\mathcal{A}^\sigma_{\delta,r})$ and*
- *$\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{sVal}(S, \beta) - \varepsilon \mid \mathrm{Inf} \subseteq S] \geq 1 - \gamma$ for all $(S, \beta) \in \mathrm{MEC}_{\mathcal{A}_{\delta,r}}$ such that (S, β) is weakly good and $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\mathrm{Inf} \subseteq S] > 0$.*

Proof sketch. The strategy σ we construct follows a parity-winning strategy σ_{par} until a state contained in a weakly good MEC, that has not been visited before, is entered. In this case, the strategy follows τ (the strategy from Proposition 14). Observe that when τ switches

to σ_{par} (a parity-winning strategy) it may exit the end component. If this happens, then the component is marked as visited and σ_{par} is followed until a new – not previously visited – maximal good end component is entered. In that case, we switch to τ once more. Crucially, the new strategy σ ignores MECs that are revisited ◀

▶ Remark (On the choice of MECs to reach). The strategy constructed for the proof of Theorem 16 has to deal with leaving a MEC due to the fallbacks to the parity-winning strategy σ_{par}. However, surprisingly, instead of actually following σ_{par}, upon entering a new MEC it has to restart the process of achieving a satisfactory mean-payoff. Indeed, otherwise the overall mass of sub-optimal runs from various MECs (each smaller than γ) could get concentrated in a single MEC, thus violating the advertised guarantees.

The strategy could be simplified as follows. First, we follow any strategy to reach a bottom MEC (BMEC) – that is, a MEC from which no other MEC is reachable. By definition, the winning strategy can be played here and the MEC cannot be escaped. Therefore, in the BMEC we run the strategy as described, and after the fallback we indeed simply follow σ_{par}. If we did not reach a BMEC after a long time, we could switch to the fallback, too. While this strategy is certainly simpler, our general strategy has the following advantage. Intuitively, we can force the strategy to stay in any current good MEC, even if it is not bottom, and thus maybe achieve a more satisfactory mean-payoff. Further, whenever we want, we can force the strategy to leave the current MEC and go to a lower one. For instance, if the current estimate of the mean payoff is lower than what we hope for, we can try our luck in a lower MEC. We further comment on the choice of unknown MECs in the conclusions.

5 Learning for MP under an Almost-Sure Parity Constraint

In this section, we turn our attention to learning strategies that must ensure a parity objective not with certainty (as in previous section) but *almost surely*, i.e. with probability 1. As winning almost surely is less stringent, we can hope both for a stricter yardstick (i.e. better target values) and also better ways of achieving such high values. We show here that this is indeed the case. Additionally, we argue that several important learning results can now be obtained with finite-memory strategies.

As previously, we make the hypothesis that we have removed from $\mathcal{A}$ all states from which the parity objective cannot be forced with probability 1 (no such state can ever be entered). Note that to compute the set of states to remove, we do not need the knowledge of δ but only the support as given by $\mathcal{A}$. States to remove can be computed in polynomial time using graph-based algorithms (see, e.g., [5]). An automaton $\mathcal{A}$ which contains only almost-surely winning states for the parity objective is called *almost-surely good*.

We have, as in the previous section, that for all automata $\mathcal{A}$ there exists a memoryless deterministic strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$, for all r, the measure of the subset of $\varrho \in \text{Runs}^{q_0}(\mathcal{A}^{\sigma}_{\mu,r})$ such that $\varrho \models \textsc{Parity}$ is equal to 1 (see e.g. [5]). Such a strategy is said to be *uniformly almost-sure winning (for the parity objective)*. In the sequel, we denote such a strategy $\sigma^{\text{as}}_{\text{par}}$.

We now study the design of learning strategies for mean-payoff optimization under *almost-sure* parity constraints for increasingly complex cases.

5.1 The case of a good end component

Consider an automaton $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is a GEC, and some $\pi_{\min} \in (0, 1]$.

Yardstick. For this case, we use as a yardstick the optimal expected mean-payoff value: $\mathbf{Val}(Q, \alpha_T) = \sup_\sigma \mathbb{E}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\underline{\mathbf{MP}}]$ for any $q_0 \in Q$.

Learning strategies. We start by noting that σ_∞ from Section 3 also ensures that the parity objective is satisfied almost surely when exercised in a GEC.

▶ **Proposition 17.** *One can compute a sequence $(L_i, O_i)_{i\in\mathbb{N}}$ such that for the resulting strategy σ_∞ we have that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma_\infty}_{\delta,r}}[\textsc{Parity}] = 1$ and $\mathbb{P}^{q_0}_{\mathcal{A}^{\sigma_\infty}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T)] = 1$.*

Proof. By Proposition 10, one can choose parameter sequences such that $L_i \geq |Q|$ for all $i \in \mathbb{N}$ and so that we obtain the second part of the claim. Then, since in every episode we have a non-zero probability of visiting a minimal even priority state, we obtain the first part of the claim as a simple consequence of the second Borel-Cantelli lemma. ◀

We now turn to learning using finite memory only. Consider parameters $\varepsilon, \gamma \in (0, 1)$. Let $\eta = \min\{\pi_{\min}, \eta_{\varepsilon/4}\}$ for $\eta_{\varepsilon/4}$ as defined for Lemma 5. The strategy τ_{fin} that we construct does the following.

1. First, it computes δ' such that $\delta' \sim^\eta \delta$ with probability at least $1 - \gamma$ and a reward function r' by following an exploration strategy λ for α_T during J steps (see Lemma 7).
2. Then, it computes a unichain deterministic expectation-optimal strategy $\sigma^{\delta'}_{\text{MP}}$ for $\mathcal{A}_{\delta',r'}$ and repeats the following forever: follow $\sigma^{\delta'}_{\text{MP}}$ for O steps, then follow λ for $|Q|$ steps.

Using the fact that, in a finite MC with a single BSCC, almost all runs obtain the expected mean payoff and the assumption that the EC is good, one can then prove the following result.

▶ **Proposition 18.** *For all $\varepsilon, \gamma \in (0, 1)$ one can compute $L, O \in \mathbb{N}$ such that for the resulting strategy τ_{fin}, for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\mathbb{P}^{q_0}_{\mathcal{A}^{\tau_{\text{fin}}}_{\delta,r}}[\textsc{Parity}] = 1$ and $\mathbb{P}^{q_0}_{\mathcal{A}^{\tau_{\text{fin}}}_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{Val}(Q, \alpha_T) - \varepsilon] \geq 1 - \gamma$.*

Optimality. Obviously, the result of Proposition 17 is optimal as we obtain the best possible value with probability one. We claim that the result of Proposition 18 is also optimal as we have seen that when we use finite learning, we cannot do better than ε-optimality with high probability, this can be proved on the example of Fig. 2 with a similar argument to the one that has been developed for the proof of Proposition 15.

5.2 The case of a single end component

Consider an almost-surely-good automaton $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is an EC and some $\pi_{\min} \in (0, 1]$. The EC is not necessarily good but as the automaton is almost-surely-good, then we have the analogue of Lemma 13 in this context.

▶ **Lemma 19.** *For all almost-surely-good automata $\mathcal{A} = (Q, A, T, p)$ such that (Q, α_T) is an EC there exists $(S, \beta) \subseteq (Q, \alpha_T)$ such that (S, β) is a GEC in $\mathcal{A}_{\delta,r}$ for all δ compatible with $\mathcal{A}$ and all reward functions r, i.e. (Q, α_T) is weakly good.*

Yardstick. As a yardstick for this case, we use the following value: $\mathbf{asVal}(Q, \alpha_T) \stackrel{\text{def}}{=} \max\{\mathbf{Val}(S,\beta) \mid (S,\beta) \subseteq (Q,\alpha_T) \text{ and } (S,\beta) \text{ is a GEC}\}$. That is, $\mathbf{asVal}(Q,\alpha_T)$ is the best expected mean-payoff value that can be obtained in a GEC included in the EC. Such a good EC exists by Lemma 19.

Learning strategy. We will now prove an analogue of Proposition 14. For any given $\varepsilon, \gamma \in (0,1)$ we define the strategy σ as follows.

1. First, it follows an exploration strategy λ for α_T during sufficiently many steps (say K) to compute an approximation δ' of δ such that $\delta' \sim^{\eta_{\varepsilon/4}} \delta$ with probability at least $1-\gamma/2$; and a reward function r' (see Lemma 7).
2. Next, it selects a GEC (S,β) with maximal value $\pm\frac{\varepsilon}{4}$ (see Lemma 19) and computes for it a strategy τ as in Proposition 18 with $\varepsilon_1/2$ and $\gamma/2$.
3. Finally, σ follows λ until (S,β) is reached, then switches to τ.

▶ **Proposition 20.** *For all $\varepsilon, \gamma \in (0,1)$ one can construct a finite-memory strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and for all reward functions r, we have $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\text{PARITY}] = 1$ and $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{asVal}(Q,\alpha_T) - \varepsilon] \geq 1-\gamma$.*
See the remark in Sect. 4.3 for a comment on the finite-memory implementability of σ.

Optimality. Using the same example and reasoning as in Proposition 15, we can show that this result is optimal and cannot be improved. Also note that using infinite memory would not help as shown with the example of Fig. 2, where the learning needs to be finite and enforcing the almost sure parity does not require infinite memory.

5.3 General almost-surely-good automata

We now generalize our approach to general almost-surely-good automata.

▶ **Theorem 21.** *Consider an almost-surely-good automaton $\mathcal{A} = (Q, A, T, p)$ and some $\pi_{\min} \in (0,1]$. For all $\varepsilon, \gamma \in (0,1)$ one can compute a finite-memory strategy σ such that for all $q_0 \in Q$, for all δ compatible with $\mathcal{A}$ and $\pi_{\min}$, and all reward functions r, we have*

- $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\text{PARITY}] = 1$ *and*
- $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\varrho : \underline{\mathbf{MP}}(\varrho) \geq \mathbf{asVal}(S,\beta) - \varepsilon \mid \text{Inf} \subseteq S] \geq 1-\gamma$ *for all $(S,\beta) \in \text{MEC}_{\mathcal{A}_{\delta,r}}$ such that (S,β) is weakly good and $\mathbb{P}^{q_0}_{\mathcal{A}^\sigma_{\delta,r}}[\text{Inf} \subseteq S] > 0$.*

Proof sketch. The argument to prove the above result is simple: σ follows a strategy $\sigma^{\text{as}}_{\text{par}}$ that ensures satisfying the parity objective almost surely. Then, if the run reaches a state contained in a weakly good MEC, σ switches to τ as described in Proposition 20. ◀

See the remark in Sect. 3.1 for a word on how to modify σ to favour some unknown MECs.

6 Conclusion

As future work, we would like to study different configurations resulting from relaxations of the assumptions we make in this work (i.e. full support, $\pi_{\min}$, and bounded reward). Further, we would like to obtain model-free learning algorithms ensuring the same guarantees we give here. Finally, we have left open the choice of strategy driving the visits to MECs in Theorems 16 and 21 (as long as it satisfies the parity objective). Indeed, the question of computing an "optimal" such strategy in view of the unknown components of the MDP can be addressed in different ways. One such way would be to model the problem as a Canadian traveler problem [24].

References

1. Shaull Almagor, Orna Kupferman, and Yaron Velner. Minimizing expected cost under hard boolean constraints, with applications to quantitative synthesis. 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 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.9.
2. Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. *CoRR*, abs/1708.08611, 2017. arXiv:1708.08611.
3. Benjamin Aminof and Sasha Rubin. First-cycle games. *Information and Compution*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.
4. Krzysztof R. Apt and Erich Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.
5. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
6. Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 2017. URL: http://proceedings.mlr.press/v70/bellemare17a.html.
7. Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *ITA*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.
8. 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.
9. Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *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 98–114. Springer, 2014.
10. Tomás Brázdil, Antonín Kucera, and Petr Novotný. Optimizing the expected mean payoff in energy Markov decision processes. In *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.
11. Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 199–213. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.199.
12. 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.

13 Krishnendu Chatterjee. Concurrent games with tail objectives. *Theoretical Computer Science*, 388(1-3):181–198, 2007. doi:10.1016/j.tcs.2007.07.047.

14 Krishnendu Chatterjee. Robustness of structurally equivalent concurrent parity games. 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 270–285. Springer, 2012. doi:10.1007/978-3-642-28729-9_18.

15 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity markov decision processes. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011. doi:10.1007/978-3-642-22993-0.

16 Krishnendu Chatterjee, Petr Novotný, Guillermo A. Pérez, Jean-François Raskin, and Dorde Zikelic. Optimizing expectation with guarantees in POMDPs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3725–3732. AAAI Press, 2017. URL: http://www.aaai.org/Library/AAAI/aaai17contents.php.

17 Lorenzo Clemente and Jean-François Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 257–268. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.33.

18 Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. In *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 112–129. Springer, 2016. doi:10.1007/978-3-662-49674-9.

19 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 *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.

20 Hugo Gimbert. Pure stationary optimal strategies in Markov decision processes. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007. doi:10.1007/978-3-540-70918-3_18.

21 Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for MDPs. 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 130–146. Springer, 2016. doi:10.1007/978-3-662-49674-9_8.

22 Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285, 1996. doi:10.1613/jair.301.

23 James R. Norris. *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998.

24 Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

25 Martin L. Puterman. *Markov Decision Processes.* Wiley-Interscience, 2005.

26 Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010. URL: `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html`.

27 Eilon Solan. Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability*, 16(4):831–845, 2003.

28 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction.* Adaptive computation and machine learning. MIT Press, 2018. URL: `http://www.incompleteideas.net/book/the-book-2nd.html`.

29 Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13, 1995. `doi:10.1007/3-540-59042-0_57`.

30 Leslie G. Valiant. A theory of the learnable. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 436–445. ACM, 1984. `doi:10.1145/800057.808710`.

31 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 327–338. IEEE Computer Society, 1985. `doi:10.1109/SFCS.1985.12`.

32 Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992. `doi:10.1007/BF00992698`.

33 Min Wen, Rüdiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 4983–4990. IEEE, 2015. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7347169`.

34 Min Wen and Ufuk Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3630–3636. IJCAI/AAAI Press, 2016. URL: `http://www.ijcai.org/Proceedings/2016`.

Deciding Probabilistic Bisimilarity Distance One for Probabilistic Automata

Qiyi Tang
Department of Computing, Imperial College, London, United Kingdom

Franck van Breugel
Department of Electrical Engineering and Computer Science, York University, Toronto, Canada

Abstract

Probabilistic bisimilarity, due to Segala and Lynch, is an equivalence relation that captures which states of a probabilistic automaton behave exactly the same. Deng, Chothia, Palamidessi and Pang proposed a robust quantitative generalization of probabilistic bisimilarity. Their probabilistic bisimilarity distances of states of a probabilistic automaton capture the similarity of their behaviour. The smaller the distance, the more alike the states behave. In particular, states are probabilistic bisimilar if and only if their distance is zero.

Although the complexity of computing probabilistic bisimilarity distances for probabilistic automata has already been studied and shown to be in $\mathbf{NP} \cap \mathbf{coNP}$ and **PPAD**, we are not aware of any practical algorithm to compute those distances. In this paper we provide several key results towards algorithms to compute probabilistic bisimilarity distances for probabilistic automata. In particular, we present a polynomial time algorithm that decides distance one. Furthermore, we give an alternative characterization of the probabilistic bisimilarity distances as a basis for a policy iteration algorithm.

2012 ACM Subject Classification Mathematics of computing → Markov processes, Theory of computation → Concurrency

Keywords and phrases probabilistic automaton, probabilistic bisimilarity, distance

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.9

Funding Natural Sciences and Engineering Research Council of Canada

Acknowledgements The authors would like to thank the referees for their detailed and constructive feedback.

1 Introduction

Behavioural equivalences, such as bisimilarity, are one of the cornerstones of concurrency theory. Recall that a behavioural equivalence $\sim \subseteq S \times S$, where S is the set of states of the model, satisfies

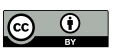
$s \sim s$

if $s \sim t$ then $t \sim s$

if $s \sim t$ and $t \sim u$ then $s \sim u$

for all s, t, $u \in S$. If $s \sim t$ then states s and t behave the same.

As first observed by Giacalone, Jou and Smolka [17], behavioural equivalences are *not robust* for models that contain quantitative information such as probabilities and time. This lack of robustness is caused by the discrepancy between the discrete nature of behavioural



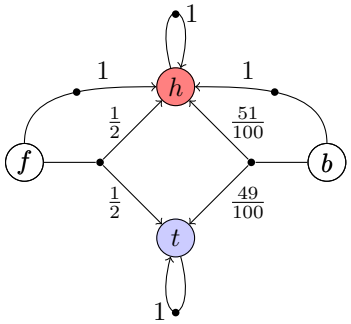
equivalence and the continuous nature of the quantitative information on the which the behavioural equivalence relies. In particular, even small changes to the quantitative information may cause behaviourally equivalent states become inequivalent or vice versa.

Giacalone et al. proposed *behavioural pseudometrics* as a robust quantitative generalization of behavioural equivalences. A behavioural pseudometric $d : S \times S \to [0, 1]$ satisfies

$d(s, t) = 0$ if and only if $s \sim t$
$d(s, t) = d(t, s)$
$d(s, u) \leq d(s, t) + d(t, u)$

for all s, t, $u \in S$. The distance $d(s, t)$ measures the similarity of the behaviour of states s and t. The smaller this distance, the more alike the states behave. Distance zero captures that states are behaviourally equivalent.

In this paper, we focus on *probabilistic automata*. This model was first studied by Segala in [27]. It captures both nondeterminism (and, hence, concurrency) and probabilities. Let us consider a simple example.



The states of a probabilistic automaton are labelled. These labels provide a partition of the states so that states satisfying the same basic properties of interest are in the same partition. In the above example, the labels are represented by colours. Each state has one or more probabilistic transitions. For example, the state t has a single probabilistic transition that takes state t to itself with probability one. State f has two probabilistic transitions. The one takes state f to state h with probability one. The other represents a fair coin toss, that is, it transitions to state h with probability $\frac{1}{2}$ and to state t with probability $\frac{1}{2}$. Also state b has two transitions, one of which represents a biased coin toss.

Segala and Lynch [28] introduced *probabilistic bisimilarity*. This behavioural equivalence for probabilistic automata generalises the one introduced by Larsen and Skou [25]. The latter is applicable to models without nondeterminism, known as labelled Markov chains. States s and t of a probabilistic automaton are probabilistic bisimilar if for each outgoing probabilistic transition of state s there exists a matching outgoing probabilistic transition of state t, and vice versa. Two probabilistic transitions match if they both transition to each probabilistic bisimilarity equivalence class with the exact same probability. States f and b in the above example are not probabilistic bisimilar. Although the transition from state f to state h can be matched by the transition from state b to state h, the probabilistic transitions representing a fair and biased coin toss do not match since the probabilities are slightly different.

Deng, Chothia, Palamidessi and Pang [12] introduced a behavioural pseudometric for probabilistic automata that generalises probabilistic bisimilarity. The Hausdorff metric [18] and the Kantorovich metric [22] are key ingredients of this pseudometric. The former is used

to capture nondeterminism. This idea dates back to the work of De Bakker and Zucker [4]. The latter was first used by Van Breugel and Worrell [7] to capture probabilistic behaviour. On the one hand, the behaviours of the states h and t of the above example 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 these states have probabilistic bisimilarity distance $\frac{1}{100}$.

Tracol, Desharnais and Zhioua [34] also introduced a behavioural pseudometric for probabilistic automata. Their probabilistic bisimilarity distances generalise probabilistic bisimilarity as well, but are different from the ones introduced by Deng et al. An example showing the difference can be found in [34, Example 5]. To compute their probabilistic bisimilarity distances, they developed an iterative algorithm. In each iteration, a maximum flow problem needs to be solved. The resulting algorithm is polynomial time.

The complexity of computing the probabilistic bisimilarity distances for probabilistic automata a la Deng et al. was first studied by Fu [15]. He showed that these probabilistic bisimilarity distances are rational. Furthermore, he proved that the problem of deciding whether the distance of two states is smaller than a given rational is in $\mathbf{NP} \cap \mathbf{coNP}$. The proof can be adapted to show that the decision problem is in $\mathbf{UP} \cap \mathbf{coUP}$ [16]. Recall that $\mathbf{UP}$ contains those problems in $\mathbf{NP}$ with a unique accepting computation. Van Breugel and Worrell [8] have shown that the problem of computing the probabilistic bisimilarity distances is in $\mathbf{PPAD}$, which is short for polynomial parity argument in a directed graph.

For the behavioural pseudometric of Deng et al., states are probabilistic bisimilar if and only if they have distance zero. Since probabilistic bisimilarity can be decided in polynomial time, as shown by Baier [2], distance zero can be decided in polynomial time as well. In Section 5 we present a polynomial time algorithm that decides distance one.

As we have already shown in [32] in the context of labelled Markov chains, being able to decide distance zero and distance one in polynomial time has significant impact on computing probabilistic bisimilarity distances. For example, we can determine in polynomial time how many, if any, distances are non-trivial, that is, greater than zero and smaller than one. The technical details in this paper are considerably more involved than those in [32].

Deng et al. define their pseudometric as a least fixed point. In Section 4 we present an alternative characterization of the probabilistic bisimilarity distances. This characterization is similar to the one presented for labelled Markov chains by Chen, Van Breugel and Worrell [9]. The latter characterization provided the foundation for the policy iteration algorithm to compute the probabilistic bisimilarity distances for labelled Markov chains by Bacci, Bacci, Larsen and Mardare [1] (see also [31]). Our alternative characterization plays a key role in the correctness proof of our algorithm.

2 Order and Distances

In this section, we provide some definitions and results from the literature about orders and distances that we will use in the remainder of this paper. For more details we refer the reader to, for example, [11] and [3]. Given a set S, we denote the set of functions from $S \times S$ to $[0,1]$ by $[0,1]^{S\times S}$. As in the work of Desharnais et al. [13], we endow the set $[0,1]^{S\times S}$ with the following natural order.

▶ **Definition 1.** The relation $\sqsubseteq \subseteq [0,1]^{S\times S} \times [0,1]^{S\times S}$ is defined by

$$d \sqsubseteq e \text{ if } d(s,t) \leq e(s,t) \text{ for all } s,t \in S.$$

▶ **Proposition 2.** $\langle [0,1]^{S\times S}, \sqsubseteq \rangle$ is a complete lattice.

Proof. See, for example, [13, Lemma 3.2]. ◀

Let $\langle X, \leq \rangle$ be an ordered set. Let $f : X \to X$. Following [11, Definition 8.14], we define the following three notions:

- $x \in X$ is a *fixed point* of f if $f(x) = x$,
- $x \in X$ is a *pre-fixed point* of f if $f(x) \leq x$, and
- $x \in X$ is a *post-fixed point* of f if $x \leq f(x)$.

A function $f : X \to X$ is *monotone* if for all $x, y \in X$, $x \leq y$ implies $f(x) \leq f(y)$. The following result is known as the Knaster-Tarski fixed point theorem [24, 33].

▶ **Theorem 3.** *Let X be a complete lattice and let $f : X \to X$ be a monotone function.*
(a) *f has a greatest fixed point.*
(b) *The greatest fixed point of f is the greatest post-fixed point of f.*
(c) *f has a least fixed point.*
(d) *The least fixed point of f is the least pre-fixed point of f.*

Proof. See, for example, [11, Theorem 2.35] and [11, Theorem 8.20]. ◀

We denote the greatest and least fixed point of a function f by $\boldsymbol{\nu} f$ and $\boldsymbol{\mu} f$, respectively. Given a set X, we denote the set of subsets of X by 2^X. The correctness of our iterative algorithm to decide distance one relies on the following theorem.

▶ **Theorem 4.** *Let X be a finite set and let $\Phi : 2^X \to 2^X$ be a monotone function.*
(a) *$\boldsymbol{\mu}\Phi = \Phi^n(\emptyset)$ for some $n \in \mathbb{N}$.*
(b) *$\boldsymbol{\nu}\Phi = \Phi^n(X)$ for some $n \in \mathbb{N}$.*
(c) *If $Y \subseteq \boldsymbol{\mu}\Phi$ then $\boldsymbol{\mu}\Phi = \Phi^n(Y)$ for some $n \in \mathbb{N}$.*

Proof. See, for example, [10, Lemma 8]. ◀

The set $[0,1]^{S \times S}$ also carries the following natural metric.

▶ **Definition 5.** The function $\| \cdot - \cdot \| : [0,1]^{S \times S} \times [0,1]^{S \times S} \to [0,1]$ is defined by

$$\|d - e\| = \sup_{s,t \in S} |d(s,t) - e(s,t)|.$$

▶ **Proposition 6.** $\langle [0,1]^{S \times S}, \| \cdot - \cdot \| \rangle$ is a nonempty complete metric space.

Proof. See, for example, [3, Section 1.1.2]. ◀

Let $\langle X, d \rangle$ be a metric space and $c \in (0,1]$. A function $f : X \to X$ is *c-Lipschitz* if for all $x, y \in X$, $d(f(x), f(y)) \leq c\, d(x,y)$. A 1-Lipschitz function is also called *nonexpansive*. A function is *contractive* if it is c-Lipschitz for some $c \in (0,1)$. The following result is known as Banach's fixed point theorem [5].

▶ **Theorem 7.** *Let X be a nonempty complete metric space and $f : X \to X$ a contractive function. Then f has a unique fixed point.*

Proof. See, for example, [3, Theorem 1.34]. ◀

The Hausdorff metric [18] is defined as follows.

▶ **Definition 8.** The function $H : [0,1]^{X \times X} \to [0,1]^{2^X \times 2^X}$ is defined by

$$H(d)(M,N) = \max \left\{ \max_{\mu \in M} \min_{\nu \in N} d(\mu,\nu), \max_{\nu \in N} \min_{\mu \in M} d(\mu,\nu) \right\}.$$

Given a nonempty finite set X, we denote the set of probability distributions on X by $\mathit{Distr}(X)$. For $\mu \in \mathit{Distr}(X)$, we define its support by $\mathrm{support}(\mu) = \{\, x \in X \mid \mu(x) > 0 \,\}$.

▶ **Definition 9.** Let $\mu, \nu \in \mathit{Distr}(X)$. The set $\Omega(\mu,\nu)$ of *couplings* of μ and ν is defined by

$$\Omega(\mu,\nu) = \left\{ \omega \in \mathit{Distr}(X \times X) \;\middle|\; \sum_{x \in X} \omega(x,y) = \mu(y) \text{ and } \sum_{y \in X} \omega(x,y) = \nu(x) \right\}.$$

In general, the set $\Omega(\mu,\nu)$ is infinite. The set of vertices of the convex polytope $\Omega(\mu,\nu)$ is denoted by $V(\Omega(\mu,\nu))$. The latter set is finite (see, for example, [23, page 259]). This fact will be crucial in the proof of Lemma 20. The Kantorovich metric [22] is defined as follows.

▶ **Definition 10.** The function $K : [0,1]^{X \times X} \to [0,1]^{\mathit{Distr}(X) \times \mathit{Distr}(X)}$ is defined by

$$K(d)(\mu,\nu) = \min_{\omega \in V(\Omega(\mu,\nu))} \sum_{u,v \in S} \omega(u,v)\, d(u,v).$$

The Hausdorff metric and the Kantorovich metric are key ingredients of the definition of the probabilistic bisimilarity distances, as we will see in the next section.

3 Probabilistic Automata

Also in this section, we recall some definitions and results from the literature. In particular, we introduce the model of interest, probabilistic automata, its best known behavioural equivalence, probabilistic bisimilarity, and its quantitative generalization. Probabilistic automata were first studied in the context of concurrency by Segala [27].

▶ **Definition 11.** A *probabilistic automaton* is a tuple $\langle S, L, \to, \ell \rangle$ consisting of

- a nonempty finite set S of *states*,
- a nonempty finite set L of *labels*,
- a finitely branching *transition relation* $\to \subseteq S \times \mathit{Distr}(S)$, and
- a *labelling function* $\ell : S \to L$.

Instead of $(s,\mu) \in \to$, we write $s \to \mu$. A transition relation is finitely branching if for all $s \in S$, the set $\{\, \mu \in \mathit{Distr}(S) \mid s \to \mu \,\}$ is *nonempty* and finite. For the remainder of this paper we fix a probabilistic automaton $\langle S, L, \to, \ell \rangle$.

In order to define probabilistic bisimilarity, we first show how a relation on states can be lifted to a relation on distributions over states. This notion of lifting is due to Jonsson and Larsen [21].

▶ **Definition 12.** The *lifting* of a relation $\mathcal{R} \subseteq S \times S$ is the relation $\mathcal{R}\!\uparrow \subseteq \mathit{Distr}(S) \times \mathit{Distr}(S)$ defined by $(\mu,\nu) \in \mathcal{R}\!\uparrow$ if there exists $\omega \in V(\Omega(\mu,\nu))$ such that $\mathrm{support}(\omega) \subseteq \mathcal{R}$.

Probabilistic bisimilarity, a notion due to Segala and Lynch [28], is introduced next. States are probabilistic bisimilar if they have the same label and each probabilistic transition of the one state can be matched by a probabilistic transition of the other state, and vice versa. Two probabilistic transitions match if they transition with *exactly* the same probability to states that behave *exactly* the same.

▶ **Definition 13.** An equivalence relation $\mathcal{R} \subseteq S \times S$ is a *probabilistic bisimulation* if for all $s, t \in S$, if $(s,t) \in \mathcal{R}$ then

- $\ell(s) = \ell(t)$,

- for all $s \to \mu$ there exists $t \to \nu$ such that $(\mu, \nu) \in \mathcal{R}\uparrow$ and
- for all $t \to \nu$ there exists $s \to \mu$ such that $(\nu, \mu) \in \mathcal{R}\uparrow$.

Probabilistic bisimilarity, denoted $\sim$, is the largest probabilistic bisimulation.

For a proof that a largest probabilistic bisimulation exists, we refer the reader to, for example, [6, Proposition 4.3]. Relying on exact matching is the cause for a lack of robustness. To address this shortcoming, we define a quantitative generalization of probabilistic bisimilarity, the probabilistic bisimilarity distances, as the least fixed point of the function Δ_1. To prove an alternative characterization of the probabilistic bisimilarity distances in the next section, we also introduce a family of discounted versions of Δ_1, namely Δ_c with $c \in (0, 1)$.

▶ **Definition 14.** Let $c \in (0, 1]$. The function $\Delta_c : [0, 1]^{S \times S} \to [0, 1]^{S \times S}$ is defined by

$$\Delta_c(d)(s,t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ c\, H(K(d))(\{\, \mu \mid s \to \mu \,\}, \{\, \nu \mid t \to \nu \,\}) & \text{otherwise.} \end{cases}$$

▶ **Proposition 15.** For all $c \in (0, 1]$, the function Δ_c is monotone.

Proof. See [12, Lemma 2.10]. ◀

Since $\langle [0, 1]^{S \times S}, \sqsubseteq \rangle$ is a complete lattice according to Proposition 2 and Δ_c is a monotone function by Proposition 15, we can conclude from Theorem 3(c) that Δ_c has a least fixed point $\boldsymbol{\mu}\Delta_c$. The fact that the probabilistic bisimilarity distances $\boldsymbol{\mu}\Delta_1$ provide a quantitative generalization of probabilistic bisimilarity is captured by the following theorem due to Deng et al. [12].

▶ **Theorem 16.** *For all s, $t \in S$, $\boldsymbol{\mu}\Delta_1(s, t) = 0$ if and only if $s \sim t$.*

Proof. See [12, Corollary 2.14]. ◀

4 An Alternative Characterization

In the previous section, we defined the probabilistic bisimilarity distances as a least fixed point. Next, we present an alternative characterization. This generalizes the characterization of probabilistic bisimilarity distances for labelled Markov chains due to Chen et al. [9, Theorem 8]. First, we partition the set of state pairs as follows.

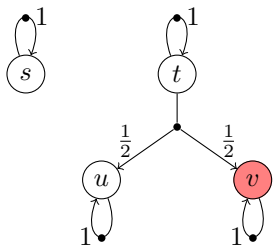
$$\begin{aligned} S_0^2 &= \{\, (s,t) \in S \times S \mid s \sim t \,\} \\ S_1^2 &= \{\, (s,t) \in S \times S \mid \ell(s) \neq \ell(t) \,\} \\ S_?^2 &= (S \times S) \setminus (S_0^2 \cup S_1^2) \end{aligned}$$

Note that, due to Theorem 16 the state pairs in S_0^2 have distance zero. From Definition 14 we can infer that the state pairs in S_1^2 have distance one. The state pairs in $S_?^2$ cannot have distance zero, again due to Theorem 16, but can have any distance in the interval $(0, 1]$, including distance one.

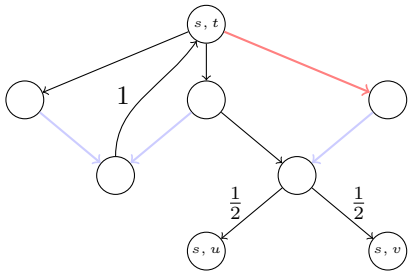
The characterization can be viewed as a two player game, a max player and a min player, similar to the one presented in [8]. The game can be considered a quantitative generalization of the game that characterizes bisimilarity (see [30]). In this turn based game, starting in a pair of states (s, t), the max player chooses a probabilistic transition from either s or t. Subsequently, the min player chooses a probabilistic transition from the other state and also chooses a coupling. For example, if the max player picks $s \to \mu$ and the min player picks $t \to \nu$, then the min player also has to choose $\omega \in V(\Omega(\mu, \nu))$. This will be formalized in

Definition 17. Recall that such a coupling ω is a probability distribution on $S \times S$. From a coupling ω the game moves to state pair (u, v) with probability $\omega(u, v)$.

Consider, for example, the following probabilistic automaton.



Note that the states s and u are probabilistic bisimilar. The corresponding game graph can be depicted as follows.



Since the game will be used to characterize the probabilistic bisimilarity distances, the state pairs for which we can easily determine their distance have no outgoing edges in the game graph. In particular, state pairs with different labels, which have distance one, and state pairs that are probabilistic bisimilar, which have distance zero, have no outgoing edges.

The objective of the max player is to maximize the expectation of reaching a state pair with different labels. The min player tries to minimize this expectation. In the above example, the max player tries to reach the state pair (s, v), whereas the min player tries to avoid that from happening. The policies, also known as strategies, for the max and min player are introduced next.

▶ **Definition 17.** The set $\mathcal{A}$ of *max policies* is defined by

$$\mathcal{A} = \left\{ A \in (S_?^2 \to (S \times \mathit{Distr}(S))) \;\middle|\; \begin{array}{l} \forall (s,t) \in S_?^2 : \\ (\exists \nu \in \mathit{Distr}(S) : A(s,t) = (s,\nu) \wedge t \to \nu) \vee \\ (\exists \mu \in \mathit{Distr}(S) : A(s,t) = (t,\mu) \wedge s \to \mu) \end{array} \right\}.$$

The set $\mathcal{I}$ of *min policies* is defined by

$$\mathcal{I} = \left\{ I \in ((S \times \mathit{Distr}(S)) \to \mathit{Distr}(S \times S)) \;\middle|\; \begin{array}{l} \forall (s,\nu) \in S \times \mathit{Distr}(S) : \exists \mu \in \mathit{Distr}(S) : \\ I(s,\nu) \in V(\Omega(\mu,\nu)) \wedge s \to \mu \end{array} \right\}.$$

Given a policy A for the max player and a policy I for the min player, we define the value function as the least fixed point of the function $\Gamma_1^{A,I}$. This least fixed point captures the expectation of reaching a state pair with different labels if both players use the given policies. We also introduce a family of discounted versions of $\Gamma_1^{A,I}$, namely $\Gamma_c^{A,I}$ with $c \in (0, 1)$, that we will use later in this section.

▶ **Definition 18.** Let $A \in \mathcal{A}$, $I \in \mathcal{I}$ and $c \in (0,1]$. The function $\Gamma_c^{A,I} : [0,1]^{S\times S} \to [0,1]^{S\times S}$ is defined by

$$\Gamma_c^{A,I}(d)(s,t) = \begin{cases} 0 & \text{if } (s,t) \in S_0^2 \\ 1 & \text{if } (s,t) \in S_1^2 \\ c \sum\limits_{u,v\in S} I(A(s,t))(u,v)\, d(u,v) & \text{otherwise.} \end{cases}$$

▶ **Proposition 19.** For all $A \in \mathcal{A}$, $I \in \mathcal{I}$ and $c \in (0,1]$, the function $\Gamma_c^{A,I}$ is monotone and c-Lipschitz.

From Theorem 3(c) we can conclude that $\Gamma_c^{A,I}$ has a least fixed point, which we denote by $\boldsymbol{\mu}\Gamma_c^{A,I}$. In the remainder of this section we will show that there exist an optimal max policy A^* and an optimal min policy I^* such that the corresponding value function captures the probabilistic bisimilarity distances. In the above game graph, the red edge represents the optimal max policy and the blue edges represent the optimal min policy. The proof of $\boldsymbol{\mu}\Delta_1 = \boldsymbol{\mu}\Gamma_1^{A^*,I^*}$ consists of two parts. First, we prove that there exists an optimal min policy.

▶ **Lemma 20.** $\exists I \in \mathcal{I} : \forall A \in \mathcal{A} : \boldsymbol{\mu}\Gamma_1^{A,I} \sqsubseteq \boldsymbol{\mu}\Delta_1$.

Proof. Towards the construction of $I^* \in \mathcal{I}$, let $s \in S$ and $\nu \in \mathit{Distr}(S)$. Since we restrict our attention to finitely branching probabilistic automata,

$$\mu_{s,\nu} = \operatorname*{argmin}_{s\to\mu} K(\boldsymbol{\mu}\Delta_1)(\mu,\nu) \tag{1}$$

exists. Because the set $V(\Omega(\mu_{s,\nu},\nu))$ is nonempty and finite, we can define

$$I^*(s,\nu) = \operatorname*{argmin}_{\omega\in V(\Omega(\mu_{s,\nu},\nu))} \sum_{u,v\in S} \omega(u,v)\, \boldsymbol{\mu}\Delta_1(u,v). \tag{2}$$

By construction $I^* \in \mathcal{I}$.

Let $A \in \mathcal{A}$. Since $\boldsymbol{\mu}\Gamma_1^{A,I^*}$ is the least pre-fixed point of Γ_1^{A,I^*} according to Theorem 3(d), to conclude that $\boldsymbol{\mu}\Gamma_1^{A,I^*} \sqsubseteq \boldsymbol{\mu}\Delta_1$ it suffices to show that $\boldsymbol{\mu}\Delta_1$ is a pre-fixed point of Γ_1^{A,I^*}, that is, $\Gamma_1^{A,I^*}(\boldsymbol{\mu}\Delta_1) \sqsubseteq \boldsymbol{\mu}\Delta_1$. Let $s,t \in S$. We distinguish three cases.

- If $(s,t) \in S_0^2$, then

$$\begin{aligned} \Gamma_1^{A,I^*}(\boldsymbol{\mu}\Delta_1)(s,t) &= 0 \\ &= \boldsymbol{\mu}\Delta_1(s,t) \qquad \text{[Theorem 16]} \end{aligned}$$

- If $(s,t) \in S_1^2$, then

$$\begin{aligned} \Gamma_1^{A,I^*}(\boldsymbol{\mu}\Delta_1)(s,t) &= 1 \\ &= \Delta_1(\boldsymbol{\mu}\Delta_1)(s,t) \\ &= \boldsymbol{\mu}\Delta_1(s,t). \end{aligned}$$

- Otherwise, $(s,t) \in S_?^2$. Without any loss of generality, we assume that $A(s,t) = (s,\nu)$

with $t \to \nu$. Then

$$\begin{aligned}
\Gamma_1^{A,I^*}(\boldsymbol{\mu}\Delta_1)(s,t) &= \sum_{u,v\in S} I^*(A(s,t))(u,v)\,\boldsymbol{\mu}\Delta_1(u,v) \\
&= \sum_{u,v\in S} I^*(s,\nu)(u,v)\,\boldsymbol{\mu}\Delta_1(u,v) \qquad [A(s,t)=(s,\nu)] \\
&= \min_{\omega\in V(\Omega(\mu_{s,\nu},\nu))} \sum_{u,v\in S} \omega(u,v)\,\boldsymbol{\mu}\Delta_1(u,v) \qquad [(2)] \\
&= K(\boldsymbol{\mu}\Delta_1)(\mu_{s,\nu},\nu) \\
&= \min_{s\to\mu} K(\boldsymbol{\mu}\Delta_1)(\mu,\nu) \qquad [(1)] \\
&\leq \max_{t\to\nu}\min_{s\to\mu} K(\boldsymbol{\mu}\Delta_1)(\mu,\nu) \\
&\leq H(K(\boldsymbol{\mu}\Delta_1))(\{\,\mu \mid s\to\mu\,\},\{\,\nu \mid t\to\nu\,\}) \\
&= \Delta_1(\boldsymbol{\mu}\Delta_1)(s,t) \\
&= \boldsymbol{\mu}\Delta_1(s,t).
\end{aligned}$$

◀

In the remainder of this paper, we denote the optimal min policy constructed in the above proof by I^*. It remains to prove that there exists an optimal max policy. The proof of this second part turns out to be more involved than the proof of the first part contained in above lemma. The proof has the following three major components.

- For all $A \in \mathcal{A}$ and $I \in \mathcal{I}$, the value function $\boldsymbol{\mu}\Gamma_1^{A,I}$ is the limit of the discounted value functions $\boldsymbol{\mu}\Gamma_c^{A,I}$. This result is inspired by [14, Theorem 4.4.1].
- Similarly, the probabilistic bisimilarity distances captured by $\boldsymbol{\mu}\Delta_1$ are the limit of their discounted counterparts represented by $\boldsymbol{\mu}\Delta_c$.
- There exists an optimal max policy in the discounted setting.

Combining the above three components, we arrive at an optimal max policy. The first two components are formalized next.

▶ **Proposition 21.** For all $A \in \mathcal{A}$ and $I \in \mathcal{I}$, $\lim_{c\uparrow 1} \boldsymbol{\mu}\Gamma_c^{A,I} = \boldsymbol{\mu}\Gamma_1^{A,I}$ and $\lim_{c\uparrow 1} \boldsymbol{\mu}\Delta_c = \boldsymbol{\mu}\Delta_1$.

The major component of the proof consists of showing that there exists an optimal max policy in the discounted setting.

▶ **Proposition 22.** For all $c \in (0,1)$, $\exists A \in \mathcal{A} : \forall I \in \mathcal{I} : \boldsymbol{\mu}\Delta_c \sqsubseteq \boldsymbol{\mu}\Gamma_c^{A,I}$.

Proof. Let $c \in (0,1)$. Let s, $t \in S$. If

$$\max_{s\to\mu}\min_{t\to\nu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu) \geq \max_{t\to\nu}\min_{s\to\mu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu) \tag{3}$$

then we define $A_c^*(s,t)$ by

$$A_c^*(s,t) = \left(t, \operatorname*{argmax}_{s\to\mu}\min_{t\to\nu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu)\right).$$

Because the probabilistic automaton is finitely branching, the above exists. Otherwise, we define $A_c^*(s,t)$ by

$$A_c^*(s,t) = \left(s, \operatorname*{argmax}_{t\to\nu}\min_{s\to\mu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu)\right).$$

By construction, $A_c^* \in \mathcal{A}$.

Let $I \in \mathcal{I}$. Since $\langle [0,1]^{S\times S}, \|\cdot - \cdot\|\rangle$ is a nonempty complete metric space according to Proposition 6 and the function $\Gamma_c^{A_c^*,I}$ is contractive by Proposition 19, we can conclude from Theorem 7 that $\Gamma_c^{A_c^*,I}$ has a unique fixed point. Therefore, $\boldsymbol{\mu}\Gamma_c^{A_c^*,I}$ is not only the least fixed point but also the greatest fixed point of $\Gamma_c^{A_c^*,I}$. According to Theorem 3(b), $\boldsymbol{\mu}\Gamma_c^{A_c^*,I}$ is the greatest post-fixed point of $\Gamma_c^{A_c^*,I}$. Hence, to conclude that $\boldsymbol{\mu}\Delta_c \sqsubseteq \boldsymbol{\mu}\Gamma_c^{A_c^*,I}$ it suffices to show that $\boldsymbol{\mu}\Delta_c$ is a post-fixed point of $\Gamma_c^{A_c^*,I}$, that is, $\boldsymbol{\mu}\Delta_c \sqsubseteq \Gamma_c^{A_c^*,I}(\boldsymbol{\mu}\Delta_c)$. Let s, $t \in S$. We distinguish three cases.

- If $(s,t) \in S_0^2$, then

$$\begin{aligned}\boldsymbol{\mu}\Delta_c(s,t) &\le \boldsymbol{\mu}\Delta_1(s,t)\\ &= 0 \qquad \text{[Theorem 16]}\\ &= \Gamma_c^{A_c^*,I}(\boldsymbol{\mu}\Delta_c)(s,t).\end{aligned}$$

- If $(s,t) \in S_1^2$, then

$$\begin{aligned}\boldsymbol{\mu}\Delta_c(s,t) &= \Delta_c(\boldsymbol{\mu}\Delta_c)(s,t)\\ &= 1\\ &= \Gamma_c^{A_c^*,I}(\boldsymbol{\mu}\Delta_c)(s,t).\end{aligned}$$

- Otherwise, $(s,t) \in S_?^2$. Without loss of any generality, assume that $A_c^*(s,t) = (t,\mu)$. This assumption implies that (3) and

$$\Delta_1(\boldsymbol{\mu}\Delta_c)(s,t) = \min_{t\to\nu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu). \tag{4}$$

Hence,

$$\begin{aligned}\boldsymbol{\mu}\Delta_c(s,t) &= \Delta_c(\boldsymbol{\mu}\Delta_c)(s,t)\\ &= c\,\Delta_1(\boldsymbol{\mu}\Delta_c)(s,t)\\ &= c\min_{t\to\nu} K(\boldsymbol{\mu}\Delta_c)(\mu,\nu) \qquad [(4)]\\ &\le c\sum_{u,v\in S} I(A_c^*(s,t))(u,v)\,\boldsymbol{\mu}\Delta_c(u,v)\\ &= c\,\Gamma_1^{A_c^*,I}(\boldsymbol{\mu}\Delta_c)(s,t)\\ &= \Gamma_c^{A_c^*,I}(\boldsymbol{\mu}\Delta_c)(s,t).\end{aligned}$$

◀

Combining the above three components, we obtain the second part of the proof.

▶ **Lemma 23.** $\exists A \in \mathcal{A} : \forall I \in \mathcal{I} : \boldsymbol{\mu}\Delta_1 \sqsubseteq \boldsymbol{\mu}\Gamma_1^{A,I}$.

Proof. According to Proposition 22,

$$\forall n \in \mathbb{N} : \exists A_n \in \mathcal{A} : \forall I \in \mathcal{I} : \boldsymbol{\mu}\Delta_{\frac{n}{n+1}} \sqsubseteq \boldsymbol{\mu}\Gamma_{\frac{n}{n+1}}^{A_n,I}. \tag{5}$$

Since the set $\mathcal{A}$ is finite, the sequence $(A_n)_{n\in\mathbb{N}}$ has a subsequence $(A_{\sigma(n)})_{n\in\mathbb{N}}$ that is constant, that is, there exists $A^* \in \mathcal{A}$ such that for all $n \in \mathbb{N}$, $A_{\sigma(n)} = A^*$. From Proposition 21 we can conclude that

$$\lim_{n\in\mathbb{N}} \boldsymbol{\mu}\Delta_{\frac{\sigma(n)}{\sigma(n)+1}} = \boldsymbol{\mu}\Delta_1 \text{ and } \lim_{n\in\mathbb{N}} \boldsymbol{\mu}\Gamma^{A,I}_{\frac{\sigma(n)}{\sigma(n)+1}} = \boldsymbol{\mu}\Gamma_1^{A,I}.$$

From (5) we can deduce that $\forall I \in \mathcal{I} : \boldsymbol{\mu}\Delta_1 \sqsubseteq \boldsymbol{\mu}\Gamma_1^{A^*,I}$. ◀

In the remainder of this paper, we denote the optimal max policy that satisfies Lemma 23 by A^*. Combining Lemma 20 and 23, we arrive at the following alternative characterization of the probabilistic bisimilarity distances.

▶ **Theorem 24.** $\boldsymbol{\mu}\Delta_1 = \boldsymbol{\mu}\Gamma_1^{A^*,I^*}$.

5 Deciding Distance One

In this section, we present an algorithm to compute the set D_1 of state pairs that have distance one, that is

$$D_1 = \{\, (s,t) \in S \times S \mid \boldsymbol{\mu}\Delta_1(s,t) = 1 \,\}.$$

The key ingredient of our algorithm is the following function.

▶ **Definition 25.** The function $\Lambda : 2^{S\times S} \times 2^{S\times S} \to 2^{S\times S}$ is defined by

$$\Lambda(X,Y) = S_1^2 \cup \left\{ (s,t) \in S_?^2 \;\middle|\; \begin{array}{l} \exists s \to \mu : \forall t \to \nu : \forall \omega \in V(\Omega(\mu,\nu)) : \\ \qquad \mathrm{support}(\omega) \subseteq X \wedge \mathrm{support}(\omega) \cap Y \neq \emptyset \vee \\ \exists t \to \nu : \forall s \to \mu : \forall \omega \in V(\Omega(\mu,\nu)) : \\ \qquad \mathrm{support}(\omega) \subseteq X \wedge \mathrm{support}(\omega) \cap Y \neq \emptyset \end{array} \right\}.$$

The set $\Lambda(X,Y)$ contains all state pairs with different labels and those state pairs for which there exists a move by the max player so that every subsequent move of the min player always ends up in X and with some positive probability in Y. The function Λ has the following monotonicity properties.

▶ **Proposition 26.** For all X, Y, $Z \subseteq S \times S$ with $X \subseteq Y$,

(a) $\Lambda(Z,X) \subseteq \Lambda(Z,Y)$.

(b) $\boldsymbol{\mu}Z.\Lambda(X,Z) \subseteq \boldsymbol{\mu}Z.\Lambda(Y,Z)$.

Since $\langle 2^{S\times S}, \subseteq \rangle$ is a complete lattice and for each $X \subseteq S \times S$ the function $\boldsymbol{\lambda}Y.\Lambda(X,Y)$ is monotone, the least fixed point $\boldsymbol{\mu}Y.\Lambda(X,Y)$ exists according to Theorem 3(c). The set $\boldsymbol{\mu}Y.\Lambda(X,Y)$ contains all state pairs (s,t) for which there exists a max policy such that for all min policies, (s,t) can reach a state pair with different labels and all state pairs reachable from (s,t) are element of X.

Since the function $\boldsymbol{\lambda}X.\boldsymbol{\mu}Y.\Lambda(X,Y)$ is monotone as well, we can conclude from Theorem 3(a) that the greatest fixed point $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X,Y)$ exists. The set $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X,Y)$ contains all state pairs (s,t) for which there exists a max policy such that for all min policies, all state pairs reachable from (s,t) can reach a state pair with different labels. In the next section, we will prove that $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X,Y)$ captures the set D_1. According to Theorem 4(a) and (b), these greatest and least fixed points can be obtained iteratively as follows.

```
1  X_c = S × S
2  do
3     Y_c = ∅
4     do
5        Y_p = Y_c
6        Y_c = Λ(X_c, Y_p)
7     while Y_p ≠ Y_c
8     X_p = X_c
9     X_c = Y_c
10 while X_p ≠ X_c
```

The inner loop (line 3–7) computes the least fixed point $\boldsymbol{\mu}Y.\Lambda(X_c, Y)$. The outer loop (line 1–10) computes the greatest fixed point $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X, Y)$, which equals D_1 as we will prove in the next section. Due to the monotonicity of Λ we can conclude that both the inner and outer loop terminate after at most $|S|^2$ iterations. To conclude that the above algorithm is polynomial time, it remains to show that $\Lambda(X_c, Y_p)$ in line 6 can be computed in polynomial time.

▶ **Proposition 27.** For all μ, $\nu \in \mathit{Distr}(S)$ and $X \subseteq S \times S$,

$$\forall \omega \in V(\Omega(\mu,\nu)) : \text{support}(\omega) \subseteq X \text{ if and only if } K(d)(\mu,\nu) = 1$$

and

$$\forall \omega \in V(\Omega(\mu,\nu)) : \text{support}(\omega) \cap X \neq \emptyset \text{ if and only if } K(d)(\mu,\nu) > 0$$

where

$$d(s,t) = \begin{cases} 1 & \text{if } (s,t) \in X \\ 0 & \text{otherwise.} \end{cases}$$

Computing $K(d)(\mu,\nu)$ boils down to solving a minimum cost network flow problem, where d captures the cost. This problem can be solved in polynomial time using, for example, Orlin's network simplex algorithm [26]. Hence, $\Lambda(X_c, Y_p)$ can be computed in polynomial time.

6 Correctness Proof

To conclude that the algorithm presented in the previous section is correct, it remains to show that $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X, Y)$ equals D_1. We start by providing an iterative characterization of $\boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X, Y)$.

▶ **Definition 28.** For each $i \in \mathbb{N}$, the set $X_i \subseteq S \times S$ is defined by

$$X_i = \begin{cases} S \times S & \text{if } i = 0 \\ \boldsymbol{\mu}Y.\Lambda(X_{i-1}, Y) & \text{otherwise.} \end{cases}$$

For each $i, j \in \mathbb{N}$, the set $Y_i^j \subseteq S \times S$ is defined by

$$Y_i^j = \begin{cases} D_1 & \text{if } j = 0 \\ \Lambda(X_i, Y_i^{j-1}) & \text{otherwise.} \end{cases}$$

The above definition differs from the iterative algorithm presented in the previous section in that $Y_i^0 = D_1$ whereas the algorithm starts its iteration towards the least fixed point from $\emptyset$.

▶ **Proposition 29.**
(a) $X_m = \boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X, Y)$ for some $m \in \mathbb{N}$.
(b) $Y_m^n = \boldsymbol{\mu}Y.\Lambda(X_m, Y)$ for some $n \in \mathbb{N}$.
(c) $X_m = Y_m^n$.

Proof sketch. Part (a) follows from Theorem 4(b) and Proposition 26(b). Part (b) can be proved as follows. First, we observe that

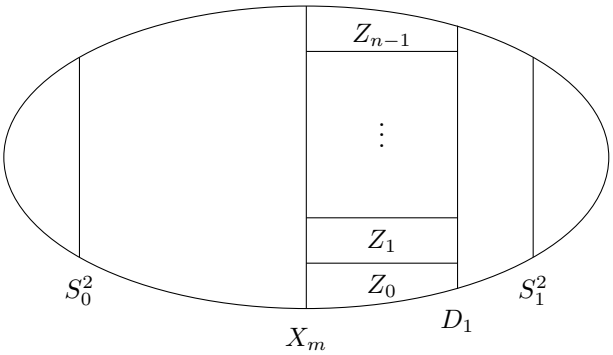
$$D_1 \subseteq \boldsymbol{\mu}Y.\Lambda(X_m, Y) = X_m \tag{6}$$

by part (a). The desired result follows from the latter fact and Theorem 4(c) and Proposition 26(a). Part (c) follows from part (a) and (b). ◀

From part (a) of the above proposition and (6) we can conclude that it suffices to prove $X_m \subseteq D_1$.

▶ **Definition 30.** For each $0 \leq i < n$, the set $Z_i \subseteq S \times S$ is defined by

$$Z_i = Y_m^{i+1} \setminus Y_m^i.$$



▶ **Proposition 31.**
(a) For all $0 \leq i < n$, $Z_i \subseteq S_?^2$.
(b) For all $0 \leq i < j < n$, $Z_i \cap Z_j = \emptyset$.
(c) $\bigcup_{0 \leq i < n} Z_i = X_m \setminus D_1$.
(d) For all $0 \leq i \leq n$, $Y_m^i = D_1 \cup \bigcup_{0 \leq j < i} Z_j$.

According to Proposition 31(b) and (c), the sets $Z_0, \ldots, Z_{n-1}$ form a partition of $X_m \setminus D_1$.

▶ **Proposition 32.** For all $0 \leq i < n$ and $(s,t) \in Z_i$,

$$\exists s \to \mu : \forall t \to \nu : \forall \omega \in V(\Omega(\mu,\nu)) : \text{support}(\omega) \subseteq X_m \wedge \text{support}(\omega) \cap Y_m^i \neq \emptyset \vee \tag{7}$$

$$\exists t \to \nu : \forall s \to \mu : \forall \omega \in V(\Omega(\nu,\mu)) : \text{support}(\omega) \subseteq X_m \wedge \text{support}(\omega) \cap Y_m^i \neq \emptyset \tag{8}$$

Based on the above proposition, we construct a max policy A'.

▶ **Definition 33.** The function $A' : S_?^2 \to (S \times \mathit{Distr}(S))$ is defined by

$$A'(s,t) = \begin{cases} (t,\mu) & \text{if } (s,t) \in Z_i \text{ and (7)} \\ (s,\nu) & \text{if } (s,t) \in Z_i \text{ and (8)} \\ A^*(s,t) & \text{if } (s,t) \in S_?^2 \setminus (X_m \setminus D_1). \end{cases}$$

Given the max policy A' and an arbitrary min policy I, from Proposition 31(d) and 32 we can conclude that each state pair in Z_i can reach a state pair in D_1 or Z_j with $j < i$. Consequently, each state pair in Z_i can reach a state pair in D_1. Given the max policy A' and the optimal min policy I^*, we define the function Ψ as follows.

▶ **Definition 34.** The function $\Psi : [0,1]^{S \times S} \to [0,1]^{S \times S}$ is defined by

$$\Psi(d)(s,t) = \begin{cases} \Gamma_1^{A',I^*}(d)(s,t) & \text{if } (s,t) \in X_m \\ 0 & \text{otherwise} \end{cases}$$

▶ **Proposition 35.** The function Ψ is monotone.

Since $\langle [0,1]^{S \times S}, \sqsubseteq \rangle$ is a complete lattice and Ψ is monotone, Ψ has a greatest fixed point $\boldsymbol{\nu}\Psi$ and a least fixed point $\boldsymbol{\mu}\Psi$ by Theorem 3(a) and (c). Next, we will show that Ψ has a unique fixed point.

▶ **Proposition 36.** Ψ has a unique fixed point.

Proof sketch. It is sufficient to prove that $\boldsymbol{\mu}\Psi = \boldsymbol{\nu}\Psi$. Let

$$m = \max\{\, \boldsymbol{\nu}\Psi(s,t) - \boldsymbol{\mu}\Psi(s,t) \mid (s,t) \in S \times S \,\}$$
$$M = \{\, (s,t) \in S \times S \mid \boldsymbol{\nu}\Psi(s,t) - \boldsymbol{\mu}\Psi(s,t) = m \,\}$$

We can show that $m = 0$ and, hence, we can conclude that $\boldsymbol{\mu}\Psi = \boldsymbol{\nu}\Psi$. ◀

From the fact that Ψ has a unique fixed point and the alternative characterization of the probabilistic bisimilarity distances presented in the previous section, we can infer the main result of this section.

▶ **Theorem 37.** $D_1 = \boldsymbol{\nu}X.\boldsymbol{\mu}Y.\Lambda(X,Y)$.

Proof sketch. We can show that the function $d \in S \times S \to [0,1]$ defined by

$$d(s,t) = \begin{cases} 1 & \text{if } (s,t) \in X_m \\ 0 & \text{otherwise} \end{cases}$$

is a fixed point of Ψ. Let $(s,t) \in X_m$. Then

$$\begin{aligned} \boldsymbol{\mu}\Delta_1(s,t) &\geq \boldsymbol{\mu}\Gamma_1^{A',I^*}(s,t) && \text{[Lemma 20]} \\ &= \boldsymbol{\mu}\Psi(s,t) && [(s,t) \in X_m] \\ &= d(s,t) && [d \text{ is a fixed point of } \Psi \text{ and Proposition 36}] \\ &= 1 && [(s,t) \in X_m] \end{aligned}$$

Hence, $(s,t) \in D_1$. Therefore, $X_m \subseteq D_1$. According to (6), $D_1 \subseteq X_m$. Thus, $X_m = D_1$. Proposition 29(a) completes the proof. ◀

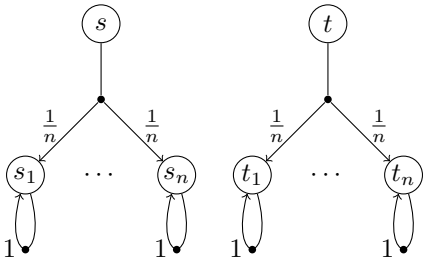
7 Conclusion

Chen et al. [9] have provided an alternative characterization of the probabilistic bisimilarity distances for labelled Markov chains. This characterization forms the basis for the algorithm to compute the probabilistic bisimilarity distances for labelled Markov chains by Bacci et al. [1]. Their algorithm is similar to Howard's policy iteration algorithm [20]. In this paper we have presented an alternative characterization of the probabilistic bisimilarity distances for probabilistic automata. In future work, we plan to use this characterization as the foundation for an algorithm to compute the probabilistic bisimilarity distances for probabilistic automata based on the policy iteration algorithm due to Hoffman and Karp [19].

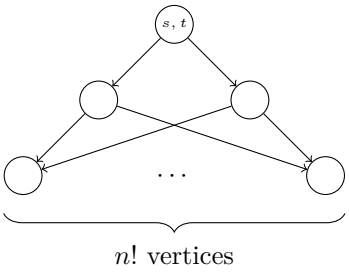
As shown by Baier [2], probabilistic bisimilarity distance zero for probabilistic automata can be decided in polynomial time. In this paper we have shown that distance one can also be decided in polynomial time. As a consequence, we can determine in polynomial time how many, if any, distances are non-trivial, that is, greater than zero and smaller than one. As we have already shown in [32] in the context of labelled Markov chains, being able to decide distance zero and distance one in polynomial time has significant impact on computing probabilistic bisimilarity distances for labelled Markov chains. The algorithm by Bacci et al. [1], that does not decide distance one before computing the non-trivial distances using policy iteration, can compute distances for labelled Markov chains up to 150 states. For one such labelled Markov chain, their algorithm takes more than 49 hours. Our algorithm that we present in [32] decides distance zero and distance one before using policy iteration to

compute the non-trivial distances. Our algorithm takes 13 milliseconds instead of 49 hours. Furthermore, our algorithm can compute distances for labelled Markov chains with more than 10,000 states in less than 50 minutes.

Consider the following probabilistic automaton.



This probabilistic automaton induces the following game graph.



If μ and ν are both the uniform distribution on n elements, then the vertices of $\Omega(\mu, \nu)$ can be viewed as permutations (see, for example, [29, Theorem 8.4]). As a result, from the state pair (s, t) after one move by the max player and one move by the min player, $n!$ vertices can be reached. Hence, we may encounter an exponential blow-up when we transform a probabilistic automaton into a game. As a consequence, it is not immediately obvious which results from game theory can be transferred to our setting. We leave this for future research.

To prove Lemma 23, which provides the second part of the proof of the alternative characterization of the probabilistic bisimilarity distances, we rely on the discounted functions Δ_c and $\Gamma_c^{A_c^*, I}$ for $c \in (0, 1)$. In particular, in the proof of Proposition 22 we use the fact that $\Gamma_c^{A_c^*, I}$ has a unique fixed point. If we were able to prove that $\Gamma_1^{A^*, I}$ has a unique fixed point, then we would be able to give a proof of Lemma 23 that does not rely on discounted functions. We also leave that for future research.

References

1. Giorgio Bacci, Giovanni Bacci, Kim Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In Nir Piterman and Scott Smolka, editors, *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 1–15, Rome, Italy, 2013. Springer-Verlag.
2. Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and Thomas Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 50–61, New Brunswick, NJ, USA, 1996. Springer-Verlag.
3. Jaco de Bakker and Erik de Vink. *Control flow semantics*. MIT Press, Cambridge, MA, USA, 1996.

4 Jaco de Bakker and Jeffery Zucker. Denotational semantics of concurrency. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 153–158, San Francisco, 1982. ACM.

5 Stefan Banach. Sur les opérations dans les ensembles abstraits et leurs applications aux equations intégrales. *Fundamenta Mathematicae*, 3:133–181, 1922.

6 Franck van Breugel. Probabilistic bisimilarity distances. *SIGLOG News*, 4(4):33–51, 2017.

7 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic systems. In Fernando Orejas, Paul Spirakis, and Jan van Leeuwen, editors, *Proceedings of 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432, Crete, 2001. Springer-Verlag.

8 Franck van Breugel and James Worrell. The complexity of computing a bisimilarity pseudometric on probabilistic automata. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind – A Tribute to Prakash Panangaden*, volume 8464 of *Lecture Notes in Computer Science*, pages 191–213. Springer-Verlag, 2014.

9 Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In Lars Birkedal, editor, *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451, Tallinn, Estonia, 2012. Springer-Verlag.

10 Edmund Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

11 Brian Davey and Hilary Priestley. *Introduction to lattices and order*. Cambridge University Press, Cambridge, United Kingdom, 2002.

12 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. In Antonio Cerone and Herbert Wiklicky, editors, *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages*, volume 153(2) of *Electronic Notes in Theoretical Computer Science*, pages 79–96, Edinburgh, Scotland, 2005. Elsevier.

13 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422, Copenhagen, Denmark, 2002. IEEE.

14 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer-Verlag, New York, NY, USA, 1997.

15 Hongfei Fu. Computing game metrics on Markov decision processes. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 227–238, Warwick, UK, 2012. Springer-Verlag.

16 Hongfei Fu. Personal communication, 2013.

17 Alessandro Giacalone, Chi-Chang Jou, and Scott 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, Sea of Gallilee, Israel, 1990. North-Holland.

18 Felix Hausdorff. *Grundzüge der Mengenlehre*. Von Veit & Comp., Leipzig, 1914.

19 Alan Hoffman and Richard Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

20 Ronald Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, MA, USA, 1960.

21 Bengt Jonsson and Kim Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th Annual Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, The Netherlands, 1991. IEEE.

22 Leonid Kantorovich. On the transfer of masses (in Russian). *Doklady Akademii Nauk*, 5(1):1–4, 1942. Translated in *Management Science*, 5(1):1–4, 1958.

23 Viktor Klee and Christoph Witzgall. Facets and vertices of transportation polytopes. In George Dantzig and Arthur Veinott, editors, *Proceedings of 5th Summer Seminar on the Mathematis of the Decision Sciences*, volume 11 of *Lectures in Applied Mathematics*, pages 257–282, Stanford, CA, USA, 1967. AMS.

24 Bronisław Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématique*, 6:133–134, 1928.

25 Kim Larsen and Arne Skou. Bisimulation through probabilistic testing. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 344–352, Austin, TX, USA, 1989. ACM.

26 James Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, 1997.

27 Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

28 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the 5th International Conference on Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496, Uppsala, Sweden, 1994. Springer-Verlag.

29 Denis Serre. *Matrices: theory and applications*. Springer-Verlag, New York, NY, USA, 2010.

30 Colin Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of the IGPL*, 7(1):103–124, 1999.

31 Qiyi Tang and Franck van Breugel. Computing probabilistic bisimilarity distances via policy iteration. In Josée Desharnais and Radha Jagadeesan, editors, *Proceedings of the 27th International Conference on Concurrency Theory*, volume 59 of *Leibniz International Proceedings in Informatics*, pages 22:1–22:15, Quebec City, QC, Canada, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

32 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for labelled Markov chains. In Hana Chockler and Georg Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification*, volume 10981 of *Lecture Notes in Computer Science*, pages 681–699, Oxford, UK, 2018. Springer-Verlag.

33 Alfred Tarski. A lattice-theoretic fixed point theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

34 Mathieu Tracol, Josée Desharnais, and Abir Zhioua. Computing distances between probabilistic automata. In Mieke Massink and Gethin Norman, editors, *Proceedings 9th Workshop on Quantitative Aspects of Programming Languages*, volume 57 of *Electronic Proceedings in Theoretical Computer Science*, pages 148–162, Saarbrücken, Germany, 2011. Elsevier.

Non-deterministic Weighted Automata on Random Words

Jakub Michaliszyn
University of Wrocław

Jan Otop
University of Wrocław

Abstract

We present the first study of non-deterministic weighted automata under probabilistic semantics. In this semantics words are random events, generated by a Markov chain, and functions computed by weighted automata are random variables. We consider the probabilistic questions of computing the expected value and the cumulative distribution for such random variables.

The exact answers to the probabilistic questions for non-deterministic automata can be irrational and are uncomputable in general. To overcome this limitation, we propose an approximation algorithm for the probabilistic questions, which works in exponential time in the automaton and polynomial time in the Markov chain. We apply this result to show that non-deterministic automata can be effectively determinised with respect to the standard deviation metric.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects, Theory of computation → Quantitative automata, Mathematics of computing → Markov networks

Keywords and phrases quantitative verification, weighted automata, expected value

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.10

Acknowledgements This work was supported by the National Science Centre (NCN), Poland under grant 2014/15/D/ST6/04543. We would like to thank anonymous reviewers for their valuable comments on this paper. Our special thanks go to Günter Rote who pointed out a blooper in an earlier version of our running example.

1 Introduction

Weighted automata are finite automata in which transitions carry weights [13]. We study here weighted automata (on finite and infinite words) whose semantics is given by *value functions* (such as sum or average) [8]. In such a weighted automaton transitions are labeled with rational numbers and hence every run yields a sequence of rationals, which the value function aggregates into a single (real) number. This number is the value of the run, and the value of a word is the infimum over values of all accepting runs on that word.

The value function approach has been introduced to express quantitative system properties (performance, energy consumption, etc.) and it serves as a foundation for *quantitative verification* [8, 18]. Basic decision questions for weighted automata are quantitative counterparts of the emptiness and universality questions obtained by imposing a threshold on the values of words.

Probabilistic semantics. The emptiness and the universality problems correspond to the best-case and the worst-case analysis. For the average-case analysis, weighted automata are considered under probabilistic semantics, in which words are random events generated by a

Markov chain [7, 9]. In this setting, functions from words to reals computed by deterministic weighted automata are measurable and hence can be considered as random variables. The fundamental probabilistic questions are to compute *the expected value* and *the cumulative distribution* for a given automaton and a Markov chain.

The deterministic case. Weighted automata under probabilistic semantics have been studied only in the deterministic case. In [7], a close relationship between weighted automata under probabilistic semantics and weighted Markov chains has been established. For a weighted automaton $\mathcal{A}$ and a Markov chain $\mathcal{M}$ representing the distribution over words, the probabilistic problems for $\mathcal{A}$ and $\mathcal{M}$ coincide with the probabilistic problem of the weighted Markov chain $\mathcal{A} \times \mathcal{M}$. Weighted Markov chains have been intensively studied with single and multiple quantitative objectives [3, 15, 24, 10]. The above reduction does not extend to non-deterministic weighted automata [9, Example 30].

Significance of nondeterminism. Non-deterministic weighted automata are provably more expressive than their deterministic counterpart [8]. Many important system properties can be expressed with weighted automata only in the nondeterministic setting. This includes minimal response time, minimal number of errors and the edit distance problem [18], which serves as the foundation for the *specification repair* framework from [5].

Non-determinism can also arise as a result of abstraction. The exact systems are often too large and complex to operate on and hence they are approximated with smaller non-deterministic models [11]. The abstraction is especially important for multi-threaded programs, where the explicit model grows exponentially with the number of threads [17].

Our contributions

We study non-deterministic weighted automata under probabilistic semantics. We work with weighted automata as defined in [8], where a value function f is used to aggregate weights along a run, and the value of the word is the infimum over values of all runs. (The infimum can be changed to supremum as both definitions are dual). We primarily focus on the two most interesting value functions: the sum of weights over finite runs, and the limit average over infinite runs. The main results presented in this paper are as follows.

- We show that the answers to the probabilistic questions for weighted automata with the sum and limit-average value functions can be irrational (Theorem 5) and cannot be computed by any effective representation (Theorem 6).
- We establish approximation algorithms for the probabilistic questions for weighted automata with the sum and limit-average value functions. The approximation is #P-complete for (total) weighted automata with the sum value function (Theorem 10), and it is PSpace-hard and solvable in exponential time for weighted automata with the limit-average value function (Theorem 16).
- We show that weighted automata with the limit-average value function can be approximately determinised (Theorem 18). Given an automaton $\mathcal{A}$ and $\epsilon > 0$, we show how to compute a deterministic automaton $\mathcal{A}_D$ such that the expected difference between the values returned by both automata is at most ϵ.

Applications

We briefly discuss applications of our contributions in quantitative verification.

- The expected-value question corresponds to the average-case analysis in quantitative verification [7, 9]. Using results from this paper, we can perform the average-case analysis with respect to quantitative specifications given by non-deterministic weighted automata.
- The universality problem for non-deterministic automata, which asks whether all words have the value below a given threshold, forms a basis for some quantitative-model-checking frameworks [8]. Unfortunately, the universality problem is undecidable for weighted automata with the sum or the limit average values functions. The distribution question can be considered as a computationally-attractive variant of universality, i.e., we ask whether almost-all words have value below some given threshold. We show that if the threshold can be approximated, the distribution question can be computed effectively.
- Weighted automata have been used to formally study online algorithms [2]. Online algorithms have been modeled by deterministic weighted automata, which make choices based solely on the past, while offline algorithms have been modeled by non-deterministic weighted automata. Relating deterministic and non-deterministic models allowed for formal verification of the worst-case competitiveness ratio of online algorithms. Using the result from our paper, we can extend the analysis from [2] to the average-case competitiveness.

Related work

Probabilistic verification of qualitative properties. Probabilistic verification asks for the probability of the sets of traces satisfying a given property. For non-weighted automata, it has been extensively studied [26, 12, 3] and implemented [22, 19]. The prevalent approach in this area is to work with deterministic automata, and apply determinisation as needed. To obtain better complexity bounds, the probabilistic verification problem has been directly studied for unambiguous Büchi automata in [4]; the authors explain there the potential pitfalls in the probabilistic analysis of non-deterministic automata.

Weighted automata under probabilistic semantics. Probabilistic verification of weighted automata and their extensions has been studied in [9]. All automata considered there are deterministic.

Markov Decision Processes (MDPs). MDPs are a classical extension of Markov chains, which allow to model control in a stochastic environment [3, 15]. In MDPs probabilistic and non-deterministic transitions are interleaved. Intuitively, the non-determinism in MDPs is resolved based on the past, i.e., already generated events. In our setting, non-deterministic weighted automata work over completely generated words and hence non-determinism may be resolved based on following letters, considered as future events.

Approximation determinisation. As weighted automata are not determinisable, Boker and Henzinger [6] studied *approximate* determinisation defined as follows. The distance $d_{\sup}$ between weighted automata $\mathcal{A}_1, \mathcal{A}_2$ is defined as $d_{\sup}(\mathcal{A}_1, \mathcal{A}_2) = \sup_w |\mathcal{A}_1(w) - \mathcal{A}_2(w)|$. A nondeterministic weighted automaton $\mathcal{A}$ can be *approximately* determinised if for every $\epsilon > 0$ there exists a deterministic automaton $\mathcal{A}_D$ such that $d_{\sup}(\mathcal{A}, \mathcal{A}_D) \leq \epsilon$. Unfortunately, weighted automata with the limit average value function cannot be approximately determinised [6]. In this work we show that the approximate determinisation is possible for the standard deviation metric d_{std} defined as $d_{\text{std}}(\mathcal{A}_1, \mathcal{A}_2) = \mathbb{E}(|\mathcal{A}_1(w) - \mathcal{A}_2(w)|)$.

2 Preliminaries

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 other sequences defined later on. By $|w|$ we denote the length of w.

A *(non-deterministic) finite automaton* (NFA) is a tuple $(\Sigma, Q, Q_0, F, \delta)$ consisting of an input alphabet Σ , a finite set of states Q, a set of initial states $Q_0 \subseteq Q$, a set of final states F, and a finite transition relation $\delta \subseteq Q \times \Sigma \times Q$.

We denote by $\delta(q,a)$ the set of states $\{q' \mid \delta(q,a,q')\}$ and by $\delta(S,a)$ the set of states $\bigcup_{q\in S} \delta(q,a)$. We extend this to $\widehat{\delta}\colon 2^Q \times \Sigma^* \to 2^Q$ in the following way: $\widehat{\delta}(S,\epsilon) = S$ (where ϵ is the empty word) and $\widehat{\delta}(S,aw) = \widehat{\delta}(\delta(S,a),w)$, i.e., $\widehat{\delta}(S,w)$ is the set of states reachable from S via δ over the word w.

Weighted automata. A *weighted automaton* is a finite automaton whose transitions are labeled by rational numbers called *weights*. Formally, a weighted automaton is a tuple $(\Sigma, Q, Q_0, F, \delta, C)$, where the first five elements are as in the finite automata, and $C\colon \delta \to \mathbb{Q}$ is a function that defines *weights* of transitions. An example of a weighted automaton is depicted in Figure 1.

The size of a weighted automaton $\mathcal{A}$, denoted by $|\mathcal{A}|$, is $|Q| + |\delta| + \sum_{q,q',a} \mathrm{len}(C(q,a,q'))$, where len is the sum of the lengths of the binary representations of the numerator and the denominator of a given rational number.

A *run* π of an automaton $\mathcal{A}$ on a word w is a sequence of states $\pi[0]\pi[1]\dots$ such that $\pi[0]$ is an initial state and for each i we have $(\pi[i-1], w[i], \pi[i]) \in \delta$. A finite run π of length k is *accepting* if and only if the last state $\pi[k]$ belongs to the set of accepting states F. As in [8], we do not consider ω-accepting conditions and assume that all infinite runs are accepting. Every run π of an automaton $\mathcal{A}$ on a (finite or infinite) word w defines a sequence of weights of successive transitions of $\mathcal{A}$ as follows. Let $(C(\pi))[i]$ be the weight of the i-th transition, i.e., $C(\pi[i-1], w[i], \pi[i])$. Then, $C(\pi) = (C(\pi)[i])_{1\le i\le |w|}$. A *value functions* f is a function that assigns real numbers to sequences of rational numbers. The value $f(\pi)$ of the run π is defined as $f(C(\pi))$.

The value of a (non-empty) word w assigned by the automaton $\mathcal{A}$, denoted by $\mathcal{L}_\mathcal{A}(w)$, is the infimum of the set of values of all accepting runs on w. The value of a word that has no (accepting) runs is infinite. To indicate a particular value function f that defines the semantics, we will call a weighted automaton $\mathcal{A}$ an f-automaton.

Value functions. We consider the following value functions. For finite runs, functions Min and Max are defined in the usual manner, and the function Sum is defined as

$$\textsc{Sum}(\pi) = \sum_{i=1}^{|C(\pi)|} (C(\pi))[i]$$

For infinite runs we consider the supremum Sup and infimum Inf functions (defined like Max and Min but on infinite runs) and the limit average function LimAvg defined as

$$\textsc{LimAvg}(\pi) = \limsup_{k\to\infty} \textsc{Avg}(\pi[0,k])$$

where for finite runs π we have $\textsc{Avg}(\pi) = \textsc{Sum}(\pi)/|C(\pi)|$.

2.1 Probabilistic semantics

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 that $\sum_{a \in \Sigma, s' \in S} E(s,a,s') = 1$. By $|\mathcal{M}| = |S| + |E| + \sum_{q,q',a} \mathrm{len}(E(q,a,q'))$ we denote the size of the Markov chain $\mathcal{M}$. An example of a single-state Markov chain is depicted in Figure 1.

The probability of a finite word u w.r.t. a Markov chain $\mathcal{M}$, denoted $\mathbb{P}_{\mathcal{M}}(u)$, 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 (by Carathéodory's extension theorem [14]). We will denote the unique probability measure defined by $\mathcal{M}$ as $\mathbb{P}_{\mathcal{M}}$, and the associated expectation measure as $\mathbb{E}_{\mathcal{M}}$. For example, for the Markov chain $\mathcal{M}$ presented in Figure 1, we have that $\mathbb{P}_{\mathcal{M}}(ab) = \frac{1}{4}$, and so $\mathbb{P}_{\mathcal{M}}(\{w \in \{a,b\}^\omega \mid w[0,1] = ab\}) = \frac{1}{4}$, whereas $\mathbb{P}_{\mathcal{M}}(X) = 0$ for any finite set of infinite words X.

A *terminating* Markov chain $\mathcal{M}^T$ is a tuple $\langle \Sigma, S, s_0, E, T \rangle$, where Σ, S and s_0 are as usual, $E \colon S \times (\Sigma \cup \{\epsilon\}) \times S \mapsto [0,1]$ is the edge probability function, such that if $E(s,a,t)$, then $a = \epsilon$ if and only if $t \in T$, and for every $s \in S$ we have $\sum_{a \in \Sigma \cup \{\epsilon\}, s' \in S} E(s,a,s') = 1$, and T is a set of terminating states such that the probability of reaching a terminating state from any state s is positive. Notice that the only ϵ-transitions in a terminating Markov chain are those that lead to a terminating state.

The probability of a finite word u w.r.t. $\mathcal{M}$, denoted $\mathbb{P}_{\mathcal{M}^T}(u)$, is the sum of probabilities of paths from s_0 labeled by u such that the only terminating state on this path is the last one. Notice that $\mathbb{P}_{\mathcal{M}^T}$ is a probability distribution on finite words whereas $\mathbb{P}_{\mathcal{M}}$ is not (because the sum of probabilities may exceed 1).

Automata as random variables. A weighted automaton defines the function $\mathcal{L}_{\mathcal{A}}(w) \colon \Sigma^\omega \mapsto \mathbb{R}$ that assigns values to words. This function is measurable for all the automata types we consider in this paper (see Remark 2 below). Thus, this function can be interpreted as random variables w.r.t. the probabilistic space we consider. Hence, for a given automaton $\mathcal{A}$ and a Markov chain $\mathcal{M}$, we consider the following quantities:

$\mathbb{E}_{\mathcal{M}}(\mathcal{A})$ – the expected value of the random variable defined by $\mathcal{A}$ w.r.t. the probability measure defined by $\mathcal{M}$.
$\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda) = \mathbb{P}_{\mathcal{M}}(\{w \mid \mathcal{L}_{\mathcal{A}}(w) \leq \lambda\})$ – the (cumulative) distribution function of the random variable defined by $\mathcal{A}$ w.r.t. the probability measure defined by $\mathcal{M}$.

In the finite words case, the expected value $\mathbb{E}_{\mathcal{M}^T}$ and the distribution $\mathbb{D}_{\mathcal{M}^T,\mathcal{A}}$ are defined in the same manner.

▶ **Remark 1** (Bounds on the expected value and the distribution). *Both quantities can be easily bounded: the value of the distribution function is always between* 0 *and* 1. *For a* LimAvg*-automaton* $\mathcal{A}$*, we have* $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) \in [\min_{\mathcal{A}}, \max_{\mathcal{A}}] \cup \{\infty\}$*, where* $\min_{\mathcal{A}}$ *and* $\max_{\mathcal{A}}$ *denote the minimal and the maximal weight of* $\mathcal{A}$*. For a* Sum*-automaton* $\mathcal{A}$*, we have* $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) \in [L_{\mathcal{M}} \cdot \min_{\mathcal{A}}, L_{\mathcal{M}} \cdot \max_{\mathcal{A}}] \cup \{\infty\}$*, where* $L_{\mathcal{M}}$ *is the expected length of a word generated by* $\mathcal{M}$ *(it can be computed in a standard way [16, Section 11.2]). In both cases,* $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) = \infty$ *if and only if the probability of the set of words with no accepting runs in* $\mathcal{A}$ *is positive. Note that we consider no* ω*-accepting conditions, and hence all infinite runs of* Sum*-automata are accepting. Still there can be infinite words, on which a given* Sum*-automaton has no infinite runs. We show in Section 3.2 that the distribution and expected value may be irrational, even for integer weights and uniform distributions.*

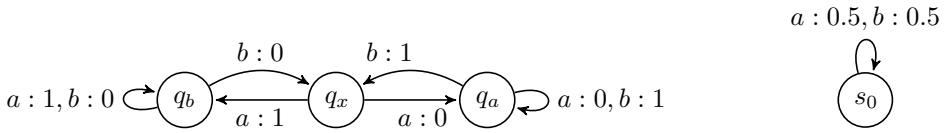


Figure 1 The automaton $\mathcal{A} = \{\{a,b\}, \{q_x, q_a, q_b\}, \{q_a, q_b\}, \emptyset, \delta, C\}$, where $\delta = \{(q_a, a, q_a), (q_a, b, q_a), (q_a, b, q_x), (q_x, a, q_a), (q_x, a, q_b), (q_b, a, q_x), (q_b, a, q_b), (q_b, b, q_b)\}$ and C such that $C(q_a, b, q_a) = C(q_b, a, q_b) = C(q_a, b, q_x) = C(q_x, a, q_b) = 1$ and for all other inputs the value of C is 0 (left) and the Markov chain $\mathcal{M} = \{\{a,b\}, \{s_0\}, \{s_0\}, E\}$ where E always returns 0.5 (right).

▶ **Remark 2** (Measurability of functions represented by automata). *For automata on finite words,* Inf*-automata and* Sup*-automata, measurability of $\mathcal{L}_\mathcal{A}$ is straightforward. To show that $\mathcal{L}_\mathcal{A}(w)\colon \Sigma^\omega \mapsto \mathbb{R}$ is measurable for any non-deterministic* LimAvg*-automaton $\mathcal{A}$, it suffices to show that for every $x \in \mathbb{R}$, the preimage $\mathcal{L}_\mathcal{A}^{-1}(-\infty, x]$ is measurable. Let Q be the set of states of $\mathcal{A}$. Consider the set $\Sigma^\omega \times \mathbb{Q}^\omega$. We can define a subset $A_x \subseteq \Sigma^\omega \times \mathbb{Q}^\omega$ of the pairs, the word and the run on it, where the value of the run is less than or equal to x. The set A_x can be presented as a countable intersection of open sets, and hence it is Borel. Observe that $\mathcal{L}_\mathcal{A}^{-1}(-\infty, x]$ is the projection of A_x on the first component Σ^ω. The projection of a Borel set is* analytic*, which is measurable [20]. Thus, $\mathcal{L}_\mathcal{A}$ defined by a non-deterministic* LimAvg*-automaton is measurable.*

The above proof of measurability requires some knowledge of descriptive set theory. We will give a direct proof of measurability of $\mathcal{L}_\mathcal{A}$ in the paper (Theorem 16).

2.2 Computational questions

We consider the following basic computational questions:

The expected value question: Given an f-automaton $\mathcal{A}$ and a (terminating) Markov chain $\mathcal{M}$, compute $\mathbb{E}_\mathcal{M}(\mathcal{A})$.
The distribution question: Given an f-automaton $\mathcal{A}$, a (terminating) Markov chain $\mathcal{M}$ and a threshold λ, compute $\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda)$.

Each of the above questions have its decision variant (useful for lower bounds), where instead of computing the value we ask whether the value is less than a given threshold t.

The above questions have their approximate variants:

The approximate expected value question: Given an f-automaton $\mathcal{A}$, a (terminating) Markov chain $\mathcal{M}$, $\epsilon > 0$, compute a number η such that $|\eta - \mathbb{E}_\mathcal{M}(\mathcal{A})| \leq \epsilon$.
The approximate distribution question: Given an f-automaton $\mathcal{A}$, a (terminating) Markov chain $\mathcal{M}$, a threshold λ and $\epsilon > 0$ compute a number $\eta \in [\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda - \epsilon), \mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda + \epsilon)]$.

In the later case, we use the Skorokhod's notion. One could expect there "$\eta \in [\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda) - \epsilon, \mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda) + \epsilon]$" instead. However, this would lead to undecidable approximation in the LimAvg case (cf. Theorem 6).

3 Basic properties

Consider an f-automaton $\mathcal{A}$, a Markov chain $\mathcal{M}$ and a set of words X. We denote by $\mathbb{E}_\mathcal{M}(\mathcal{A} \mid X)$ the expected value of $\mathcal{A}$ w.r.t. $\mathcal{M}$ restricted only to words in the set X (see [14]).

The following says that we can disregard a set of words with probability 0 (e.g. containing only some of the letters under uniform distribution) while computing the expected value.

▶ **Fact 3.** *If* $\mathbb{P}(X) = 1$ *then* $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) = \mathbb{E}_{\mathcal{M}}(\mathcal{A} \mid X)$.

The proof is rather straightforward; the only interesting case is when there are some words not in X with infinite values. But for all the functions we consider, one can show that in this case there is a set of words with infinite value that has a non-zero probability, and therefore $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) = \mathbb{E}_{\mathcal{M}}(\mathcal{A} \mid X) = \infty$.

One important corollary of Fact 3 is that if $\mathcal{M}$ is, for example, uniform, then because the set Y of ultimately-periodic words (i.e., words of the form vw^ω) has probability 0, we have $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) = \mathbb{E}_{\mathcal{M}}(\mathcal{A} \mid \Sigma^\omega \setminus Y)$. This suggests a possibility that the expected value may not be realised by any ultimately periodic word. We exemplify this in Remark 13.

3.1 Example of computing expected value by hand

Consider a LimAvg-automaton $\mathcal{A}$ and a Markov chain $\mathcal{M}$ depicted in Figure 1. We encourage the reader to take a moment to study this automaton and try to figure out its expected value.

The idea behind $\mathcal{A}$ is as follows. Assume that $\mathcal{A}$ is in a state q_l for some $l \in \{a, b\}$. Then, it reads a word up to the first occurrence a subword ba, where it has a possibility to go to q_x and then to non-deterministically choose q_a or q_b as the next state. Since going to q_x and back to q_l costs the same as staying in q_l, we will assume that the automaton always goes to q_x in such a case. When an automaton is in the state q_x and has to read a word $w = a^j b^k$, then average cost of a run on w is $\frac{j}{j+k}$ if the run goes to q_b and $\frac{k}{j+k}$ otherwise. So the run with the lowest value is the one that goes to q_a if $j > k$ and q_b otherwise.

To compute the expected value of the automaton, we focus on the set X of words w such that for each positive $n \in \mathbb{N}$ there are only finitely many prefixes of w of the form $w'a^j b^k$ such that $\frac{j+k}{|w'|+j+k} \geq \frac{1}{n}$. Notice that this means that w contains infinitely many a and infinitely many b. It can be proved in a standard manner that $\mathbb{P}_{\mathcal{M}}(X) = 1$.

Let $w \in X$ be a random event, which is a word generated by $\mathcal{M}$. Since w contains infinitely many letters a and b, it can be partitioned in the following way. Let $w = w_1 w_2 w_3 \dots$ be a partition of w such that each w_i for $i > 0$ is of the form $a^j b^k$ for $j \geq 0, k > 0$, and for $i > 1$ we also have $j > 0$. For example, the partition of $w = baaabbbaabbbaba\dots$ is such that $w_1 = b$, $w_2 = aaabbb$, $w_3 = aabbb$, $w_4 = ab$, $\dots$. Let $s_i = |w_1 w_2 \dots w_i|$.

We now define a run π_w on w as follows:

$$q_1^w \dots q_1^w q_x q_2^w \dots q_2^w q_x q_3^w \dots q_3^w q_x q_4^w \dots$$

where the length of each block of q_i is $|w_i| - 1$, $q_0^w = q_a$ and $q_i^w = q_a$ if $w_i = a^j b^k$ for some $j > k$ and $q_i^w = q_b$ otherwise. It can be shown by a careful consideration of all possible runs that this run's value is the infimum of values of all the runs on this word.

▶ **Lemma 4.** $\mathcal{L}_{\mathcal{A}}(w) = \text{LimAvg}(\pi_w)$.

By Fact 3 and Lemma 4, it remains to compute the expected value of $\text{LimAvg}(\{\pi_w \mid w \in X\})$. As the expected value of the sum is the sum of expected values, we can state that

$$\mathbb{E}_{\mathcal{M}}(\text{LimAvg}(\{\pi_w \mid w \in X\})) = \limsup_{s\to\infty} \frac{1}{s} \cdot \sum_{i=1}^{s} \mathbb{E}_{\mathcal{M}}\left(\{(C(\pi_w))[i] \mid w \in X\}\right)$$

It remains to compute $\mathbb{E}_{\mathcal{M}}((C(\pi_w)[i])$. If i is large enough (and since the expected value does not depend on a finite number of values, we assume that it is), the letter $\pi_w[i]$ is in some block $w_s = a^j b^k$. There are $j + k$ possible letters in this block, and the probability that

the letter $\pi_w[i]$ is an ith letter in such a block is $2^{-(j+k+2)}$ ("+2", because the block has to be maximal, so we need to include the letters before the block and after the block). So the probability that a letter is in a block $a^j b^k$ is $\frac{j+k}{2^{j+k+2}}$. The average cost of a such a letter is $\frac{\min(j,k)}{j+k}$, as there are $j+k$ letters in this block and the block contributes $\min(j,k)$ to the sum.

It can be analytically checked that

$$\sum_{j=1}^{\infty}\sum_{k=1}^{\infty}\frac{j+k}{2^{j+k+2}}\cdot\frac{\min(j,k)}{j+k}=\sum_{j=1}^{\infty}\sum_{k=1}^{\infty}\frac{\min(j,k)}{2^{j+k+2}}=\frac{1}{3}$$

We can conclude that $\mathbb{E}_{\mathcal{M}}(\textsc{LimAvg}(\pi_w))=\frac{1}{3}$ and, by Lemma 4, $\mathbb{E}_{\mathcal{M}}(\mathcal{A})=\frac{1}{3}$.

The bottom line is that even for such a simple automaton with only one strongly connected component consisting of three states (and two of them being symmetrical), the analysis is complicated. On the other hand, we conducted a simple Monte Carlo experiment in which we computed the value of this automaton on 10000 random words of length 2^{22} generated by $\mathcal{M}$, and observed that the obtained values are in the interval $[0.3283, 0.3382]$, with the average of 0.33336, which is a good approximation of the expected value 0.(3). This foreshadows our results for LimAvg-automata: we show that computing the expected value is, in general, impossible, but it is possible to approximate it with arbitrary precision. Furthermore, the small variation of the results is not accidental – we show that for strongly-connected LimAvg-automata, almost all words have the same value (which is equal to the expected value).

3.2 Irrationality of the distribution and the expected value

We argue that the exact values of $\mathbb{E}_{\mathcal{M}}(\mathcal{A})$ and $\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda)$ for Sum-automata and LimAvg-automata may be irrational.

For the rest of this section we assume that the distribution of words is uniform. In the infinite case, this means that the Markov chain contains a single state where it loops over any letter with probability $\frac{1}{|\Sigma|}$, where Σ is the alphabet. In the finite case, this amounts to a terminating Markov chain with one regular state and one terminating state; it loops over any letter in the non-terminating state with probability $\frac{1}{|\Sigma|+1}$ or go to the terminating state over ϵ with probability $\frac{1}{|\Sigma|+1}$. Below we omit the Markov chain as it is fixed (for a given alphabet).

We define a Sum-automaton $\mathcal{A}$ (Figure 2) over the alphabet $\Sigma=\{a,\#\}$ such that $\mathcal{A}(w)=0$ if $w=a\#a^2\#\ldots\#a^{2^n}$ and $\mathcal{A}(w)\leq -1$ otherwise. Such an automaton basically picks a block with an inconsistency and verifies it. For example, if w contains a block $\#a^i\#a^j\#$, the automaton $\mathcal{A}$ first assigns -2 to each letter a and upon $\#$ it switches to the mode in which it assigns 1 to each letter a. Then, $\mathcal{A}$ returns the value $j-2\cdot i$. Similarly, we can encode the run that returns the value $2\cdot i-j$. Therefore, all the runs return 0 if and only if each block of a's is twice as long as the previous block. Finally, $\mathcal{A}$ checks whether the first block of a's has length 1 and returns -1 otherwise.

A word of the form $a\#a^2\#\ldots\#a^{2^n}$ has length $2^{n+1}+n-1$ and its probability is $3^{-(2^{n+1}+n)}$ (as the probability of any given word with n letters over a two-letters alphabet is $3^{-(n+1)}$). Therefore, the probability γ that a word is of the form $a\#a^2\#\ldots\#a^{2^n}$ is equal to $\sum_{n=0}^{\infty}3^{-(2^{n+1}+n)}$. Observe that γ written in base 3 has arbitrary long sequences of 0's and hence its representation is acyclic. Thus, γ is irrational. Observe that $\gamma=1-\mathbb{D}_{\mathcal{A}}(-1)$. Therefore, $\mathbb{D}_{\mathcal{A}}(-1)$ is irrational.

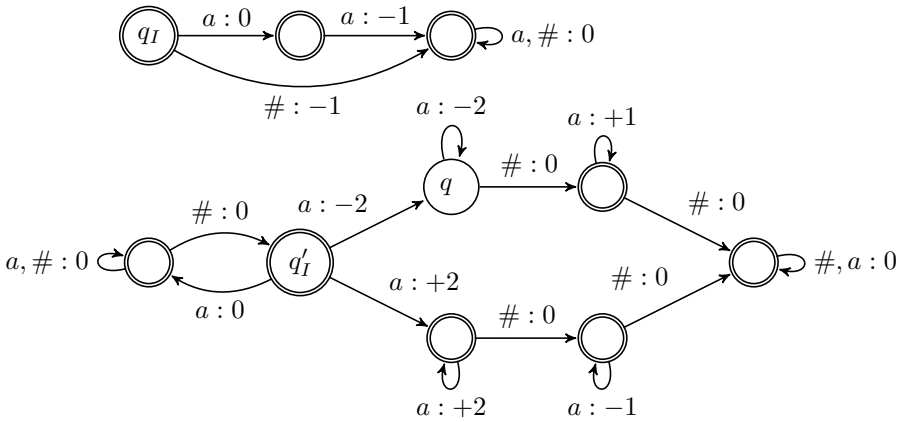


Figure 2 The automaton $\mathcal{A}$ from Section 3.2. States q_I and q'_I are initial and states but q are accepting. Any word that starts with $\#$ or aa has the value at most -1 because of a run that starts in q_I. For all other words, the runs starting in q_I have value -1. The accepting runs starting in q'_I have negative value only if the input word contains a (maximal) subword $a^i\#a^j$ such that $j \neq 2i$.

For the expected value, we construct $\mathcal{A}'$ such that for every word w we have $\mathcal{L}_{\mathcal{A}'}(w) = min(\mathcal{L}_{\mathcal{A}}(w), -1)$. This can be done by adding to $\mathcal{A}$ an additional initial state q_0, which starts an automaton that assigns to all words value -1. Observe that $\mathcal{A}$ and $\mathcal{A}'$ differ only on words w of the form $a\#a^2\#\ldots\#a^{2^n}$, where $\mathcal{A}(w) = 0$ and $\mathcal{A}'(w) = -1$. On all other words, both automata return the same values. Therefore, $\mathbb{E}(\mathcal{A}) - \mathbb{E}(\mathcal{A}') = \gamma$. It follows that at least one of the values $\mathbb{E}(\mathcal{A})$, $\mathbb{E}(\mathcal{A}')$ is irrational.

The same construction works for LimAvg. We take $\mathcal{A}$ (resp., $\mathcal{A}'$) and we convert it to a LimAvg-automaton $\mathcal{A}_\infty$ (resp., $\mathcal{A}'_\infty$) over $\Sigma' = \Sigma \cup \{\$\}$. The new letter \$ resets the automaton, i.e., $\mathcal{A}_\infty$ (resp., $\mathcal{A}'_\infty$) has transitions from the final states of $\mathcal{A}$ (resp., $\mathcal{A}'$) to its initial states labeled with \$. We can show that $1 - \mathbb{D}_{\mathcal{A}_\infty}(-1)$ and $\mathbb{E}(\mathcal{A}_\infty) - \mathbb{E}(\mathcal{A}'_\infty)$ over the uniform distribution are equal to γ and hence $\mathbb{D}_{\mathcal{A}_\infty}(-1)$ is irrational and one of the values $\mathbb{E}(\mathcal{A}_\infty)$, $\mathbb{E}(\mathcal{A}'_\infty)$ is irrational.

▶ **Theorem 5.** *There exist a* Sum*-automaton and a* LimAvg*-automaton whose distributions and expected values w.r.t. the uniform distribution are irrational.*

4 The exact value problems

In this section we consider the probabilistic questions for non-deterministic Sum-automata and LimAvg-automata, i.e., the problems of computing the exact values of the expected value $\mathbb{E}_{\mathcal{M}}(\mathcal{A})$ and the distribution $\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda)$ w.r.t. a Markov chain $\mathcal{M}$ and an f-automaton $\mathcal{A}$. We showed that these values may be irrational. But one can perhaps argue that there might be some representation of irrational numbers that can be employed to avoid this problem. We prove that this is not the case by showing that computing the exact value to any representation with decidable equality of two numbers is not possible. The proof is by a (Turing) reduction from the quantitative universality problem for Sum-automata:

The quantitative universality problem for Sum*-automata*: Given a Sum-automaton with weights -1, 0 and 1, decide whether for all words w we have $\mathcal{L}_{\mathcal{A}}(w) \leq 0$.

The quantitative universality problem for Sum-automata is undecidable [21, 1].

We first discuss reductions to the probabilistic problems for Sum-automata. Consider an instance of the quantitative universality problem, which is a Sum-automaton $\mathcal{A}$. If there is a word w with the value greater than 0, then $\mathbb{P}(w) > 0$, and thus $\mathbb{D}_{\mathcal{A}}(0) < 1$. Otherwise, clearly $\mathbb{D}_{\mathcal{A}}(0) = 1$. Therefore, solving the universality problem amounts to computing whether the $\mathbb{D}_{\mathcal{A}}(0) = 1$, and thus the latter problem is undecidable. For the expected value, we construct a Sum-automaton $\mathcal{A}'$ such that for every word w we have $\mathcal{L}_{\mathcal{A}'}(w) = min(\mathcal{L}_{\mathcal{A}}(w), 0)$. Observe that $\mathbb{E}(\mathcal{A}) = \mathbb{E}(\mathcal{A}')$ if and only if for every word w we have $\mathcal{L}_{\mathcal{A}}(w) \leq 0$, i.e., the answer to the universality problem is YES. Therefore, there is no Turing machine, which given a Sum-automaton $\mathcal{A}$ computes $\mathbb{E}(\mathcal{A})$.

For LimAvg case, we construct a LimAvg-automaton $\mathcal{A}_\infty$ from the Sum-automaton $\mathcal{A}$, by connecting all accepting states (of $\mathcal{A}$) with all initial states by transitions of weight 0 labeled by an auxiliary letter #. For the expected value we construct $\mathcal{A}'_\infty$ from $\mathcal{A}'$ in the same way. Again, the distribution $\mathbb{D}_{\mathcal{A}_\infty}(0) = 1$ if and only if for all words we have $\mathcal{L}_{\mathcal{A}}(w) \leq 0$. Then, observe that $\mathbb{E}(\mathcal{A}_\infty) = \mathbb{E}(\mathcal{A}'_\infty)$ if and only if for every word w we have $\mathcal{L}_{\mathcal{A}}(w) \leq 0$. Therefore, there is no Turing machine computing the expected value or the distribution of a given LimAvg-automaton.

▶ **Theorem 6.** *The expected value and the distribution of (non-deterministic)* Sum*-automata (resp.,* LimAvg*-automata) are uncomputable even for the uniform distribution.*

4.1 Extrema automata

We discuss the distribution problem for Min-, Max-, Inf- and Sup-automata, where Min and Max return the minimal and the maximal (resp.) element of a finite sequence, and Inf and Sup return the minimal and the maximal (resp.) element of an infinite sequence. The expected value of an automaton can be easily computed based on the distribution as there are only finitely many possible values of a run (each possible value is a label of some transition).

▶ **Theorem 7.** *For* Min-, Max-, Inf- *and* Sup*-automata $\mathcal{A}$ and a Markov chain $\mathcal{M}$, the distribution problem can be solved in exponential time in $|\mathcal{A}|$ and polynomial time in $|\mathcal{M}|$.*

Proof. We discuss the case of $f =$ Inf as the other cases are similar. Consider an Inf-automaton $\mathcal{A}$. For each weight x of $\mathcal{A}$, we can construct a (non-deterministic) ω-automaton $\mathcal{A}_x$ that accepts only words of value greater than x – we take $\mathcal{A}$, remove the transitions of weight at most x, and drop all the weights. Therefore, the set of words with the value greater than x is regular, and hence it is measurable. We can compute its probability p_x w.r.t. $\mathcal{M}$ in exponential time in $|\mathcal{A}|$ and polynomial in $|\mathcal{M}|$ [3]. Note that $p_x = 1 - \mathbb{D}_{\mathcal{M},\mathcal{A}}(x)$. ◀

5 The approximation problems

We start the discussion on the approximation problems by showing a hardness result that holds for a wide range of value functions. We say that a function is 0-preserving if its value is 0 whenever the input consists only of 0s. Notice that functions such as Sum, LimAvg, Min, Max, Inf, Sup and virtually all the functions from the literature [8] are 0-preserving. The hardness results follow from the fact that accepted words have finite values, which we can force to be 0, while words without accepting runs have infinite values.

The answers in the approximation problems are numbers and to study the lower bounds, we consider their decision variants, called the *separation problems*. In these variants, the input is enriched with numbers a, b such that $b - a > 2\epsilon$ and the instance is such that

$\mathbb{E}_{\mathcal{M}}(\mathcal{A}) \notin [a, b]$ (resp. $\mathbb{D}_{\mathcal{M}}(\mathcal{A}) \notin [a, b]$), and the question is whether $\mathbb{E}_{\mathcal{M}}(\mathcal{A}) < a$ (resp. $\mathbb{D}_{\mathcal{M}}(\mathcal{A}) < a$). Note that having an algorithm computing one of the approximate problems (for the distribution or the expected value), we can use it to decide the separation question. Conversely, using the separation problem as an oracle, we can perform binary search on the domain to compute solve the corresponding approximation problem in polynomial time.

▶ **Theorem 8.** *For a 0-preserving function f, the separation problems for non-deterministic f-automata are* PSpace*-hard.*

Total automata. Theorem 8 gives us a general hardness result, which is due to accepting conditions rather than values returned by weighted automata. In the following, we focus on weights and we assume that weighted automata are *total*, i.e., they accept all the words (resp., almost all the words in the infinite-word case). For Sum-automata under the totality assumption, the approximate probabilistic questions become #P-complete.

▶ **Theorem 9.** *The approximate expected value and the approximate distribution questions for non-deterministic total* Sum*-automata are* #P*-complete.*

The lower bound can be obtained by a reduction from the problem of counting the number of satisfying assignment of a given propositional formula φ in Conjunctive Normal Form (CNF) [25, 23], which is #P-complete. We construct an automaton that, for a formula with n variables, accepts only words of length n that encode valuations that make the formula satisfied. It follows that the expected value of the automaton equals $3^{-(n+1)} \cdot C$, where $3^{-(n+1)}$ is the probability of generating a word of length n under uniform distribution and C is the number of variable assignments satisfying φ.

The upper bound follows the idea that the probability that $\mathcal{M}^T$ emits a word of length greater than n decreases exponentially with n. This means that there is N of polynomial size such that the distribution (resp., the expected value) of $\mathcal{A}$ and the distribution (resp., the expected value) of $\mathcal{A}$ over words up to length N differ by less than ϵ. Based on this, we can build a Turing machine that imitates the distribution of $\mathcal{M}^T$ over words up to length N with its non-deterministic computations.

We show that the approximation problem for LimAvg-automata is PSpace-hard over the class of total automata.

▶ **Theorem 10.** *The separation problems for non-deterministic total* LimAvg*-automata are* PSpace*-hard.*

Proof. Given a non-deterministic finite-word automaton $\mathcal{A}$, we construct an infinite-word LimAvg-automaton $\mathcal{A}_\infty$ from $\mathcal{A}$ in the following way. We introduce an auxiliary symbol # and we add transitions labeled by # between any final state of $\mathcal{A}$ and any initial state of $\mathcal{A}$. Then, we label all transitions of $\mathcal{A}_\infty$ with 0. Finally, we connect all non-accepting states of $\mathcal{A}$ with an auxiliary state q_{sink}, which is a sink state with all transitions of weight 1. The automaton $\mathcal{A}_\infty$ is total.

Observe that if $\mathcal{A}$ is universal, then $\mathcal{A}_\infty$ has a run of value 0 on every word. Otherwise, if $\mathcal{A}$ rejects a word w, then upon reading a subword $\#w\#$, the automaton $\mathcal{A}_\infty$ reaches q_{sink}, i.e., the value of the whole word is 1. Almost all words contain an infix $\#w\#$ and hence almost all words have value 1. ◀

6 Approximating LimAvg-automata in exponential time

The case of LimAvg is significantly more complex than the other cases. First, we restrict our attention to *recurrent* LimAvg-automata and the uniform distribution over infinite words. Then, we comment on the extension to all distributions given by Markov chains. Finally, we show the proof for all LimAvg-automata over probability measures given by Markov chains.

6.1 Recurrent automata

A non-deterministic LimAvg-automaton $\mathcal{A} = (\Sigma, Q, Q_0, \delta)$ is recurrent if and only if for every set $S \subseteq Q$ such that $|S| = 1$ or $\widehat{\delta}(Q_0, w) = S$ for some word w, there is a finite word u such that $\widehat{\delta}(S, u) = Q_0$.

For every $\mathcal{A}$, which is strongly connected as a graph, there exists a set of initial states T with which it becomes recurrent. Moreover, the probability of words accepted by $\mathcal{A}$ is either 0 or 1. Indeed, consider $\mathcal{A}$ as an unweighted ω-automaton and construct a deterministic ω-automaton $\mathcal{A}^D$ through the power-set construction applied to $\mathcal{A}$. Note that $\mathcal{A}^D$ has a single bottom strongly-connected component (BSCC) and Q_0 belongs to that component. Conversely, for any strongly connected automaton $\mathcal{A}$, if Q_0 belongs to the BSCC of $\mathcal{A}^D$, then $\mathcal{A}$ is recurrent. Moreover, since $\mathcal{A}^D$ has a single BSCC, for almost all words, all runs end up in that BSCC and hence the probability of the set of words having any infinite run in $\mathcal{A}$ is either 0 or 1.

6.2 Nearly-deterministic approximations

While words are generated by a Markov chain letter by letter, the run on that word can be defined only when the complete word is generated. This precludes application of standard techniques for probabilistic verification, which relies on the fact that the word and the run on it are generated simultaneously [26, 12, 3].

Key ideas. Our main idea is to change the non-determinism to *bounded look-ahead*. This must be inaccurate, as the expected value of a deterministic automaton with bounded look-ahead is always rational, whereas Theorem 5 shows that the values of non-deterministic automata may be irrational. Nevertheless, we show that bounded look-ahead is sufficient to *approximate* the probabilistic questions for recurrent automata (Lemma 11). Furthermore, the approximation can be done effectively (Lemma 14), which in turn gives us an exponential-time approximation algorithm for recurrent automata (Lemma 15).

Jumping runs. Let $k > 0$. A *k-jumping run* ξ of $\mathcal{A}$ on a word w is an infinite sequence of states such that for every i there is a run π of $\mathcal{A}$ on w such that $\xi[ki, k(i+1) - 1] = \pi[ki, k(i+1) - 1]$.

A *block* of a k-jumping run is a sequence $\xi[ki, k(i+1) - 1]$ for some i; positions $k, 2k, \ldots$ are *jumps*, where the sequence ξ need not obey the transition relation of $\mathcal{A}$.

The cost C of a transition of a k-jumping run ξ within a block is defined as usual, while the cost of a jump is defined as the minimal weight of $\mathcal{A}$. The value of a k-jumping run ξ is defined as the limit average computed for such costs.

Optimal and block-deterministic jumping runs. We say that a k-jumping run ξ on a word w is *optimal* if its value is the infimum over values of all k-jumping runs on w. We show that optimal k-jumping runs can be constructed nearly deterministically, i.e., only looking ahead to see the whole current block.

For every $S \subseteq Q$ and $u \in \Sigma^k$ we fix a run $\xi_{S,u}$ on u starting in one of states of S, which has the minimal average weight. Then, given a word $w \in \Sigma^\omega$, we define a k-jumping run ξ as follows. We divide w into k-letter blocks $u_1, u_2, \ldots$ and we put $\xi = \xi_{S_0,u_1}\xi_{S_1,u_2}\ldots$, where $S_0 = \{q_0\}$ and for $i > 0$, S_i is the set of states reachable from q_0 on the word $u_1 \ldots u_i$. The run ξ_o is a k-jumping run and it is indeed optimal. We call such runs *block-deterministic* – they can be constructed based on finite memory – the set of reachable states S_i and the current block of the input word.

Since all runs of $\mathcal{A}$ are in particular k-jumping runs, the value of (any) optimal k-jumping run on w is less or equal to $\mathcal{A}(w)$. We show that for recurrent LimAvg-automata, the values of k-jumping runs on w converge to $\mathcal{A}(w)$ as k tends to infinity. To achieve this, we construct a run of $\mathcal{A}$ which tries to "follow" a given jumping run, i.e., after most all of the jumps it is able to synchronize with the jumping run quickly.

▶ **Lemma 11.** *Let $\mathcal{A}$ be a recurrent* LimAvg*-automaton. For every $\epsilon > 0$, there exists k such that for almost all words w, the value $\mathcal{A}(w)$ and the value an optimal k-jumping run on w differ by at most ϵ. The value k is doubly-exponential in $|\mathcal{A}|$ and polynomial in $\frac{1}{\epsilon}$.*

6.3 Random variables

Given a recurrent LimAvg-automaton $\mathcal{A}$ and $k > 0$, we define a function $g[k] : \Sigma^\omega \to \mathbb{R}$ such that $g[k](w)$ is the value of some optimal k-jumping run ξ_o on w. We can pick ξ_o to be block-deterministic and hence $g[k]$ corresponds to a Markov chain $M[k]$. More precisely, we define $M[k]$ labeled by Σ^k such that for every word w, the limit average of the path in $M[k]$ labeled by blocks of w (i.e., blocks $w[1,k]w[k+1,2k]\ldots$) equals $g[k](w)$. Moreover, the distribution of blocks Σ^k is uniform and hence $M[k]$ corresponds to $g[k]$ over the uniform distribution over Σ. The Markov chain $M[k]$ is a labeled weighted Markov chain [15], such that its states are all subsets of Q, the set of states of $\mathcal{A}$. For each state $S \subseteq Q$ and $u \in \Sigma^k$, the Markov chain $\mathcal{M}$ has an edge $(S, \widehat{\delta}(S,u))$ of probability $\frac{1}{|\Sigma|^k}$. The weight of an edge (S,S') labeled by u is the minimal average of weights of any run from some state of S to some state of S' over the word w.

We have the following:

▶ **Lemma 12.** *Let $\mathcal{A}$ be a recurrent* LimAvg*-automaton and $k > 0$. (1) The functions $g[k]$ and $\mathcal{L}_\mathcal{A}$ are random variables. (2) For almost all words w we have $g[k](w) = \mathbb{E}(g[k])$ and $\mathcal{L}_\mathcal{A}(w) = \mathbb{E}(\mathcal{L}_\mathcal{A})$.*

Proof. Since $\mathcal{A}$ is recurrent, $M[k]$ has a single BSCC and hence $M[k]$ and $g[k]$ return the same value for almost all words [15]. This implies that the preimage through $g[k]$ of each set has measure 0 or 1, and hence $g[k]$ is measurable [14]. Lemma 11 implies that (measurable functions) $g[k]$ converge to $\mathcal{L}_\mathcal{A}$ with probability 1, and hence $\mathcal{L}_\mathcal{A}$ is measurable [14]. As the limit of $g[k]$, $\mathcal{L}_\mathcal{A}$ also has the same value for almost all words. ◀

▶ **Remark 13.** *The automaton $\mathcal{A}$ from the proof of Theorem 5 is recurrent (it resets after each \$), so the value of $\mathcal{A}$ on almost all words is irrational. Yet, for every ultimately periodic word vw^ω, the value of $\mathcal{A}$ is rational. This means that while the expected value is realised by almost all words, it is not realised by any ultimately periodic word.*

6.4 Approximation algorithms

We show that the expected value of $g[k]$ can be efficiently approximated. The approximation is exponential in the size of $\mathcal{A}$, but only logarithmic in k (which is doubly-exponential due to Lemma 11).

▶ **Lemma 14.** *Given a recurrent* LimAvg*-automaton $\mathcal{A}$, $k = 2^l$ and $\epsilon > 0$, the expected value $\mathbb{E}(g[k])$ can be approximated up to ϵ in exponential time in $|\mathcal{A}|$, logarithmic time in k and polynomial time in $\frac{1}{\epsilon}$.*

Lemma 11 and Lemma 14 imply the following:

▶ **Lemma 15.** *Given a recurrent* LimAvg*-automaton $\mathcal{A}$, Markov chain $\mathcal{M}$, $\epsilon > 0$ and $\lambda \in \mathbb{Q}$, we can compute ϵ-approximations of the distribution $\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda)$ and the expected value $\mathbb{E}_{\mathcal{M}}(\mathcal{A})$ in exponential time in $|\mathcal{A}|$ and polynomial time in $|\mathcal{M}|$ and $\frac{1}{\epsilon}$.*

Proof. For uniform distributions, by Lemma 11, for every $\epsilon > 0$, there exists k such that $|\mathbb{E}(\mathcal{A}) - \mathbb{E}(g[k])| \leq \frac{\epsilon}{2}$. The value k is doubly-exponential in $|\mathcal{A}|$ and polynomial in $\frac{1}{\epsilon}$. Then, Lemma 14, we can compute γ such that $|\gamma - \mathbb{E}(g[k])| \leq \frac{\epsilon}{2}$ in exponential time in $|\mathcal{A}|$ and polynomial in $\frac{1}{\epsilon}$. Thus, γ differs from $\mathbb{E}(\mathcal{A})$ by at most ϵ. Since almost all words have the same value, we can approximate $\mathbb{D}_{\mathcal{A}}(\lambda)$ by comparing λ with γ, i.e., 1 is an ϵ-approximation of $\mathbb{D}_{\mathcal{A}}(\lambda)$ if $\lambda \leq \gamma$, and otherwise 0 is an ϵ-approximation of $\mathbb{D}_{\mathcal{A}}(\lambda)$.

The case of the nonuniform distributions can be solved similarly, by encoding the Markov chain in the automaton. ◀

We lift Lemma 15 from recurrent to all LimAvg-automata and formally show that $\mathcal{L}_{\mathcal{A}} : \Sigma^\omega \to \mathbb{R}$ is measurable. To do so, we take the product of a given automaton and Markov chain and observe that its BSCC correspond to recurrent automata. A careful analysis allows to compute the values for the whole automata based on the values for the BSCC.

▶ **Theorem 16.** *(1) For a non-deterministic* LimAvg*-automaton $\mathcal{A}$ the function $\mathcal{L}_{\mathcal{A}} : \Sigma^\omega \to \mathbb{R}$ is measurable. (2) Given a non-deterministic* LimAvg*-automaton $\mathcal{A}$, Markov chain $\mathcal{M}$, $\epsilon > 0$, and $\lambda \in \mathbb{Q}$, we can ϵ-approximate the distribution $\mathbb{D}_{\mathcal{M},\mathcal{A}}(\lambda)$ and the expected value $\mathbb{E}(\mathcal{A})$ in exponential time in $|\mathcal{A}|$ and polynomial time in $|\mathcal{M}|$ and $\frac{1}{\epsilon}$.*

7 Determinising and approximating LimAvg-automata

For technical simplicity, we assume that the distribution of words is uniform. However, the results presented here extend to all distributions given by Markov chains.

Recall that for the LimAvg automata, the value of almost all words, whose optimal runs end up in the same SSC, is the same. This means that there is a finite set of values (not greater than the number of SSCs of the automaton) such that almost all the words have their values in this set.

LimAvg-automata are not determinisable [8]. We say that a non-deterministic LimAvg-automaton $\mathcal{A}$ is *weakly determinisable* if there is a deterministic LimAvg-automaton B such that A and B have the same value over almost all the words. From [9] we know that deterministic automata return rational values for almost all the words, so not all LimAvg-automata are weakly determinisable. However, we can show the following.

▶ **Theorem 17.** *A* LimAvg*-automaton $\mathcal{A}$ is weakly determinisable if and only if it returns rational values for almost all words.*

Proof sketch. Assume an automaton $\mathcal{A}$ with SSCs $C_1, \ldots, C_m$. For each i let v_i be defined as the expected value of $\mathcal{A}$ when its set of initial states is C_i and the run is bounded to stay in C_i. If $\mathcal{A}$ has no such runs for some C_i, then $v_i = \infty$.

We now construct a deterministic automaton B with rational weights using the standard power-set construction. We define the cost function such that the cost of any transition from

a state Y is the minimal value v_i such that v_i is rational and Y contains a state from C_i. If there are no such v_i, then we set the cost to the maximal cost of $\mathcal{A}$. Roughly speaking, B tracks in which SSCs A can be and the weight corresponds to the SSC with the lowest value.

To see that B weakly determinises A observe that for almost all words w, a run with the lowest value over w ends in some SSC and its value then equals the expected value of this component, which is rational as the value of this word is rational. ◀

A straightforward corollary is that every non-deterministic LimAvg-automaton can be weakly determinised by an LimAvg-automaton with real weights.

Theorem 17 does not provide an implementable algorithm for weakly-determinisation, because of the hardness of computing the values v_i. It is possible, however, to approximate this automaton. We say that a deterministic LimAvg-automaton B *ϵ-approximates* $\mathcal{A}$ if for almost every word w we have that $\mathcal{L}_B(w) \in [\mathcal{L}_{\mathcal{A}}(w) - \epsilon, \mathcal{L}_{\mathcal{A}}(w) + \epsilon]$.

▶ **Theorem 18.** *For every $\epsilon > 0$ and a non-deterministic* LimAvg*-automaton $\mathcal{A}$, one can compute in exponential time a deterministic* LimAvg*-automaton that ϵ-approximates $\mathcal{A}$.*

The proof of this theorem is similar to the proof of Theorem 17, except now it is enough to approximate the values v_i, which can be done in exponential time.

References

1. Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491. LNCS 6996, Springer, 2011.
2. Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms (TALG)*, 6(2):28, 2010.
3. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
4. Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. Markov chains and unambiguous büchi automata. In *CAV 2016*, pages 23–42. Springer, 2016.
5. Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *LICS 2011*, pages 335–344, 2011.
6. Udi Boker and Thomas A. Henzinger. Approximate determinization of quantitative automata. In *FSTTCS 2012*, pages 362–373. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
7. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. A survey of stochastic games with limsup and liminf objectives. In *ICALP 2009*, pages 1–15, 2009.
8. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.
9. Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative automata under probabilistic semantics. In *LICS 2016*, pages 76–85. ACM, 2016.
10. Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *CONCUR 2012*, pages 115–131, 2012.
11. Edmund Clarke, Thomas Henzinger, Helmut Veith, and Roderick. Bloem. *Handbook of Model Checking*. Springer International Publishing, 2016.
12. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.
13. Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.
14. William Feller. *An introduction to probability theory and its applications*. Wiley, 1971.

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

16 Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012.

17 Ashutosh Gupta, Corneliu Popeea, and Andrey Rybalchenko. Predicate abstraction and refinement for verifying multi-threaded programs. In *POPL 2011*, pages 331–344, 2011.

18 Thomas A. Henzinger and Jan Otop. Model measuring for discrete and hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 23:166–190, 2017.

19 Andrew Hinton, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS 2006*, pages 441–444, 2006.

20 Alexander Kechris. *Classical descriptive set theory*, volume 156. Springer Science & Business Media, 2012.

21 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. J. Algebr. Comput.*, 4(3):405–426, 1994. doi:10.1142/S0218196794000063.

22 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electr. Notes Theor. Comput. Sci.*, 153(2):5–31, 2006.

23 Christos H Papadimitriou. *Computational complexity*. Wiley, 2003.

24 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional markov decision processes. In *CAV 2015*, pages 123–139, 2015.

25 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189–201, 1979.

26 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS 1985*, pages 327–338. IEEE Computer Society, 1985.

Ergodic Mean-Payoff Games for the Analysis of Attacks in Crypto-Currencies

Krishnendu Chatterjee
IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria
krishnendu.chatterjee@ist.ac.at

Amir Kafshdar Goharshady
IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria
amir.goharshady@ist.ac.at

Rasmus Ibsen-Jensen
IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria
ribsen@ist.ac.at

Yaron Velner
Hebrew University of Jerusalem, Jerusalem, Israel
yaron.welner@mail.huji.ac.il

Abstract

Crypto-currencies are digital assets designed to work as a medium of exchange, e.g., Bitcoin, but they are susceptible to attacks (dishonest behavior of participants). A framework for the analysis of attacks in crypto-currencies requires (a) modeling of game-theoretic aspects to analyze incentives for deviation from honest behavior; (b) concurrent interactions between participants; and (c) analysis of long-term monetary gains. Traditional game-theoretic approaches for the analysis of security protocols consider either qualitative temporal properties such as safety and termination, or the very special class of one-shot (stateless) games. However, to analyze general attacks on protocols for crypto-currencies, both stateful analysis and quantitative objectives are necessary. In this work our main contributions are as follows: (a) we show how a class of concurrent mean-payoff games, namely ergodic games, can model various attacks that arise naturally in crypto-currencies; (b) we present the first practical implementation of algorithms for ergodic games that scales to model realistic problems for crypto-currencies; and (c) we present experimental results showing that our framework can handle games with thousands of states and millions of transitions.

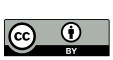
2012 ACM Subject Classification Software and its engineering → Formal software verification

Keywords and phrases Crypto-currency, Quantitative Verification, Mean-payoff Games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.11

Related Version A full version of the paper is available at [10], `https://arxiv.org/abs/1806.03108`.

Acknowledgements The research was partially supported by Vienna Science and Technology Fund (WWTF) Project ICT15-003, Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE), ERC Starting Grant (279307: Graph Games), and an IBM PhD Fellowship.



29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 11; pp. 11:1–11:17

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

1 Introduction

Economic effects of security violations. Traditionally, automated security analysis of protocols using game-theoretic frameworks focused on qualitative properties, such as safety or liveness [26, 16, 1], to ensure absolute security. In many cases absolute security is too expensive, and security violations are inevitable. In such scenarios rather than security, the economic implications of violations should be accounted for. In general, economic consequences of security violations are hard to measure. However, there is a new application area of crypto-currencies, in which the economic impact of an attack can be measured in terms of the number of coins that are lost. These currencies have considerable market value, in the order of hundreds of billions of dollars [18], thus developing a framework to formally analyze the security violations and their economic consequences for crypto-currencies is an interesting problem.

Crypto-currencies. There are many active crypto-currencies today, some with considerable market values. Currently, the main crypto-currency is Bitcoin with a value of over 150 billion dollars at the time of writing [18]. Virtually all of these currencies are free from outside governance and authority and are not controlled by any central bank. Instead, they work based on the decentralized *blockchain* protocol. This protocol, which was first developed for monetary transactions in Bitcoin [31], sets down the rules for creating new units of currency and valid transactions. However, it only defines the outcomes of actions taken by involved parties and cannot dictate the actions themselves. So, the whole ecosystem operates in a game-theoretic manner. The lack of an authority also leads to irreversibility of transactions, so if an amount of currency is transferred unintentionally or due to a bug, it cannot be reclaimed. This, together with the huge market values, makes it imperative to develop formal methods for quantifying the economic consequences before deploying the protocols.

Dishonest interaction. The fact that protocols define only the outcomes of actions and do not force the actions themselves, means that in some scenarios they might give one of the parties unfair or unintended advantage over others and an incentive to act dishonestly, i.e. to take an unintended action. Such behavior is called an attack. We succinctly describe some attacks.

- The most fundamental attack in every crypto-currency is *double-spending*, where one party could in some circumstances use the same coin twice in two different purchases. While this vulnerability is inherent in every blockchain protocol, people still use crypto-currencies as the probability (and the economic consequences) of such an attack can be bounded over time.
- Another line of attacks follow from dishonest behavior of the *blockchain miners* who are responsible for the underlying security of the blockchain protocol and are rewarded for their operations. It was shown that undesirable behavior, such as block withholding [19] or selfish mining [20], could increase the dishonest miner's reward, at the expense of other (honest) miners. We explain the block withholding attack in more detail in Section 5.1.

Research Questions. Analyzing attacks on crypto-currencies requires a formal framework to handle: (a) game-theoretic aspects and incentives for dishonest behavior; (b) simultaneous interaction of the participants; and (c) quantitative properties corresponding to long-term monetary gains and losses. These properties cannot be obtained from standard temporal or qualitative properties which have been the focus of previous game-theoretic

frameworks [26, 16]. On the other hand, game-theoretic incentives are also analyzed in the security community (e.g., see [8]), but their methods are normally considering the very special case of one-shot (stateless) or short-term games. One-shot games cannot model the different states of the ecosystem or the history of actions taken.

Concurrent mean-payoff games. These games were introduced in the seminal work of Shapley [37], and later extended by Gillette [22]. A concurrent mean-payoff game is played by two players over a finite state space, where at each state both players simultaneously choose actions. The transition to the next state is determined by their joint actions, and each transition is assigned a reward. The goal of one player is to maximize the long-run average of the rewards, and the other player tries to minimize it. These games provide a very natural and general framework to study stateful games with simultaneous interactions and quantitative objectives. They lead to a very elegant and mathematically rich framework, and the theoretical complexity of such games has been studied for six decades [37, 22, 5, 24, 30, 14, 23]. However, the analysis of concurrent mean-payoff games is computationally intractable and no practical (such as strategy-iteration) algorithms exist to solve these games. Existing algorithmic approaches either require the theory of reals and quantifier elimination [14] or have doubly-exponential time complexity in the number of states [23].

Our contributions. Our main contributions are as follows:

1. **Modeling.** We propose to model long-term (infinite-horizon) economic aspects of security violations as concurrent mean-payoff games, between the attacker and the defender. The guaranteed payoff in the game corresponds to the maximal loss of the defender. In particular, for blockchain protocols, where the utility of every transition is naturally measurable, we show how to model various interesting scenarios as a sub-class of concurrent mean-payoff games, namely, *concurrent ergodic games*. In these games all states are visited infinitely often with probability 1.
2. **Practical implementation.** Second, while for concurrent ergodic games a theoretical algorithm (strategy-iteration algorithm) exists that does not use theory of reals and quantifier elimination, no previous implementation exists. Moreover, the implementation of the theoretical algorithm poses practical challenges: (a) the algorithm guarantees convergence only in the limit; and (b) the algorithm requires high numerical precision and the straightforward implementation of the algorithm does not converge in practice. We present (i) a simple stopping criterion for approximation, and (ii) resolve the numerical precision problem; and to our knowledge present the first practical implementation of a solver for concurrent ergodic games.
3. **Experimental results.** Finally, we present experimental results and show that the solver for ergodic games scales to thousands of states and nearly a million transitions to model realistic analysis problems from crypto-currencies. Note that in comparison, approaches for general concurrent mean-payoff games cannot handle even ten transitions (see the Remark in Section 3). Thus we present orders of magnitude of improvement.

2 Crypto-Currencies

Monetary system. A crypto-currency is a *monetary system* that allows secure transactions of currency units and dictates how new units are formed. Each transaction has a unique id and the following components: (i) a set of inputs; and (ii) a set of outputs and (iii) locking scripts. Each input has a pointer to an output of a previous transaction, and each output

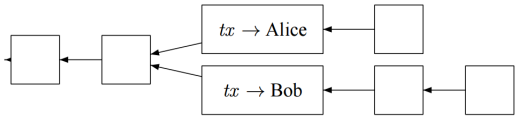


Figure 1 The longest chain dictates that the transaction tx belongs to Bob.

has an assigned monetary value. A locking script on an output defines a condition for using the funds stored in that output, e.g. the need for a digital signature. An input can only use funds of an output by passing its locking script.

Validity. A transaction is *valid* if these conditions hold: (a) the total value brought by the outputs is greater than or equal to the total value of the inputs; (b) the inputs have not been spent before; (c) the inputs satisfy locking scripts.

A transaction-based system is not secure if transactions are sent directly between users to transfer units. While validity conditions are enough to make sure that only valid recipients could redirect units they once truly held, there is nothing in the transactions themselves to limit the user from spending the same output twice (in two different transactions). For this purpose a public ledger of all valid transactions, called a *blockchain*, is maintained.

Blockchain. A ledger is a distributed database that maintains a growing *ordered* list of *valid* transactions. Its main novelty is that it enforces consensus among untrusted and possibly adversarial parties [31]. In Bitcoin (and most other major crypto-currencies) the public ledger is implemented as a series of *blocks* of transactions, each containing a reference to its previous block, and is hence called a blockchain. A consensus on the chain is obtained by a decentralized pseudonymous protocol. Any party tries to collect new transactions, form a block and add it to the chain (this process is called block mining). However, in order to do so, they must solve a challenging computational puzzle (which depends on the last block of the chain). The process of choosing the next block is as follows:

1. The first announced valid block that solves the puzzle is added to the chain.
2. If two valid blocks are found approximately at the same time (depending on network latency), then there is a temporary fork in the chain.

Every party is free to choose either fork, and try to extend it. Hence, the underlying structure of the blockchain is a tree. At any given time, the longest path in the tree, aka the *longest chain*, is the consensus blockchain (see Figure 1). Due to the random nature of the computational puzzle one branch will eventually become strictly longer than the other, and all parties will adopt it.

Mining process. The puzzle asks for a block consisting of valid transactions, hash of the previous block and an arbitrary integer *nonce*, whose hash is less than a target value. The random nature of the hash function dictates a simple strategy for mining: try random nonces until a solution is found. So the chance of a miner to find the next block is proportional to their computational power.

Incentives for mining. There are two incentives for miners: (i) Every transaction can donate to the miner who finds a new block that contains it, (ii) Each block creates a certain number of new coins which are then given to the miner.

Pool mining. To lower the variance of their revenue, miners often collaborate in *pools* [35, 8]. The pools have a manager who collects the rewards from valid blocks found by the members and allocates funds to them in proportion to the amount of work they did. Members prove their work by sending *partial solution* blocks, which are blocks with valid transactions but lower difficulty level, i.e., the hash of the block is not smaller than the network threshold, but it is lower than some threshold that was defined by the manager. As a result, pool members obtain lower variance in rewards, but have a small drop in expected revenue to cover the manager's fee. Members will get the same reward for a partial and full solution, but the member cannot claim the full block reward for themselves. More precisely, a block also dictates where the block reward goes to. Hence, even if a member broadcasts the new block, the reward will still go to the manager.

Proof of stake mining. An emerging criticism over the huge amount of energy that is wasted in the mining process led to development of *proof of stake protocols*. In proof of stake mining the miner is elected with probability that is proportional to their *stake* in the network (i.e., number of coin units he holds), rather than their computation power. Current proof of stake protocols assume a synchronous setting [32, 40, 28] where a miner is chosen in every time slot t_0. However, they differ in the way they reach consensus. We study a simplified version of [28].

1. At time t_0 a miner is randomly elected. She broadcasts the next block.
2. Until time $t_0 + t$ other miners who receive the block, verify it and if it were valid, sign it and broadcast the signature.
3. The block is added to the chain only if a majority of the network sign it.

To encourage honest behavior, the elected miner and signers get rewards when the suggested block is accepted.

3 Concurrent and Ergodic Games

Probability distributions. For a finite set A, a *probability distribution* on A is a function $\delta\colon A \to [0,1]$ such that $\sum_{a\in A} \delta(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. Given a distribution $\delta \in \mathcal{D}(A)$, we denote by $\mathrm{Supp}(\delta) = \{x \in A \mid \delta(x) > 0\}$ the *support* of the distribution.

Concurrent game structures. A *concurrent stochastic game structure* $G = (S, A, \Gamma_1, \Gamma_2, \delta)$ has the following components:

- A finite state space S and a finite set A of actions (or moves).
- Two move assignments $\Gamma_1, \Gamma_2\colon S \to 2^A \setminus \emptyset$. For $i \in \{1,2\}$, assignment Γ_i associates with each state $s \in S$ the non-empty set $\Gamma_i(s) \subseteq A$ of moves available to Player i at state s.
- A probabilistic transition function $\delta\colon S \times A \times A \to \mathcal{D}(S)$, which associates with every state $s \in S$ and moves $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$, a probability distribution $\delta(s, a_1, a_2) \in \mathcal{D}(S)$ for the successor state.

We denote by n the number of states (i.e., $n = |S|$), and by m the maximal number of actions available for a player at a state (i.e., $m = \max_{s\in S} \max\{|\Gamma_1(s)|, |\Gamma_2(s)|\}$). The size of the transition relation of a game structure is defined as
$|\delta| = \sum_{s\in S} \sum_{a_1\in\Gamma_1(s)} \sum_{a_2\in\Gamma_2(s)} |\mathrm{Supp}(\delta(s, a_1, a_2))| \leq n^2 \cdot m^2$.

Plays. At every state $s \in S$, Player 1 chooses a move $a_1 \in \Gamma_1(s)$, and simultaneously and independently Player 2 chooses a move $a_2 \in \Gamma_2(s)$. The game then proceeds to the successor state t with probability $\delta(s, a_1, a_2)(t)$, for all $t \in S$. A *path* or a *play* of G is an infinite

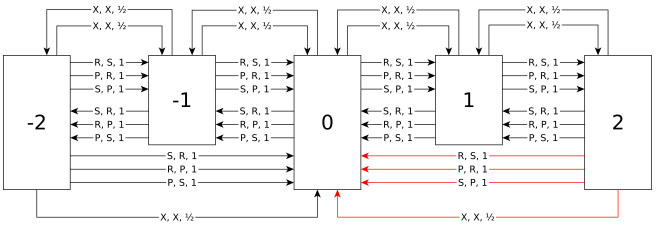


Figure 2 A repetitive rock-paper-scissors game.

sequence $\pi = \big((s_0, a_1^0, a_2^0), (s_1, a_1^1, a_2^1), (s_2, a_1^2, a_2^2) \dots\big)$ of states and action pairs such that for all $k \geq 0$ we have (i) $a_i^k \in \Gamma_i(s_k)$; and (ii) $s_{k+1} \in \mathrm{Supp}(\delta(s_k, a_1^k, a_2^k))$. We denote by Π the set of all paths.

▶ **Example 1.** Consider a repetitive game of rock-paper-scissors, consisting of an infinite number of laps, in which each lap is made of a number of rounds as illustrated in Figure 2. When a lap begins, the two players play rock-paper-scissors repetitively until one of them wins 3 rounds more than her opponent, in which case she wins the current lap of the game and a new lap begins. In each round, the winner is determined by the usual rules of rock-paper-scissors, i.e. rock beats scissors, scissors beat paper and paper beats rock. In case of a tie, each player wins the round with probability $\frac{1}{2}$.

Here we have $S = \{-2, -1, 0, 1, 2\}$ and $\Gamma_1 = \Gamma_2 \equiv \{\mathrm{R}, \mathrm{P}, \mathrm{S}\}$. The game starts at state 0 and state s corresponds to the situation where Player 1 has won s rounds more than Player 2 in the ongoing lap. Edges in the figure correspond to possible transitions in the game. Each edge is labeled with three values a_1, a_2, p to denote that the game will transition from the state at the beginning of the edge to the state at its end with probability p if the two players decide on actions a_1 and a_2, respectively. For example, there is an edge from state 2 to state 0 labeled $\mathrm{R}, \mathrm{S}, 1$, which corresponds to $\delta(2, \mathrm{R}, \mathrm{S})(0) = 1$. In the figure, we use X, X in place of a_1, a_2 to denote that they are equal. Hence every *play* in this game corresponds to an infinite walk on the graph in Figure 2.

Strategies. A *strategy* is a recipe to extend prefixes of a play. Formally, a strategy for Player i is a mapping $\sigma_i \colon (S \times A \times A)^* \times S \to \mathcal{D}(A)$ that associates with every finite sequence $x \in (S \times A \times A)^*$ of state and action pairs, representing the past history of the game, and the current state s in S, a probability distribution $\sigma_i(x \cdot s)$ used to select the next move. The strategy σ_i can only prescribe moves that are available to Player i; that is, for all sequences $x \in (S \times A \times A)^*$ and states $s \in S$, we require $\mathrm{Supp}(\sigma_i(x \cdot s)) \subseteq \Gamma_i(s)$. We denote by Σ_i the set of all strategies for Player i. Once the starting state s and the strategies σ_1 and σ_2 for the two players have been chosen, then the probabilities of measurable events are uniquely defined [39]. For an event $\mathcal{A} \subseteq \Pi$, we denote by $\Pr_s^{\sigma_1,\sigma_2}(\mathcal{A})$ the probability that a path belongs to $\mathcal{A}$ when the game starts from s and the players use the strategies σ_1 and σ_2. We call a pair of strategies $(\sigma_1, \sigma_2) \in \Sigma_1 \times \Sigma_2$ a *strategy profile*.

Stationary (memoryless) and positional strategies. In general, strategies use randomization, and can use finite or even infinite memory to remember the history. Simpler strategies, that either do not use memory, or randomization, or both, are significant, as they are simple to implement and interpret. A strategy σ_i is *stationary* (or memoryless) if it is independent of the history but only depends on the current state, i.e., for all $x, x' \in (S \times A \times A)^*$ and all $s \in S$, we have $\sigma_i(x \cdot s) = \sigma_i(x' \cdot s)$, and thus can be expressed as a function $\sigma_i : S \to \mathcal{D}(A)$.

A strategy is *pure* if it does not use randomization, i.e., for any history there is a unique action a that is played with probability 1. A pure stationary strategy σ_i is called *positional*, and denoted as a function $\sigma_i : S \to A$.

Mean-payoff objectives. We consider maximizing *limit-average* (or mean-payoff) objectives for Player 1, and the objective of Player 2 is the opposite (i.e., the games are zero-sum). We consider concurrent games with a reward function $\mathrm{R} : S \times A \times A \to \mathbb{R}$ that assigns a reward value $\mathrm{R}(s, a_1, a_2)$ for all $s \in S$, $a_1 \in \Gamma_1(s)$, and $a_2 \in \Gamma_2(s)$. For a path $\pi = \big((s_0, a_1^0, a_2^0), (s_1, a_1^1, a_2^1), \dots\big)$, the average for T steps is $\mathsf{Avg}_T(\pi) = \frac{1}{T} \cdot \sum_{i=0}^{T-1} \mathrm{R}(s_i, a_1^i, a_2^i)$, and the limit-inferior average (resp. limit-superior average) is defined as follows: $\mathsf{LimInfAvg}(\pi) = \liminf_{T\to\infty} \mathsf{Avg}_T(\pi)$ (resp. $\mathsf{LimSupAvg}(\pi) = \limsup_{T\to\infty} \mathsf{Avg}_T(\pi)$). We denote concurrent mean-payoff games as CMPGs.

▶ **Example 2.** Consider the game in Figure 2. In this game, Player 1 wins a lap whenever a red edge is crossed. Therefore, in order to capture the number of laps won by Player 1, rewards can be assigned as: $R(2, R, S) = R(2, P, R) = R(2, S, P) = 1$; $R(2, X, X) = \frac{1}{2}$ and 0 in all other cases.

Values and ϵ-optimal strategies. Given a CMPG G and a reward function R, the *lower value* $\underline{v}_s$ (resp. the *upper value* $\overline{v}_s$) at a state s is defined as follows:
$\underline{v}_s = \sup_{\sigma_1\in\Sigma_1} \inf_{\sigma_2\in\Sigma_2} \mathbb{E}_s^{\sigma_1,\sigma_2}[\mathsf{LimInfAvg}]$; $\overline{v}_s = \inf_{\sigma_2\in\Sigma_2} \sup_{\sigma_1\in\Sigma_1} \mathbb{E}_s^{\sigma_1,\sigma_2}[\mathsf{LimSupAvg}]$.

The *determinacy* result of [30] shows that the upper and lower values coincide and give the *value* of the game denoted as v_s. For $\epsilon \geq 0$, a strategy σ_1 for Player 1 is *ϵ-optimal* if we have $v_s - \epsilon \leq \inf_{\sigma_2\in\Sigma_2} \mathbb{E}_s^{\sigma_1,\sigma_2}[\mathsf{LimInfAvg}]$.

Ergodic Games. A CMPG G is *ergodic* if for all states $s, t \in S$, for all strategy profiles (σ_1, σ_2), if we start at s, then t is visited infinitely often with probability 1 in the random walk $\pi_s^{\sigma_1,\sigma_2}$. The game in Figure 2 is not ergodic. If Player 1 keeps playing rock and Player 2 scissors, then the states -1 and -2 are visited at most once. However, a more realistic version of this game is also ergodic.

▶ **Example 3.** Consider two players playing the repetitive game of rock-paper-scissors over a network, e.g. the Internet. The game is loaded on a central server that asks the players for their moves and provides them with rewards and information about changes in the state of the game. Given that the network is not perfect, there is always a small probability that one of the players is unable to announce his move in time to the server. In such cases, the player will lose the current round. Assume that this scenario happens with probability $\epsilon > 0$. Then all probabilities in Figure 2 have to be multiplied by $(1 - \epsilon)$ and new transitions, which are not under players' control and are a result of uncertainty in the network connection, should be added to the game. These new transitions are illustrated in Figure 3. Here a star can be replaced by any permissible action of the players. It is easy to check that this variant of the game is ergodic, given that starting from any state, there is a positive probability of visiting any other state within 3 steps using the new transitions only.

Results about general CMPGs. The main results for CMPGs are as follows:

1. The celebrated result of existence of values was established in [30].
2. For CMPGs, stationary or finite-memory strategies are not sufficient for optimality, and even in CMPGs with three states (the well-known Big Match game), very complex infinite-memory strategies are required for ϵ-optimality [5].

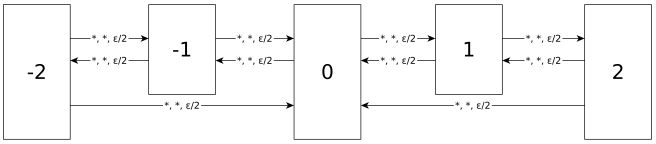


Figure 3 Transitions due to network connectivity issues in the repetitive RPS.

3. The value problem, that given a CMPG, a state s, and a threshold λ, asks whether the value at state s is at least λ, can be decided in PSPACE [14]; and also in $m^{2^{O(n)}}$ time, which is doubly exponential in the worst case, but polynomial-time in m, for n constant [23]. Both the above algorithms use the theory of reals and quantifier elimination for analysis.

▶ Remark (Inefficiency). The quantifier elimination approach for general CMPGs considers formulas in the theory of reals with alternation, where the variables represent the transitions [14]. With as few as ten transitions, quantifier elimination produces formulas with hundreds of variables over the existential theory of reals. In turn, the existential theory of reals has exponential-time complexity, is notoriously hard to solve, and its existing solvers cannot handle hundreds of variables. Hence, CMPGs with as few as ten transitions are not tractable.

Results about ergodic CMPGs. The main results for ergodic CMPGs are as follows:

1. Stationary optimal strategies exist[24], but positional strategies are not sufficient for optimality. For precise strategy complexity see [13].
2. Even in ergodic games, values and probabilities of optimal strategies can be irrational [13], and hence the relevant question is the approximation problem of values which is solvable in non-deterministic polynomial-time [13].
3. The most well-known algorithm for ergodic mean-payoff games is the Hoffman-Karp *strategy-iteration* algorithm [24]. See [10] for a more detailed treatment of this algorithm.

Note that since in ergodic games, every state is reached from every other state with probability 1, the value at all states is the same.

4 Modeling Framework

In this section we present an abstract framework to model economical consequences of attacks with mean-payoff games. In particular we show how broad classes of attacks can be modeled as ergodic games. In the next section we present concrete examples that arise from blockchain protocols.

4.1 Mean-payoff games modeling

We describe two aspects of mean-payoff games modeling.

1. *Game graph modeling.* Graph games are a standard model for reactive systems as well as protocols. The states and transitions of the graph represent states and transitions of the reactive system, and paths in the graphs represent traces of the system [33, 34]. Similarly, in modeling of protocols with different variables for the agents, the states of the game represent various scenarios of the protocols along with the valuation of the variables. The transitions represent a change of the scenario along with change in the valuation of the variables (for example see [16] for game graph modeling of protocols for digital-contract signing).

2. *Mean-payoff objective modeling.* In mean-payoff objectives, the costs (or rewards) of every transition can represent, for example, delays, execution times, cost of context switches, cost of concurrency, or monetary gains and losses. The mean-payoff objective represents the long-term average of the rewards or the costs. The mean-payoff objective has been used for synthesis of better reactive systems [7], synthesis of synchronization primitives for concurrent data-structures to minimize average context-switch costs [9], model resource-usage in container analysis and frequency of function calls [15], as well as analysis of energy-related objectives [3, 2, 21].

4.2 Crypto-currency Protocols as Mean-payoff Games

We describe how to apply the general framework of CMPGs to crypto-currencies:

- *General setting.* We propose to analyze protocols as a game between a defender and an attacker. The defender and the attacker have complete freedom to decide on their moves. The decisions of the other parties in the ecosystem can be modeled as stochastic choices that are not adversarial to either of the players.
- *Reward function.* The reward function will reflect the monetary gain or loss of the defender. The attacker gain is not modeled as we consider the worst-case scenario in which the attacker's objective is to minimize the defender's utility.
- *States.* States of the game can represent the information that is relevant for the analysis of the protocol, such as the abstract state of the blockchain.
- *Stochastic transitions.* Probabilities over the transitions can model true stochastic processes e.g., mining, or abstract complicated situations where the exact behavior cannot be directly computed (see Section 5.2) or in order to simulate the social behavior of a group (see Section 5.1).
- *Concurrent interactions.* Concurrent games are used when both players decide on their action simultaneously or when a single action models a behavior that continues over a time period and the players can only reason about their opponent's behavior after a while (Sections 5.1 and 5.2).
- *Result of the game.* In this work we want to reason on defender's security in a protocol wrt a malicious attacker who aims to decrease defender's gain at any cost. The result of the mean-payoff game will describe the inevitable expected loss that the defender will have in the presence of an attacker and defender's strategy describes the best way to defend himself against such an attacker.

4.3 Modeling with Ergodic Games

In this section we describe two classes of attacks, which can be naturally modeled with ergodic games. Our description here is high-level and informal, and concrete instances are considered in the next section. The attacks we describe are in a more general setting than crypto-currencies; however, for crypto-currencies the economic consequences are more natural to model.

First class of attacks. In the first class of attacks the setting consists of two companies and the revenues of the companies depend on the number of users each has. Thus states represent the number of users. Each company can decide to attack its competing company. Performing an attack entails some economic costs, however it could increase the number of users of the attacking company at the expense of the attacked one. For example, consider two competing social networks, Alice and Bob. Alice can decide to launch a distributed-denial-of-service

(DDOS) attack on Bob, and vice-versa. Such attacks entail a cost, but provide incentives for Bob users to switch to Alice. The rewards depend on the network revenues (i.e., number of users) and on the amount of funds the company decides to spend for the attack. The migration of users is a stochastic process that is biased towards the stronger network, but with smaller probability some users migrate to the other network. Thus the game is ergodic. This class represents pool attacks in the context of crypto-currencies (Sections 5.1 and 5.3).

Second class of attacks. Consider the scenario where the state of the game represents aspects of the dynamic network topology. The network evolves over the course of the time, and the actions of the participants also affect the network topology. However, the effect of the actions only makes local changes. The combination of the global changes and the local effects still ensure that different network states can be reached, and the game is ergodic. Attacks in such a scenario where the network topology determines the outcome of attack can be modeled as ergodic games. This class of attacks represent the zero-confirmation double-spending attack in the context of crypto-currencies (see Section 5.2).

5 Formal Modeling of Real Attacks

In this section we show how to model several real-world examples. These examples were described in the literature but were never analyzed as stateful games.

5.1 Block Withholding Pool Attack

Pools are susceptible to the classic block withholding attack [35], where a miner sends only partial solutions to the pool manager and discards full solutions. In this section we analyze block withholding attacks among two pools, pool A and pool B. We describe how pool A can attack pool B, and the converse direction is symmetric. To employ the pool block withholding attack, pool A registers at pool B as a regular miner. It receives tasks from pool B and transfers them to some of its own miners. Following the notions in [19], we call these infiltrating miners, and their mining power is called infiltration rate. When pool A's infiltrating miners deliver partial solutions, pool A's manager submits them to pool B's manager and proves the portion of work they did. When the infiltrating miners deliver a full solution, the attacking pool manager discards it.

At first, the total revenue of the victim pool does not change (as its effective mining rate was not changed), but the same sum is now divided among more miners. Thus, since the pool manager fees are nominal (fixed percentage of the total revenue [4]), in the short term, the manager of the victim pool will not lose. The attacker's mining power is reduced, since some of its miners are used for block withholding, but it earns additional revenue through its infiltration of the other pool. Finally, the total effective mining power in the system is reduced, causing the blockchain protocol to reduce the difficulty. Hence, in some scenarios, the attacker can gain, even in the short run, from performing the attack [19].

In the long run, if miners see a decrease in their profits (since they have to split the same revenue among more participants), it is likely that they consider to migrate to other pools. As a result, the victim pool's total revenue will decrease.

Our modeling. We aim to capture the long term consequences of pool attacks. We have two pools A and B, where B is the victim pool and A is the malicious pool who wishes to decrease B's profits. There is also a group of miners C who are honest and represent the rest of the network. In return, pool B can defend itself by attacking back. To simulate the

long term effect, in every round pool members from A and B may migrate from one pool to another or to and from C. The migration is a stochastic process that favors the pool with maximum profitability for miners. We note that given sufficient amount of time (say a week), a pool manager can evaluate with very high probability the fraction of infiltrating miners in his pool. This can be done by looking at the ratio between full and partial solutions. Hence, in retrospect of a week, the pools are aware of each other's decisions, but within this week there is uncertainty. Therefore, we use concurrent games to analyze the worst case scenario for pool B.

▶ **Theorem 4.** *Consider a pair of pools A and B capable of attacking each other. Let C be the pool of remaining miners. If the miners in each pool migrate stochastically according to the attractiveness levels (as detailed below), then B can ensure a revenue of at least v on average per round, against any behavior of A, where v is the value of the concurrent ergodic game described below.*

5.1.1 Details of Modeling

We provide details of our modeling on some of the attacks to demonstrate how they can be thought of in terms of ergodic games. Details of all other attacks can be found in [10].

- *Game states.* We consider two pools, A and B and assume that any miner outside these two is mining independently for himself. Each state is defined by two values, i.e. the fractions of total computation power that belongs to A and B. We use a discretized version of this idea to model the game in a finite number of states and let $S = \{1, 2, \ldots, n\}^2$ and define $\epsilon = \frac{1}{2n+1}$, where a state $(i_1, i_2) \in S$ corresponds to the case where pool A owns a fraction $\alpha_{i_1} = i_1\epsilon = \frac{i_1}{2n+1}$ of the total hash power and pool B controls a fraction $\beta_{i_2} = i_2\epsilon = \frac{i_2}{2n+1}$ of it. In this case the miners who work independently own a fraction $\gamma_{i_1,i_2} = 1 - \alpha_{i_1} - \beta_{i_2}$ of the total hash power.
- *Actions at each state.* Each pool can choose how much of its hash power it devotes to attacking the other pool. More formally, at each state $s = (i_1, i_2)$, pool A has i_1 choices of actions and $\Gamma_1(s) = \{a_1^0, a_1^1, a_1^2, \ldots, a_1^{i_1-1}\}$ where a_1^j corresponds to attacking pool B with a fraction $j\epsilon$ of the total computing power of the network. Similarly $\Gamma_2(s) = \{a_2^0, a_2^1, a_2^2, \ldots, a_2^{i_2-1}\}$.
- *Rewards.* We want the rewards to model the revenue (profit) of pool A, denoted by r_A, so we let $R(s, a_1^i, a_2^j) = r_A(s, a_1^i, a_2^j)$, for $a_1 \in \Gamma_1(s), a_2 \in \Gamma_2(s)$. We write r_A instead of $r_A(s, a_1^i, a_2^j)$ when there is no risk of confusion. We define r_B and r_C similarly and normalize the revenues: $r_A + r_B + r_C = 1$. To compute these values, we define "attractiveness". The attractiveness of a pool is its revenue divided by the total computing power of its miners. If pool A chooses the action a_1^i and pool B chooses the action a_2^j, then pool A is using a fraction $\alpha' = i\epsilon$ of the total network computing power to attack B and is receiving a corresponding fraction of B's revenue while not contributing to it. Therefore the attractiveness of pool B will be equal to: $attr_B = \frac{r_B}{\beta+\alpha'}$. Similarly we have $attr_A = \frac{r_A}{\alpha+\beta'}$, where $\beta' = j\epsilon$.
  Now consider the sources for pool A's revenue. It either comes from A's own mining process or from collecting shares of B's revenue, therefore:

  $$r_A = (\alpha - \alpha') + \alpha' \times attr_B,$$

  and similarly $r_B = (\beta - \beta') + \beta' \times attr_A$. The previous four equations provide us with a system of linear equations which we can solve to obtain the values of r_A, r_B, $attr_A$ and $attr_B$. Since a fraction $\alpha' + \beta'$ of total computation power is used on attacking other pools, we have: $attr_C = \frac{1}{1-\alpha'-\beta'}$.

- *Game transitions (δ).* Miners migrate between pools and a pool gains or loses mining power based on its attractiveness. If a pool is the most attractive option among the two, it gains ϵ new mining power with probability $\frac{2}{3}$, retains its current power with probability $\frac{1}{6}$ and loses ϵ power with probability $\frac{1}{6}$. On the other hand a pool that is not the most attractive option loses ϵ power with probability $\frac{2}{3}$, retains its current power with probability $\frac{1}{6}$ and attracts ϵ new mining power with probability $\frac{1}{6}$. These values were chosen for the purpose of demonstration of our algorithm and our implementation results. In practice, one can obtain realistic probabilities experimentally.
- *Ergodicity.* The game is ergodic because for each two states $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$ where $|s_1 - s'_1| \leq 1$ and $|s_2 - s'_2| \leq 1$, there is at least $\frac{1}{36}$ probability of going from s to s' no matter what choices the players make.

Proof of Theorem 4. Ergodicity was established in the final part above. The rest follows from the modeling and the determinacy result.

5.2 Zero-confirmation Double-spending

Nowadays, Bitcoin is increasingly used in "fast payments" such as online services, ATM withdrawals and vending machines [17], where the payment is followed by fast delivery of goods. While the blockchain consensus is appropriate for slow payments, it requires tens of minutes to confirm a transaction and is therefore inappropriate for fast payments. We consider a transaction confirmed when it is added to the blockchain and several blocks are added after it. This mechanism is essential for the detection of double-spending attacks in which an adversary attempts to use some of her coins for two or more payments. However, even in the absence of a confirmation, it is far from trivial to perform a double-spending attack. In a double spending attack, the attacker publishes two transactions that consume the same input. The attack is successful only if the victim node received one transaction and provided the goods before he became aware of the other, but eventually the latter was added to the blockchain. In an ideal world the attacker can increase his odds by broadcasting one transaction directly to the victim and the other at a far apart location, while on the other hand the victim can defend itself by deploying several nodes in the network in *strategic* locations. In the real world, however, the full topology of the network is never known to either of the parties. Nevertheless, based on history and network statistics one can estimate the odds of a successful attack given the current state of the network [6].

The victim has to decide on a policy for accepting zero-confirmation transactions. In particular he has to decide on the probability of whether to wait for a confirmation or not. If he waits for confirmation, then the payment is guaranteed, but customer satisfaction is damaged, and as a result the utility is smaller than the actual payment. If he does not wait for a confirmation, then the payment might be double spent. In the long term, the victim could decide to change the topology of the network. As it does not have full control over the topology, the outcome of the change is stochastic. Moreover, even when the victim does not initiate a change, the network topology is dynamic and keeps changing all the time. Hence, the odds of a successful attack are constantly changing in small stochastic steps.

Our modeling. We aim to analyze the worst case long run loss of the victim. In our model we abstract the network topology state and consider only the odds of successful double spending. We consider a scenario where the victim's honest customers typically purchase goods worth 10 units per round. In every round, the victim decides on a policy for accepting fast payment, and the attacker, concurrently, unaware of the victim's policy, has to decide the

size of the attack. After every round, the victim decides if he wants to do a thorough change in the network topology. If he decides on a change, then the next state is chosen uniformly from all possible states (this represents the fact that neither players has full knowledge on the topology). If he decides to make no change, then the network state might still change, due to the dynamic nature of the network. In this case the next state is with high probability either the current state, or a state which is slightly better or slightly worse for the victim, but with low probability the state changes completely to an arbitrary state in the network (as sometimes small changes in the topology have big impact). The rewards stem from the outcome of each round in the following way: The payment is the sum of the honest customer purchases and the payment of the attacker (if it gets into the blockchain). The reward is the payment minus some penalty in case the victim has decided to wait for a confirmation. The fact that the network state is constantly changing makes our model ergodic.

▶ **Theorem 5** (Proof in [10]). *Consider a seller and an attacker in the zero-confirmation double spending problem. The seller can ensure profit of at least v on average per round, where v is the value of the corresponding CMPG.*

5.3 Proof of Stake Pool Attack

Proof of stake protocols let miners centralize their stakes in a pool. In such pools the withholding attack is not relevant as mining does not require physical resources. However, pool A might attack an opponent pool B by not signing or broadcasting its blocks. A successful attack would prevent the block from getting signed by a majority of the network and result in a loss of mining fees for B and can encourage miners to migrate from B. An unsuccessful attack decreases A's signing revenue.

Our modeling. We assume a setting similar to that of Section 5.1, where there are two opponent pools A and B, and the rest of the network consists of honest pools who sign every block that arrives on time. The states of the game are the stakes of each pool, namely α for pool A and β for pool B. In every round, with probability $1-(\alpha+\beta)$ neither of the pools is elected to mine a block, and no decisions are made. Otherwise, with probability $\frac{\alpha}{\alpha+\beta}$ pool A is elected and otherwise pool B is elected. When a pool is elected, the other pool decides whether to sign and broadcast the resulting block or not. In addition the network state and connectivity induce a distribution over the fraction of honest miners that receive the block. If the block is accepted, then its creator is rewarded with mining fees, and the other pool will get its signing fees only if it signed the block.

▶ **Theorem 6** (Proof in [10]). *Consider two pools A and B in a proof of stake mining system that can choose to attack each other by not signing blocks mined by the other pool. Consider that the rest of the network consists of independent miners who observe published blocks according to a predefined probability distribution and sign every valid block they observe. If the miners migrate according to the attractiveness levels (as described in Section 5.1), then B can ensure an average revenue of v against any behavior of A, where v is the value of the corresponding CMPG.*

6 Implementation and Experimental Results

Implementation. We have implemented the strategy-iteration algorithm for ergodic games (see [10] for pseudo-code and more details). The implementation is available at `http://ist.ac.at/~akafshda/concur2018`. To the best of our knowledge, this is the first implementation

Table 1 Experimental results for block-withholding pool attack (left), zero-confirmation double-spending (center) and proof of stake pool attack (right).

#T	States	#SI	Time(s)
17050	100	4	69
56252	196	2	291
135252	289	2	389
236000	400	2	1059
331816	484	2	3880
508032	576	2	6273
720954	676	2	17014
966281	784	2	53103
1269450	900	2	100435

#T	States	#SI	Time(s)
19940	100	2	426
40040	200	2	800
60140	300	2	1141
80240	400	2	1586
100340	500	2	2069
120440	600	2	1253
140540	700	2	2999
160640	800	2	3496
180740	900	2	3917

#T	States	#SI	Time(s)
6076	99	18	471
20956	275	8	1338
31744	396	9	2520
44764	539	4	1073
77500	891	16	22125
119164	1331	27	32636
169756	1859	10	31597
262384	2816	12	89599

of this algorithm. The straightforward implementation of the strategy-iteration algorithm for ergodic games has two practical problems, which we describe below.

1. *No stopping criteria.* First, the strategy-iteration algorithm only guarantees convergence of values in the limit, and since values and probabilities in strategies can be irrational, convergence cannot be guaranteed in a finite number of steps. Hence we need a stopping criterion for approximation.
2. *Numerical precision issues.* Second, the stationary strategies in each iteration are obtained by solving LPs, which has numerical errors, and the probabilities sum to less than 1. If these errors remain, they cascade over iterations, and do not ensure convergence in practice for large examples. Hence we need to ensure numerical precision on top of the strategy-iteration algorithm.

Our solution for the above two problems are as follows:

1. *Stopping criteria.* We first observe that the value sequence which is obtained converges from below to the value of the game. In other words, the value sequence provide a lower bound to the lower value of the game. Hence we consider a symmetric version which is the strategy-iteration algorithm for player 2, and run each iteration of the two algorithms in sequence. The version for player 2 provides a lower bound on the lower value for player 2, and thus from that we can obtain an upper bound on the upper value of player 1. Since the upper and lower values coincide, we thus have both an upper and lower bound on the values, and once the difference is smaller than $\epsilon > 0$, then the algorithm has correctly approximated the value within ϵ and can stop and return the value and the strategy obtained as approximation.
2. *Numerical precision.* For numerical precision, instead of obtaining the results from the linear program, we obtain the set of *tight* and *slack* constraints, where the tight constraints represent the constraints where equality is obtained, and the other constraints are slack ones. From the tight constraints, which are equalities, we obtain the result using Gaussian elimination, which provides more precise values to the solution. We also tried other heuristics, such as adding the remaining probability to the greatest probability action, which led to similar results on convergence.

Experimental Results. Our experimental results are reported in Table 1. We show number of transitions in the game (#T), number of states in the game, the running time and number of strategy iterations (#SI). It is noteworthy that in all cases the number of iterations required is quite small. We also note that since the number of iterations is small, the crucial computational step is every iteration, where many LPs are solved. The outputs provided the following results (more details in [10]):

- For the block withholding pool attack game, the algorithm could guarantee a mean-payoff of 0.49 for the victim pool. In absence of an attacker the mean-payoff will be 1.
- For the zero-confirmation double-spending game, the algorithm verified that the seller is guaranteed to maintain at least half of her revenue, i.e., in presence of a malicious attacker, the value for the seller converges to 5 as the number of states increase, while it is 10 in absence of it.
- For the proof of stake pool attack game, by increasing the number of states, i.e., by refining the discretization, the guaranteed value (game value) decreases and tends to zero. In absence of an attacker, a pool A can achieve an expected payoff of $11s_A$ at a turn where s_A is the stake it holds.

7 Related Work

- *Pools attack.* The danger of a block withholding attack is as old as Bitcoin pools. The attack was described by Rosenfeld [35], as pools were becoming a dominant player in Bitcoin. While it was obvious that a pool is vulnerable to a malicious attacker, Eyal [19] showed that in some circumstances a pool can benefit by attacking another pool, and thus pool mining is vulnerable also in the presence of rational attackers. However, the analysis only considered the short term, i.e., the profit that the pool can get only in the short period after the attack. Laszka et al. [29] studied the long term impact of pools attack. In their framework miners are allowed to migrate from one pool to another. They analyzed the steady equilibrium in which the size of the pools become stable (although there is no guarantee that the game will converge to such a scenario). Our framework is the first to allow analysis of long term impacts without convergence assumptions.
- *Zero-confirmation double-spending.* Zero-confirmation double-spending was experimentally analyzed by Karame et al. [25] who gave numerical figures for the odds of successful double spending for different network states. However, their analysis did not consider that the victim may change his connectivity state. Our work is the first analysis of the long term impact of this attack.
- *Stateful analysis.* A stateful analysis of blockchain attacks was done by Sapirshtein et al. [36] and by Sompolinsky and Zohar [38]. In their analysis the different states of the blockchain were taken into account during the attack. The analysis was done using MDPs in which only the attacker decides on his actions and the victim follows a predefined protocol. A recent work [11] also considers abstraction-refinement for finite-horizon games based on smart contracts. However, it neither considers long-term behavior, nor mean-payoff objectives, nor can it model attacks such as double-spending and interactions between pools.
- *Quantitative verification with mean-payoff games.* The mean-payoff games problem has been studied extensively as a theoretical problem [33, 34]. It has also been studied in the context of verification and synthesis for performance related issues [7, 9, 15, 3, 2, 21]. However, all these works focus on turn-based games, and none of them consider concurrent games. To the best of our knowledge concurrent mean-payoff games have not been studied in the setting of security that we consider, where the quantitative objective is as crucial as safety critical issues. Practical implementation of algorithms for ergodic CMPGs do not exist in the literature.

8 Conclusion and Future Work

In this work we considered concurrent mean-payoff games, and in particular the subclass of ergodic games, to analyze attacks on crypto-currencies. There are several interesting directions to pursue: First, various notions of rationality are relevant to analyze games where the attacker is rational, rather than malicious, and aims to maximize his own utility instead of minimizing the defender's utility (e.g., secure-equilibria [12] or other related notions). Second, we consider two-player games, and the extension to multi-player games to model crypto-currency attacks is another interesting problem. Third, the modeling assumptions should be empirically validated and the parameters used to generate the games, e.g. the rates of migration, should be empirically obtained. Fourth, we consider the rest of the network to be neutral and stochastic. An interesting extension would be to consider a rational network, possibly consisting of coalitions of cooperating miners, as defined e.g. in [27].

References

1 M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *Proceedings of the IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer, 2000.

2 C. Baier, C. Dubslaff, J. Klein, S. Klüppelholz, and S Wunderlich. Probabilistic model checking for energy-utility analysis. In *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, pages 96–123, 2014.

3 C. Baier, S. Klüppelholz, H. de Meer, F. Niedermeier, and S. Wunderlich. Greener bits: Formal analysis of demand response. In *ATVA*, pages 323–339, 2016.

4 Bitcoin Wiki. Comparison of mining pools, 2017. URL: `http://en.bitcoin.it/Comparison_of_mining_pools`.

5 D. Blackwell and T. Ferguson. The big match. *The Annals of Mathematical Statistics*, 39(1):159–163, 1968.

6 blockcypher.com. Confidence factor, 2017. URL: `http://dev.blockcypher.com/#confidence-factor`.

7 R. Bloem, K. Chatterjee, T.A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156, 2009.

8 J. Bonneau, A. Miller, J. Clark, A. Narayanan, J.A. Kroll, and E.W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 104–121. IEEE, 2015.

9 P. Cerný, K. Chatterjee, T.A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *CAV*, pages 243–259, 2011.

10 K. Chatterjee, A.K. Goharshady, R. Ibsen-Jensen, and Y. Velner. Ergodic mean-payoff games for the analysis of attacks in crypto-currencies. *arXiv*, 2018. `arXiv:1806.03108`.

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

12 K. Chatterjee, T.A. Henzinger, and M. Jurdzinski. Games with secure equilibria. In *LICS*, pages 160–169, 2004.

13 K. Chatterjee and R. Ibsen-Jensen. The complexity of ergodic mean-payoff games. In *ICALP II*, pages 122–133, 2014.

14 K. Chatterjee, R. Majumdar, and T. A. Henzinger. Stochastic limit-average games are in EXPTIME. *Int. J. Game Theory*, 37(2):219–234, 2008.

15 K. Chatterjee, A. Pavlogiannis, and Y. Velner. Quantitative interprocedural analysis. In *POPL*, pages 539–551, 2015.

16 K. Chatterjee and V. Raman. Assume-guarantee synthesis for digital contract signing. *Formal Asp. Comput.*, 26(4):825–859, 2014.

17 CNN Money. Bitcoin's uncertain future as currency, 2011. URL: `http://money.cnn.com/video/technology/2011/07/18/t_bitcoin_currency.cnnmoney/`.

18 coinmarketcap.com. Crypto-currency market capitalizations, 2017. URL: `http://coinmarketcap.com/`.

19 I. Eyal. The miner's dilemma. In *IEEE Symposium on Security and Privacy*, pages 89–103. IEEE, 2015.

20 I. Eyal and E.G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, 2014.

21 V. Forejt, M. Z. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *ATVA*, pages 317–332, 2012.

22 D. Gillette. Stochastic games with zero stop probabilitites. In *CTG*, pages 179–188. Princeton University Press, 1957.

23 K. A. Hansen, M. Koucký, N. Lauritzen, P. B. Miltersen, and E. P. Tsigaridas. Exact algorithms for solving stochastic games: extended abstract. In *STOC*, pages 205–214, 2011.

24 A.J. Hoffman and R.M. Karp. On nonterminating stochastic games. *Management Sciences*, 12(5):359–370, 1966.

25 G. Karame, E. Androulaki, and S. Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012:248, 2012.

26 S. Kremer and J.F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 2003.

27 Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Strategic analysis of trust models for user-centric networks. In *SR*, pages 53–59, 2013.

28 J. Kwon. Tendermint: Consensus without mining, 2015. URL: `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/`.

29 A. Laszka, B. Johnson, and J. Grossklags. When bitcoin mining pools run dry. In *International Conference on Financial Cryptography and Data Security*, pages 63–77. Springer, 2015.

30 J.F. Mertens and A. Neyman. Stochastic games. *IJGT*, 10:53–66, 1981.

31 S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

32 NxtCommunity. Nxt whitepaper, 2014. URL: `http://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf`.

33 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

34 P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

35 Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.

36 A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. *arXiv preprint arXiv:1507.06183*, 2015.

37 L.S. Shapley. Stochastic games. *PNAS*, 39:1095–1100, 1953.

38 Y. Sompolinsky and A. Zohar. Bitcoin's security model revisited. *CoRR*, abs/1605.09193, 2016.

39 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *FOCS*, pages 327–338. IEEE, 1985.

40 V. Zamfir. Introducing casper, the friendly ghost, 2015. URL: `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/`.

Bounded Context Switching for Valence Systems

Roland Meyer
TU Braunschweig, Germany
roland.meyer@tu-braunschweig.de
https://orcid.org/0000-0001-8495-671X

Sebastian Muskalla
TU Braunschweig, Germany
s.muskalla@tu-braunschweig.de
https://orcid.org/0000-0001-9195-7323

Georg Zetzsche[1]
IRIF (Université Paris-Diderot, CNRS), France
zetzsche@irif.fr
https://orcid.org/0000-0002-6421-4388

Abstract

We study valence systems, finite-control programs over infinite-state memories modeled in terms of graph monoids. Our contribution is a notion of bounded context switching (BCS). Valence systems generalize pushdowns, concurrent pushdowns, and Petri nets. In these settings, our definition conservatively generalizes existing notions. The main finding is that reachability within a bounded number of context switches is in NP, independent of the memory (the graph monoid). Our proof is genuinely algebraic, and therefore contributes a new way to think about BCS. In addition, we exhibit a class of storage mechanisms for which BCS reachability belongs to P.

2012 ACM Subject Classification Theory of computation → Parallel computing models, Theory of computation → Formal languages and automata theory, Theory of computation → Logic and verification

Keywords and phrases valence systems, graph monoids, bounded context switching

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.12

Related Version The full version is available on arXiv [47], `https://arxiv.org/abs/1803.09703`.

1 Introduction

Bounded context switching (BCS) is an under-approximate verification technique typically applied to safety properties. It was introduced for concurrent and recursive programs [50]. There, a context switch happens if one thread leaves the processor for another thread to be scheduled. The analysis explores the subset of computations where the number of context switches is bounded by a given constant. Empirically, it was found that safety violations occur within few context switches [48, 45]. Algorithmically, the complexity of the analysis drops from undecidable to NP [50, 26]. The idea received considerable interest from both practice and theory, a detailed discussion of related work can be found below.

[1] Supported by a fellowship of the Fondation Sciences Mathématiques de Paris and partially funded by the DeLTA project (ANR-16-CE40-0007).

Our contribution is a generalization of bounded context switching to programs operating over arbitrary memories. To be precise, we consider valence systems, finite-control programs equipped with a potentially infinite-state memory modeled as a monoid [23, 56, 57]. In valence systems, both the data domain and the operations are represented by monoid elements, and an operation o will change the current memory value m to the product $m \cdot o$. Of course, the monoid has to be given in some representation.

We consider so-called graph monoids that capture the memories commonly found in programs, like stacks, counters, and tapes, but also combinations thereof. A graph monoid is represented by a graph. Each vertex is interpreted as a symbol (say c) on which the operations push (c^+) and pop (c^-) are defined. A computation is a sequence of such operations. The edges of the graph define an independence relation among the symbols that is used to commute the corresponding operations in a computation. To give an example, if c and d are independent, the computation $d^+.c^+.d^-$ acts on two counters c and d and yields the values 1 and 0, respectively. Pushdowns are represented by valence systems over graphs without edges and concurrent pushdowns by complete m-partite graphs (for m stacks). Petri nets yield complete graphs, blind counter systems complete graphs with self-loops on all vertices.

Our definition of context switches concentrates on the memory and does not reference the control flow. This frees us from having to assume a notion of thread, and makes the analysis applicable to sequential programs as well. We define a context switch as two consecutive operations in a computation that act on different and independent (in the above sense) symbols. This conservatively generalizes existing notions and yields intuitive behavior where a notion of context switch is not defined. When modeling concurrent pushdowns, a context switch indeed corresponds to switching the stack. For Petri nets and blind counter systems, it means switching the counter. Note, however, that the restriction can be applied to all memories expressible in terms of graph monoids.

Our main result shows that reachability within a bounded number of context switches is in NP, *for all graph monoids*. The result requires a uniform representation for the computations over very different memories. We prove that a computation can always be split into quadratically-many blocks (in the number of context switches) – independent of the monoid. These blocks behave like single operations in that they commute or form inverses (in the given monoid). With this decomposition result, we develop an automata-theoretic approach to checking reachability. A more elaborate explanation of the proof approach can be found in Section 3, where we have the required terminology at hand.

In addition, we investigate the precise complexity of the problem for individual graph monoids. While there are graph monoids for which our problem is NP-complete (such as those corresponding to the setting of concurrent pushdowns), we show that for an important subclass, those induced by transitive forests, the problem can be solved in polynomial time. Moreover, we describe those graph monoids for which the problem is NL-complete.

Taking a step back, our approach provides the first algebraic view to context-bounded computation, and hence enriches the tool box so far containing graph-theoretic interpretations and logical encodings of computations. We elaborate on the related work.

Related Work. There are two lines of work on BCS that are closely related to ours in that they apply to various memory structures. Aiswarya [6] and Madhusudan and Parlato [46] define a graph-theoretic interpretation of computations that manipulate a potentially infinite memory. They restrict the analysis to computations where graph-based measures like the split-width or the tree-width are bounded, and obtain general decidability results by reductions to problems on tree automata. The graph interpretation has been applied to multi

pushdowns [7], timed systems [9, 10], and has been generalized to controller synthesis [8]. It also gives a clean formulation of existing restrictions and uniformizes the corresponding analysis algorithms, in particular for [50, 36, 37, 40, 31]. Different from under-approximations based on split- or tree-width, we are able to handle counters, even nested within stacks. We cannot handle, however, the queues to which those technique apply. Indeed, our main result is NP-completeness whereas graph-based analyses may have a higher complexity. Our approach thus applies to an incomparable class of models. Moreover, it contributes an algebraic view to bounded computations that complements the graph-theoretic interpretation.

The second line of related work are reductions of reachability under BCS to satisfiability in existential Presburger arithmetic [26, 30]. Hague and Lin propose an expressive model, concurrent pushdowns communicating via reversal-bounded counters. Their main result is NP-completeness, like in our setting. The model does not admit the free combination of stacks and counters that we support. The way it is presented, we in turn do not handle reversal boundedness, where the counters may change as long as the mode (increasing/decreasing) does not switch too often. Our approach should be generalizable to reversal boundedness by replacing the emptiness test in the free automata reduction of Section 5 by a satisfiability check, using [53]. The details remain to be worked out. Besides providing an incomparable class of models, our approach complements the logical view to computations.

Reductions to existential Presburger arithmetic often restrict the set of computations by an intersection with a bounded language [29], like in [26, 5]. The importance of bounded languages for under-approximation has been observed by Ganty et al. [28, 25].

Besides the above unifying approaches, there has been a body of work on generalizations of BCS, towards exploring a larger set of computations [36, 41, 24, 12, 52, 2] and handling more expressive programming models [37, 14, 31, 16]. An unconventional instantance of the former direction are restrictions to the network topology [15]. As particularly relevant instantiations of the latter, the BCS under-approximation has been applied to programs operating on relaxed memories [13, 4] and programs manipulating data bases [3].

The practical work on BCS concentrated on implementing fast context-bounded analyses. Sequentialization techniques [51] were successful in bridging the gap between the parallel program at hand and the available tooling, which is often limited to sequential programs. The idea is to translate the BCS instance into a sequential safety verification problem. The first sequentialization for BCS has been proposed in [42], [38] gave a lazy formulation, and [17] a systematic study of when sequentialization can be achieved. The approach now applies to full C-programs [33] and has won the concurrency track in the softare verification competition. Current work is on parallelizing the analysis by further restricting the interleavings and in this way obtaining instances that are easier to solve [49].

Also with the goal of parallelization, recent works study the multi-variate complexity of context-bounded analyses. While [26, 27] focus on P and NP, [20] studies fixed-parameter tractability, and [21] the fine-grained complexity. The goal of the latter work is to achieve an analysis of comlexity $2^k poly(n)$, with k a parameter and n the input size. Ideally, this analysis could be performed by 2^k independent threads, each solving a poly-time problem.

Our results contribute to a line of work on valence systems over graph monoids [57]. They have previously been studied with respect to elimination of silent transitions [55], semi-linearity of Parikh images [19], decidability of unrestricted reachability [58], and decidability of first-order logic with reachability [23]. See [56] for a general overview.

2 Valence Systems over Graph Monoids

We introduce the basics on graph monoids and valence systems following [57].

Graph Monoids. Let $G = (V, I)$ be an undirected graph, without parallel edges, but possibly with self-loops. This means $I \subseteq V \times V$, which we refer to as the *independence relation*, is symmetric but neither necessarily reflexive nor necessarily anti-reflexive. We use infix notation and write $o_1 \; I \; o_2$ for $(o_1, o_2) \in I$.

To understand how the graph induces a monoid (a memory), think of the nodes $o \in V$ as stack symbols or counters. To each symbol o, we associate two operations, a positive operation o^+ that can be understood as *push* o or *increment* o and a negative operation o^-, *pop* o or *decrement* o. We call $+$ and $-$ the polarity of the operation. By $o^\pm$ we denote an arbitrary element from $\{o^+, o^-\}$. Let $\mathcal{O} = \{o^+, o^- \mid o \in V\}$ denote the set of all operations. We refer to sequences of operations from $\mathcal{O}^*$ as computations. We lift the independence relation to operations by setting ${o_1}^\pm \; I \; {o_2}^\pm$ if $o_1 \; I \; o_2$. We also write $v_1 \; I \; v_2$ for $v_1, v_2 \in \mathcal{O}^*$ if the operations in the computations are pairwise independent, and similar for subsets of operations $\mathcal{O}_1 \; I \; \mathcal{O}_2$ with $\mathcal{O}_1, \mathcal{O}_2 \subseteq \mathcal{O}$.

We obtain the monoid by factorizing the set of all computations. The congruence will identify computations that order independent operations differently. Moreover, it will implement that o^+ followed by o^- should have no effect, like a push followed by a pop. Formally, we define $\cong$ as the smallest congruence (with respect to concatenation) on $\mathcal{O}^*$ containing ${o_1}^\pm.{o_2}^\pm \cong {o_2}^\pm.{o_1}^\pm$ for all $o_1 \; I \; o_2$ and $o^+.o^- \cong \varepsilon$ for all o.

The *graph monoid for graph* G is $\mathbb{M}_G = \mathcal{O}^*/_\cong$. For a word $w \in \mathcal{O}^*$, we use $[w]_\mathbb{M} \in \mathbb{M}_G$ to denote its equivalence class. Multiplication is $[u]_\mathbb{M} \cdot [v]_\mathbb{M} = [u.v]_\mathbb{M}$, which is well-defined as $\cong$ is a congruence. The neutral element of $\mathbb{M}_G$ is the equivalence class of ε, $1_\mathbb{M} = [\varepsilon]_\mathbb{M}$.

Recall that an element x of a monoid M is called *right-invertible* if there is $y \in M$ such that $x \cdot y = 1_M$. We lift this notation to $\mathcal{O}^*$ by saying that $w \in \mathcal{O}^*$ is *right-invertible* if its equivalence class $[w]_\mathbb{M} \in \mathbb{M}_G$ is.

Valence Systems. Given a graph G, a *valence system* over the graph monoid $\mathbb{M}_G$ is a pair $A = (Q, \to)$, where Q is a finite set of control states and $\to \subseteq Q \times (\mathcal{O} \cup \{\varepsilon\}) \times Q$ is a set of transitions. A transition $q_1 \xrightarrow{x} q_2$ is labeled by an operation on the memory. We write $q_1 \to q_2$ if the label is ε, indicating that no operation is executed. The size of A is $|A| = |{\to}|$. We use $\mathcal{O}(A)$ to access the set of operations that label transitions in A.

A *configuration* of A is a tuple $(q, w) \in Q \times \mathcal{O}^*$ consisting of a control state and the sequence of storage operations that has been executed. We will restrict ourselves to configurations where w is right-invertible. More precisely, in (q, w) a transition $q_1 \xrightarrow{x} q_2$ is *enabled* if $q = q_1$ and $w.x$ is right-invertible. In this case, the transition leads to the configuration $(q_2, w.x)$, and we write $(q, w) \to (q_2, w.x)$. A *run* is a sequence of consecutive transitions.

This restriction to right-invertible configurations is justified by the definition of the *reachability problem* for valence systems. It asks, given a valence system with two states q_{init}, q_{fin}, whether we can reach q_{fin} with neutral memory from q_{init} with neutral memory, i.e. whether there is a run from (q_{init}, ε) to (q_{fin}, w) with $[w]_\mathbb{M} = 1_\mathbb{M}$. To be able to reach such a configuration (q_{fin}, w) from some configuration (q, w'), w' has to be right-invertible.

Examples. Figure 1 depicts various graphs. The graph monoid of each of these graph models a commonly used storage mechanism, i.e. it represents the behavior of the storage.
(a) Valence systems for this graph are pushdown systems over the stack alphabet $\{a, b, c\}$.

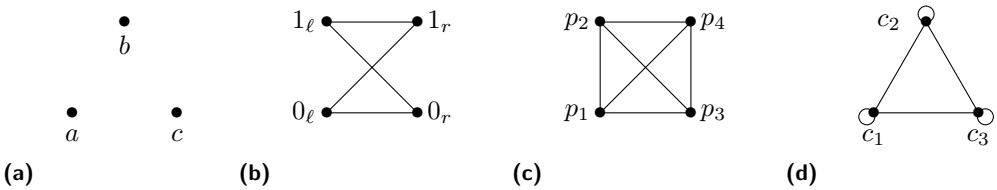


Figure 1 Various examples of graphs representing commonly used storage mechanism.

(b) Valence systems for this graph can be seen as concurrent pushdown systems with two stacks, each over a binary alphabet.

(c) Petri nets resp. vector addition systems with four counters/places p_1, p_2, p_3, p_4 can be modeled as valence systems for this graph. Since the valence system labels transitions by single increments or decrements, the transition multiplicities are encoded in unary.

(d) Integer vector addition systems resp. blind counter automata with counters c_1, c_2, c_3 (that may assume negative values) can be seen as valence systems for this graph.

What about Queues? Let us quickly comment on why it is hard to fit queues into this framework. An appealing aspect of valence automata over graph monoids is that by using the monoid identity as the target for reachability problems (resp. as an acceptance condition [19, 55, 57, 58]), we can realize a range of storage mechanisms by only varying the underlying monoid. This is because in the mechanisms that we can realize, the actions (or compositions of actions) that transform the empty storage into the empty storage are precisely those that equal the identity transformation.

In order to keep this aspect, we would need to construct a monoid whose generators can be interpreted as queue actions so that a sequence of generators transforms the empty queue into the empty queue if and only if this sequence evaluates to the identity of the monoid. This, however, is not possible: Suppose that a and b represent enqueue operations and that $\bar{a}$ and $\bar{b}$ are the corresponding dequeue operations. Each of the action sequences $a.\bar{a}$ and $b.\bar{b}$ transforms the empty queue into the empty queue, but $a.b.\bar{b}.\bar{a}$ does not (it is undefined on the empty queue). Hence, in the monoid, we would want to have $a\bar{a} = 1$, $b\bar{b} = 1$, but $ab\bar{b}\bar{a} \neq 1$, which violates associativity. Hence, although it is possible to model queue behavior in a monoid [32, 34, 35], one would need a different target element (or set).

3 Bounded Context Switching

We introduce a notion of bounded context switching that applies to all valence systems, over arbitrary graph monoids. The idea is to let a new context start with an operation that is independent of the current computation, and hence intuitively belongs to a different thread. We elaborate on the notion of dependence.

We call a set of symbols $V' \subseteq V$ *dependent*, if it does not contain $o_1, o_2 \in V$, $o_1 \neq o_2$ with $o_1 \; I \; o_2$. A set of operations $\mathcal{O}' \subseteq \mathcal{O}$ is dependent if its underlying set of symbols $\{o \mid o^+ \in \mathcal{O}' \text{ or } o^- \in \mathcal{O}'\}$ is. A computation is dependent if it is over a dependent set of operations. A valence system is said to be dependent if the operations labeling the transitions form a dependent set.

▶ **Definition 3.1.** Given $w \in \mathcal{O}^+$, its context decomposition is defined inductively: If w is dependent, w is a single context and does not decompose. Else, the first context w_1 of w is the (non-empty) maximal dependent prefix of w. Then, the context decomposition of w is

$w = w_1, \ldots, w_k$, where $w_2, \ldots, w_k$ is the context decomposition of the rest of the word. The number of context switches in w, $cs(w)$, is the number of contexts minus one. For technical reasons, it will be convenient to define $cs(\varepsilon) = -1$.

We study reachability under a restricted number of context switches.

Reachability under bounded context switching (BCSREACH)	
Given:	Valence system A, initial state q_{init}, final state q_{fin}, bound k in unary.
Decide:	Is there a run from (q_{init}, ε) to (q_{fin}, w) so that $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ and $cs(w) \leqslant k$?

In all abovementioned graph monoids, the restriction has an intuitive meaning that generalizes existing results. Using the finite states, our notion of BCS also permits a finite shared memory among the threads. In addition, our definition applies to all storage structures expressible in terms of graph monoids, including combinations like stacks of counters.

▶ **Lemma 3.2.** (BCSREACH) *yields the following restriction:*

(1) *On pushdowns, the notion does not incur a restriction.*

(2) *On concurrent pushdowns, the notion corresponds to changing the stack k-times and hence yields the original definition [50].*

(3) *On Petri nets and blind counters, the notion corresponds to changing the counter k-times.*

Our main result is this.

▶ **Theorem 3.3.** (BCSREACH) *is in* NP*, independent of the storage graph.*

Note that the NP upper bound matches the lower bound in the case of concurrent pushdowns [39]. We consider the proof technique the main contribution of the paper. Different from existing approaches, which are based on graph interpretations of computations or encodings into Presburger, ours is of algebraic nature. With an algebraic analysis, given in Section 4, we simplify the problem of checking whether a given computation reduces to one, $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$. We show that such a reduction exists if and only if the computation admits a decomposition into so-called blocks that reduce to one in a strong sense. There are two surprising aspects about the block decomposition. First, the strong reduction is defined by either commuting two blocks or canceling them if they are inverses. This means the blocks behave like operations, despite being full subcomputations. Second, the decomposition yields only quadratically-many blocks in the number of context switches (important for NP-membership). The block decomposition is the main technical result of the paper.

The second step, presented in Section 5, is a symbolic check for whether a computation exists whose block decomposition admits a strong reduction. We rely on automata-theoretic techniques to implement the operations of a strong reduction. Key is a saturation based on which we give a complete check of whether two automata accept blocks that are inverses.

4 Block Decomposition

In this section, we show how to decompose a computation that reduces to the neutral element into polynomially-many blocks such that the decomposition admits a syntactic reduction to ε. The size of the decomposition will only depend on the number of contexts of the computation and not on its length. This result will later provide the basis for our algorithm.

To be precise, we restrict ourselves to computations with so-called irreducible contexts. In the next section, we will prove that the restriction to this setting is sufficient.

▶ **Definition 4.1.** We call a computation $w \in \mathcal{O}^*$ *irreducible* if it cannot be written as $w = w'.a.w_I.b.w''$ such that $a = o^+$, $b = o^-$ and o commutes with every symbol in w_I, or $a = o^-, b = o^+, o\ I\ o$ and o commutes with every symbol in w_I.

In other words, a computation is irreducible if we cannot eliminate a pair $o^+.o^-$ after using commutativity. This is in fact the standard definition of irreducibility in the so-called trace monoid, which we do not introduce here.

Our goal is to decompose irreducible contexts such that the decomposition of all contexts in the computation admits a syntactic reduction defined as follows.

▶ **Definition 4.2** ([44])**.** Let $w_1, w_2, \ldots, w_n$ be a sequence of computations in $\mathcal{O}^*$. A *free reduction* is a finite sequence of applications of the following rewriting rules to consecutive entries of the sequence that transforms $w_1, \ldots, w_n$ into the empty sequence.
(FR1) $w_i, w_j \mapsto_{free} \varepsilon$, applicable if $[w_i.w_j]_{\mathbb{M}} = 1_{\mathbb{M}}$.
(FR2) $w_i, w_j \mapsto_{free} w_j, w_i$, applicable if $w_i\ I\ w_j$
We call $w_1, w_2, \ldots, w_n$ *freely reducible* if it admits a free reduction.

Being freely reducible is a strictly stronger property than $[w_1.w_2.\ldots.w_n]_{\mathbb{M}} = 1_{\mathbb{M}}$: It means that the sequence can be reduced to $1_{\mathbb{M}}$ by block-wise canceling, Rule (FR1), and swapping whole blocks, Rule (FR2). Indeed, consider ${o_1}^+.{o_2}^+, {o_2}^-, {o_1}^-$ where no two symbols commute. We have $[{o_1}^+.{o_2}^+.{o_2}^-.{o_1}^-]_{\mathbb{M}} = 1_{\mathbb{M}}$, but the sequence is not freely reducible.

The decomposition of a computation w with $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ into its single operations is always freely reducible. The main result of this section is that for a computation with irreducible contexts, we can always find a freely-reducible decomposition whose length is independent of the length of the computation.

▶ **Theorem 4.3.** *Let w be a computation with $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ and let $w = w_1 \ldots w_k$ be its decomposition into irreducible contexts. There is a decomposition of each $w_i = w_{i,1}.w_{i,2} \ldots w_{i,m_i}$ such that $m_i \leqslant k - 1$ and the sequence*

$$w_{1,1}, w_{1,2}, \ldots, w_{1,m_1}, w_{2,1}, w_{2,2}, \ldots, w_{2,m_2}, \ldots, w_{k,1}, w_{k,2}, \ldots, w_{k,m_k}$$

is freely reducible.

Note that the number of words occurring in the decomposition is bounded by k^2. Theorem 4.3 can be seen as a strengthened version of Lemma 3.10 from [44]: We use the bound on the number of contexts to obtain a polynomial-size decomposition instead of an exponential one. However, the proofs of the two results are vastly different.

Constructing a Freely-Reducible Decomposition. The rest of this section will be dedicated to the proof of Theorem 4.3. Let $w \in \mathcal{O}^*$ be the computation of interest with $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$. We assume that it has length n and $w = w_1 \ldots w_k$ is its decomposition into contexts. For the first part of the proof, we do not require that each w_i is irreducible. As $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$, w can be transformed into ε by finitely often swapping letters and canceling out operations. We formalize this by defining transition rules, similar to the definition of a free reduction.

For the technical development, it will be important to keep track of the original position of each operation in the computation. To this end, we see w as a word over $\mathcal{O} \times \{1, \ldots, n\}$, i.e. we identify the x^{th} operation a of w with the tuple (a, x). For ease of notation, we write $w[x]$ for the x^{th} operation of w. The annotation of letters by their original position will be preserved under the transition rules.

▶ **Definition 4.4.** A *reduction* of w is a finite sequence of applications of the following rewriting rules that transforms w into into ε.

(R1) $w'.w[x].w[y].w'' \mapsto_{red} w'.w''$, applicable if $w[x] = o^+$, $w[y] = o^-$ for some o.

(R2) $w'.w[x].w[y].w'' \mapsto_{red} w'.w''$, applicable if $w[x] = o^-$, $w[y] = o^+$ for $o\ I\ o$.

(R3) $w'.w[x].w[y].w'' \mapsto_{red} w'.w[y].w[x].w''$, applicable if $w[x] \in {o_1}^{\pm}, w[y] \in {o_2}^{\pm}$ for $o_1\ I\ o_2$, $o_1 \neq o_2$.

If a word u can be transformed into v using these rules, we write $u \mapsto^*_{red} v$. Note that a reduction of w to ε can be seen as a free reduction of the sequence we obtain by decomposing w into single operations.

▶ **Lemma 4.5.** *For a word w, we have $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ iff w admits a reduction.*

Consequently, we may fix a reduction $\pi = w \mapsto^*_{red} \varepsilon$ that transforms w into ε. The following definitions will depend on this fixed π.

▶ **Definition 4.6.** We define a relation R_π that relates positions of w that cancel in π, i.e.

$$w[x]\ R_\pi\ w[y] \quad \text{if} \quad w'.w[x].w[y].w'' \mapsto_{red} w'.w'' \text{ or } w'.w[y].w[x].w'' \mapsto_{red} w'.w'' \text{ is used in } \pi\ .$$

We lift it to infixes of w by defining inductively

$$t_1 s_1\ R_\pi\ s_2 t_2 \quad \text{if} \quad \begin{array}{l}\text{there are contexts } w_i = w_{i1}.t_1.s_1.w_{i2} \text{ and } w_j = w_{j1}.s_2.t_2.w_{j2} \\ \text{of } w \text{ such that } s_1\ R_\pi\ s_2 \text{ and } t_1\ R_\pi\ t_2\ .\end{array}$$

An infix u of a context w_i is called a *cluster* if there is an infix u' of a context w_j such that $u\ R_\pi\ u'$. Moreover, if u is a maximal cluster in w_i, then it is called a *block*.

Note that R_π is symmetric by definition. In the following, when we write $s_1\ R_\pi\ s_2$, we will assume that s_1 appears before s_2 in w, i.e. $w = w'.s_1.w''.s_2.w'''$. We now show that each context has a unique decomposition into blocks. Afterwards, we will see that the resulting block decomposition is the decomposition required by Theorem 4.3.

▶ **Lemma 4.7.** *Every context has a unique factorization into blocks.*

To prove the lemma, we show that each position belongs to at least one block and to at most one block. We call the unique factorization of a context w_i into blocks the *block decomposition* of w_i (induced by π) and denote it by

$$w_i = w_{i,1}, \ldots, w_{i,m_i}.$$

The *block decomposition* of w (induced by π) is the concatenation of the block decompositions of its contexts,

$$w = w_{1,1}, \ldots, w_{1,m_1}, \ldots, w_{k,1}, \ldots, w_{k,m_k}\ .$$

Note that if u is a block and $u\ R_\pi\ v$, then v is a block as well. Therefore, R_π is a one-to-one correspondence of blocks. It remains to prove that the block decomposition of w admits a free reduction. We will show that we can inductively cancel out blocks pairwise, starting with an *innermost* pair. Being innermost is formalized by the following relation.

▶ **Definition 4.8.** We define relation $\leqslant_w$ on R_π-related pairs of blocks by $(s_1\ R_\pi\ s_2) \leqslant_w (t_1\ R_\pi\ t_2)$ if $w = w^{(1)}.t_1.w^{(2)}.s_1.w^{(3)}.s_2.w^{(4)}.t_2.w^{(5)}$ for appropriately chosen $w^{(1)}, \ldots, w^{(5)}$. A pair $s_1\ R_\pi\ s_2$ minimal wrt. this order is called *minimal nesting* in w.

Note that we still assume that all letters are annotated by their position. This means if $w^{(1)}, \dots, w^{(5)}$ exist, they are uniquely determined.

▶ **Lemma 4.9.** *$\leqslant_w$ has a minimal nesting.*

The next lemma states that $s_1 \; R_\pi \; s_2$ implies that s_2 is (a representative of) a right inverse of s_1. While we already know that the operations in s_1 cancel with those in s_2, it could ostensibly be the case that $[s_2]_{\mathbb{M}}$ is a left-inverse to $[s_1]_{\mathbb{M}}$.

▶ **Lemma 4.10.** *If $s_1 \; R_\pi \; s_2$, then $[s_1.s_2]_{\mathbb{M}} = 1_{\mathbb{M}}$.*

▶ **Proposition 4.11.** *Let $\pi \colon w \to^*_{red} \varepsilon$ be a reduction of w. The block decomposition of w induced by π is freely reducible.*

Proof. If $w = \varepsilon$, then there is nothing to do. Otherwise, w decomposes into at least two blocks. We proceed by induction on the number of blocks. In the base case, let us assume that $w = u, v$ is the block decomposition, where $u \; R_\pi \; v$ has to hold. Using Lemma 4.10, $u, v \xmapsto{\text{(FR1)}}_{free} \varepsilon$ is the desired free reduction.

In the inductive step, we pick a minimal nesting $s_1 \; R_\pi \; s_2$ in w. As argued in Lemma 4.9, this is always possible. We may write

$$w = w_1 \dots \underbrace{w_{i_1} s_1 w_{i_2}}_{\text{context } w_i} \dots \underbrace{w_{j_1} s_2 w_{j_2}}_{\text{context } w_j} \dots w_k \; .$$

Since $s_1 \; R_\pi \; s_2$, we know that by definition of R_π, π has to move each letter from s_1 next to the corresponding letter of s_2 or vice versa.

Let us consider the effect of π on the infix $w_{i_2} \dots w_{j_1}$. Without further arguments, the reduction π could cancel some letters inside this infix, and it can swap the remaining letters with the letters in s_1 or s_2. In fact, there can be no canceling within $w_{i_2} \dots w_{j_1}$, as $s_1 \; R_\pi \; s_2$ was chosen to be a minimal nesting: Assume that $w_{i_2} \dots w_{j_1}$ contains some letters a, b with $a \; R_\pi \; b$. Pick the unique blocks u, v to which they belong, and note that we have $(u \; R_\pi \; v) <_w (s_1 \; R_\pi \; s_2)$, i.e. $(u \; R_\pi \; v) \leqslant_w (s_1 \; R_\pi \; s_2)$ and $(u, v) \neq (s_1, s_2)$, a contradiction to the minimality of $s_1 \; R_\pi \; s_2$.

Hence, the reductions needs to swap all letters in $w_{i_2} \dots w_{j_1}$ with s_1 or s_2 and we have $s_1 \; I \; w_{i_2} \dots w_{j_1} \; I \; s_2$. We construct a free reduction as follows:

$$\begin{aligned}
& w_1 \dots w_{i_1} s_1 w_{i_2} w_{i+1} \dots w_{j-1} w_{j_1} s_2 w_{j_2} \dots w_k \\
\xmapsto{\text{(FR2)}}{}^*_{free} \quad & w_1 \dots w_{i_1} w_{i_2} w_{i+1} \dots w_{j-1} w_{j_1} s_1 s_2 w_{j_2} \dots w_k \\
\xmapsto{\text{(FR1)}}_{free} \quad & w_1 \dots w_{i_1} w_{i+1} \dots w_{j-1} w_{j_2} \dots w_k =: w' \; .
\end{aligned}$$

The applications of Rule (FR2) are valid as $s_1 \; I \; w_{i_2} \dots w_{j_1} \; I \; s_2$ holds. The application of Rule (FR1) to s_1, s_2 is valid by Lemma 4.10.

Let us denote by w' the result of these reduction steps. We consider the reduction π' that is obtained by restricting π to transitions that work on letters still present in w'. Indeed, π' reduces w' to ε. In particular, for each operation in w', the operation it cancels with is the same in π and π'. Consequently, the relation $R_{\pi'}$ is the restriction of R_π to the operation still occurring in w', and the block decomposition of w' induced by π' is the block decomposition of π minus the blocks s_1, s_2 that have been removed.

We may apply induction to obtain that w' admits a free reduction. We prepend the above reduction steps to this free reduction to obtain the desired reduction for w.

We emphasize the fact that we have not used in the proof that the w_i are contexts. This is important, as the context decompositions of w and w' can differ substantially. Potentially, we have that w consists of four contexts, $w = w_1, s_1, w_2, s_2$, but after canceling s_1 with s_2, w_1 and w_2 merge to a single context, $w' = w_1.w_2$. As we have preserved R_π and its induced block decomposition, this does not hurt the validity of the proof. ◀

A Bound on the Number of Blocks. It remains to prove the desired bound on the number of blocks. To this end, we will exploit that each context w_i is irreducible.

▶ **Proposition 4.12.** *Let w be a computation with irreducible contexts and $\pi\colon w \to^*_{red} \varepsilon$ a reduction. In the block decomposition of w induced by π, $m_i \leqslant k-1$ holds for all i.*

We prove the proposition in the form of two lemmas.

▶ **Lemma 4.13.** *The relation R_π never relates blocks from the same context.*

The following lemma allows us to bound the number of blocks in a context by the total number k of contexts.

▶ **Lemma 4.14.** *For any two contexts w_i and w_j, there is at most one block in w_i that is R_π-related to a block in w_j.*

Proof. Towards a contradiction, assume that some context contains two blocks that are R_π-related to a block from the same context. Let us consider the minimal i such that w_i contains such blocks. Let w_j be the context to which the two blocks are related. By the choice of i, w_i occurs in w before w_j does.

We pick s_1, t_1 as a pair of blocks in w_i canceling with blocks from w_j with minimal distance, i.e. $w_i = w_{i_1} s_1 w_{i_2} t_1 w_{i_3}$ where w_{i_2} contains no block that is canceled by some block in w_j. Let s_2, t_2 be the blocks in w_j such that $s_1 \; R_\pi \; s_2$, $t_1 \; R_\pi \; t_2$. We have to distinguish two cases, depending on the order of occurrence of s_2 and t_2 in w_j. In the first case, we have $w_j = w_{j_1} t_2 w_{j_2} s_2 w_{j_3}$ and thus

$$w = w_1 \dots w_{i-1} \underbrace{w_{i_1} s_1 w_{i_2} t_1 w_{i_3}}_{\text{context } w_i} w_{i+1} \dots w_{j-1} \underbrace{w_{j_1} t_2 w_{j_2} s_2 w_{j_3}}_{\text{context } w_j} w_{j+1} \dots w_k \ .$$

Our goal is to show that w_{i_2} and w_{j_2} have to be empty. We then obtain $s_1 t_1 \; R_\pi \; t_2 s_2$, a contradiction to the definition of blocks as maximal R_π-related infixes in each context.

We start by assuming that w_{i_2} contains some operation b. As π reduces w to ε, w contains some operation c that b cancels with. We first note that c cannot be contained in w_j, as we have chosen s_1, t_1 such that w_{i_2} contains no block that cancels with a block of w_j. Assume that c is contained in the prefix $w_1 \dots w_{i-1} w_{i_1}$. Reduction π either needs to swap b or c with s_1, or it needs to swap s_2 with b (to cancel s_1). In any case, by definition of $\mapsto_{red}$, this means s_1 contains an operation that commutes with b and is distinct from b. However, this is impossible, as s_1 and b are contained in the same context w_i, and contexts do not contain distinct independent symbols. For the same reason, c cannot be contained in the suffix $w_{j_3} w_{j+1} \dots w_k$.

If c is contained in the infix $w_{i+1} \dots w_{j-1}$, π needs to swap b with t_1, or c with t_1, or t_2 with c. In any case, this means t_1 contains an operation that commutes with b and is distinct from b. However, this is impossible, as t_1 and b are contained in the same context w_i, and contexts do not contain distinct independent symbols.

Consequently w_{i_2} needs to be empty. Let us assume that w_{j_2} contains an operation b, and let c denote the operation it cancels with. As for w_{i_2}, we can show that c can

neither be contained in the prefix $w_1 \dots w_{i-1} w_{i_1}$, nor in the suffix $w_{j_3} w_{j+1} \dots w_k$, nor in the infix $w_{i+1} \dots w_{j-1}$. We conclude that w_{j_2} is also empty and obtain a contradiction to the maximality of the blocks as explained above.

It remains to consider the second case, i.e. $w_j = w_{j_1} s_2 w_{j_2} t_2 w_{j_3}$ and

$$w = w_1 \dots w_{i-1} \underbrace{w_{i_1} s_1 w_{i_2} t_1 w_{i_3}}_{\text{context } w_i} w_{i+1} \dots w_{j-1} \underbrace{w_{j_1} s_2 w_{j_2} t_2 w_{j_3}}_{\text{context } w_j} w_{j+1} \dots w_k \ .$$

Reduction π either needs to swap s_1 with t_1 or equivalently s_2 with t_1. Again by definition of $\mapsto_{red}$, this means there is an operation a in s_1 and an operation b in t_1 such that $a \ I \ b$ and a, b have distinct symbols. Since s_1, t_1 and s_2, t_2 belong to the same context, this is impossible. ◀

Lemma 4.13 and Lemma 4.14 together prove Proposition 4.12, finishing the proof of Theorem 4.3.

5 Decision Procedure

Given a valence system A with states q_{init} and q_{fin}, and a bound k, we give an algorithm that checks whether there is a run from (q_{init}, ε) to (q_{fin}, w) such that $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ and $cs(w) \leqslant k$.

Implementing Irreducibility. The theory we have developed above applies to irreducible contexts. To determine the irreducible versions of contexts in A, we define a saturation operation on valence systems. The algebraic idea behind the saturation is the following.

▶ **Lemma 5.1.** *Let w be a dependent computation. Then w can be turned into an irreducible computation by applying the following rules: $o^+.o^- \mapsto \varepsilon$ and, provided $o \ I \ o$, $o^-.o^+ \mapsto \varepsilon$.*

To see the lemma, note that in a dependent computation, reducible operations o^+ and o^- cannot be separated by an operation on a different symbol. Hence, o^+ and o^- are placed side by side (potentially after further reductions). If $o \ I \ o$ does not hold, the first rule is sufficient for the reduction. If $o \ I \ o$ does holds, we may find $o^-.o^+$ and need both rules.

The saturation operation implements these two rules. Since Lemma 5.1 assumes a dependent computation, we consider a dependent valence system $B = (P, \rightsquigarrow)$. The *saturation* is the valence system $sat(B) = (P, \rightsquigarrow_{sat})$ with the same set of control states. The transitions are defined by requiring $\rightsquigarrow \subseteq \rightsquigarrow_{sat}$ and exhaustively applying the following rules:

(1) If $p_1 \overset{o^+}{\rightsquigarrow}_{sat} p \rightsquigarrow^*_{sat} p' \overset{o^-}{\rightsquigarrow}_{sat} p_2$, add an ε-transition $p_1 \rightsquigarrow_{sat} p_2$.

(2) If $p_1 \overset{o^-}{\rightsquigarrow}_{sat} p \rightsquigarrow^*_{sat} p' \overset{o^+}{\rightsquigarrow}_{sat} p_2$ and $o \ I \ o$, add an ε-transition $p_1 \rightsquigarrow_{sat} p_2$.

Here, $p \rightsquigarrow^*_{sat} p'$ denotes that p' is reachable from p by a sequence of ε-transitions.

▶ Remark. In the worst case, we add $|P|^2$ many transitions.

▶ **Lemma 5.2.** *There is a computation $(q_1, \varepsilon) \to (q_2, u)$ in B if and only if there is a computation $(q_1, \varepsilon) \to (q_2, v)$ with v irreducible and $u \cong v$ in $sat(B)$.*

The valence system $A = (Q, \to)$ of interest may not be dependent. We will determine dependent versions of it (one for each context) by restricting to a dependent set of operations $\mathcal{O}' \subseteq \mathcal{O}$. The *restriction* is defined by $A[\mathcal{O}'] = (Q, \to \cap (Q \times (\mathcal{O}' \cup \{\varepsilon\}) \times Q))$.

Representing Block Decompositions. Theorem 4.3 considers a computation decomposed into irreducible contexts w_1 to w_k. It shows that each context w_i can be further decomposed into at most k blocks such that the overall sequence of blocks $w_{1,1}, \dots, w_{k,m_k}$ freely reduces to $1_{\mathbb{M}}$. Our goal is to represent the block decompositions of all candidate computations in a finite way. To this end, we analyze the result more closely.

The decomposition into contexts means there are dependent sets $\mathcal{O}_1, \dots, \mathcal{O}_k \subseteq \mathcal{O}$ such that each context w_i only uses operations from the set $\mathcal{O}_i$. The decomposition into blocks means there are $n = k^2$ computations v_1 to v_n and states q_1 to q_{n-1} such that v_i leads from q_{i-1} to q_i with $q_0 = q_{init}$ and $q_n = q_{fin}$. The last thing to note is that a block itself does not have to be right-invertible. This means we should represent block decompositions by (non-deterministic finite) automata rather than valence systems.

We define, for each pair of states $q_i, q_f \in Q$, each dependent set of operations $\mathcal{O}_{con} \subseteq \mathcal{O}$, and each subset $\mathcal{O}_{bl} \subseteq \mathcal{O}_{con}$ the automaton

$$N(q_i, q_f, \mathcal{O}_{con}, \mathcal{O}_{bl}) = \mathit{2nfa}(q_i, q_f, \mathit{sat}(A[\mathcal{O}_{con}])[\mathcal{O}_{bl}]) \ .$$

Function *2nfa* understands the given valence system $\mathit{sat}(A[\mathcal{O}_{con}])[\mathcal{O}_{bl}]$ as an automaton, with the first parameter as the initial and the second as the final state. The set $\mathcal{O}_{con}$ will be the operations used in the context of interest. As these operations are dependent, $\mathit{sat}(A[\mathcal{O}_{con}])$ will include the irreducible versions of all computations in $A[\mathcal{O}_{con}]$, Lemma 5.2. The second restriction to $\mathcal{O}_{bl}$ identifies the operations of one block in the context.

With this construction at hand, we define our representation of block decompositions.

▶ **Definition 5.3.** A *test* for the given (BCSREACH)-instance is a sequence $N_1, \dots, N_n$ of $n = k^2$ automata $N_i = N(q_{i-1}, q_i, \mathcal{O}_j, \mathcal{O}_{j,i})$ with $j = \lceil \frac{i}{k} \rceil$, $q_0 = q_{init}$, and $q_n = q_{fin}$.

The following lemma links Theorem 4.3 and the notion of tests. With Theorem 4.3, we have to check whether there is a computation w from q_{init} to q_{fin} with $cs(w) \leqslant k$ whose block decomposition admits a free reduction. With the analysis above, such a computation exists iff there is a test N_1 to N_n whose automata accept the blocks in the decomposition.

▶ **Lemma 5.4.** *We have $(q_{init}, \varepsilon) \to (q_{fin}, w)$ with $cs(w) \leqslant k$ and $[w]_{\mathbb{M}} = 1$ in A iff there is a test $N_1, \dots, N_n$ and computations $v_1 \in \mathcal{L}(N_1)$ to $v_n \in \mathcal{L}(N_n)$ that freely reduce to $1_{\mathbb{M}}$.*

Determining Free Reducibility. Given a test $N_1, \dots, N_n$, we have to check whether the automata accept computations that freely reduce to $1_{\mathbb{M}}$. To get rid of the reference to single computations, we now define a notion of free reduction directly on sequences of automata. This means we have to lift the following operations from computations to automata. On computations u and v, a free reduction may check commutativity, $u \ I \ v$, and whether the computations are inverses, $[u]_{\mathbb{M}} \cdot [v]_{\mathbb{M}} = 1_{\mathbb{M}}$. Consider N_u and N_v from $N_1, \dots, N_n$.

Rather than checking whether N_u and N_v accept computations that commute, the free reduction on automata will check whether the alphabets are independent, $\mathcal{O}(N_u) \ I \ \mathcal{O}(N_v)$. To see that this yields a complete procedure, note that Lemma 5.4 existentially quantifies over all tests, and hence all sets of operations to construct N_u and N_v. If there are computations u and v that commute in the free reduction, we can construct the automata N_u and N_v by restricting to the letters in these words. This will still guarantee $u \in \mathcal{L}(N_u)$ and $v \in \mathcal{L}(N_v)$.

To check whether N_u and N_v accept computations that multiply up to $1_{\mathbb{M}}$, we rely on the syntactic inverse. Consider a computation u that contains negative operations o^- only for symbols with $o \ I \ o$. In this case, the *syntactic inverse* $\mathit{sinv}(u)$ is defined by reversing the letters and inverting the polarity of operations. The operation is not defined otherwise. The following lemma is immediate.

▶ **Lemma 5.5.** *If $u, v \in \mathcal{O}^*$ are irreducible, dependent with $[u]_{\mathbb{M}} \cdot [v]_{\mathbb{M}} = 1_{\mathbb{M}}$, then $v = sinv(u)$.*

The idea is to admit v as the inverse of u if $v = sinv(u)$ holds. The equality will of course entail that v is the inverse of u, for any pair of computations. Lemma 5.5 moreover shows that for irreducible, dependent computations the check is complete. Since N_u and N_v are dependent and saturated, it will be complete (Lemma 5.2) to use the syntactic inverse also on the level of automata.

The definition swaps initial and final state, turns around the transitions, removes the negative operations on non-commutative symbols, and inverts the polarity of the others. Formally, the *syntactic inverse* yields $sinv(N_u) = (Q, q_{u,fin}, remswap(\rightarrow_u^{-1}), q_{u,init})$. The reverse relation contains $(q_2, o^{\pm}, q_1) \in \rightarrow_u^{-1}$ iff $(q_1, o^{\pm} q_2) \in \rightarrow_u$. Function *remswap* removes transitions with operations o^- for which $o \; I \; o$ does not hold and inverts the remaining polarities. The construction guarantees that $sinv(\mathcal{L}(N_u)) = \mathcal{L}(sinv(N_u))$. With this, the check of whether N_u and N_v contain computations u and v with $v = sinv(u)$ amounts to checking whether N_v and $sinv(N_u)$ have a computation in common.

▶ **Lemma 5.6.** *There are $u \in \mathcal{L}(N_u), v \in \mathcal{L}(N_v)$ with $v = sinv(u)$ iff $\mathcal{L}(N_v) \cap \mathcal{L}(sinv(N_u)) \neq \emptyset$.*

The analogue of the free reduction defined on automata is the following definition.

▶ **Definition 5.7.** A *free automata reduction* on a test N_1 to N_n is a sequence of operations
(FRA1) $N_i, N_j \mapsto_{free} \varepsilon$, if $\mathcal{L}(N_j) \cap \mathcal{L}(sinv(N_i)) \neq \emptyset$.
(FRA2) $N_i, N_j \mapsto_{free} N_j, N_i$, if $\mathcal{O}(N_i) \; I \; \mathcal{O}(N_j)$.
Since we quantify over all tests, free automata reductions are complete as follows.

▶ **Lemma 5.8.** *There is a test $N_1, \ldots, N_n$ and computations $u_1 \in \mathcal{L}(N_1)$ to $u_n \in \mathcal{L}(N_n)$ that freely reduce to $1_{\mathbb{M}}$ iff there is a test $N_1, \ldots, N_n$ that admits a free automata reduction to ε.*

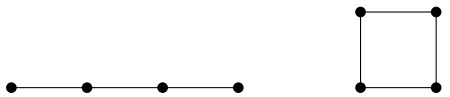
Together, Lemma 5.4 and Lemma 5.8 yield a decision procedure for (BCSREACH). We guess a suitable test and for this test a suitable free automata reduction. The restrictions, the saturation, the automata conversion, and the independence and disjointness tests require time polynomial in $|A| + k$. Moreover, the free automata reduction contains polynomially-many (in k) steps. Together, this yields membership in NP and proves Theorem 3.3.

6 Complexity for Fixed Graphs

We have seen that reachability under bounded context switching can always be decided in NP, even if the graph describing the storage mechanism is part of the input. In this section, we study how the complexity of the problem depends on the storage mechanism, i.e. the graph. We fix the graph G and consider the problem BCSREACH(G). We will see that for some graphs, the complexity is lower than NP: We exhibit a class of graphs G for which BCSREACH(G) is solvable in polynomial time and we describe those graphs for which the problem is NL-complete. Of course, for any graph G, the problem BCSREACH(G) is NL-hard, because reachability in directed graphs is. In some cases, we also have an NL upper bound.

A loop-free graph is a *clique* if any two distinct vertices are adjacent. By G^- we denote the graph obtained from G by removing all self-loops. If G^- is a clique, then valence systems over G are systems with access to a fixed number of independent counters, some of which are blind and some of which are partially blind.

▶ **Theorem 6.1.** *If G^- is a clique, then* BCSREACH(G) *is* NL*-complete. Otherwise,* BCSREACH(G) *is* P*-hard.*



(a) The graph P4. **(b)** The graph C4.

Figure 2 The graphs P4 and C4.

In some cases, BCSREACH is P-complete. A loop-free graph is a *transitive forest* if it is obtained from the empty graph using *disjoint union* and *adding a universal vertex*. A universal vertex is a vertex that is adjacent to all other vertices. Adding one means that we take a graph $G = (V, I)$ and add a new vertex $v \notin V$ and make it adjacent to every vertex in G. Hence, we obtain $(V \cup \{v\}, I \cup \{\{u, v\} \mid u \in V\})$.

▶ **Theorem 6.2.** *If G^- is a transitive forest, then* BCSREACH(G) *is in* P.

In the area of graph monoids, transitive forests are an important subclass. For many decision problems, they characterize those graphs for which the problem becomes decidable [58, 43] or tractable [44]. Intuitively, the storage mechanisms represented by graphs G where G^- is a transitive forest are those obtained by *building stacks* and *adding counters*, see [58, 57].

If $G = (V, I)$ is a graph, then H is an *induced subgraph of* G if H is isomorphic to a graph (V', I'), where $V' \subseteq V$ and $I' = \{e \in I \mid e \subseteq V'\}$. See Fig. 2 for the graphs C4 and P4.

▶ **Theorem 6.3.** *If* C4 *is an induced subgraph of G^-, then* BCSREACH(G) *is* NP*-complete.*

It is an old combinatorial result that a simple graph is a transitive forest if and only if it does not contain the two graphs P4 and C4 as induced subgraphs [54]. Hence, if one could also show that BCSREACH(G) is NP-hard when $G^- =$ P4, then Theorem 6.2 would cover all cases with polynomial complexity (unless P = NP). However, we currently do not know whether BCSREACH(P4) is NP-hard.

Proof Sketches. The rest of this section is devoted to sketching the proofs of Theorems 6.1, 6.2, and 6.3. The first step is a reformulation of the problem BCSREACH(G) if G is obtained from two disjoint graphs G_0 and G_1 by drawing edges everywhere between G_0 and G_1. Suppose $G_i = (V_i, I_i)$ is a graph for $i = 0, 1$ such that $V_0 \cap V_1 = \emptyset$. Then the graph $G_0 \times G_1$ is defined as (V, I), where $V = V_0 \cup V_1$ and $I = I_0 \cup I_1 \cup \{\{v_0, v_1\} \mid v_0 \in V_0, v_1 \in V_1\}$.

The reformulation also involves valence automata, which can read input. Let $G = (V, I)$ be a graph and let $\mathcal{O} = \{o^+, o^- \mid o \in V\}$. A *valence automaton* over G is a tuple $A = (Q, \Sigma, q_0, E, q_f)$, where Q is a finite set of *states*, Σ is an alphabet, $q_0 \in Q$ is its *initial state*, $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\mathcal{O} \cup \{\varepsilon\}) \times Q$ is its set of *transitions*, and $q_f \in Q$ is its *final state*. A *configuration* is a tuple (q, u, v), where $q \in Q$, $u \in \Sigma^*$, and $v \in \mathcal{O}^*$, where v is right-invertible. Intuitively, a transition (q, s, w, q') changes the state from q to q', reads the input s, and puts w into the storage. We write $(q, u, v) \to (q', u', v')$ if there is a transition (q, s, w, q') such that $u' = us$ and $v' = vw$. For any $k \in \mathbb{N}$, the *language accepted by A with at most k context switches* is denoted $\mathcal{L}_k(A)$ and defined as the set of all $u \in \sigma^*$ such that from $(q_0, \varepsilon, \varepsilon)$, we can reach (q_f, u, w) for some $w \in \mathcal{O}^*$ with $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ and $cs(w) \leq k$. The following problem will be used to reformulate BCSREACH$(G \times H)$.

Intersection under bounded context switching (BCSINT(G, H))	
Given:	Alphabet Σ, valence automata A, B over graphs G, H, resp., with input alphabet Σ, and bounds k, ℓ, m in unary.
Decide:	Is the intersection $\mathcal{L}_k(A) \cap \mathcal{L}_\ell(B) \cap \Sigma^{\leq m}$ non-empty?

We are now ready to state the reformulation, which is not difficult to prove.

▶ **Proposition 6.4.** *If* $G = G_0 \times G_1$*, then* BCSREACH(G) *is logspace-interreducible with* BCSINT(G_0, G_1)*.*

We can use Proposition 6.4 to show that adding a universal vertex does not change the complexity.

▶ **Proposition 6.5.** *If* G *has a universal vertex* v*, then* BCSREACH(G) *reduces to* BCSREACH$(G \setminus v)$ *in logspace.*

This can be deduced from Proposition 6.4 as follows. If v is a universal vertex, then $G = (G \setminus v) \times H$, where H is a one-vertex graph. In this situation, a valence automaton over H is equivalent to a one-counter automaton (OCA). It is folklore that an n-state OCA accepts a word of length m if and only if it does so with counter values at most $O((mn)^2)$ [22]. We can thus compute in logspace a finite automaton for the language $R = \mathcal{L}_\ell(B) \cap \Sigma^{\leq m}$. This means, our instance of BCSINT$(G \setminus v, H)$ reduces to emptiness of $\mathcal{L}_k(A) \cap R$. Using the automaton for R, this is easily turned into an instance of BCSREACH$(G \setminus v)$. Note that Proposition 6.5 yields the upper bound of Theorem 6.1. The P-hardness follows from P-hardness of reachability in pushdown automata.

The P upper bound in Theorem 6.2 follows from Proposition 6.5 and the following.

▶ **Proposition 6.6.** *rpropositionrestatePropositionComplexityUnion If* BCSREACH(G_i) *is in* P *for* $i = 0, 1$*, then* BCSREACH$(G_0 \uplus G_1)$ *is in* P *as well.*

Proposition 6.6 is shown using a saturation procedure similar to the one in Section 5. In the latter, we shortcut paths that read two (complementary) instructions. Here, in contrast, we find states p, q between which there is an arbitrarily long path that reads instructions w over one graph G_i for $i = 0, 1$ such that $[w]_{\mathbb{M}} = 1_{\mathbb{M}}$ and $cs(w) \leq k$. Then, we add an ε-transition between p and q.

Finally, let us comment on the NP-hardness in Theorem 6.3. If $G =$ C4, this is the well-known NP-hardness of reachability under bounded context switching. If G contains self-loops, we employ Proposition 6.4: If $G^- =$ C4, then $G = G_0 \times G_1$ for some graphs where each G_i contains two non-adjacent vertices. In this case, it is known that that valence automata over G_i accept the same languages as those over G_i^- [58, 57]. Therefore, the formulation in terms of BCSINT(G_0, G_1) allows us to conclude hardness.

7 Conclusion

We have shown that for every storage represented by a graph monoid, reachability under bounded context switches (BCSREACH) is decidable in NP. To this end, we show that after some preprocessing in a saturation procedure, any computation with bounded context switches decomposes into quadratically many blocks. These blocks then cancel and commute with each other so as to reduce to the identity element. Thus, one can guess a decomposition into blocks and verify the cancellation and commutation relations among them.

For the subclass of graph monoids whose underlying simple graph is a transitive forest, we have provided a polynomial-time algorithm (Theorem 6.2). However, we leave open whether there are other graph monoids for which the problem is in P.

One has NP-hardness in the case that the underlying simple graph contains C4 as an induced subgraph, which corresponds to the classical case of bounded context switching in concurrent recursive programs. Since transitive forests are precisely those simple graphs

that contain neither C4 nor P4 as induced subgraphs [54], showing NP-hardness for P4 would imply that Theorem 6.2 captures all graphs with polynomial-time algorithms (unless P = NP). Unfortunately, the known hardness techniques for problems involving graph groups or Mazurkiewicz traces over P4 [1, 43, 44, 58] do not seem to apply.

Moreover, there is a variety of under-approximations for concurrent recursive programs [36, 11, 18, 41, 24, 12, 52]. It appears to be a promising direction for future research to study generalizations of these under-approximations to valence systems.

References

1 IJ. J. Aalbersberg and H. J. Hoogeboom. Characterizations of the decidability of some problems for regular trace languages. *Mathematical Systems Theory*, 22(1):1–19, 1989.

2 P. A. Abdulla, C. Aiswarya, and M. F. Atig. Data multi-pushdown automata. In *CONCUR*, volume 85 of *LIPIcs*, pages 38:1–38:17. Dagstuhl, 2017.

3 P. A. Abdulla, C. Aiswarya, M. F. Atig, M. Montali, and O. Rezine. Recency-bounded verification of dynamic database-driven systems. In *PODS*, pages 195–210. ACM, 2016.

4 P. A. Abdulla, M. F. Atig, A. Bouajjani, and T. P. Ngo. Context-bounded analysis for POWER. In *TACAS*, volume 10206 of *LNCS*, pages 56–74. Springer, 2017.

5 P. A. Abdulla, M. F. Atig, R. Meyer, and M. S. Salehi. What's decidable about availability languages? In *FSTTCS*, volume 45 of *LIPIcs*, pages 192–205. Dagstuhl, 2015.

6 C. Aiswarya. *Verification of communicating recursive programs via split-width.* PhD thesis, École normale supérieure de Cachan, France, 2014.

7 C. Aiswarya, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

8 C. Aiswarya, P Gastin, and K. N. Kumar. Controllers for the verification of communicating multi-pushdown systems. In *CONCUR*, volume 8704 of *LNCS*, pages 297–311. Springer, 2014.

9 S. Akshay, P. Gastin, and S. N. Krishna. Analyzing timed systems using tree automata. In *CONCUR*, volume 59 of *LIPIcs*, pages 27:1–27:14. Dagstuhl, 2016.

10 S. Akshay, P. Gastin, S. N. Krishna, and I. Sarkar. Towards an efficient tree automata based technique for timed systems. In *CONCUR*, volume 85 of *LIPIcs*, pages 39:1–39:15. Dagstuhl, 2017.

11 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *DLT*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

12 M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS*, volume 29 of *LIPIcs*, pages 611–623. Dagstuhl, 2014.

13 M. F. Atig, A. Bouajjani, and G. Parlato. Getting rid of store-buffers in TSO analysis. In *CAV*, volume 6806 of *LNCS*, pages 99–115. Springer, 2011.

14 M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *TACAS*, volume 5505 of *LNCS*, pages 107–123. Springer, 2009.

15 M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *LNCS*, pages 356–371. Springer, 2008.

16 A. Bouajjani and M. Emmi. Bounded phase analysis of message-passing programs. *STTT*, 16(2):127–146, 2014.

17 A. Bouajjani, M. Emmi, and G. Parlato. On sequentializing concurrent programs. In *SAS*, volume 6887 of *LNCS*, pages 129–145. Springer, 2011.

18 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

19 P. Buckheister and Georg Zetzsche. Semilinearity and context-freeness of languages accepted by valence automata. In *MFCS*, volume 8087 of *LNCS*, pages 231–242. Springer, 2013.

20 P. Chini, J. Kolberg, A. Krebs, R. Meyer, and P. Saivasan. On the complexity of bounded context switching. In *ESA*, volume 87 of *LIPIcs*, pages 27:1–27:15. Dagstuhl, 2017.

21 P. Chini, R. Meyer, and P. Saivasan. Fine-grained complexity of safety verification. In *TACAS*, volume 87 of *LNCS*. Springer, 2018.

22 D. Chistikov, W. Czerwinski, P. Hofman, M. Pilipczuk, and M. Wehar. Shortest paths in one-counter systems. In *FOSSACS*, pages 462–478, 2016.

23 E. D'Osualdo, R. Meyer, and G. Zetzsche. First-order logic with reachability for infinite-state systems. In *LICS*, pages 457–466. ACM, 2016.

24 M. Emmi, S. Qadeer, and Z. Rakamaric. Delay-bounded scheduling. In *POPL*, pages 411–422. ACM, 2011.

25 J. Esparza, P. Ganty, and R. Majumdar. A perfect model for bounded verification. In *LICS*, pages 285–294. IEEE, 2012.

26 J. Esparza, P. Ganty, and T. Poch. Pattern-based verification for multithreaded programs. *ACM ToPLaS*, 36(3):9:1–9:29, 2014.

27 F. Furbach, R. Meyer, K. Schneider, and M. Senftleben. Memory-model-aware testing: A unified complexity analysis. *ACM Trans. Embedded Comput. Syst.*, 14(4):63:1–63:25, 2015.

28 P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. In *CAV*, volume 6174 of *LNCS*, pages 600–614. Springer, 2010.

29 S. Ginsburg and E. Spanier. Bounded ALGOL-like languages. *Trans. Amer. Math. Soc.*, 113:333—368, 1964.

30 M. Hague and A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, volume 7358 of *LNCS*, pages 260–276. Springer, 2012.

31 A. Heussner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. *LMCS*, 8(3), 2012.

32 Martin Huschenbett, Dietrich Kuske, and Georg Zetzsche. The monoid of queue actions. *Semigroup Forum*, 95:475–508, 2017.

33 O. Inverso, T. L. Nguyen, B. Fischer, S. La Torre, and G. Parlato. Lazy-CSeq: A context-bounded model checking tool for multi-threaded C-programs. In *ASE*, pages 807–812. IEEE, 2015.

34 C. Köcher. Rational, recognizable, and aperiodic sets in the partially lossy queue monoid. In *STACS*, LIPIcs, pages 45:1–45:14. Dagstuhl, 2018.

35 C. Köcher and D. Kuske. The transformation monoid of a partially lossy queue. In *CSR*, volume 10304 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2017.

36 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE, 2007.

37 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.

38 S. La Torre, P. Madhusudan, and G. Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, volume 5643 of *LNCS*, pages 477–492. Springer, 2009.

39 S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN*, pages 96–107. Springer, 2010.

40 S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.

41 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

42 A. Lal and T. W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

43 M. Lohrey and B. Steinberg. The submonoid and rational subset membership problems for graph groups. *Journal of Algebra*, 320(2):728–755, 2008.

44 M. Lohrey and G. Zetzsche. Knapsack in graph groups. *Theory of Computing Systems*, 62:192–246, 2018.

45 S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, pages 329–339. ACM, 2008.

46 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294. ACM, 2011.

47 R. Meyer, S. Muskalla, and G. Zetzsche. Bounded Context Switching for Valence Systems. *ArXiv e-prints*, 2018. `arXiv:1803.09703`.

48 M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *PLDI*, pages 446–455. ACM, 2007.

49 T. L. Nguyen, P. Schrammel, B. Fischer, S. La Torre, and G. Parlato. Parallel bug-finding in concurrent programs via reduced interleaving instances. In *ASE*, pages 753–764. IEEE, 2017.

50 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

51 S. Qadeer and D: Wu. KISS: Keep it simple and sequential. In *PLDI*, pages 14–24. ACM, 2004.

52 E. Tomasco, O. Inverso, B. Fischer, S. La Torre, and G. Parlato. Verifying concurrent programs by memory unwinding. In *TACAS*, volume 9035 of *LNCS*, pages 551–565. Springer, 2015.

53 K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *CADE*, volume 3632 of *LNCS*, pages 337–352. Springer, 2005.

54 E. S. Wolk. A note on "the comparability graph of a tree". *Proceedings of the American Mathematical Society*, 16(1):17–20, 1965.

55 G. Zetzsche. Silent transitions in automata with storage. In *ICALP*, volume 7966 of *LNCS*, pages 434–445. Springer, 2013.

56 G. Zetzsche. Monoids as storage mechanisms. *Bulletin of the EATCS*, 120:237–249, 2016.

57 G. Zetzsche. *Monoids as Storage Mechanisms*. PhD thesis, Technische Universität Kaiserslautern, 2016.

58 G. Zetzsche. The emptiness problem for valence automata over graph monoids, 2018. To appear in Information and Computation.

Alternating Nonzero Automata

Paulin Fournier
LS2N, Université de Nantes, France

Hugo Gimbert
CNRS, LaBRI, Université de Bordeaux, France

Abstract

We introduce a new class of automata on infinite trees called *alternating nonzero automata*, which extends the class of non-deterministic nonzero automata. The emptiness problem for this class is still open, however we identify a subclass, namely limited choice, for which we reduce the emptiness problem for alternating nonzero automata to the same problem for non-deterministic ones, which implies decidability. We obtain, as corollaries, algorithms for the satisfiability of a probabilistic temporal logic extending both CTL* and the qualitative fragment of pCTL*.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases zero-automata, probabilities, temporal logics

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.13

Funding Both authors received support for this work from the French ANR projet "Stoch-MC".

Acknowledgements The authors wish to thank Mikołaj Bojańczyk, Henryk Michalewski and Matteo Mio for interesting discussions on TMSO+ZERO as well as zero- and nonzero-automata; and the referees for their helpful comments.

1 Introduction

The theory of automata on infinite trees is rooted in Rabin's seminal theorem which establishes an effective correspondence between the monadic second order logic (MSO) theory of the infinite binary tree and the non-deterministic automata on this tree [18]. In this correspondence, the satisfiability of the logic is dual to the emptiness of the algorithm and both these algorithmic problems are mutually reducible to one another.

This elegant setting has been partially extended to probabilistic logics [13, 6, 14, 15, 2] and automata with probabilistic winning conditions [18, 17, 1, 7, 2]. In this paper we make another step in this direction: we show a correspondence between the logic $\mathrm{CTL}^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ and nonzero alternating automata with limited choice. Moreover we show that the emptiness problem of the automata is decidable and obtain as a corollary the decidability of the satisfiability of the logic.

Automata. Alternating nonzero automata are an alternating version of *non-deterministic nonzero automata* introduced in [3], which themselves are equivalent to *non-deterministic zero automata* introduced in [2].

An alternating nonzero automaton takes as input a binary tree. Some states of the automaton are controlled by Eve, while other states are controlled by Adam, and the player controlling the current state chooses the next transition. Some transitions are *local transitions*, in which case the automaton stays on the same node of the input tree while other are *split*

transitions in which case the automaton proceeds to the left son or to the right son of the current node with equal probability $\frac{1}{2}$.

This interaction between Eve and Adam is seen as a game where Eve and Adam play according to some strategies. Once the strategies are fixed, one obtains a Markov chain whose trajectories are all possible plays consistent with the strategies. The winner is determined with respect to winning conditions introduced in [2, 3], using a total order on the set of states (used to compute the limsup of a play which is the largest state seen infinitely often during the play) and three subsets of states, respectively called the *sure*, *almost-sure* and *positive states*. Eve wins if and only if the three acceptance conditions hold:

sure winning: every play has limsup in sure states; and

almost-sure winning: almost-every play has limsup in almost-sure states; and

positive winning: whenever the play enters a positive state there is positive probability that the play never exits positive states.

The input tree is accepted by the alternating automaton iff Eve has a winning strategy.

Alternating nonzero automata generalize both classical alternating automata with parity conditions [8, 16] (when all states are almost-sure and positive) as well as non-deterministic nonzero automata [3] (in case Eve controls all states).

We do not know whether the emptiness problem for these automata is decidable or not, however we show that the answer is positive for the subclass of alternating nonzero automata with *limited choice for Adam*. In these automata, some choices of Adam are canonical, at most one in every state, and Adam may perform at most a bounded number of non-canonical choices during a single play.

First, we show that the emptiness problem for alternating nonzero automata with limited choice for Adam is in NEXPTIME$\cap$ co-NEXPTIME (Theorem 22). The proof is an EXPTIME reduction to the emptiness problem for non-deterministic automata. This proof relies on the positional determinacy of the acceptance games for Eve (Lemma 10) and a characterization of positional winning strategies for Eve (Lemmas 12, 13 and 17).

Second, we show that in the particular case where the sure winning condition is a Büchi condition, emptiness of non-deterministic nonzero automata is in PTIME (Theorem 3). It follows that, in case of a trivial sure winning condition, emptiness of alternating nonzero automata with limited choice for Adam is in EXPTIME (Theorem 22).

Logic. The temporal logic CTL* introduced by Emerson and Halpern [9] and its fragments CTL and LTL are prominent tools to specify properties of discrete event systems.

A variant of CTL* is the logic pCTL* [11] in which the universal and existential path quantifiers are *replaced* by probabilistic path quantifiers which set upper or lower bounds on the probability of a path property in a Markov chain. For example the formula $\mathbb{P}_{\geq \frac{1}{2}}(FGa)$ specify that with probability at least $\frac{1}{2}$ eventually all the visited states are labelled with a. To our knowledge, the satisfiability problem for this logic is an open problem.

However, for the qualitative fragment of pCTL*, where only two probabilistic quantifiers $\mathbb{P}_{>0}$ and $\mathbb{P}_{=1}$ are available, the satisfiability is decidable [6]. In a variant of pCTL* called pECTL the path subformula are replaced by deterministic Büchi automaton, and the satisfiability of the qualitative fragment is 2-EXPTIME complete [6], the same complexity as for CTL* [19].

Remark that neither pCTL* nor pECTL includes the path operators $\forall$ and $\exists$, thus these two logics are incomparable in expressivity with CTL*. For example, on the alphabet $\{a, b\}$, the CTL* formula $\phi_1 = \forall FG\neg b$, and the pCTL* formula $\phi_2 = \mathbb{P}_{=1}(FG\neg b)$ specify, that

every branch, respectively *almost-every* branch, of the model has finitely many b. Neither ϕ_1 can be expressed in pCTL* nor ϕ_2 can be expressed in CTL*.

In this paper, we consider the logic CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ which is an extension of both CTL* and qualitative pCTL*and establish several properties of this logic.

The satisfiability by an arbitrary Σ-labelled Markov chain reduces to the satisfiability by $(\Sigma \cup \{\circ\})$-labelled a binary tree with $\circ$ a fresh letter (Theorem 24).

The satisfiability of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ reduces to the emptiness of alternating nonzero automata with finite choice for Adam thus it is decidable in 3-NEXPTIME$\cap$co-3-NEXPTIME. In the variant ECTL$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$, where path formula are deterministic Büchi automata, this reduction gives a 2-NEXPTIME$\cap$ co-2-NEXPTIME complexity and for the fragment CTL$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ the complexity is NEXPTIME$\cap$ co-NEXPTIME (Theorem 23).

For the fragments CTL$^*[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$, ECTL$[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$ and CTL$[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$ (i.e. qualitative pCTL*, pECTL and pCTL respectively), the $F_\forall$ acceptance condition of the automaton is a Büchi condition, and we retrieve the optimal complexity bounds of [6, 5], i.e. 3-EXPTIME, 2-EXPTIME and EXPTIME, respectively.

A motivation for the study of alternating nonzero automata is the recent research on the logic TMSO+ZERO. The logic TMSO+ZERO is an extension of Monadic second-order logic on infinite binary trees with a new probabilistic operator [14, 15, 2]. The satisfiability of this logic is reducible to the emptiness problem for nonzero non-deterministic automata [2] which is decidable [3]. Since CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ is a fragment of TMSO+ZERO, this result implies that the satisfiability of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ is decidable with non-elementary complexity. The reduction to the emptiness of alternating nonzero automata given in the present paper provides a better complexity bound.

2 Alternating nonzero automata

An alternating nonzero automaton on a finite alphabet Σ is a finite-state machine processing binary trees, equipped with a game semantics: every tree is either accepted or rejected by the machine depending on who wins the acceptance game on the tree.

Trees. A Σ-labelled binary tree (or Σ-tree for short) is a function $t : \{0,1\}^* \to \Sigma$. An element $n \in \{0,1\}^*$ is called a *node* of the tree and has exactly two sons $n0$ and $n1$. We use the usual notions of ancestors and descendants. A node n' is *(strictly) below* n if n is a (strict) prefix of n'. A *path* in the tree is a finite or infinite sequence of nodes $n_0, n_1, \ldots$ such that for every k the node n_{k+1} is a son of the node n_k.

A branch b is an element of $\{0,1\}^\omega$. If a node n is a prefix of b we say that n *belongs* to b or that b *visits* n. The set of branches is equipped with the uniform probability measure, denoted μ, corresponding to an infinite random walk taking at each step either direction 0 or 1 with equal probability $\frac{1}{2}$.

A set of nodes $T \subseteq \{0,1\}^*$ is a *subtree* if it contains a node r, called the root of T, such that every node $n \in T$ is a descendant of r, T contains all nodes on the path from r to n. A subtree is full if T contains all descendants of r.

Automata. An alternating nonzero automaton on alphabet Σ is presented as a tuple $\mathcal{A} = (Q, q_0, Q_E, Q_A, \to, F_\forall, F_1, F_{>0})$ where:

- Q is a finite set of states, equipped with a total order $\leq$, containing the initial state q_0.
- (Q_E, Q_A) is a partition of Q into Eve and Adam states.

- $\to$ is the set of transitions, there are two types of transitions: *local transitions* which are tuples (q, a, q') with $q, q' \in Q$ and $a \in \Sigma$, denoted $q \to_a q'$; and *split transitions* which are tuples $(q, a, q_0, q_1) \in Q \times \Sigma \times Q^2$, denoted $q \to_a (q_0, q_1)$.
- $F_\forall$, F_1 and $F_{>0}$ are subsets of Q defining the acceptance condition.

The input of such an automaton is an infinite binary tree $t : \{0,1\}^* \to \Sigma$. The source (resp. the target) of a local transition $q \to_a q'$ is q (resp q'). The source (resp. the targets) of a split transition $q \to_a (q_0, q_1)$ is q (resp q_0 and q_1). A state is said to be controlled by Eve or Adam whether it belongs to Q_E or Q_A. The controller of a transition is the controller of its source state. We always assume that

(HC) the automaton is **complete**: for every state q and letter a there is at least one transition with source q on a.

The (HC) condition makes it easier to define the game semantics of the automaton.

Game semantics. The acceptance of an input binary tree by the automaton is defined by mean of a stochastic game between Eve and Adam called the *acceptance game*.

The game of acceptance of a binary tree $t : \{0,1\}^* \to \Sigma$ by $\mathcal{A}$ is a two-player stochastic game with perfect information played by two strategic players Eve and Adam. The vertices of the game are all pairs (n, q) where $n \in \{0,1\}^*$ is a node of the infinite binary tree and q is a state of the automaton. The game starts in the initial vertex (ϵ, q_0).

Each vertex (n, q) is controlled by either Eve or Adam depending on whether $q \in Q_E$ or $q \in Q_A$. The controller of the current state chooses any transition with source q and letter $t(n)$. Intuitively, depending on whether the transition is a local or a split transition, the automaton stays on the current node n or move with equal probability $\frac{1}{2}$ to either node $n0$ or $n1$. If the transition is a local transition $q \to_{t(n)} q'$, the new vertex of the game is (n, q'). If the transition is a split transition $q \to_{t(n)} (r_0, r_1)$ then the new vertex is chosen randomly with equal probability $\frac{1}{2}$ between vertices $(n0, r_0)$ or $(n1, r_1)$.

A play is a finite or infinite sequence of vertices $\pi = (n_0, q_0)(n_1, q_1)\ldots$. We denote $\text{first}(\pi) = (n_0, q_0)$ and $\text{last}(\pi) = (n_k, q_n)$ (for finite plays).

A strategy for Eve associates with every finite play whose last vertex is controlled by Eve a transition with source q_n and letter $t(n_k)$ (such a transition always exists since the automaton is complete). Strategies for Adam are defined in a symmetric way. Strategies of Eve are usually denoted σ while strategies for Adam are denoted τ.

Measuring probabilities. Once both players Eve and Adam have chosen some strategies σ and τ, this defines naturally a non-homogenous Markov chain whose states are the vertices of the game. According to Tulcea theorem, if we equip the set of plays with the σ-field generated by cylinders, then there is a unique probability measure $\mathbb{P}^{\sigma,\tau}$ such that after a play $\pi = (n_0, q_0)\ldots(n_k, q_k)$, if $\delta(\pi)$ denotes the transition chosen by Eve or Adam after π (depending on whether $q_k \in Q_E$ or $q_k \in Q_A$), the probability to go to vertex (n_{k+1}, q_{k+1}) is:

$$\begin{cases} 1 & \text{if } \delta(\pi) \text{ is the local transition } q_k \to_{t(n_k)} q_{k+1} \ , \\ \frac{1}{2} & \text{if } \delta(\pi) \text{ is the split transition } q_k \to_{t(n_k)} (r_0, r_1) \text{ and } \begin{cases} n_{k+1} = n_k 0 \text{ and } q_{k+1} = r_0 \ ; \textit{or} \\ n_{k+1} = n_k 1 \text{ and } q_{k+1} = r_1 \ . \end{cases} \\ 0 & \text{otherwise .} \end{cases}$$

This way we obtain a probability measure $\mathbb{P}^{\sigma,\tau}$ on the set of infinite plays.

Consistency and reachability. If a finite play π is the prefix of another finite or infinite play π' we say that π' is a *continuation* of π. A finite π play is *consistent* with a strategy σ or, more simply, is a σ*-play* if there exists a strategy τ such that π may occur in the non-homogenous Markov chain induced by σ and τ. In this case, the number N of split transitions which occurred in π is exactly the depth of the node of $\mathrm{last}(\pi)$ and $\mathbb{P}^{\sigma,\tau}(\{ \text{ continuations of } \pi \}) = 2^{-N}$. A vertex w is σ*-reachable* if there exists a finite σ-play from the initial vertex to w. An infinite play is consistent with σ if all its prefixes are.

Bounded vs. unbounded plays. There are two kinds of infinite plays: *bounded plays* are plays whose sequence of nodes is ultimately constant, or equivalently which ultimately use only local transitions while *unbounded plays* use infinitely many split transitions.

Bounded plays consistent with σ and τ are the atoms of $\mathbb{P}^{\sigma,\tau}$: a play π is bounded and consistent with σ and τ iff $\mathbb{P}^{\sigma,\tau}(\{\pi\}) > 0$.

In this paper we will focus on subclasses of automata whose structural restrictions forbids the existence of bounded plays (see the **(NLL)** hypothesis below).

So in practice, every play $\pi = (n_0, q_0)(n_1, q_1)\dots$ we consider will visit a sequence of nodes $n_0, n_1, n_2, \dots$ which enumerates all finite prefixes of an infinite branch $b \in \{0,1\}^\omega$ of the binary tree, in a weakly increasing order: for every index i either $n_{i+1} = n_i$ (the player controlling (n_i, q_i) played a local transition) or $n_{i+1} = n_i d$ for some $d \in \{0,1\}$ (the player controlling (n_i, q_i) played a split transition and the play followed direction d).

Winning strategies and language. Whether Eve wins the game is defined as follows. The *limsup* of an infinite play $(n_0, q_0)(n_1, q_1)\dots$ is $\limsup_i q_i$ i.e. the largest automaton state visited infinitely often. An infinite play π' is a *positive continuation* of π if all states of π' visited after π belongs to $F_{>0}$.

Eve wins with σ against τ if the three following conditions are satisfied.

Sure winning. Every play consistent with σ and τ has limsup in $F_\forall$.

Almost-sure winning. Almost-every play consistent with σ and τ has limsup in F_1.

Positive winning. For every finite play π consistent with σ and τ whose last state belongs to $F_{>0}$, the set of positive continuations of π has nonzero probability.

A Büchi condition is a set of states $R \subseteq Q$ which is upper-closed with respect to $\leq$. Then a play has limsup in R iff it visits R infinitely often.

We say that *Eve wins* the acceptance game if she has a *winning strategy* i.e. a strategy which wins the acceptance game against any strategy of Adam.

▶ **Definition 1** (Acception and language)**.** A binary tree is *accepted* by the automaton if Eve has a winning strategy in the acceptance game. The language of the automaton is the set of its accepted trees.

We are interested in the following decision problem:

Emptiness problem: Given an automaton, decide whether its language is empty or not.

Alternation and the use of game semantics makes the following closure properties trivial.

▶ **Lemma 2** (Closure properties)**.** *The class of languages recognized by alternating nonzero automata is closed under union and intersection.*

Normalization. We assume all automata to be normalized in the sense where they satisfy:

(N1) every split transition whose source is in $F_{>0}$ has at least one successor in $F_{>0}$; and

(N2) every local transition whose source is in $F_{>0}$ has its target in $F_{>0}$ as well.

We can normalize an arbitrary automaton by removing all transitions violating **(N1)** and **(N2)**. This will not change the language because such transitions are never used by positively winning strategies of Eve. This normalization could lead to a violation of the completeness hypothesis, **(HC)**. In this case we can also delete the corresponding states without modifying the language of the automaton.

If one would drop **(HC)** then the game graph may have dead-ends and the rules of the game would have to be extended to handle this case, typically the player controlling the state in the dead-end loses the game. This extension does not bring any extra expressiveness to our model of automaton, we can always make an automaton complete by adding local transitions leading to losing absorbing states.

Moreover, we assume:

(N3) $F_1 \subseteq F_\forall$.

This is w.l.o.g. since replacing F_1 with $F_1 \cap F_\forall$ does not modify the language of the automaton.

Non-deterministic nonzero automata Non-deterministic *zero* automata were introduced in [2], followed by a variant of equivalent expressiveness, non-deterministic *nonzero* automata [4, Lemma 5]. In those automata, Adam is a dummy player, i.e. $Q_A = \emptyset$ and moreover all transitions are split-transitions.

▶ **Theorem 3.** *The emptiness problem for non-deterministic nonzero automata is in* NP∩coNP. *If $F_\forall$ is a Büchi condition then emptiness can be decided in* PTIME.

The first statement is established in [3, Theorem 3]. The second statement is proved in the appendix. The proof idea is as follows. Assume the alphabet to be a singleton, which is w.l.o.g. for non-deterministic automata. The existence of a winning strategy for Eve can be witnessed by a subset $W \subseteq Q$ which contains the initial state and two positional winning strategies $\sigma_1, \sigma_2 : W \to W \times W$. Strategy σ_1 should be almost-surely and positively winning while strategy σ_2 should be surely winning. These two strategies can be combined into a (non-positional) strategy for Eve which satisfies the three objectives, thus witnesses non-emptiness of the automaton.

3 An example: the language of PUCE trees

A $\{a, b\}$-tree is *positively ultimately constant everywhere* (*PUCE*) if for every node n,

- **i)** the set of branches visiting n and with finitely many a-nodes has > 0 probability; and
- **ii)** the set of branches visiting n and with finitely many b-nodes has > 0 probability.

No regular tree is PUCE. There are two cases. If the regular tree has a node n which is the root of a full subtree labelled with a single letter (either a or b) then clearly the tree is not PUCE. Otherwise, by a standard pumping argument, every node labelled a (resp. b) has a descendant labelled b (resp. a) at some depth $\leq |S|$, where S is the set of states of the regular tree. But in this second case from every node n there is probability at least $\frac{1}{2^{|S|}}$ to reach a descendant with a different label, thus almost-every branch of the regular tree has infinitely many a and b, and the tree is not PUCE either.

There exists a PUCE tree. However it is possible to build a non-regular tree t whose every node satisfies both i) and ii). For that, we combine together two partial non-regular trees. Let $H \subseteq \{0, 1\}^*$ be a subset of nodes such that a) the set of branches which visit no node in

H has probability $\frac{1}{2}$, b) every node in $\{0,1\}^*$ is either a descendant or an ancestor of a node in H, but not both (H is a cut).

To obtain t, we combine two partial trees t_a and t_b whose domain is $\{0,1\}^* \setminus (H\{0,1\}^+)$ and t_a is fully labeled with a while t_b is fully labelled with b. Since H is a cut, the nodes in H are exactly the leaves of t_a and t_b. To obtain t, we plug a copy of t_b on every leaf of t_a and a copy of t_a on every leaf of t_b. Then from every node, according to b) there is non-zero probability to enter either t_a or t_b and according to a) there is non-zero probability to stay in there forever.

An automaton recognizing PUCE trees. We can design one automaton for each of the two conditions and combine them together with an extra state controlled by Adam (cf proof of Lemma 2). We provide an alternating nonzero automaton checking condition ii), the automaton for condition i) is symmetric. The state space is: $Q = \{s < w < g < \sharp\}$.

Intuitively, Adam uses states s to search for a node n from which condition ii) does not hold. Once on node n, Adam switches to state w and challenges Eve to find a path to an a-node n' which is the root of an a-labelled subtree T_n of > 0 probability. For that Eve navigates the tree in state w to node n', switches to state g on node n', stays in g as long as the play stays in T_n and switches definitively to $\sharp$ whenever leaving T_n.

Formally, the only state controlled by Adam is s, i.e. $Q_A = \{s\}$, from which Adam can choose, independently of the current letter, between two split transitions $s \to (s, \sharp)$ and $s \to (\sharp, s)$ and a local transition $s \to w$. The state $\sharp$ is absorbing. From state w, Eve can guess the path to n' using the split transitions: $w \to (\sharp, w) \quad w \to (w, \sharp)$.

Once n' is reached Eve can switch to state g with a local transition $w \to g$ and, whenever the current node is an a-node, she can choose among three split transitions: $g \to_a (g, g) \quad g \to_a (g, \sharp) \quad g \to_a (\sharp, g)$ to identify T_n.

The acceptance conditions are: $F_\forall = F_1 = Q \setminus \{w\}$ and $F_{>0} = \{g\}$, so that from w Eve is forced to eventually switch to g (otherwise $\limsup = w \notin F_\forall$) and the a-subtree labelled by g must have positive probability for Eve to win. Adam may never exit the pathfinding state s, in which case Eve wins.

4 Deciding emptiness of automata with limited choice for Adam

In this section, we introduce the class of automata with *limited choice for Adam*, and show that emptiness of these automata is decidable.

For that we rely on a characterization of positional strategies of Eve which satisfy the surely and almost-surely winning conditions (Lemma 12, Lemma 13) and the positively winning condition (Lemma 17). Then we represent the positional strategies of Eve as labelled trees, called *strategic trees* (Definition 18). Finally, we show that the language of strategic trees whose corresponding positional strategy is winning can be recognized by a non-deterministic nonzero automaton (Theorem 19).

4.1 Automata with limited choice for Adam

In the rest of the paper, we focus on the class of automata with limited choice for Adam. Our motivation is that these automata capture the logic we are interested in and their acceptance games have good properties. In particular the existence of positional winning strategies for Eve is one of the key properties used to decide emptiness.

To define the class of automata with limited choice for Adam, we rely on the transition graph of the automaton.

▶ **Definition 4** (Equivalent and transient states)**.** The transitions of the automaton define a directed graph called the *transition graph* and denoted $G_{\to}$. The vertices of $G_{\to}$ are Q and the edges are labelled with Σ, those are all triplets (q, a, r) such that $q \to_a r$ is a local transition or such that $q \to_a (r, q')$ or $q \to_a (q', r)$ is a split transition for some state q'.

Two states q, r are *equivalent*, denoted $q \equiv r$, if they are in the same connected component of $G_{\to}$. A state is *transient* if it does not belong to any connected component of $G_{\to}$, or equivalently if there is no cycle on this state in $G_{\to}$.

▶ **Definition 5.** An automaton has limited choice for Adam if for every state q controlled by Adam, all transitions with source q are local transitions; and for every letter a, at most one of the (local) transitions $q \to_a q'$ satisfies $q \equiv q'$. Such a transition is called a *canonical* transition.

In a limited choice for Adam automaton, the only freedom of choice of Adam, apart from playing canonical transitions, is deciding to go to a lower connected component of the transition graph. This non-canonical decision can be done only finitely many times, hence the name *limited choice*.

In the classical (non-probabilistic) theory of alternating automata, similar notions of limited alternation have already been considered, for example *hesitant alternating automata* [12].

▶ **Definition 6** (Canonical plays and transient vertices)**.** A *canonical play* is a play in which Adam only plays canonical transitions. A vertex (n, q) of an acceptance game is *transient* if it has no immediate successor (n', q') (by a local or a split transition) such that $q \equiv q'$.

In the acceptance game of an automaton with limited choice for Adam, every infinite play visit finitely many transient vertices and has a canonical suffix.

The automaton recognizing PUCE trees described in Section 3 has not limited choice for Adam, since Adam can play the non-local transitions $s \to (s, \sharp)$ and $s \to (\sharp, s)$. However, it is an easy exercise to turn this automaton into an automaton with limited choice for Adam recognizing the same language: add a new state e controlled by Eve from which Eve has a single split transition $e \to (s, s)$. This new state e belongs to both $F_\forall$ and F_1. From s Adam can choose between two local transitions: the canonical transition $s \to e$ (keep navigating in the tree) or the transient transition $s \to w$ (start verification).

The no local loop assumption. We assume that every automata with limited choice for Adam also satisfies:

(NLL) the automaton has **no local loop**: there is no letter a and sequence of local transitions $q_0 \to_a q_1 \to_a \cdots \to_a q_i$ such that $q_0 = q_i$.

Under the hypothesis **(NLL)**, for every infinite play π there is a unique branch of the binary tree $b \in \{0, 1\}^\omega$ whose every prefix is visited by π. We say that π *projects* to b. It is direct that, under the hypothesis **(NLL)**, the measures on plays and on branches are linked.

▶ **Lemma 7.** *Under the hypothesis **(NLL)**, given a tree t, two strategies σ and τ in the acceptance game and X is a measurable set of plays we have that $\mathbb{P}^{\sigma,\tau}(X) = \mu(\tilde{X})$ where μ is the usual uniform measure on the set of branches of t and $\tilde{X}$ is the set of infinite branches that X projects to.*

Moreover, assuming **(NLL)** does not reduce expressiveness.

▶ **Lemma 8.** *Given an automaton $\mathcal{A}$ with limited choice for Adam and set of states Q one can effectively construct another automaton $\mathcal{A}'$ with limited choice for Adam satisfying **(NLL)** and recognizing the same language.*

The interest of the **(NLL)** assumption is to make the acceptance game acyclic, which in turn guarantees positional determinacy for Eve, as shown in the next section. The transformation performed in the proof of Lemma 8 creates an exponential blowup of the state space of the automaton, which is bad for complexity. We could do without this blowup by dropping the **(NLL)** assumption, in which case Eve might need one extra bit of memory in order to implement local loops with priority in $F_\forall \setminus F_1$.

However, we prefer sticking to the **(NLL)** assumption, which makes the alternating automata and their accepting games simpler and is anyway not restrictive when it comes to translating temporal logics into alternating automata: the natural translation produces automata with no local loop.

4.2 Positional determinacy of the acceptance game

A crucial property of automata with limited choice for Adam is that their acceptance games are positionally determined for Eve.

▶ **Definition 9** (Positional strategies)**.** A strategy σ of Eve in an acceptance game is *positional* if for every finite plays π, π' whose last vertices are controlled by *Eve* and coincide, i.e. $\text{last}(\pi) = \text{last}(\pi') \in \{0,1\}^* \times Q_E$, then $\sigma(\pi) = \sigma(\pi')$.

▶ **Lemma 10** (Positional determinacy for Eve)**.** *Every acceptance game of an automaton with limited choice for Adam is positionally determined for Eve: if Eve wins then she has a positional winning strategy.*

Sketch of proof. Since the **(NLL)** hypothesis is assumed, the underlying acceptance game is acyclic. The construction of a positional winning strategy σ' from a (non-positional) winning strategy σ relies on the selection of a canonical way of reaching a σ-reachable vertex w with a σ-play $\pi(w)$ and setting $\sigma'(w) = \sigma(\pi(w))$. ◀

4.3 On winning positional strategies of Eve

In the next section we show how to use use automata-based techniques to decide the existence of a (positional) winning strategy for Eve. These techniques rely on characterizing whether a positional strategy of Eve is surely, almost-surely and positively winning.

4.3.1 Surely and almost-surely winning conditions

We characterize (almost-)surely winning strategies.

▶ **Definition 11** (q-branches)**.** Let $q \in Q$ and σ a strategy. An infinite branch of the binary tree is a q-branch in σ if at least one σ-play which projects to this branch has limsup q.

▶ **Lemma 12.** *Assume the automaton has limited choice for Adam. Let σ be a positional strategy for Eve. Then σ is surely winning iff for every $q \in (Q \setminus F_\forall)$ there is no q-branch in σ. Moreover σ is almost-surely winning iff for every $q \in (Q \setminus F_1)$ the set of q-branches in σ has measure 0.*

Whether a branch is a q-branch can be checked by computing a system of σ-indexes. Intuitively, all σ-reachable vertices receives a finite index, such that along a σ-play the index does not change except when Adam performs a non-canonical move or when two plays merge on the same vertex, in which case the smallest index is kept. After a non-canonical move of Adam, a new play may start in which case it receives a fresh index not used yet in the current neither in the parent node. For this less than $2|Q|$ indices are required. The important properties of σ-indexes are:

▶ **Lemma 13** (Characterization of q-branches). *Every positional strategy σ of Eve can be associated with a function $\sigma : \{0,1\}^* \times Q \to \{0,1,\dots,2|Q|,\infty\}^Q$ with the following properties.*

First, σ can be computed on-the-fly along a branch. For every node n denote σ_n the restriction of σ on $\{n\} \times Q$. Then $\sigma(\epsilon)$ only depends on σ_ϵ. And for every node n and $d \in \{0,1\}$, $\sigma(nd)$ only depends on $\sigma(n)$ and σ_{nd}.

Second, a vertex (n,q) is reachable from the initial vertex by a σ-play iff $\sigma(n)(q)$ is finite.

Third, let $b \in \{0,1\}^\omega$ be an infinite branch of the binary tree, visiting successively the nodes $n_0, n_1, n_2, \dots$. Denote $R^\infty(b)$ the set of pairs $(k,q) \in \{0,\dots,2|Q|\} \times Q$ such that: $k \in \sigma(n_i)(Q)$ for every $i \in \mathbb{N}$ except finitely many; and $k = \sigma(n_i)(q)$ for infinitely many $i \in \mathbb{N}$.

Finally, for every state q, the branch b is a q-branch if and only if there exists $k \in \{0,1,\dots,2|Q|\}$ such that $q = \max\{r \in Q \mid (k,r) \in R^\infty(b)\}$.

4.3.2 Checking the positively winning condition

In order to check with a non-deterministic automaton whether a positional strategy is positively winning, we rely on the notion of *positive witnesses*. The point of positive witnesses is to turn the verification of up to $|Q|$ positively-winning conditions - depending on the decisions of Adam, there may be up to $|Q|$ different σ-reachable vertices on a given node - into a single one. This single condition can then be checked by a non-deterministic nonzero automaton equipped with a single positively-winning condition.

Everywhere thick subtrees. We need the notion of everywhere thick subtrees. We measure sets of infinite branches with the uniform probability measure μ on $\{0,1\}^\omega$.

▶ **Definition 14** (Everywhere thick sets of nodes). For every set $T \subseteq \{0,1\}^*$ of nodes denote $\vec{T}$ the set of branches in $\{0,1\}^\omega$ whose every prefix belongs to T. Then T is *everywhere thick* if starting from every node $n \in T$ there is nonzero probability to stay in T, i.e. if $\mu\left(\vec{T} \cap n\{0,1\}^\omega\right) > 0$.

Everywhere thick subtrees are almost everywhere.

▶ **Lemma 15.** *Let $P \subseteq \{0,1\}^\omega$ be a measurable set of infinite branches. Assume $\mu(P) > 0$. Then there exists an everywhere thick subtree T, with root ϵ such that $\vec{T} \subseteq P$.*

The proof relies on the inner-regularity of μ, so that P can be assumed to be a closed set, i.e. a subtree from which we can prune leaves whose subtree has probability 0.

Positive witnesses. Positive witnesses can be used to check whether a strategy is positively winning:

▶ **Definition 16** (Positive plays and witnesses). Let t be a Σ-labelled binary tree and σ a positional strategy of Eve in the acceptance game of t. Let Z be the set of σ-reachable vertices whose state is in $F_{>0}$.

A play is positive if all vertices it visits belong to $\{0,1\}^* \times F_{>0}$. A positive witness for σ is a pair (W,E) where: $W \subseteq Z$ are the *active* vertices, and $E \subseteq \{0,1\}^* \times \{0,1\}$ is the set of *positive edges*, and they have the following properties.

a) From every vertex $z \in Z$ there is a positive and canonical finite σ-play starting in z which reaches a vertex in W or a transient vertex.

b) Let $z = (n, q) \in W$. Then $(n, 0) \in E$ or $(n, 1) \in E$, or both. If $z \to z'$ is a local transition then $z' \in W$ as well whenever ($q \in Q_E$ and $z \to z'$ is consistent with σ) or ($q \in Q_A$ and $z \to z'$ is canonical). If z is controlled by Eve and $\sigma(z)$ is a split transition $q \to (q_0, q_1)$ then $((n, 0) \in E \implies (n0, q_0) \in W)$ and $((n, 1) \in E \implies (n1, q_1) \in W)$.

c) The set of nodes $\{nd \in \{0, 1\}^* \mid (n, d) \in E\}$ is everywhere thick.

▶ **Lemma 17** (Characterization of positively winning strategies)**.** *Assume the automaton has limited choice for Adam. A positional strategy σ for Eve is positively winning iff there exists a positive witness for σ.*

4.4 Deciding emptiness

A Σ-labelled binary tree t and a positional strategy σ in the corresponding acceptance game generate a tree $T_{t,\sigma} : \{0, 1\}^* \to (Q \cup Q \times Q)^{Q_E}$.

For every vertex (n, q) controlled by Eve, if $\sigma(n, q)$ is a local transition $q \to_{t(n)} q'$ then $T_{t,\sigma}(n)(q) = q'$ and if $\sigma(n, q)$ is a split transition $q \to_{t(n)} (q_0, q_1)$ then $T_{t,\sigma}(n)(q) = (q_0, q_1)$.

▶ **Definition 18** (Strategic tree)**.** A tree $T : \{0, 1\}^* \to (Q \cup Q \times Q)^{Q_E}$ is *strategic* if there exists a tree $t : \{0, 1\}^* \to \Sigma$ and a positional strategy σ for Eve such that $T = T_{t,\sigma}$.

We are interested in the strategic trees associated to winning strategies. The rest of the section is dedicated to the proof of the following theorem.

▶ **Theorem 19.** *Fix an alternating nonzero automata with limited choice for Adam. The language of strategic trees $T_{t,\sigma}$ such that σ wins the acceptance game of t can be recognized by a non-deterministic nonzero automaton of size exponential in $|Q|$. If $F_\forall = Q$ in the alternating automaton, then the sure condition of the non-deterministic automaton is Büchi.*

Proof. The characterizations of surely, almost-surely and positively winning strategies given in lemmas 12, 13 and 17 can be merged as follows.

▶ **Corollary 20.** *Let σ be a positional strategy σ for Eve. For every branch b denote $M(b) = \{\max\{q \mid (k, q) \in R^\infty(b)\} \mid k \in 0 \dots 2|Q|\}$.*

Then σ is winning if and only if for every branch b, $M(b) \subseteq F_\forall$; for almost-every branch b, $M(b) \subseteq F_1$; and there exists a positive witness for σ.

First of all, the non-deterministic automaton $\mathcal{B}$ checks whether the input tree is a strategic tree, for that it guesses on the fly the input tree $t : \{0, 1\}^* \to \Sigma$ by guessing on node n the value of $t(n)$ and checking that for every $q \in Q_E$, $q \to_{t(n)} T(n)(q)$ is a transition of the automaton.

On top of that $\mathcal{B}$ checks the three conditions of Corollary 20. For the first two conditions, it computes (asymptotically) along every branch b the value of $R^\infty(b)$ and thus of $M(b)$. For that the automaton relies on a Last Appearance Record memory (LAR) [10] whose essential properties are:

▶ **Lemma 21** (LAR memory [10])**.** *Let C be a finite set of symbols. There exists a deterministic automaton on C called the* LAR memory *on C with the following properties. First, the set of states, denoted Q, has size $\leq |C|^{|C|+1}$ and is totally ordered. Second, for every $u \in C^\omega$ denote $L^\infty(u)$ the set of letters seen infinitely often in u and $\mathrm{LAR}(u)$ the largest state seen infinitely often during the computation on u. Then $L^\infty(u)$ can be inferred from $\mathrm{LAR}(u)$, precisely there is a mapping $\phi : Q \to 2^C$ such that: $\forall u \in C^\omega, L^\infty(u) = \phi(\mathrm{LAR}(u))$.*

In order to compute $R^\infty(b)$ along a branch b, the non-deterministic automaton $\mathcal{B}$ computes deterministically on the fly the σ-index of the current node n, as defined in Lemma 13, and implements a LAR memory on the alphabet $C = \{0, \dots, 2|Q|\} \times (Q \cup \{\bot\})$.

When visiting node n, $\mathcal{B}$ injects into the LAR memory all pairs $(\sigma(q), q)$ such that $q \in Q$ and $\sigma(q) \neq \infty$ plus all pairs $(k, \bot)$ such that $k \notin \sigma(n)(Q)$. For every branch b, the set $R^\infty(b)$ is equal to all pairs (k, q) seen infinitely often such that $(k, \bot)$ is seen only finitely often. Thus, the LAR memory can be used to check the first two conditions of Corollary 20, more details are given at the end of the proof.

For now, we describe how the non-deterministic automaton $\mathcal{B}$ checks whether there exists a positive witness (W, E) (Definition 16). Denote by Z the set of σ-reachable vertices whose state is in $F_{>0}$. On node n the automaton guesses (resp. computes) the vertices of W (resp. Z) of the current node and guesses the elements of E by storing three sets of states: $W_n = \{q \in Q \mid (n, q) \in W\}$; $Z_n = \{q \in F_{>0} \mid \sigma(n, q) < \infty\}$; and $E_n = \{b \in \{0, 1\} \mid (n, b) \in E\}$.

Then $\mathcal{B}$ checks conditions a), b) and c) in the definition of a positive witness as follows.

$\mathcal{B}$ checks condition a) in the definition of a positive witness by guessing on the fly for every vertex in Z a canonical positive σ-play to a vertex which is either transient or in W, in which case we say the canonical positive play *terminates*.

For that $\mathcal{B}$ maintains an ordered list P_n of states. On the root node, P_ϵ is $Z_\epsilon \setminus W_\epsilon$. When the automaton performs a transition, it guesses for each state q in P_n and direction b_q a successor s_q, such that (nb_q, s_q) can be reached from (q, n) by a positive canonical σ-play. In direction b, every state q for which $b_q \neq b$ is removed from the list, while every state q for which $b_q = b$ is replaced by the corresponding s_q. Then all states in Z_{nb} are added at the end of the list. In case of duplicates copies of the same state in the list, only the first copy is kept. In case the head of the list is in W_{nb} or is transient, a Büchi condition is triggered and the head is moved at the back of the list. Finally, all entries of the list which are in W_{nb} are removed.

This way, condition a) holds iff the Büchi condition is triggered infinitely often on every branch. We discuss below how to integrate this Büchi condition in the sure accepting condition of the automaton.

$\mathcal{B}$ checks condition b) in the definition of a positive witness by entering an absorbing error state as soon as

1) there is some local transition $(n, q) \to_{t(n)} (n, q')$ such that $q \in W_n$ and ($q \in Q_E$ and $z \to z'$ is consistent with σ) or ($q \in Q_A$ and $z \to z'$ is canonical); or

2) there is some $q \in W_n$ controlled by Eve and $b \in E_n$ such that $\sigma(n, q)$ is a split transition $q \to_{t(n)} (q_0, q_1)$ but $q_b \notin W_{nb}$.

The guessed sets W_n are bound to satisfy condition 1) and condition 2) is checked by storing a subset of Q.

$\mathcal{B}$ checks condition c) in the definition of a positive witness by triggering the positive acceptance condition whenever it moves in direction b on a node n such that $b \in E_n$.

The sure and almost-sure acceptance condition are defined as follows. The Büchi condition necessary for checking condition a) in the definition of a positive witness is integrated in the LAR memory, for that we add to the alphabet C of the LAR memory a new symbol $\top$ which is injected in the LAR memory whenever the Büchi condition is triggered. The order between states of $\mathcal{B}$ is induced by the order of the LAR memory.

This way, according to Lemma 21, the largest state seen infinitely often along a branch b reveals whether $\top$ was seen infinitely often, and reveals the value of $R^\infty(b)$ (the set of pairs (k, q) seen infinitely often such that $(k, \bot)$ was seen finitely often) hence of $M(b)$ as well. The state is surely (resp. almost-surely) accepting iff $\top$ was seen infinitely often and $M(b) \subseteq F_\forall$

(resp. $M(b) \subseteq F_1$). In case $F_\forall = Q$ in the alternating automaton then the sure condition boils down to the Büchi condition.

According to Corollary 20, and by construction of $\mathcal{B}$, the computation of $\mathcal{B}$ is accepting iff the input is a strategic tree whose corresponding strategy of Eve is winning. ◀

▶ **Theorem 22.** *Emptiness of alternating nonzero automata with limited choice for Adam is decidable in* NEXPTIME∩*co*-NEXPTIME. *If $F_\forall = Q$, emptiness can be decided in* EXPTIME.

Proof. Emptiness of an alternating automaton reduces to the emptiness of a non-deterministic automaton of exponential size. This non-deterministic automaton guesses on-the-fly a tree $\{0,1\}^* \to (Q \cup Q \times Q)^{Q_E}$ and checks it is a winning strategic tree, using the automaton given by Theorem 19. In case the alternating automaton is $F_\forall$-trivial, the sure condition of the non-deterministic automaton is Büchi (Theorem 19). We conclude with Theorem 3. ◀

5 Satisfiability of CTL*$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$

Our result on alternating nonzero automata can be applied to decide the satisfiability of the logic CTL*$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$, a generalization of CTL* which integrates both deterministic and probabilistic state quantifiers.

Markov chains. The models of CTL*$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ formulas are Markov chains. A Markov chain with alphabet Σ is a tuple $\mathcal{M} = (S, p, t)$ where S is the (countable) set of *states*, $p : S \to \Delta(S)$ are the *transition probabilities* and $t : S \to \Sigma$ is the *labelling function*.

For every state $s \in S$, there is a unique probability measure denoted $\mathbb{P}_{\mathcal{M},s}$ on S^ω such that $\mathbb{P}_{\mathcal{M},s}(sS^\omega) = 1$ and for every sequence $s_0 \cdots s_n s_{n+1} \in S^*$, $\mathbb{P}_{\mathcal{M},s}(s_0 \cdots s_n s_{n+1} S^\omega) = p(s_n, s_{n+1}) \cdot \mathbb{P}_{\mathcal{M},s}(s_0 s_1 \cdots s_n S^\omega)$. When $\mathcal{M}$ is clear from the context this probability measure is simply denoted $\mathbb{P}_s$. A *path* in $\mathcal{M}$ is a finite or infinite sequence of states $s_0 s_1 \cdots$ such that $\forall n \in \mathbb{N}, p(s_n, s_{n+1}) > 0$ .. We denote $\mathrm{Path}_\mathcal{M}(s_0)$ the set of such paths.

A binary tree $t : \{0,1\}^* \to \Sigma$ is seen as a specific type of Markov chain, where from every node $n \in \{0,1\}^*$ there is equal probability $\frac{1}{2}$ to perform transitions to $n0$ or $n1$.

Syntax. For a fixed alphabet Σ, there are two kinds of formula: state formula (typically denoted ψ) and path formula (denoted ϕ), generated by the following grammar:

$$\psi ::= \top \mid \bot \mid a \in \Sigma \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid \exists\phi \mid \forall\phi \mid \mathbb{P}_{>0}(\phi) \mid \mathbb{P}_{=1}(\phi)$$
$$\phi ::= \psi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi \mid G\phi \ .$$

Semantics. Let $\mathcal{M} = (S, t, p)$ a Markov chain. We define simultaneously and inductively the satisfaction $\mathcal{M}, s \models \psi$ of a state formula ψ by a state $s \in S$ and the satisfaction $\mathcal{M}, w \models \phi$ of a path formula ϕ by a path $w \in \mathrm{Path}_\mathcal{M}$. When $\mathcal{M}$ is clear from the context, we simply write $s \models \psi$ and $w \models \phi$.

If a state formula is produced by one of the rules $\top \mid \bot \mid p \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi$, its satisfaction is defined as usual. If ϕ is a path formula and $\psi \in \{\exists\phi, \forall\phi, \mathbb{P}_{>0}(\phi), \mathbb{P}_{=1}(\phi)\}$ then

$$\begin{aligned} s &\models \exists\phi && \text{if } \exists w \in \mathrm{Path}_\mathcal{M}(s), w \models \phi \\ s &\models \forall\phi && \text{if } \forall w \in \mathrm{Path}_\mathcal{M}(s), w \models \phi \\ s &\models \mathbb{P}_{\sim b}(\phi) && \text{if } \mathbb{P}_{\mathcal{M},s}(w \in \mathrm{Path}_\mathcal{M}(s) \mid w \models \phi) \sim b \ . \end{aligned}$$

The satisfaction of a path formula ϕ by an infinite path $w = s_0 s_1 \cdots \in \mathrm{Path}_\mathcal{M}(s_0)$ is defined as follows. If ϕ is produced by one of the rules $\neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$ then its satisfaction

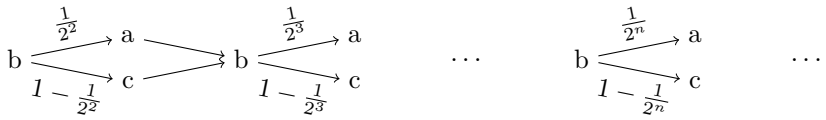


Figure 1 A model of $(\forall(G\exists(\top U a))) \wedge (\mathbb{P}_{>0}(G\neg a))$.

is defined as usual. If ϕ is a state formula (rule $\phi := \psi$) then $w \models \psi$ if $s_0 \models \psi$. Otherwise, $\phi \in \{X\phi', G\phi', \phi_1 U \phi_2\}$ where ϕ', ϕ_1 and ϕ_2 are path formulas. For every integer k, we denote $w[k]$ the path $s_k s_{k+1} \cdots \in \mathrm{Path}_{\mathcal{M}}(s_k)$. Then:

$$
\begin{array}{ll}
w \models X\phi' & \text{if } w[1] \models \phi' \\
w \models G\phi' & \text{if } \forall i \in \mathbb{N}, w[i] \models \phi' \\
w \models \phi_1 U \phi_2 & \text{if } \exists n \in \mathbb{N}, (\forall 0 \leq i < n, w[i] \models \phi_1 \wedge w[n] \models \phi_2).
\end{array}
$$

The Markov chain given in Figure 1 satisfies the formula $(\forall(G\exists(\top U a))) \wedge (\mathbb{P}_{>0}(G\neg a))$.

A formula for PUCE trees. The language of PUCE trees introduced in Section 3 can be described by the following CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ formula:

$$\forall G(\mathbb{P}_{>0}(\top U(G\neg a)) \wedge \mathbb{P}_{>0}(\top U(G\neg b))) \ .$$

Variants and fragments. A formula of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ belongs to the fragment CTL if in each of its state subformula ψ of type $\exists\phi \mid \forall\phi \mid \mathbb{P}_{>0}(\phi) \mid \mathbb{P}_{=1}(\phi)$ the path formula ϕ has type $X\psi' \mid \psi' U \psi'' \mid G\psi'$ where ψ' and ψ'' are state subformulas.

In the variant ECTL, every path formula ϕ is described as the composition of a deterministic Büchi automata on some alphabet $\{0,1\}^k$ with k state subformulas. A path satisfies ϕ if the Büchi automaton accepts the sequence of letters obtained by evaluating the k state subformulas on every state along the path. This variant augments both the expressivity and the conciseness of the logic at the cost of a less intuitive syntax. For more details see [6].

We are also interested in the fragments where the operators $\exists$ and $\forall$ are not used, i.e. the qualitative fragments of the logics pCTL*, pECTL and pCTL.

Satisfiability problem. A Markov chain $\mathcal{M}$ *satisfies* a formula ξ at state s, or equivalently $(\mathcal{M}, s)$ *is a model* of ξ, if $\mathcal{M}, s \models \xi$. We are interested in the problem:
MC-SAT: given a formula, does it have a model?

This logic is an extension of monadic second-order logic on infinite binary trees with a new probabilistic operator [14, 15, 2]. The satisfiability of this logic is reducible to the emptiness problem for nonzero non-deterministic automata [2] which is decidable [3]. Since CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ is a fragment of TMSO+ZERO, this result implies that the satisfiability of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ is decidable with non-elementary complexity. The reduction to the emptiness of alternating nonzero automata given in the present paper provides a better complexity bound.

▶ **Theorem 23.** *For CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ the satisfiability problem is in 3-*NEXPTIME ∩ *co-3-*NEXPTIME*. The following table summarizes complexities of the satisfiability problem for various fragments and variants of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$:*

	CTL*	*ECTL*	*CTL*
$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$	*3-*NEXPTIME ∩ *co-3-*NEXPTIME	*2-*NEXPTIME ∩ *co-2-*NEXPTIME	NEXPTIME ∩ *co-*NEXPTIME
$[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$	*3-*EXPTIME *[6] (qualitative pCTL*)*	*2-*EXPTIME *[6] (qualitative pECTL)*	EXPTIME *[5] (qualitative pCTL)*

According to [5, 6], the complexities for ECTL$[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$ and CTL$[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$ are optimal.

The first step in the proof of Theorem 23 is a linear-time reduction from **MC-SAT** to: **BIN-SAT:** given a formula, does it have a model among binary trees?

▶ **Theorem 24.** *Any formula ξ of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ on alphabet Σ can be effectively transformed into a formula ξ' of linear size on alphabet $\Sigma \cup \{\circ\}$ such that ξ is **MC-SAT** iff ξ' is **BIN-SAT**. As a consequence, **MC-SAT** linearly reduces to **BIN-SAT**. This transformation preserves the fragment CTL$^*[\mathbb{P}_{>0}, \mathbb{P}_{=1}]$.*

The second step is a standard translation from logic to alternating automata [12].

▶ **Lemma 25.** *For every formula ξ of CTL$^*[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$ (resp. ECTL$[\exists, \forall, \mathbb{P}_{>0}, \mathbb{P}_{=1}]$), there is an alternating automaton $\mathcal{A}$ with limited choice for Adam whose language is the set of binary trees satisfying the formula at the root. The automaton is effectively computable, of size $O(2^{2^{|\xi|}})$ (resp. $\mathcal{O}(2^{|\xi|})$). If ξ is a CTL formula, the size of $\mathcal{A}$ is $\mathcal{O}(|\xi|)$. In case the formula does not use the $\exists$ and $\forall$ operators, the $F_\forall$ condition is trivial i.e. $F_\forall = Q$.*

Proof of Theorem 23. All the complexity results are obtained by reduction of **MC-SAT** to the emptiness problem for an alternating nonzero automaton with limited choice for Adam, which is decidable in NEXPTIME∩co-NEXPTIME (Theorem 22). The size of the automaton varies from doubly-exponential to linear size depending on whether the formula is in CTL*, ECTL or CTL (Lemma 25). In case the formula does not use the deterministic operators $\exists$ and $\forall$ (i.e. for qualitative pCTL*, pECTL and pCTL) the $F_\forall$ condition of the alternating automaton is trivial thus its emptiness is decidable in EXPTIME (Theorem 22). ◀

Conclusion

We have introduced the class of *alternating nonzero* automata, proved decidability of the emptiness problem for the subclass of automata with limited choice for Adam and obtained as a corollary algorithms for the satisfiability of a temporal logic extending both CTL* and the qualitative fragment of pCTL*.

A natural direction for future work is to find more general classes of alternating nonzero automata with a decidable emptiness problem, which requires some more insight on the properties of the acceptance games, in particular the existence of positional strategies in the acceptance game.

References

1. Christel Baier, Marcus Größer, and Nathalie Bertrand. Probabilistic ω-automata. *J. ACM*, 59(1):1, 2012.
2. Mikołaj Bojańczyk. Thin MSO with a probabilistic path quantifier. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 96:1–96:13, 2016.
3. Mikołaj Bojańczyk, Hugo Gimbert, and Edon Kelmendi. Emptiness of zero automata is decidable. *CoRR*, abs/1702.06858, 2017. URL: `http://arxiv.org/abs/1702.06858`.
4. Mikołaj Bojańczyk, Hugo Gimbert, and Edon Kelmendi. Emptiness of zero automata is decidable. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 106:1–106:13, 2017.
5. Tomás; Brázdil, Vojtech Forejt, Jan Kretínský, and Antonín Kucera. The satisfiability problem for probabilistic CTL. In *Proc. of LICS*, pages 391–402, 2008.

6 Tomáš Brázdil, Vojtěch Forejt, and Antonín Kučera. Controller synthesis and verification for markov decision processes with qualitative branching time objectives. *Automata, Languages and Programming*, pages 148–159, 2008.

7 Arnaud Carayol, Axel Haddad, and Olivier Serre. Randomization in automata on infinite trees. *ACM Trans. Comput. Log.*, 15(3):24:1–24:33, 2014.

8 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

9 E Allen Emerson and Joseph Y Halpern. Sometimes and not never revisited: on branching versus linear time temporal logic. *Journal of the ACM (JACM)*, 33(1):151–178, 1986.

10 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of STOC'82*, pages 60–65, New York, NY, USA, 1982. ACM.

11 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

12 Orna Kupferman, Moshe Y Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

13 Daniel Lehmann and Saharon Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–1983, 1982.

14 Henryk Michalewski and Matteo Mio. Measure quantifier in monadic second order logic. In *Proc. of LFCS 2016*, pages 267–282, 2016. doi:10.1007/978-3-319-27683-0_19.

15 Henryk Michalewski, Matteo Mio, and Mikołaj Bojańczyk. On the regular emptiness problem of subzero automata. In *Proc. of ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pages 1–23, 2016.

16 David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2):267 – 276, 1987. doi:http://dx.doi.org/10.1016/0304-3975(87)90133-2.

17 A. Paz. *Introduction to probabilistic automata.* Academic Press, 1971.

18 M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

19 Moshe Y Vardi and Larry Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 240–251. ACM, 1985.

Affine Extensions of Integer Vector Addition Systems with States

Michael Blondin[1]
Technische Universität München, Germany
blondin@in.tum.de
https://orcid.org/0000-0003-2914-2734

Christoph Haase
University of Oxford, United Kingdom
christoph.haase@cs.ox.ac.uk
https://orcid.org/0000-0002-5452-936X

Filip Mazowiecki[2]
LaBRI, Université de Bordeaux, France
filip.mazowiecki@u-bordeaux.fr

Abstract

We study the reachability problem for affine $\mathbb{Z}$-VASS, which are integer vector addition systems with states in which transitions perform affine transformations on the counters. This problem is easily seen to be undecidable in general, and we therefore restrict ourselves to affine $\mathbb{Z}$-VASS with the finite-monoid property (afmp-$\mathbb{Z}$-VASS). The latter have the property that the monoid generated by the matrices appearing in their affine transformations is finite. The class of afmp-$\mathbb{Z}$-VASS encompasses classical operations of counter machines such as resets, permutations, transfers and copies. We show that reachability in an afmp-$\mathbb{Z}$-VASS reduces to reachability in a $\mathbb{Z}$-VASS whose control-states grow polynomially in the size of the matrix monoid. Our construction shows that reachability relations of afmp-$\mathbb{Z}$-VASS are semilinear, and in particular enables us to show that reachability in $\mathbb{Z}$-VASS with transfers and $\mathbb{Z}$-VASS with copies is PSPACE-complete.

2012 ACM Subject Classification Theory of computation → Logic and verification, Theory of computation → Automata over infinite objects, Theory of computation → Complexity classes

Keywords and phrases Vector addition systems, affine transformations, reachability, semilinear sets, computational complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.14

Acknowledgements We are thankful to James Worrell for insightful discussions on transfer VASS.

1 Introduction

Vector addition systems with states (VASS) are a fundamental model of computation comprising a finite-state controller with a finite number of counters ranging over the natural numbers. When a transition is taken, a counter can be incremented or decremented provided that the resulting counter value is greater than or equal to zero. Since the counters of a

[1] Supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT).
[2] 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).

VASS are unbounded, a VASS gives rise to an infinite transition system. One of the biggest advantages of VASS is that most of the standard decision problems such as configuration reachability and coverability are decidable [26, 32, 27, 29]. Those properties make VASS and their extensions a prime choice for reasoning about and modelling concurrent, distributed and parametrised systems, see *e.g.* the recent surveys by Abdulla and Delzanno [2, 15].

In order to increase their modelling power, numerous extensions of plain VASS have been proposed and studied in the literature over the last 25 years. Due to the infinite-state nature of VASS, even minor extensions often cross the undecidability frontier. For example, while in the extension of VASS with hierarchical zero-tests on counters both reachability and coverability remain decidable [37, 10], all important decision problems for VASS with two counters which can arbitrarily be tested for zero become undecidable [33]. Another example is the extension of VASS with resets and transfers. In a *reset VASS*, transitions may set a counter to zero, whereas *transfer VASS* generalise reset VASS and allow transitions to move the contents of a counter onto another. While it was initially widely believed that any extension of VASS either renders both reachability and coverability undecidable, reset and transfer VASS have provided an example of an extension which leads to an undecidable reachability [5] yet decidable coverability problem [16]. Nevertheless, the computational costs for those extensions are high: while coverability is EXPSPACE-complete for VASS [30, 35], it becomes Ackermann-complete in the presence of resets and transfers [38, 19]. For practical purposes, the extension of VASS with transfers is particularly useful since transfer VASS allow for reasoning about broadcast protocols and multi-threaded non-recursive C programs [17, 25]. It was already observed in [17] that transfer VASS can be viewed as an instance of so-called *affine VASS*. An affine VASS is an extended VASS with transitions labelled by pairs $(\mathbf{A}, \boldsymbol{b})$, where $\mathbf{A}$ is a $d \times d$ matrix over the integers and $\boldsymbol{b} \in \mathbb{Z}^d$ is an integer vector. A transition switches the control-state while updating the configuration of the counters $\boldsymbol{v} \in \mathbb{N}^d$ to $\mathbf{A} \cdot \boldsymbol{v} + \boldsymbol{b}$, provided that $\mathbf{A} \cdot \boldsymbol{v} + \boldsymbol{b} \geq \mathbf{0}$; otherwise, the transition is blocked. Transfer VASS can be viewed as affine VASS in which the columns of all matrices are d-dimensional unit vectors [17].

Due to the symbolic state-explosion problem and Ackermann-hardness of coverability, standard decision procedures for transfer VASS such as the backward algorithm [1] do not *per se* scale to real-world instances. In recent years, numerous authors have proposed the use of over-approximations in order to attenuate the symbolic state-explosion problem for VASS and some of their extensions (see, *e.g.*, [18, 6, 8]). Most commonly, the basic idea is to relax the integrality or non-negativity condition on the counters and to allow them to take values from the integers or non-negative rational numbers. It is easily seen that if a configuration is not reachable under the relaxed semantics, then the configuration is also not reachable under the standard semantics. Hence, those over-approximations can, for instance, be used in order to prune the sets of minimal basis elements in every iteration of the backward algorithm. In this paper, we investigate reachability in *integer over-approximations* of affine VASS, *i.e.*, affine VASS in which a configuration of the counters is a point in $\mathbb{Z}^d$, and in which all transitions are non-blocking. Subsequently, we refer to such VASS as *affine $\mathbb{Z}$-VASS*.

Main contributions

We focus on affine $\mathbb{Z}$-VASS with the *finite-monoid property* (afmp-$\mathbb{Z}$-VASS), *i.e.* where the matrix monoid generated by all matrices occurring along transitions in the affine $\mathbb{Z}$-VASS is finite. By a reduction to reachability in $\mathbb{Z}$-VASS, we obtain decidability of reachability for the whole class of afmp-$\mathbb{Z}$-VASS and semilinearity of their reachability relations.

More precisely, we show that reachability in an afmp-$\mathbb{Z}$-VASS can be reduced to reachability in a $\mathbb{Z}$-VASS whose size is polynomial in the size of the original afmp-$\mathbb{Z}$-VASS and in the size of the finite monoid $\mathcal{M}$ generated by the matrices occurring along transitions,

denoted by $\|\mathcal{M}\|$. For all classes of affine transformations considered in the literature, $\|\mathcal{M}\|$ is bounded exponentially in the dimension of the matrices. This enables us to deduce a general PSPACE upper bound for extensions of $\mathbb{Z}$-VASS such as transfer $\mathbb{Z}$-VASS and copy $\mathbb{Z}$-VASS. By a slightly more elaborated analysis of this construction, we are also able to provide a short proof of the already known NP upper bound for reset $\mathbb{Z}$-VASS [21].

We also show that a PSPACE lower bound of the reachability problem already holds for the extension of reset $\mathbb{Z}$-VASS with permutations. This gives PSPACE-completeness of some interesting classes such as transfer $\mathbb{Z}$-VASS and copy $\mathbb{Z}$-VASS. Finally, we show that an affine $\mathbb{Z}$-VASS that allows for both transfers and copies may not have the finite-monoid property, and the reachability problem for this class becomes undecidable. All complexity results obtained in this paper are summarized in Figure 1.

Related work

Our work is primarily related to the work of Finkel and Leroux [20], Iosif and Sangnier [24], Haase and Halfon [21], and Cadilhac, Finkel and McKenzie [12, 13]. In [20], Finkel and Leroux consider a model more general than affine $\mathbb{Z}$-VASS in which transitions are additionally equipped with guards which are Presburger formulas defining admissible sets of vectors in which a transition does not block. Given a sequence of transitions σ, Finkel and Leroux show that the reachability set obtained from repeatedly iterating σ, *i.e.*, the *acceleration* of σ, is definable in Presburger arithmetic. Note that the model of Finkel and Leroux does not allow for control-states and the usual tricks of encoding each control-state by a counter or all control-states into three counters [22] do not work over $\mathbb{Z}$ since transitions are non blocking. Iosif and Sangnier [24] investigated the complexity of model checking problems for a variant of the model of Finkel and Leroux with guards defined by convex polyhedra and with control-states over a flat structure. Haase and Halfon [21] studied the complexity of the reachability, coverability and inclusion problems for $\mathbb{Z}$-VASS and reset $\mathbb{Z}$-VASS, two submodels of the affine $\mathbb{Z}$-VASS that we study in this paper. In [12, 13], Cadilhac, Finkel and McKenzie consider an extension of Parikh automata to affine Parikh automata with the finite-monoid restriction like in our paper. These are automata recognizing boolean languages, but the finite-monoid restriction was exploited in a similar way to obtain some decidability results in that context. We finally remark that our models capture variants of cost register automata that have only one + operation [4, 3].

Structure of the paper

We introduce general notations and affine $\mathbb{Z}$-VASS in Section 2. In Section 3, we give the reduction from afmp-$\mathbb{Z}$-VASS to $\mathbb{Z}$-VASS. Subsequently, in Section 4 we show that afmp-$\mathbb{Z}$-VASS have semilinear reachability relations and discuss semilinearity of affine $\mathbb{Z}$-VASS in general. In Section 5, we show the PSPACE and NP upper bounds of the reachability problem for some classes of afmp-$\mathbb{Z}$-VASS; and in Section 6 we show PSPACE-hardness and undecidability results for some classes of affine $\mathbb{Z}$-VASS. Some concluding remarks will be made in Section 7.

2 Preliminaries

General notation

For every $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, 2, \ldots, n\}$. For every $\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{Z}^d$ and every $i \in [d]$, we write $\boldsymbol{x}(i) \stackrel{\text{def}}{=} x_i$. We denote the identity matrix and the zero-vector by $\mathbf{I}$ and $\mathbf{0}$ in every dimension, as there will be no ambiguity. For every $\boldsymbol{x} \in \mathbb{Z}^d$ and $\mathbf{A} \in \mathbb{Z}^{d \times d}$, we define

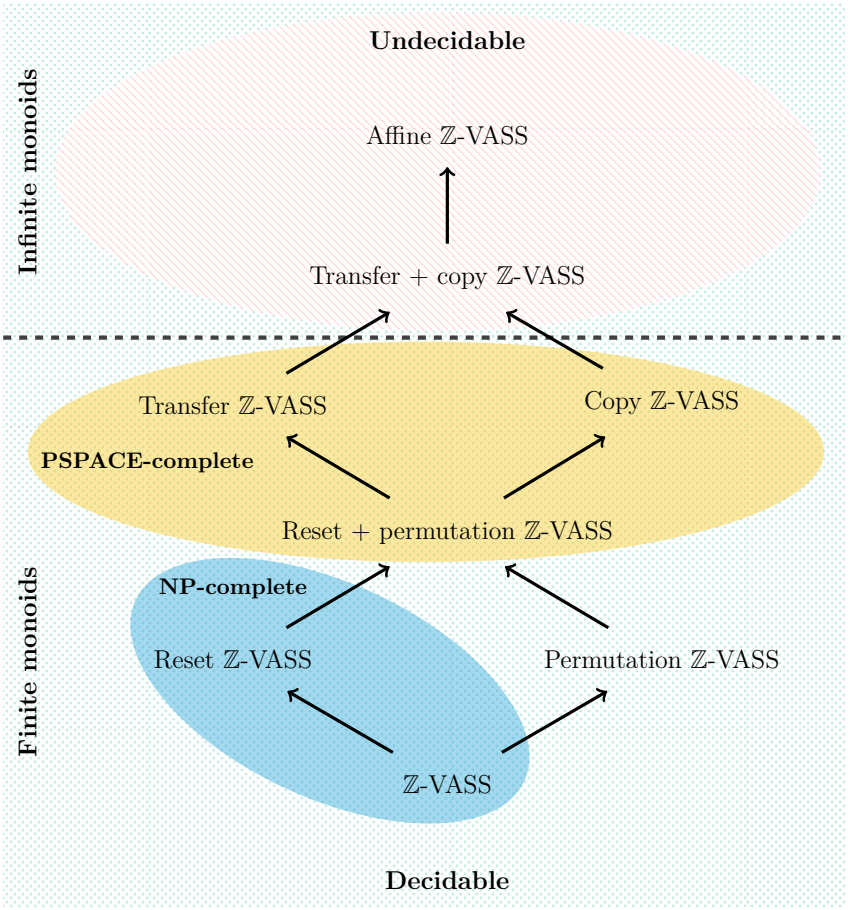


Figure 1 Classification of the complexity of reachability in affine ℤ-VASS in terms of classes of matrices. The rectangular regions below and above the horizontal dashed line correspond to classes of matrices with finite and infinite monoids respectively. The green rectangular dotted region and the red elliptical striped region correspond to the classes where reachability is decidable and undecidable, respectively. The blue elliptical region and the orange elliptical region correspond to the classes where reachability is NP-complete and PSPACE-complete respectively. Reachability in permutation ℤ-VASS is NP-hard and belongs to PSPACE.

the *max-norm* of $\boldsymbol{x}$ and $\mathbf{A}$ as $\|\boldsymbol{x}\| \stackrel{\text{def}}{=} \max\{|\boldsymbol{x}(i)| : i \in [d]\}$ and $\|\mathbf{A}\| \stackrel{\text{def}}{=} \max\{\|\mathbf{A}_i\| : i \in [d]\}$ where $\mathbf{A}_i$ denotes the i^{th} column of $\mathbf{A}$. We assume that numbers are represented in binary, hence the entries of vectors and matrices can be exponential in the size of their encodings.

Affine Integer VASS

An *affine integer vector addition system with states (affine ℤ-VASS)* is a tuple $\mathcal{V} = (d, Q, T)$ where $d \in \mathbb{N}$, Q is a finite set and $T \subseteq Q \times \mathbb{Z}^{d\times d} \times \mathbb{Z}^d \times Q$. Let us fix such a $\mathcal{V}$. We call d the *dimension* of $\mathcal{V}$ and the elements of Q and T respectively *control-states* and *transitions*. For every transition $t = (p, \mathbf{A}, \boldsymbol{b}, q)$, let $\mathrm{src}(t) \stackrel{\text{def}}{=} p$, $\mathrm{tgt}(t) \stackrel{\text{def}}{=} q$, $M(t) \stackrel{\text{def}}{=} \mathbf{A}$ and $\Delta(t) \stackrel{\text{def}}{=} \boldsymbol{b}$, and let $f_t \colon \mathbb{Z}^d \to \mathbb{Z}^d$ be the affine transformation defined by $f_t(\boldsymbol{x}) = \mathbf{A} \cdot \boldsymbol{x} + \boldsymbol{b}$. The *size* of $\mathcal{V}$, denoted $|\mathcal{V}|$, is defined as $|\mathcal{V}| \stackrel{\text{def}}{=} d + |Q| + \|T\|$ where $\|T\| \stackrel{\text{def}}{=} \sum_{t\in T} d^2 \cdot \lceil \log(\|M(t)\| + \|\Delta(t)\| + 1) \rceil$.

A *configuration* of $\mathcal{V}$ is a pair $(q, \boldsymbol{v}) \in Q \times \mathbb{Z}^d$ which we write as $q(\boldsymbol{v})$. For every $t \in T$ and $p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{Z}^d$, we write $p(\boldsymbol{u}) \xrightarrow{t} q(\boldsymbol{v})$ if $p = \mathrm{src}(t)$, $q = \mathrm{tgt}(t)$ and $\boldsymbol{v} = f_t(\boldsymbol{u})$. We naturally extend $\to$ to sequences of transitions as follows. For every $w \in T^*$ and

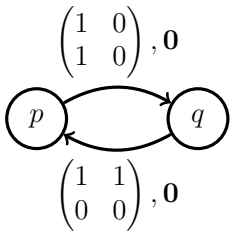


Figure 2 Example of a transfer + copy $\mathbb{Z}$-VASS $\mathcal{V}$ which does not have the finite-monoid property.

$p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{Z}^d$, we write $p(\boldsymbol{u}) \xrightarrow{w} q(\boldsymbol{v})$ if either $|w| = 0$ and $p(\boldsymbol{u}) = q(\boldsymbol{v})$, or $|w| = k > 0$ and there exist $p_0(\boldsymbol{u}_0), p_1(\boldsymbol{u}_1), \ldots, p_k(\boldsymbol{u}_k) \in Q \times \mathbb{Z}^d$ such that

$$p(\boldsymbol{u}) = p_0(\boldsymbol{u}_0) \xrightarrow{w_1} p_1(\boldsymbol{u}_1) \xrightarrow{w_2} \cdots \xrightarrow{w_k} p_k(\boldsymbol{u}_k) = q(\boldsymbol{v}).$$

We write $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ if there exists some $w \in T^*$ such that $p(\boldsymbol{u}) \xrightarrow{w} q(\boldsymbol{v})$. The relation $\xrightarrow{*}$ is called the *reachability relation* of $\mathcal{V}$. If $p(\boldsymbol{u}) \xrightarrow{w} q(\boldsymbol{v})$, then we say that w is a *run from* $p(\boldsymbol{u})$ *to* $q(\boldsymbol{v})$, or simply a *run* if the source and target configurations are irrelevant. We also say that w is a *path* from p to q, and if $p = q$ then we say that w is a *cycle*.

Let $M(\mathcal{V}) \stackrel{\text{def}}{=} \{M(t) : t \in T\}$ and $\Delta(\mathcal{V}) \stackrel{\text{def}}{=} \{\Delta(t) : t \in T\}$. If $\mathcal{V}$ is clear from the context, we sometimes simply write M and Δ. The *monoid of* $\mathcal{V}$, denoted $\mathcal{M}_\mathcal{V}$ or sometimes simply $\mathcal{M}$, is the monoid generated by $M(\mathcal{V})$, *i.e.* it is the smallest set that contains $M(\mathcal{V})$, is closed under matrix multiplication, and contains the identity matrix. We say that a matrix $\mathbf{A} \in \mathbb{Z}^{d\times d}$ is respectively a (i) *reset*, (ii) *permutation*, (iii) *transfer*, (iv) *copyless*, or (v) *copy* matrix if $\mathbf{A} \in \{0,1\}^{d\times d}$ and

- **(i)** $\mathbf{A}$ does not contain any 1 outside of its diagonal;
- **(ii)** $\mathbf{A}$ has exactly one 1 in each row and each column;
- **(iii)** $\mathbf{A}$ has exactly one 1 in each column;
- **(iv)** $\mathbf{A}$ has at most one 1 in each column;
- **(v)** $\mathbf{A}$ has exactly one 1 in each row.

Similarly, we say that $\mathcal{V}$ is respectively a *reset*, *permutation*, *transfer*, *copyless*, or *copy* $\mathbb{Z}$-VASS if all matrices of $M(\mathcal{V})$ are reset, permutation, transfer, copyless, or copy matrices. The monoids of such affine $\mathbb{Z}$-VASS are finite and respectively of size at most 2^d, $d!$, d^d, $(d+1)^d$ and d^d. Copyless $\mathbb{Z}$-VASS correspond to a model of copyless cost-register automata studied in [3] (see the remark below). If $M(\mathcal{V})$ only contains the identity matrix, then $\mathcal{V}$ is simply called a $\mathbb{Z}$-VASS. We define $\|\mathcal{M}_\mathcal{V}\| \stackrel{\text{def}}{=} |\mathcal{M}_\mathcal{V}| \cdot d^2 \cdot \max\{\log(\|\mathbf{A}\| + 1) : \mathbf{A} \in \mathcal{M}_\mathcal{V}\}$. Note that $\|\mathcal{M}_\mathcal{V}\| = |\mathcal{M}_\mathcal{V}| \cdot d^2$ for any monoid obtained from one of the above matrices types.

A *class of matrices* $\mathcal{C}$ is a union $\bigcup_{d\geq 1} \mathcal{C}_d$ where $\mathcal{C}_d$ is a finitely generated, but possibly infinite, submonoid of $\mathbb{Z}^{d\times d}$ for every $d \geq 1$. We say that $\mathcal{V}$ belongs to a class $\mathcal{C}$ of $\mathbb{Z}$-VASS if $\mathcal{M}_\mathcal{V} \subseteq \mathcal{C}$. If each $\mathcal{C}_d$ is finite, then we say that this class of affine $\mathbb{Z}$-VASS has the *finite-monoid property* (afmp-$\mathbb{Z}$-VASS). For two classes $\mathcal{C}$ and $\mathcal{C}'$ we write $\mathcal{C} + \mathcal{C}'$ to denote the smallest set $\mathcal{D} = \bigcup_{d\geq 1} \mathcal{D}_d$ such that $\mathcal{D}_d$ is a monoid that contains both $\mathcal{C}_d$ and $\mathcal{C}'_d$ for every $d \geq 1$. Notice that this operation does not preserve finiteness and for example the class of transfer + copy matrices is infinite (see Figure 2 and Section 6).

We discuss the $\mathbb{Z}$-VASS $\mathcal{V}$ in Figure 2 to give some intuition behind the names transfer and copy $\mathbb{Z}$-VASS. The transition from p to q is a copy transition and the transition from q to p is a transfer transition. Notice that for every vector $(x, y) \in \mathbb{Z}^2$, we have $p(x, y) \to q(x, x)$, *i.e.* the value of the first counter is copied to the second counter. Similarly, for the other

transition we have $q(x, y) \to p(x + y, 0)$, that is the value of the second counter is transferred to the first counter (resetting its own value to 0). Let $\mathbf{A}$ and $\mathbf{B}$ be the two matrices used in $\mathcal{V}$. Note that $(\mathbf{A} \cdot \mathbf{B})^n$ is the matrix with all entries equal to 2^{n-1}, and hence $\mathcal{M}_\mathcal{V}$ is infinite.

▶ Remark. The variants of affine $\mathbb{Z}$-VASS that we consider are related to cost register automata (CRA) with only the $+$ operation [4, 3] and without an output function. These are deterministic models with states and registers that upon reading an input, update their registers in the form $x \leftarrow y + c$, where x, y are registers and c is an integer. An affine $\mathbb{Z}$-VASS does not read any input, but is nondeterministic. Thus, one can identify an affine $\mathbb{Z}$-VASS with a CRA that reads sequences of transitions as words. In particular, the restrictions imposed on the studied CRAs correspond to copy $\mathbb{Z}$-VASS [4] and copyless $\mathbb{Z}$-VASS [3].

Decision problems

We consider the *reachability* and the *coverability* problems parameterized by classes of matrices $\mathcal{C}$:

$\mathsf{Reach}_\mathcal{C}$ (reachability problem)

GIVEN:	an affine $\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$ and configurations $p(\boldsymbol{u}), q(\boldsymbol{v})$ such that $\mathcal{M}_\mathcal{V} \subseteq \mathcal{C}$.
DECIDE:	whether $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$?

$\mathsf{Cover}_\mathcal{C}$ (coverability problem)

GIVEN:	an affine $\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$ and configurations $p(\boldsymbol{u}), q(\boldsymbol{v})$ such that $\mathcal{M}_\mathcal{V} \subseteq \mathcal{C}$.
DECIDE:	whether there exists $\boldsymbol{v}' \in \mathbb{Z}^d$ such that $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v}')$ and $\boldsymbol{v}' \geq \boldsymbol{v}$?

For standard VASS (where configurations cannot hold negative values), the coverability problem is considered much simpler than the reachability problem. However, for affine $\mathbb{Z}$-VASS, these two problems coincide as observed in [21, Lemma 2]: the two problems are inter-reducible in logarithmic space at the cost of doubling the number of counters. Therefore we will only study the reachability problem in this paper.

3 From affine $\mathbb{Z}$-VASS with the finite-monoid property to $\mathbb{Z}$-VASS

The main result of this section is that every affine $\mathbb{Z}$-VASS $\mathcal{V}$ with the finite monoid can be simulated by a $\mathbb{Z}$-VASS with twice the number of counters whose size is polynomial in $\|\mathcal{M}\|$ and $|\mathcal{V}|$. More formally, we show the following:

▶ **Theorem 1.** *For every afmp-$\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$ there exist a $\mathbb{Z}$-VASS $\mathcal{V}' = (d', Q', T')$ and $p', q' \in Q'$ such that*

- $d' = 2 \cdot d$,
- $|Q'| \leq 4 \cdot \|\mathcal{M}\|^2 \cdot |Q|$,
- $\|T'\| \leq 8d \cdot \|\mathcal{M}\|^2 \cdot |Q| + \|\mathcal{M}\|^4 \cdot \|T\|$,
- $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ *in* $\mathcal{V}$ *if and only if* $p'(\boldsymbol{u}, \mathbf{0}) \xrightarrow{*} q'(\mathbf{0}, \boldsymbol{v})$ *in* $\mathcal{V}'$.

Moreover, $\mathcal{V}'$, p' and q' are effectively computable from $\mathcal{V}$.

▶ **Corollary 2.** *The reachability problem for afmp-$\mathbb{Z}$-VASS is decidable.*

Proof. By Theorem 1, it suffices to construct, for a given afmp-$\mathbb{Z}$-VASS $\mathcal{V}$, the $\mathbb{Z}$-VASS $\mathcal{V}'$ and to test for reachability in $\mathcal{V}'$. It is known that reachability for $\mathbb{Z}$-VASS is in NP [21]. To effectively compute $\mathcal{V}'$ it suffices to provide a bound for $\|\mathcal{M}_\mathcal{V}\|$. It is known that if $|\mathcal{M}_\mathcal{V}|$ is finite then it is bounded by a computable function, which is an exponential tower (see [31]), and hence $\|\mathcal{M}_\mathcal{V}\|$ is also computable. ◀

For the remainder of this section, let us fix some affine $\mathbb{Z}$-VASS $\mathcal{V}$ such that $\mathcal{M}_\mathcal{V}$ is finite. We proceed as follows to prove Theorem 1. First, we introduce some notations and intermediary lemmas characterizing reachability in affine $\mathbb{Z}$-VASS. Next, we give a construction that essentially proves the special case of Theorem 1 where the initial configuration is of the form $p(\mathbf{0})$. Finally, we prove Theorem 1 by extending this construction to the general case.

It is worth noting that proving the general case is not necessary if one is only interested in deciding reachability. Indeed, an initial configuration $p(\boldsymbol{v})$ can be turned into one of the form $p'(\mathbf{0})$ by adding a transition that adds $\boldsymbol{v}$. The reason for proving the general case is that it establishes a stronger relation that allows us to prove semilinearity of afmp-$\mathbb{Z}$-VASS reachability relations in Section 4.

3.1 A characterization of reachability

For every $\sigma \in T^*$, $t \in T$ and $\boldsymbol{u} \in \mathbb{Z}^d$, let

$$M(\varepsilon) \stackrel{\text{def}}{=} \mathbf{I}, \qquad \varepsilon(\boldsymbol{u}) \stackrel{\text{def}}{=} \boldsymbol{u},$$
$$M(\sigma t) \stackrel{\text{def}}{=} M(t) \cdot M(\sigma), \qquad \sigma t(\boldsymbol{u}) \stackrel{\text{def}}{=} M(t) \cdot \sigma(\boldsymbol{u}) + \Delta(t).$$

Intuitively, for any sequence $w \in T^*$, $w(\boldsymbol{u})$ is the effect of w on $\boldsymbol{u}$, regardless of whether w is an actual path of the underlying graph. A simple induction yields the following characterization:

▶ **Lemma 3.** *For every $w \in T^*$ and $p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{Z}^d$, it is the case that $p(\boldsymbol{u}) \xrightarrow{w} q(\boldsymbol{v})$ if and only if*

(a) *w is a path from p to q in the underlying graph of $\mathcal{V}$, and*
(b) *$\boldsymbol{v} = w(\boldsymbol{u})$.*

Testing for reachability with Lemma 3 requires evaluating $w(\boldsymbol{u})$. This value can be evaluated conveniently as follows:

▶ **Lemma 4.** *For every $w \in T^k$ and $\boldsymbol{u} \in \mathbb{Z}^d$, the following holds:*

$$w(\boldsymbol{u}) = M(w) \cdot \boldsymbol{u} + \sum_{i=1}^{k} M(w_{i+1}w_{i+2} \cdots w_k) \cdot \Delta(w_i). \tag{1}$$

Moreover, $w(\boldsymbol{u}) = M(w) \cdot \boldsymbol{u} + w(\mathbf{0})$.

Proof of Lemma 4. We prove (1) by induction on k. The base case follows from $\varepsilon(\boldsymbol{u}) = \boldsymbol{u} = \mathbf{I} \cdot \boldsymbol{u} + \mathbf{0} = M(\varepsilon) \cdot \boldsymbol{u} + \mathbf{0}$. Assume that $k > 0$ and that the claim holds for sequences of length $k - 1$. For simplicity we denote $\sigma \stackrel{\text{def}}{=} w_1 \dots w_{k-1}$. We have:

$$\begin{aligned}
w(\boldsymbol{u}) &= \sigma w_k(\boldsymbol{u}) \\
&= M(w_k) \cdot \sigma(\boldsymbol{u}) + \Delta(w_k) && (2)\\
&= M(w_k) \cdot \left(M(\sigma) \cdot \boldsymbol{u} + \sum_{i=1}^{k-1} M(w_{i+1}w_{i+2} \cdots w_{k-1}) \cdot \Delta(w_i) \right) + \Delta(w_k) && (3)\\
&= M(w_k) \cdot M(\sigma) \cdot \boldsymbol{u} + \sum_{i=1}^{k-1} M(w_k) \cdot M(w_{i+1}w_{i+2} \cdots w_{k-1}) \cdot \Delta(w_i) + \Delta(w_k) \\
&= M(\sigma w_k) \cdot \boldsymbol{u} + \sum_{i=1}^{k-1} M(w_{i+1}w_{i+2} \cdots w_k) \cdot \Delta(w_i) + \Delta(w_k) && (4)\\
&= M(w) \cdot \boldsymbol{u} + \sum_{i=1}^{k} M(w_{i+1}w_{i+2} \cdots w_k) \cdot \Delta(w_i)
\end{aligned}$$

where (2), (3) and (4) follow respectively by definition of $\sigma w_k(\boldsymbol{u})$, by induction hypothesis and by definition of $M(\sigma w_k)$.

The last part of the lemma follows from applying (1) to $w(\mathbf{0})$ and $w(\boldsymbol{u})$, and observing that subtracting them results in $w(\boldsymbol{u}) - w(\mathbf{0}) = M(w) \cdot \boldsymbol{u}$. ◀

Observe that Lemma 4 is trivial for the particular case of $\mathbb{Z}$-VASS. Indeed, we obtain $w(\boldsymbol{u}) = \boldsymbol{u} + \sum_{i=1}^{k} \Delta(w_i)$, which is the sum of transition vectors as expected for a $\mathbb{Z}$-VASS.

3.2 Reachability from the origin

We make use of Lemmas 3 and 4 to construct a $\mathbb{Z}$-VASS $\mathcal{V}' = (d, Q', T')$ for the special case of Theorem 1 where the initial configuration is of the form $p(\mathbf{0})$. The states and transitions of $\mathcal{V}'$ are defined as:

$$Q' \stackrel{\text{def}}{=} Q \times \mathcal{M},$$

$$T' \stackrel{\text{def}}{=} \{((\mathrm{src}(t), \mathbf{A}), \mathbf{I}, \mathbf{B} \cdot \Delta(t), (\mathrm{tgt}(t), \mathbf{B})) : \mathbf{A}, \mathbf{B} \in \mathcal{M}, t \in T \text{ and } \mathbf{B} \cdot M(t) = \mathbf{A}\}.$$

The idea behind $\mathcal{V}'$ is to simulate a path w of $\mathcal{V}$ forward while evaluating $w(\mathbf{0})$ backwards. The latter can be evaluated as the sum identified in Lemma 4 provided that $\mathcal{V}'$ initially "knows" $M(w)$. More formally, $\mathcal{V}'$ and $\mathcal{V}$ are related as follows:

▶ **Proposition 5.**

(a) *For every $w \in T^*$ if $p(\mathbf{0}) \xrightarrow{w} q(\boldsymbol{v})$ in $\mathcal{V}$, then $p'(\mathbf{0}) \xrightarrow{*} q'(\boldsymbol{v})$ in $\mathcal{V}'$, where $p' = (p, M(w))$ and $q' = (q, \mathbf{I})$.*

(b) *If $p'(\mathbf{0}) \xrightarrow{*} q'(\boldsymbol{v})$ in $\mathcal{V}'$, where $p' = (p, \mathbf{A})$ and $q' = (q, \mathbf{I})$, then there exists $w \in T^*$ such that $M(w) = \mathbf{A}$ and $p(\mathbf{0}) \xrightarrow{w} q(\boldsymbol{v})$ in $\mathcal{V}$.*

Proof. **(a)** By Lemma 3, $\mathcal{V}$ has a path $w \in T^*$ such that $w(\mathbf{0}) = \boldsymbol{v}$. Let $k \stackrel{\text{def}}{=} |w|$. For every $i \in [k+1]$, let

$$\mathbf{A}_i \stackrel{\text{def}}{=} M(w_i w_{i+1} \cdots w_k)$$

with the convention that $A_{k+1} = \mathbf{I}$. For every $i \in [k]$, let

$$\boldsymbol{b}_i \stackrel{\text{def}}{=} \mathbf{A}_{i+1} \cdot \Delta(w_i),$$

$$w'_i \stackrel{\text{def}}{=} ((\mathrm{src}(w_i), \mathbf{A}_i), \mathbf{I}, \boldsymbol{b}_i, (\mathrm{tgt}(w_i), \mathbf{A}_{i+1})).$$

We claim that $w' \stackrel{\text{def}}{=} w'_1 w'_2 \cdots w'_k$ is such that $(p, \mathbf{A}_1) \xrightarrow{w'} (q, \mathbf{A}_{k+1})$ in $\mathcal{V}'$. Note that the validity of the claim completes the proof since $\mathbf{A}_1 = M(w)$ and $\mathbf{A}_{k+1} = \mathbf{I}$.

It follows immediately from the definition of T' that $w'_i \in T'$ for every $i \in [k]$ and hence that w' is a path from $(p, \mathbf{A}_1)$ to $(q, \mathbf{A}_k)$. By Lemma 3, it remains to show that $w'(\mathbf{0}) = \boldsymbol{v}$:

$$\begin{aligned}
w'(\mathbf{0}) &= \sum_{i=1}^{k} M(w'_{i+1} w'_{i+2} \cdots w'_k) \cdot \Delta(w'_i) && \text{(by Lemma 4 applied to } w'(\mathbf{0})) \\
&= \sum_{i=1}^{k} \Delta(w'_i) && \text{(by } M(w'_i) = \mathbf{I} \text{ for every } i \in [k]) \\
&= \sum_{i=1}^{k} \mathbf{A}_{i+1} \cdot \Delta(w_i) && \text{(by definition of } \Delta(w'_i)) \\
&= \sum_{i=1}^{k} M(w_{i+1} w_{i+2} \cdots w_k) \cdot \Delta(w_i) && \text{(by definition of } \mathbf{A}_{i+1}) \\
&= w(\mathbf{0}) && \text{(by Lemma 4 applied to } w(\mathbf{0})).
\end{aligned}$$

(b) Similarly, by Lemma 3, there exists a path w' of $\mathcal{V}'$ such that $w'(\mathbf{0}) = \boldsymbol{v}$, and it suffices to exhibit a path $w \in T^*$ from p to q in $\mathcal{V}$ such that $w(\mathbf{0}) = \boldsymbol{v}$ and $M(w) = \mathbf{A}$. Let $k \stackrel{\text{def}}{=} |w'|$. For every $i \in [k]$, let $w'_i = ((p_i, \mathbf{A}_i), \mathbf{I}, \boldsymbol{b}_i, (q_i, \mathbf{B}_i))$. By definition of T', for every $i \in [k]$, there exists a (possibly non unique) transition $t_i \in T$ such that $\mathrm{src}(t) = p_i$, $\mathrm{tgt}(t) = q_i$, $\boldsymbol{b}_i = \mathbf{B}_i \cdot \Delta(t_i)$ and $\mathbf{B}_i \cdot M(t_i) = \mathbf{A}_i$. We set $w \stackrel{\text{def}}{=} t_1 t_2 \cdots t_k$. It is readily seen that w is a path from p to q. To prove $w(\mathbf{0}) = \boldsymbol{v}$ and $M(w) = \mathbf{A}$, Lemma 4 can be applied as in the previous implication. ◀

3.3 Reachability from an arbitrary configuration

We now construct the $\mathbb{Z}$-VASS $\mathcal{V}'' = (2d, Q'', T'')$ of Theorem 1 which is obtained mostly from $\mathcal{V}'$. The states of $\mathcal{V}''$ are defined as

$$Q'' \stackrel{\text{def}}{=} Q_i \cup (Q \times \mathcal{M} \times \mathcal{M}) \cup (Q \times \mathcal{M}) \cup Q_f$$

where $Q_i = \{q_i : q \in Q\}$ and $Q_f = \{q_f : q \in Q\}$. To simplify the notation, given two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}^d$ we write $(\boldsymbol{u}, \boldsymbol{v})$ for the vector of $\mathbb{Z}^{2d}$ equal to $\boldsymbol{u}$ on the first d components and equal to $\boldsymbol{v}$ on the last d components. The set T'' consists of five disjoint subsets of transitions $T_{\text{init}} \cup T_{\text{simul}} \cup T_{\text{end}} \cup T_{\text{mult}} \cup T_{\text{final}}$ working in five sequential stages. Intuitively, these transitions allow $\mathcal{V}''$ to guess a matrix $\mathbf{A}_{\text{guess}}$, to simulate a path w of $\mathcal{V}$ such that $\mathbf{A}_{\text{guess}} = M(w)$, to compute $w(\mathbf{0})$ and finally to compute $w(\mathbf{0}) + \mathbf{A}_{\text{guess}} \cdot \boldsymbol{u}$.

The first set of transitions is defined as:

$$T_{\text{init}} \stackrel{\text{def}}{=} \{(q_i, \mathbf{I}, (\mathbf{0}, \mathbf{0}), (q, \mathbf{C}, \mathbf{C})) : q \in Q, \mathbf{C} \in \mathcal{M}\}.$$

Its purpose is to move from Q_i to $Q \times \mathcal{M} \times \mathcal{M}$, thereby storing two copies of the guessed matrix $\mathbf{A}_{\text{guess}}$. The second set is defined as:

$$T_{\text{simul}} \stackrel{\text{def}}{=} \{((p, \mathbf{A}, \mathbf{C}), \mathbf{I}, (\mathbf{0}, \boldsymbol{b}), (q, \mathbf{B}, \mathbf{C})) : \mathbf{C} \in \mathcal{M}, ((p, \mathbf{A}), \mathbf{I}, \boldsymbol{b}, (q, \mathbf{B})) \in T'\}.$$

Its purpose is to simulate T' in the two first components of $Q \times \mathcal{M} \times \mathcal{M}$ and to remember $\mathbf{A}_{\text{guess}}$ in the third component. The third set is defined as:

$$T_{\text{end}} \stackrel{\text{def}}{=} \{((q, \mathbf{I}, \mathbf{C}), \mathbf{I}, (\mathbf{0}, \mathbf{0}), (q, \mathbf{C})) : (q, \mathbf{I}, \mathbf{C}) \in Q''\},$$

and its purpose is to move from $Q \times \mathcal{M} \times \mathcal{M}$ to $Q \times \mathcal{M}$, thus guessing the end of a run in $\mathcal{V}'$, i.e. by reaching $\mathbf{I}$. The fourth set is defined as:

$$\begin{aligned} T_{\text{mult}} \stackrel{\text{def}}{=} &\{((q, \mathbf{C}), \mathbf{I}, (-\boldsymbol{e_i}, \mathbf{C} \cdot \boldsymbol{e_i}), (q, \mathbf{C})) : q \in Q, \mathbf{C} \in \mathcal{M}, i \in [d]\} \cup \\ &\{((q, \mathbf{C}), \mathbf{I}, (\boldsymbol{e_i}, -\mathbf{C} \cdot \boldsymbol{e_i}), (q, \mathbf{C})) : q \in Q, \mathbf{C} \in \mathcal{M}, i \in [d]\}, \end{aligned}$$

where $\boldsymbol{e_i}$ is the unit vector such that $\boldsymbol{e_i}(i) = 1$. The purpose of T_{mult} is to compute $\mathbf{A}_{\text{guess}} \cdot \boldsymbol{u}$. Finally, T_{final} is defined as:

$$T_{\text{final}} \stackrel{\text{def}}{=} \{((q, \mathbf{C}), \mathbf{I}, (\mathbf{0}, \mathbf{0}), q_f) : q \in Q, \mathbf{C} \in \mathcal{M}\},$$

and its purpose is to move from $Q \times \mathcal{M}$ to Q_f, guessing the end of the matrix multiplication performed with T_{mult}.

We may now prove Theorem 1:

Proof of Theorem 1. First, note that we obtain

$$\begin{aligned}|Q''| &= (2 + \|\mathcal{M}\| + \|\mathcal{M}\|^2) \cdot |Q| \\ &\leq 4 \cdot \|\mathcal{M}\|^2 \cdot |Q|,\end{aligned}$$

$$\begin{aligned}\|T''\| &= 2 \cdot \|\mathcal{M}\| \cdot |Q| + \|\mathcal{M}\| \cdot \|T'\| + |Q''| + 2d \cdot \|\mathcal{M}\| \cdot |Q| \\ &\leq \|\mathcal{M}\|^4 \cdot \|T\| + 8d \cdot \|\mathcal{M}\|^2 \cdot |Q|,\end{aligned}$$

where we use the fact that $\|T'\| \leq \|\mathcal{M}\|^2 \cdot \|T\| \cdot \max\{\|\mathbf{A}\| : \mathbf{A} \in \mathcal{M}\} \leq \|\mathcal{M}\|^3 \cdot \|T\|$.

It remains to show that $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ in $\mathcal{V}$ if and only if $p_i(\boldsymbol{u}, \mathbf{0}) \xrightarrow{*} q_f(\mathbf{0}, \boldsymbol{v})$ in $\mathcal{V}''$.

$\Rightarrow$) By Lemma 3, there exists a path w of $\mathcal{V}$ such that $w(\boldsymbol{u}) = \boldsymbol{v}$. By definition of T_{init}, T_{simul} and T_{end}, and by Proposition 5, it is the case that $p_i(\boldsymbol{u}, \mathbf{0}) \xrightarrow{*} r(\boldsymbol{u}, w(\mathbf{0}))$ where $r = (q, M(w))$. The transitions of T_{mult} allow to transform $(\boldsymbol{u}, w(\mathbf{0}))$ into $(\mathbf{0}, w(\mathbf{0}) + M(w) \cdot \boldsymbol{u})$. Thus, using T_{final}, we can reach the configuration $q_f(w(\mathbf{0}) + M(w) \cdot \boldsymbol{u})$. This concludes the proof since $w(\boldsymbol{u}) = w(\mathbf{0}) + M(w) \cdot \boldsymbol{u}$ by Lemma 4.

$\Leftarrow$) The converse implication follows the same steps as the previous one. It suffices to observe that the first part of a run of $\mathcal{V}''$ defines the value $w(\mathbf{0})$, while the second part of the run defines $M(w) \cdot \boldsymbol{u}$. ◀

4 Semilinearity of affine $\mathbb{Z}$-VASS

We say that a subset of $\mathbb{Z}^d$ is *semilinear* if it is definable by a Presburger formula [34], *i.e.* by a formula of $\mathsf{FO}(\mathbb{Z}, +, <)$, the first-order logic over $\mathbb{Z}$ with addition and order. Semilinear sets capture precisely finite unions of sets of the form $\boldsymbol{b} + \mathbb{N} \cdot \boldsymbol{p_1} + \mathbb{N} \cdot \boldsymbol{p_2} + \ldots + \mathbb{N} \cdot \boldsymbol{p_k}$, and are closed under basic operations such as finite sums, intersection and complement. Semilinear sets are important in formal verification, in particular because satisfiability of Presburger formulas is decidable [34] and in NP for the existential fragment [11].

The results of Section 3 allow us to show that any affine $\mathbb{Z}$-VASS with the finite-monoid property has a semilinear reachability relation:

▶ **Theorem 6.** *Given an afmp-$\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$ and $p, q \in Q$, it is possible to compute an existential Presburger formula $\varphi_{\mathcal{V},p,q}$ of size at most $O(\text{poly}(|\mathcal{V}|, \|\mathcal{M}_\mathcal{V}\|))$ such that $\varphi_{\mathcal{V},p,q}(\boldsymbol{u}, \boldsymbol{v})$ holds if and only if $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ in $\mathcal{V}$.*

Proof. By Theorem 1, there exist an effectively computable $\mathbb{Z}$-VASS $\mathcal{V}' = (d', Q', T')$ and $p', q' \in Q'$ such that $d' = 2 \cdot d$, $|Q'| \leq 4 \cdot \|\mathcal{M}\|^2 \cdot |Q|$, $\|T'\| \leq 8d \cdot \|\mathcal{M}\|^2 \cdot |Q| + \|\mathcal{M}\|^4 \cdot \|T\|$ and

$$p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v}) \text{ in } \mathcal{V} \text{ if and only if } p'(\boldsymbol{u}, \mathbf{0}) \xrightarrow{*} q'(\mathbf{0}, \boldsymbol{v}) \text{ in } \mathcal{V}'. \tag{5}$$

By [21, Sect. 3], we can compute an existential Presburger formula ψ of linear size in $\mathcal{V}'$ such that $\psi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y}, \boldsymbol{y}')$ holds if and only if $p'(\boldsymbol{x}, \boldsymbol{x}') \xrightarrow{*} q'(\boldsymbol{y}, \boldsymbol{y}')$ in $\mathcal{V}'$. By (5), the formula $\varphi_{\mathcal{V},p,q}(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \psi(\boldsymbol{x}, \mathbf{0}, \mathbf{0}, \boldsymbol{y})$ satisfies the theorem. ◀

It was observed in [20, 9] that the reachability relation of a $\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$, such that $|Q| = |M(\mathcal{V})| = 1$, is semilinear if and only if $\mathcal{M}_\mathcal{V}$ is finite. Theorem 6 shows that if we do not bound the number of states and matrices, *i.e.* drop the assumption $|Q| = |M(\mathcal{V})| = 1$, then the left implication remains true. It is natural to ask whether the right implication also remains true.

Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be the affine $\mathbb{Z}$-VASS illustrated in Figure 3 from left to right respectively. Note that $\mathcal{M}_{\mathcal{V}_1}$ and $\mathcal{M}_{\mathcal{V}_2}$ are both infinite due to the matrix made only of 1s. Moreover, the reachability relations of $\mathcal{V}_1$ and $\mathcal{V}_2$ are semilinear since the former can reach any target

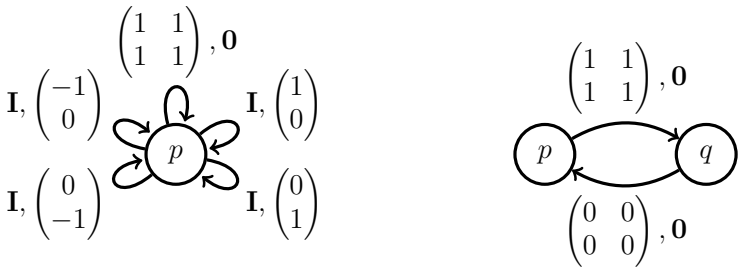


Figure 3 Examples of affine $\mathbb{Z}$-VASS with infinite monoids and semilinear reachability relations.

configuration from any initial configuration, and since the latter can only generate finitely many vectors due to the zero matrix. Since $\mathcal{V}_1$ has a single control-state, $|M(\mathcal{V}_1)| = |M(\mathcal{V}_2)| = 2$ and $\Delta(\mathcal{V}_2) = \{\mathbf{0}\}$, any simple natural extension of the characterization of semilinearity in terms of the number of control-states, matrices and vectors fails.

It is worth noting that an affine $\mathbb{Z}$-VASS with an infinite monoid may have a non semilinear reachability relation. Indeed, Figure 2 depicts a transfer + copy $\mathbb{Z}$-VASS with an infinite monoid and such that $\{\boldsymbol{v} : p(1,1) \xrightarrow{*} q(\boldsymbol{v})\} = \{(2^n, 2^n) : n \in \mathbb{N}\}$, which is known to be non semilinear.

5 Complexity of reachability

In this section, we use the results of Section 3 to show that reachability belongs to PSPACE for a large class of afmp-$\mathbb{Z}$-VASS encompassing all variants of Section 2. Moreover, we give a novel proof to the known NP membership of reachability for reset $\mathbb{Z}$-VASS.

▶ **Theorem 7.** *Let $\mathcal{C} = \bigcup_{d\geq 1} \mathcal{C}_d$ be a class of matrices such that $\mathcal{C}_d$ is finite for every $d \geq 1$. If there exists a polynomial* poly *such $\|\mathcal{C}_d\| \leq 2^{\mathrm{poly}(d)}$ for every $d \geq 1$, then $\mathsf{Reach}_{\mathcal{C}} \in$ PSPACE.*

▶ **Corollary 8.** *The reachability problem of reset, permutation, transfer, copy and copyless $\mathbb{Z}$-VASS is in PSPACE.*

Proof of Theorem 7. Let $\mathcal{V} = (d, Q, T)$ be an affine $\mathbb{Z}$-VASS from class $\mathcal{C}$. Let $\mathcal{V}' = (d, Q', T')$ be the $\mathbb{Z}$-VASS obtained from $\mathcal{V}$ in Theorem 1. Recall that, by Theorem 1, $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ in $\mathcal{V}$ if and only if $p'(\boldsymbol{u}, \mathbf{0}) \xrightarrow{*} q'(\mathbf{0}, \boldsymbol{v})$ in $\mathcal{V}'$. Therefore, it suffices to check the latter for determining reachability in $\mathcal{V}$.

We invoke a result of [7] on the flattability of $\mathbb{Z}$-VASS. By [7, Prop. 3], $p'(\boldsymbol{x}) \xrightarrow{*} q'(\boldsymbol{y})$ in $\mathcal{V}'$ if and only if there exist $k \leq |T'|$, $\alpha_0, \beta_1, \alpha_1, \ldots, \beta_k, \alpha_k \in (T')^*$ and $\boldsymbol{e} \in \mathbb{N}^k$ such that

- **(i)** $p'(\boldsymbol{x}) \xrightarrow{\alpha_0 \beta_1^{\boldsymbol{e}(1)} \alpha_1 \cdots \beta_k^{\boldsymbol{e}(k)}} q'(\boldsymbol{y})$ in $\mathcal{V}'$,
- **(ii)** β_i is a cycle for every $i \in [k]$, and
- **(iii)** $\alpha_0\beta_1\alpha_1 \cdots \beta_k\alpha_k$ is a path from p' to q' of length at most $2 \cdot |Q'| \cdot |T'|$.

For every $w \in (T')^*$, let $\Delta(w) \stackrel{\text{def}}{=} \sum_{i=1}^{|w|} \Delta(w_i)$. By Lemma 4 (see the remark below the proof of Lemma 4), we have $w(\boldsymbol{u}) = \boldsymbol{u} + \Delta(w)$ for every $\boldsymbol{u} \in \mathbb{Z}^d$. Thus, by Lemma 3, checking (i), assuming (iii), amounts to testing whether $\boldsymbol{e}$ is a solution of the following system of linear Diophantine equations:

$$\boldsymbol{x} + \sum_{i=0}^{k} \Delta(\alpha_i) + \begin{pmatrix}\Delta(\beta_1) & \Delta(\beta_2) & \cdots & \Delta(\beta_k)\end{pmatrix} \cdot \boldsymbol{e} = \boldsymbol{y}. \tag{6}$$

Let $m \stackrel{\text{def}}{=} 2 \cdot |Q'| \cdot |T'|$. Since $|T'| \leq \|T'\|$ and by Theorem 1, we have $m \leq 128 \cdot d \cdot |\mathcal{M}_\mathcal{V}|^5 \cdot |Q|^2 \cdot \|T\|$, and hence by $M(\mathcal{V}) \subseteq \mathcal{C}_d$ and by assumption on $\mathcal{C}_d$, $m \leq 128 \cdot d \cdot (2^{\text{poly}(d)})^5 \cdot |Q|^2 \cdot \|T\|$.

We describe a polynomial-space non deterministic Turing machine $\mathcal{A}$ for testing whether $p'(\boldsymbol{x}) \stackrel{*}{\to} q'(\boldsymbol{y})$ in $\mathcal{V}'$. The proof follows from NPSPACE = PSPACE. Machine $\mathcal{A}$ guesses $k \leq |T'|$, a path $\pi = \alpha_0\beta_1\alpha_1 \cdots \beta_k\alpha_k$ of length at most m from p' to q', and $\boldsymbol{e} \in \mathbb{N}^k$, and tests whether (6) holds for π. Note that we are not given $\mathcal{V}'$, but $\mathcal{V}$, so we must be careful for the machine to work in polynomial space.

Instead of fully constructing $\mathcal{V}'$ and fully guessing π, we do both on the fly, and also construct $\Delta(\alpha_0), \Delta(\beta_1), \ldots, \Delta(\beta_k), \Delta(\alpha_k)$ on the fly as partial sums as we guess π. Note that to ensure that each β_i is a cycle, we do not need to fully store β_i but only its starting control-state. Moreover, note that $\|\Delta(\alpha_i)\|, \|\Delta(\beta_i)\| \leq m \cdot \max\{\|\Delta(t)\| : t \in T\}$ for every i, and hence each α_i and β_i has a binary representation of polynomial size in $|\mathcal{V}|$.

By [14, Prop. 4], (6) has a solution if and only if it has a solution $\boldsymbol{e} \in \mathbb{N}^k$ such that

$$\|\boldsymbol{e}\| \leq \left((k+1) \cdot \max\{\|\Delta(\beta_i)\| : i \in [k]\} + \|\boldsymbol{x}\| + \|\boldsymbol{y}\| + \sum_{i=0}^{k} \|\Delta(\alpha_i)\| + 1 \right)^{d'}.$$

Since $d' = 2 \cdot d$, this means that we can guess a vector $\boldsymbol{e} \in \mathbb{N}^k$ whose binary representation is of polynomial size, and that we can thus evaluate (6) in polynomial time. ◀

▶ **Theorem 9** ([21]). *The reachability problem for reset $\mathbb{Z}$-VASS belongs to NP.*

Proof. Let $\mathcal{V} = (d, Q, T)$ be a reset $\mathbb{Z}$-VASS. The proof does not follow immediately from Theorem 1 because $\mathcal{M}_\mathcal{V}$ can be of size up to 2^d. We will analyze the construction used in the proof of Theorem 1, where reachability in $\mathcal{V}$ is effectively reduced to reachability in a $\mathbb{Z}$-VASS $\mathcal{V}' = (d', Q', T')$. Recall that $Q' = Q_i \cup (Q \times \mathcal{M}_\mathcal{V}) \cup (Q \times \mathcal{M}_\mathcal{V} \times \mathcal{M}_\mathcal{V}) \cup Q_f$, and thus that the size of $\mathcal{V}'$ depends only on the sizes of Q and $\mathcal{M}_\mathcal{V}$.

It follows from the proof of Theorem 1 and Proposition 5 that for every run $p_i(\boldsymbol{u}, \mathbf{0}) \stackrel{*}{\to} q_f(\mathbf{0}, \boldsymbol{v})$ in $\mathcal{V}'$, there is a corresponding run $p(\boldsymbol{u}) \stackrel{w}{\to} q(\boldsymbol{v})$ in $\mathcal{V}$ for some $w \in T^*$ of length $k \geq 0$. Moreover, the only states of the form $(Q, \mathbf{A}, \mathbf{B})$ or $(Q, \mathbf{A})$ occurring along the run contain matrices $\mathbf{A}, \mathbf{B} \in \mathcal{M}_\mathcal{V}$ of the form $\mathbf{A}_i = M(w_i w_{i+1} \cdots w_k)$ for $i \in [k+1]$. Recall that by definition, for every $i \in [k]$, $\mathbf{A}_i = \mathbf{A}_{i+1} \cdot \mathbf{B}$ for some $\mathbf{B} \in \mathcal{M}_\mathcal{V}$. Since $\mathcal{M}_\mathcal{V}$ consists of reset matrices, it holds that $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ is monotonic, *i.e.* if $\mathbf{A}_i$ has a 1 somewhere on its diagonal, then $\mathbf{A}_{i+1}$ also contains 1 in that position. It follows that $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k+1}$ is made of at most $d+1$ matrices.

To prove the NP upper bound we proceed as follows. We guess at most $d+1$ matrices of $\mathcal{M}_\mathcal{V}$ that could appear in sequence $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k+1}$. We construct the $\mathbb{Z}$-VASS $\mathcal{V}'$ as in Theorem 1, but we discard each control-state of Q' containing a matrix not drawn from the guessed matrices. Since the constructed $\mathbb{Z}$-VASS is of polynomial size, reachability can be verified in NP [21]. ◀

▶ Remark. Observe that the proof of Theorem 9 holds for any class of affine $\mathbb{Z}$-VASS with a finite monoid such that every path of its Cayley graph contains at most polynomially many different vertices. For a reset $\mathbb{Z}$-VASS of dimension d, the number of vertices on every path of the Cayley graph is bounded by $d+1$.

6 Hardness results for reachability

It is known that the reachability problem for $\mathbb{Z}$-VASS is already NP-hard [21], which means that reachability is NP-hard for all classes of affine $\mathbb{Z}$-VASS. In this section, we show that PSPACE-hardness holds for some classes, matching the PSPACE upper bound derived in Section 5. Moreover, we observe that reachability is undecidable for transfer + copy $\mathbb{Z}$-VASS.

▶ **Theorem 10.** *The reachability problem for permutation + reset $\mathbb{Z}$-VASS is PSPACE-hard.*

Proof. We give a reduction from the membership problem of linear bounded automata, which is known to be PSPACE-complete (see, *e.g.*, [23, Sect. 9.3 and 13]). Let $\mathcal{A} = (P, \Sigma, \Gamma, \delta, q^{\text{ini}}, q^{\text{acc}}, q^{\text{rej}})$ be a linear bounded automaton, where:

- P is the set of states,
- $\Sigma \subseteq \Gamma$ is the input alphabet,
- Γ is the tape alphabet,
- δ is the transition function, and
- $q^{\text{ini}}, q^{\text{acc}}, q^{\text{rej}}$ are the initial, accepting and rejecting states respectively.

The transition function is a mapping $\delta : P \times \Gamma \to P \times \Gamma \times \{\textsc{Left}, \textsc{Right}\}$. The intended meaning of a transition $\delta(p, a) = (q, b, D)$ is that whenever $\mathcal{A}$ is in state p and holds letter a at the current position of its tape, then $\mathcal{A}$ overwrites a with b and moves to state q and to the next tape position in direction D.

Let us fix the word that we will check for membership $w \in \Sigma^n$ (so $|w| = n$). We construct an affine $\mathbb{Z}$-VASS $\mathcal{V} = (d, Q, T)$ and configurations $r(\boldsymbol{u})$ and $r'(\boldsymbol{v})$ such that $\mathcal{A}$ accepts w if and only if $r(\boldsymbol{u}) \xrightarrow{*} r'(\boldsymbol{v})$.

We set $d \stackrel{\text{def}}{=} n \cdot |\Gamma| + 1$ and associate a counter to each position of w and each letter of the tape alphabet Γ, plus one additional counter. For readability, we denote these counters respectively as $x_{i,a}$ and y, where $i \in [n]$ and $a \in \Gamma$. The idea is to maintain, for every $i \in [n]$, a single "token" among counters $\{x_{i,a} : a \in \Gamma\}$ in order to represent the current letter in the i^{th} tape cell of $\mathcal{A}$. The initial vector is $\boldsymbol{u} \in \{0,1\}^d$ such that $\boldsymbol{u}(y) = 0$ and $\boldsymbol{u}(x_{i,a}) = 1$ if and only if $w_i = a$ for every $i \in [n]$ and $a \in \Gamma$.

The control-states of $\mathcal{V}$ are defined as:

$$Q \stackrel{\text{def}}{=} \{r_{p,i} : p \in P, i \in [n]\} \cup \{r_{p,i,a} : p \in P, i \in [n], a \in \Gamma\} \cup \{r_{\text{acc}}\}.$$

The purpose of states of the form $r_{p,i}$ is to store the current state p and tape cell i of $\mathcal{A}$. States of the form $r_{p,i,a}$ are intermediary control-states and the state r_{acc} will be the target control-state.

We associate transitions to every triple $(p, a, i) \in P \times \Gamma \times [n]$, which denotes a configuration of $\mathcal{A}$: the automaton is in state p in position i, where letter a is stored. Let us fix a transition $\delta(p, a) = (q, b, D)$; and let $j = i + 1$ if $D = \textsc{Right}$, and $j = i - 1$ if $D = \textsc{Left}$. For every $i \in [n]$, if $j \in [n]$ then we add to T the transitions

$$(r_{p,i}, \mathbf{A}, \mathbf{0}, r_{p,i,a}) \text{ and } (r_{p,i,a}, \mathbf{B}, \mathbf{0}, r_{q,j}), \tag{7}$$

where $\mathbf{A}$ is a permutation matrix that swaps the values of $x_{i,a}$ and $x_{i,b}$; and $\mathbf{B}$ resets $x_{i,\sigma}$ for every $\sigma \in \Gamma \setminus \{b\}$. The two transitions are depicted on the left of Figure 4 (for $D = \mathsf{Right}$). The purpose of the first transition is to simulate the transition of $\mathcal{A}$, upon reading a in tape cell i and state p, by moving the i^{th} "token" from $x_{i,a}$ to $x_{i,b}$. Note that this transition may be faulty, *i.e.* it can simulate reading letter a even though tape cell i contains another letter. The purpose of the second transition is to detect such faulty behaviour: if the first

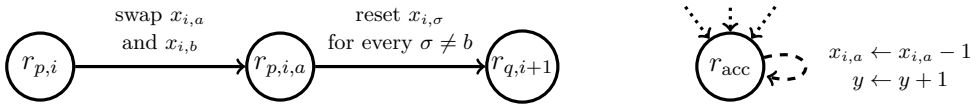


Figure 4 *Left*: transitions of $\mathcal{V}$ simulating transition $\delta(p, a) = (q, b, \mathsf{Right})$ of $\mathcal{A}$. *Right*: transitions to verify whether the accepting state has been reached with no error during the simulation.

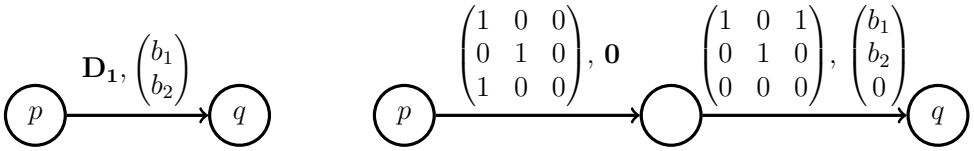


Figure 5 Gadget (on the right) made of copy and transfer transitions simulating the doubling transition on the left.

transition is taken and tape cell i does not contain a, then due to the resets, all counters of $\{x_{i,a} : a \in \Gamma\}$ end up in 0, and the i^{th} "token" is lost.

Recall that in the initial vector $\boldsymbol{u} \in \{0,1\}^d$ there were exactly n counters with 1 and $\sum_{i\in[d]} \boldsymbol{u}(i) = n$. By construction of $\mathcal{V}$, all configurations reachable from $r_{q^{\text{ini}},1}(\boldsymbol{u})$, using transitions defined in (7), have vectors in $\{0,1\}^d$ with at most n counters equal to 1. They have exactly n counters equal to 1 only if all corresponding transitions were valid for the automaton $\mathcal{A}$. We conclude that $\mathcal{A}$ accepts w if and only if there exist $i \in [n]$ and $\boldsymbol{u}' \in \{0,1\}^d$ such that $r_{q^{\text{ini}},1}(\boldsymbol{u}) \xrightarrow{*} r_{q^{\text{acc}},i}(\boldsymbol{u}')$ and $\sum_{i\in[d]} \boldsymbol{u}'(i) = n$.

To test whether such index i and vector $\boldsymbol{v}$ exist, we add some transitions to T as illustrated on the right of Figure 4. For every $i \in [n]$, we add to T the transition $(r_{q^{\text{acc}},i}, \mathbf{I}, \mathbf{0}, r_{\text{acc}})$. For every $i \in [n]$ and $a \in \Gamma$, we add to T the transition $(r_{\text{acc}}, \mathbf{I}, \boldsymbol{b}, r_{\text{acc}})$ where $\boldsymbol{b}$ is the vector whose only non zero components are $\boldsymbol{b}(x_{i,a}) = -1$ and $\boldsymbol{b}(y) = 1$. The purpose of these transitions is to (weakly) transfer the values of all counters to y. Recall that $\boldsymbol{v}$ is the vector whose only non zero component is $\boldsymbol{v}(y) = n$. We conclude that the language of $\mathcal{A}$ accepts w if and only if $r_{q^{\text{ini}},1}(\boldsymbol{u}) \xrightarrow{*} r_{\text{acc}}(\boldsymbol{v})$. ◀

▶ **Corollary 11.** *The reachability problem is PSPACE-complete for permutation + reset $\mathbb{Z}$-VASS, transfer $\mathbb{Z}$-VASS and copy $\mathbb{Z}$-VASS.*

Proof. The hardness for permutation + reset $\mathbb{Z}$-VASS follows from Theorem 10, and the upper bound for transfer $\mathbb{Z}$-VASS and copy $\mathbb{Z}$-VASS follows from Theorem 7. It remains to argue that transfers and copies can both simulate permutations and resets. By definition, permutation matrices are also transfer and copy matrices. Resetting a counter x can be simulated by adding an extra counter y. In the case of transfers, it suffices to transfer x to y and to allow for y to be arbitrarily incremented or decremented. In the case of copies, it suffices to keep $y = 0$ at all times and to copy y onto x. ◀

▶ **Proposition 12** ([36]). *The reachability problem for transfer + copy $\mathbb{Z}$-VASS is undecidable, even when restricted to three counters.*

Proof. Reichert [36] gives a reduction from the Post correspondence problem over the alphabet $\{0,1\}$ to reachability in affine $\mathbb{Z}$-VASS with two counters. The trick of the reduction is to represent two binary sequences as the natural numbers the sequences encode, one in each counter. If we add an artificial 1 at the beginning of the two binary sequences, then these sequences are uniquely determined by their numerical values. We only need to be

able to double the counter values, which corresponds to shifting the sequences. This can be achieved using the following matrices:

$$\mathbf{D_1} \stackrel{\text{def}}{=} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } \mathbf{D_2} \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.$$

The only matrices used in the construction of Reichert are $\mathbf{I}$, $\mathbf{D_1}$ and $\mathbf{D_2}$. The two last matrices can be simulated by a gadget made of copy and transfer matrices and by introducing a third counter. This gadget is depicted in Figure 5 for the case of matrix $\mathbf{D_1}$. The other gadget is symmetric. Note that if a run enters control-state p of the gadget with vector $(x, y, 0)$, then it leaves control-state q in vector $(2x + b_1, y + b_2, 0)$ as required. ◀

▶ Remark. A monoid $\mathcal{M}$ is *positive* if it contains only matrices with non negative entries. The classes of Section 2 and the matrices used in Proposition 12 have this property. The coverability problem for affine VASS with positive (and possibly infinite) monoids is known to be decidable in Ackermann time [19]. Recall that coverability and reachability are inter-reducible for affine $\mathbb{Z}$-VASS. Thus, Proposition 12 gives an example of a decision problem, namely coverability, which is more difficult for affine $\mathbb{Z}$-VASS than for affine VASS.

7 Conclusion

We have shown that the reachability problem for afmp-$\mathbb{Z}$-VASS reduces to the reachability problem for $\mathbb{Z}$-VASS, *i.e.* every afmp-$\mathbb{Z}$-VASS $\mathcal{V}$ can be simulated by a $\mathbb{Z}$-VASS of size polynomial in $|\mathcal{V}|$ and $\|\mathcal{M}_\mathcal{V}\|$. In particular, this allowed us to establish that the reachability relation of any afmp-$\mathbb{Z}$-VASS is semilinear.

For all of the variants we studied – reset, permutation, transfer, copy and copyless $\mathbb{Z}$-VASS – the size of $\|\mathcal{M}_\mathcal{V}\|$ is of exponential size, thus yielding a PSPACE upper bound on their reachability problems. We do not know whether an exponential bound on $\|\mathcal{M}_\mathcal{V}\|$ holds for any class of afmp-$\mathbb{Z}$-VASS. We are aware that the work of [31] provides an exponential tower upper bound. Moreover, an exponential upper bound holds when $\mathcal{M}_\mathcal{V}$ is generated by a single matrix [24]; and when $\mathcal{M}_\mathcal{V}$ is a group then we have an exponential bound but only on $|\mathcal{M}_\mathcal{V}|$ (see [28] for an exposition on the group case).

For all the classes of afmp-$\mathbb{Z}$-VASS studied in this paper, we have shown that the reachability problem is either PSPACE-complete or NP-complete, with the exception of permutation $\mathbb{Z}$-VASS reachability which lies between NP and PSPACE, and whose precise complexity remains open.

Another interesting open question is whether reachability is undecidable for every class of infinite matrix monoids, *i.e.* is the top rectangular region of Figure 1 equal to the red ellipse?

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 313–321, 1996. doi:10.1109/LICS.1996.561359.

2 Parosh Aziz Abdulla and Giorgio Delzanno. Parameterized verification. *International Journal on Software Tools for Technology Transfer*, 18(5):469–473, 2016. doi:10.1007/s10009-016-0424-3.

3 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Proc. Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 9:1–9:10, 2014. doi:10.1145/2603088.2603151.

4 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Proc.* 40^{th} *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 37–48, 2013. doi:10.1007/978-3-642-39212-2_7.

5 Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, 1976. doi:10.1016/0304-3975(76)90067-0.

6 Konstantinos Athanasiou, Peizun Liu, and Thomas Wahl. Unbounded-thread program verification using thread-state equations. In *Proc.* 8^{th} *International Joint Conference on Automated Reasoning (IJCAR)*, pages 516–531, 2016. doi:10.1007/978-3-319-40229-1_35.

7 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *Proc.* 30^{th} *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 32–43, 2015. doi:10.1109/LICS.2015.14.

8 Michael Blondin and Christoph Haase. Logics for continuous reachability in Petri nets and vector addition systems with states. In *Proc.* 32^{nd} *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005068.

9 Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, Belgium, 1998.

10 Rémi Bonnet. *Theory of Well-Structured Transition Systems and Extended Vector-Addition Systems*. PhD thesis, École normale supérieure de Cachan, France, 2013.

11 I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976. doi:10.2307/2041711.

12 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(8):1691–1710, 2012. doi:10.1142/S0129054112400709.

13 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *International Journal of Foundations of Computer Science*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.

14 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *Proc.* 43^{rd} *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 128:1–128:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.128.

15 Giorgio Delzanno. A unified view of parameterized verification of abstract models of broadcast communication. *International Journal on Software Tools for Technology Transfer*, 18(5):475–493, 2016. doi:10.1007/s10009-016-0412-7.

16 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *Proc.* 25^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 103–115, 1998. doi:10.1007/BFb0055044.

17 E. Allen Emerson and Kedar S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc.* 13^{th} *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998. doi:10.1109/LICS.1998.705644.

18 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. An SMT-based approach to coverability analysis. In *Proc.* 26^{th} *International Conference on Computer Aided Verification (CAV)*, pages 603–619, 2014. doi:10.1007/978-3-319-08867-9_40.

19 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Proc.* 26^{th} *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 269–278, 2011. doi:10.1109/LICS.2011.39.

20 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.

21 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *Proc. 8th International Workshop on Reachability Problems (RP)*, pages 112–124, 2014. doi:10.1007/978-3-319-11439-2_9.

22 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

23 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

24 Radu Iosif and Arnaud Sangnier. How hard is it to verify flat affine counter systems with the finite monoid property? In *Proc. 14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 89–105, 2016. doi:10.1007/978-3-319-46520-3_6.

25 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Transactions on Programming Languages and Systems*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

26 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

27 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 267–281, 1982. doi:10.1145/800070.802201.

28 James Kuzmanovich and Andrey Pavlichenkov. Finite groups of matrices whose entries are integers. *The American Mathematical Monthly*, 109(2):173–186, 2002. doi:10.2307/2695329.

29 Jérôme Leroux. Vector addition systems reachability problem (a simpler solution). In *The Alan Turing Centenary Conference*, pages 214–228, 2012.

30 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 63, Department of Computer Science, Yale University, 1976.

31 Arnaldo Mandel and Imre Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, 1977. doi:10.1016/0304-3975(77)90001-9.

32 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984. doi:10.1137/0213029.

33 Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

34 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus du I^{er} Congrès des mathématiciens des pays slaves*, pages 192–201, 1929.

35 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

36 Julien Reichert. *Reachability games with counters: decidability and algorithms*. PhD thesis, École normale supérieure de Cachan, France, 2015.

37 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008. doi:10.1016/j.entcs.2008.12.042.

38 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. 35th International Symposium Mathematical Foundations of Computer Science (MFCS)*, pages 616–628, 2010. doi:10.1007/978-3-642-15155-2_54.

Verifying Quantitative Temporal Properties of Procedural Programs

Mohamed Faouzi Atig
Uppsala University, Sweden

Ahmed Bouajjani[1]
IRIF, Paris Diderot University, France

K. Narayan Kumar[2]
Chennai Mathematical Institute and UMI RELAX, India

Prakash Saivasan
TU Braunschweig, Germany

Abstract

We address the problem of specifying and verifying quantitative properties of procedural programs. These properties typically involve constraints on the relative cumulated costs of executing various tasks (by invoking for instance some particular procedures) within the scope of the execution of some particular procedure. An example of such properties is "within the execution of each invocation of procedure P, the time spent in executing invocations of procedure Q is less than 20% of the total execution time". We introduce specification formalisms, both automata-based and logic-based, for expressing such properties, and we study the links between these formalisms and their application in model-checking. On one side, we define Constrained Pushdown Systems (CPDS), an extension of pushdown systems with constraints, expressed in Presburger arithmetics, on the numbers of occurrences of each symbol in the alphabet within invocation intervals (sub-computations between matching pushes and pops), and on the other side, we introduce a higher level specification language that is a quantitative extension of CaRet (the Call-Return temporal logic) called QCaRet where nested quantitative constraints over procedure invocation intervals are expressible using Presburger arithmetics. Then, we investigate (1) the decidability of the reachability and repeated reachability problems for CPDS, and (2) the effective reduction of the model-checking problem of procedural programs (modeled as visibly pushdown systems) against QCaRet formulas to these problems on CPDS.

2012 ACM Subject Classification Theory of computation → Logic and verification, Software and its engineering → Model checking, Software and its engineering → Software verification

Keywords and phrases Verification, Formal Methods, Pushdown systems, Visibly pushdown, Quantitative Temporal Properties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.15

1 Introduction

Reasoning about performances requires checking properties on the cumulated costs of actions along program computations. Different types of costs can be considered corresponding to consumption of resources such as time, memory, energy, etc. To be able to reason about the action costs, amounts to the ability to count numbers of occurrences of different actions in

[1] Partially supported by Indo-French project AVeCSo
[2] Partially supported by Indo-French project AVeCSo, Infosys Foundation, DST-VR Project P-02/2014

computations (since weights can be associated to actions representing their various costs). Therefore, it is important to develop formal program models and specification languages (1) that allow the expression of counting constraints in different computation segments, and (2) that are useful for algorithmic verification of programs against quantitative properties involving these counting constraints. The goal of this paper is to propose such formalisms, both automata and logic based, for reasoning about the behaviors of procedural programs, i.e., sequential programs with (potentially recursive) procedure calls.

Quantitative properties of procedural programs are typically temporal properties including cost constraints on execution intervals corresponding to procedure invocations. An example of such a property is the invariant "within the execution of every terminating call to procedure P, the cumulated cost of executing all the calls by P to the procedure Q is less that 20% of the total cost of executing P". Then, formalisms for expressing such properties must have mechanisms allowing to express counting constraints in the scope of computation intervals between procedure calls and returns.

In the framework of automata-based formalisms, it is well know that pushdown systems (PDS) are natural models for procedural programs. Our first contribution is to introduce *Constrained Pushdown Systems* (CPDS), an extension of PDS by counting constraints on execution intervals between two matching push and pop operations (i.e., the push of an element to the stack, and the corresponding pop of that element from the stack). The counting constraints, expressed in Presburger arithmetics, concern the numbers of occurrences of the input alphabet symbols in the computation segment between these two matching operations. In order to impose these constraints, we consider an extended stack alphabet, where, in addition to plain stack symbols, push operations can push to the stack a pair of stack symbol γ and a counting constraint f. When a pair (γ, f) is popped from the stack, the automaton checks the satisfaction of the constraint f by the word read since it was pushed to the stack.

In the framework of logic-based formalisms, the temporal logic CaRet [6] has been introduced as a suitable specification formalisms for procedural programs. The second contribution is to introduce *Quantitative CaRet* (QCaRet), and extension of CaRet by counting constraints over procedure call-return intervals. Counting constraints within a call-return interval concern the cumulated lengths of the *outer-most* call-return intervals of each procedure. So, basically, QCaRet is the extension of CaRet with an operator W_f parametrized by a Presburger formula f. A QCaRet formula $W_f(\varphi)$ is satisfied at a point of a computation if that point corresponds to the call of a procedure, say P, and if both f and the QCaRet subformula φ are satisfied in the call-return interval of the procedure P. Notice that this allows nesting of temporal properties with counting constraints.

Then, we investigate the decision problems for these two formalisms. First, we prove that the reachability problem of CPDS is undecidable in general. However, we prove that under the assumption that the number of constraints in the stack is bounded – we call *constraint height-bounded CPDS* the class of CPDS corresponding to this assumption, the reachability problem becomes decidable and the same holds for the repeated reachability problem. Constraint height-bounded CPDS is a powerful class of automata allowing to express interesting non context-free languages. Interestingly, this class allows also to prove that QCaRet is decidable. Indeed, we show that the satisfiability problem of QCaRet can be reduced to solving repeated reachability in CPDS. The same reduction allows to show that the model-checking problem of procedural program, modeled as visibly PDS, against QCaRet formulas is decidable. A crucial point that leads to the decidability of the satisfiability and model-checking problems for QCaRet is the fact that counting constraints are about outermost calls of procedures in a call-return interval. This is necessary for the reduction

to constraint height-bounded CPDS decision problems. Another important and nontrivial contribution is that the complexity is shown to be elementary. Indeed, a more direct way of doing the reduction would use nested computations of Parikh images that would lead to a tower of exponentials depending on the size of the formula.

Related work. Extensions of word automata with counting constraints (such as Parikh automata and CQDDs) have been studied in the literature [21, 20, 12, 13, 19, 11]. These works cannot be used for reasoning about nested words (except [19] which extends visibly pushdown automata with reversal bounded counters). There are also works allowing to reason about unbounded-width trees using counting constraints (e.g., [22, 21, 23]), however, these constraints concerning the immediate successors of a node in a tree, cannot encode the type of constraints imposed by CPDS on nested words (that would correspond to global constraints on a whole subtree). CaRet is the first logic that was tailored to the specification of procedural programs. Extensions and variants of this logic have been proposed [7, 5], but none of them allow reasoning about quantitative properties. Extensions of temporal logics with counting constraints have been studied, e.g., in [21, 22], but again they are interpreted on words or on trees but without allowing to express (nested) constraints on nested words.

Several extensions of pushdown systems with either data or time have been studied in the literature (see e.g., [3, 2, 16, 4, 9, 14, 17, 1, 10, 15]). However, all these works are orthogonal to ours since they do not allow counting constraints on execution intervals between two matching push and pop operations.

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* and Σ^+ to denote the set of all finite words and non-empty finite words respectively over Σ; and use ϵ to denote the empty word. We also write Σ_ϵ for $\Sigma \cup \{\epsilon\}$. A language is a (possibly infinite) set of words. We let $|w|$ denote the length of the word w. Let $w = a_1 a_2 \dots a_n$. We write $w(i)$ for a_i and $w[i..j]$ for $w(i) \cdots w(j)$. For $w \in \Sigma^*$, $\Gamma \subseteq \Sigma$, we use $w{\downarrow_\Gamma} \in \Gamma^*$ for the projection of w on the Γ. We will also consider infinite words and languages of infinite words over Σ.

The set of linear constraints over a set V (written $\mathcal{C}(V)$) is the set of expressions of the form $c_1 x_1 + c_2 x_2 \dots c_k x_k \ \# \ 0$, where $x_i \in V$, $\# \in \{<, >, =\}$ and $c_i \in \mathbb{Z}$. The size of such a constraint φ, written $|\varphi|$, is sum of k and the number of bits needed to describe the sequence $c_1, c_2, ...c_k$. That is, we assume that the values c_i are provided in binary notation. A valuation v is a map that assigns a value from $\mathbb{N}$ to each element of V. We write $v \models \varphi$ to mean that φ is satisfied by the valuation v (and whose meaning is evident). We shall write $\mathcal{BC}(V)$ to denote formulas over $\mathcal{C}(V)$ constructed using $\wedge$ and $\vee$. The satisfaction relation $\models$ is extended to $\mathcal{BC}(V)$ in the obvious manner.

Given a word w over an alphabet Σ, we write $\pi(w)$ to denote the *Parikh map* defined by $\pi(w)(a)$ is the number of occurrences of a in w for all $a \in \Sigma$. Let L be a language over $\Sigma \uplus \Sigma'$ and let σ be a function from Σ' to languages over Σ. We write $\sigma(L)$ for the language $\{x_1 y_1 x_2 y_2 \dots y_k x_{k+1} \mid \forall i.\ x_i \in \Sigma^*, \exists a_1, \dots, a_k \in \Sigma'.\ x_1 a_1 x_2 a_2 \dots a_k x_{k+1} \in L, \forall i.\ y_i \in \sigma(a_i)\}$.

3 Constrained Pushdown Systems

A *constrained pushdown system (CPDS) A* is a tuple $(Q, \Gamma, \Sigma, \delta, s)$ where Q is the set of states, Γ is the stack alphabet, Σ is the tape alphabet, $s \in Q$ is the initial state and δ is the transition relation. We use $\perp \notin \Gamma$ to denote the stack bottom symbol. Let $\Gamma_C = \Gamma \times \mathcal{BC}(\Sigma)$

and $\Gamma_e = \Gamma \cup \Gamma_C$. The transition set δ is a subset of $Q \times \mathcal{SO} \times \Sigma \times Q$ where $\mathcal{SO}$ is the set of stack operations given by $\{push(X), pop(X), Y?, int \mid X \in \Gamma_e, Y \in \Gamma \cup \{\bot\}\}$. The operation $push(X)$ pushes X, with $X \in \Gamma_e$, on to the stack. The operation $pop(X)$ removes such an X from the stack. The $Y?$ operation checks if the current top of stack is either Y or in $\{Y\} \times \mathcal{BC}(\Sigma)$. Finally, the operation int is an internal action (i.e., independent of the stack). Thus, CPDS are PDS enriched with the ability to add constraints to stack symbols.

A configuration of the CPDS A is a pair (q, γ) with $q \in Q$ and $\gamma \in \Gamma_e^* \bot$. The *initial configuration* is the pair $(s, \bot)$. The transition relation $\xrightarrow{\tau}_A$, with $\tau \in \delta$, on the set of configurations is defined as follows: (1) if $\tau = (q, a, int, q')$ then $(q, \gamma) \xrightarrow{\tau}_A (q', \gamma)$ (Internal move), (2) if $\tau = (q, a, push(X), q')$ then $(q, \gamma) \xrightarrow{\tau}_A (q', X\gamma)$ (Push), (3) if $\tau = (q, a, pop(X), q')$ then $(q, X\gamma) \xrightarrow{\tau}_A (q', \gamma)$ (Pop), and (4) if $\tau = (q, a, Y?, q')$ then $(q, X\gamma) \xrightarrow{\tau}_A (q', X\gamma)$ if $X = Y$ or $X \in \{Y\} \times \mathcal{BC}(\Sigma)$ (Test). We often write $\rightarrow$ for $\rightarrow_A$ when A is clear from the context.

The transition relation extends naturally to sequences of transitions: $(q, \gamma) \xrightarrow{\epsilon} (q, \gamma)$ and $(q, \gamma) \xrightarrow{\sigma.\tau} (q', \gamma')$ if there is (q'', γ'') such that $(q, \gamma) \xrightarrow{\sigma} (q'', \gamma'')$ and $(q'', \gamma'') \xrightarrow{\tau} (q', \gamma')$. We call this an unconstrained run on the sequence of transitions σ. Given two unconstrained runs $\rho_1 = (q, \gamma) \xrightarrow{\sigma_1} (q', \gamma')$ and $\rho_2 = (q', \gamma') \xrightarrow{\sigma_2} (q'', \gamma'')$, the concatenation $\rho_1\rho_2$ denotes the unique run $(q, \gamma) \xrightarrow{\sigma_1.\sigma_2} (q'', \gamma'')$.

Let $\rho := (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ be an unconstrained run. We define a binary relation $\curvearrowright$ on positions in ρ as follows: $i \curvearrowright j$ then the ith transition is a push and the symbol pushed in this transition is popped by the jth transition. Formally $i \curvearrowright j$ if $0 < i < j \leq n$ and further $\tau_i = (q_{i-1}, a_i, push(X), q_i)$, $\tau_j = (q_{j-1}, a_j, pop(X), q_j)$, for each $i \leq k < j$, $\gamma_i = X\gamma_{i-1}$ is a suffix of γ_k and $\gamma_j = \gamma_{i-1}$.

Clearly, if $i \curvearrowright j$ and $i' \curvearrowright j$ then $i = i'$ and if $i \curvearrowright j$ and $i \curvearrowright j'$ then $j = j'$. We note that that if γ_0 is a suffix of γ_k for each $0 \leq k \leq n$ then for any pop transition, say j, there is a unique i such that $i \curvearrowright j$. In particular, this is true whenever $\gamma_0 = \bot$.

We are now in a position to define the *constrained runs* (or simply *runs*) of the CPDS A. An unconstrained run $\rho = (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{k-1}, \gamma_{k-1}) \xrightarrow{\tau_k} (q_k, \gamma_k) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ with $\tau_k = (q_{k-1}, a_k, op_k, q_k)$ is a constrained run if for every transition of the form $\tau_j = (q_{j-1}, a_j, pop(Y, \varphi), q_j)$ and $i \curvearrowright j$, we have $\pi(a_i \ldots a_j) \models \varphi$. Pushing a stack symbol of the form (Y, φ) enforces the requirement that the sequence of letters read from this transition upto the transition that pops this symbol from the stack satisfies the constraint φ. However, observe that constraints that are pushed on to the stack but not popped along the run do not place any requirements. In what follows, we shall write *run* to mean *constrained run*.

We also write $(q, \gamma) \xrightarrow{a,op} (q', \gamma')$ if there is a transition $\tau = (q, a, op, q')$ and $(q, \gamma) \xrightarrow{\tau} (q', \gamma')$ as this simplifies notation at many places. We write $(q, \gamma) \xrightarrow{w} (q', \gamma')$ if either $w = \epsilon$, $q = q'$ and $\gamma = \gamma'$ or $w = a_1 a_2 \ldots a_k$, $k \geq 1$, with $a_i \in \Sigma_\epsilon$ for $1 \leq i \leq k$, and further we can find configurations (q_i, γ_i) and operations op_i, $0 \leq i < k$, such that $(q, \gamma) = (q_0, \gamma_0) \xrightarrow{a_1, op_1} (q_1, \gamma_1) \cdots (q_{k-1}, \gamma_{k-1}) \xrightarrow{a_k, op_k} (q_k, \gamma_k) = (q', \gamma')$ is a run. We write $(q, \gamma) \xrightarrow{*} (q', \gamma')$ to mean that there is some w with $(q, \gamma) \xrightarrow{w} (q', \gamma')$.

Our aim is to study the *reachability problem* for CPDS. That is, given a CPDS A and a state $q \in Q$, determine whether there is a run $(s, \bot) \xrightarrow{*} (q, \gamma)$. We will also consider the *repeated reachability problem*, where the aim is to determine if there is an infinite run $(s, \bot) \xrightarrow{*} (q, \gamma_1) \xrightarrow{*} (q, \gamma_2) \ldots$ that visits the state q infinitely often.

We may equip a CPDS $A = (Q, \Gamma, \Sigma, \delta, s)$ with a set of accepting states $F \subseteq Q$ to obtain a constrained pushdown automaton (CPDA) $A' = (Q, \Gamma, \Sigma, \delta, s, F)$. The language, $\mathcal{L}(A)$, accepted by A' is defined naturally as $\{w \in \Sigma^* \mid (s, \bot) \xrightarrow{w} (q, \gamma), q \in F\}$. Clearly, the language emptiness problem for CPDAs is equivalent to the reachability problem for CPDS.

The CPDA model is quite expressive as indicated by the following examples.

- $L_1 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$. Not a CFL, but recognized by Parikh Automata [20, 12].
- $L_2 = \{w\#w^R \mid w \in \{a,b\}^*, |\pi(w)(a) = \pi(w)(b)\}$. Not a CFL, not recognized by Parikh Automata, recognized by Parikh Pushdown Automata [20, 21]
- L_1^*, L_2^*. Unlikely to be recognizable by Parikh Pushdown Automata.
- $L^0 = \{w \in \{a_0, b_0\}^* \mid \pi(w)(a_0) = \pi(w)(b_0)\}$
  $L^{i+1} = \{w \in (a_{i+1}L^i + b_{i+1}L^i)^* \mid \pi(w)(a_{i+1}) = \pi(w)(b_{i+1})\}$.
  The automaton for L^i stores upto $i+1$ constraints at any point in the stack. Automata for L_1, L_2, L_1^* and L_2^* store at most one.

4 Visibly Pushdown Systems

Visibly Pushdown Systems (VPDS) [7] are natural formal model of procedural programs.

Formally, a VPDS is a PDS whose alphabet $\Sigma = \Sigma_\downarrow \cup \Sigma_\uparrow \cup \Sigma_L$ and any transition on a letter from $\Sigma_\downarrow$ must push a value on the stack, any transition on $\Sigma_\uparrow$ must pop a value from the stack and any transition on a letter from Σ_L must be an internal move or a test. Transitions on ϵ must also be internal or test moves and hence leave the stack unchanged. VPDSs have been extensively studied in literature and have several advantages over PDSs [7]. They enjoy a host of other algorithmic and language theoretic properties: the class of languages definable in the model is effectively closed under boolean operations, emptiness and universality are decidable.

We shall work with a specific variety of visible alphabets which makes explicit the set of procedures involved. Let Δ be a set of letters and Π be a set of *procedures*. The visible alphabet $\Sigma(\Delta, \Pi)$ is given by $\Sigma_\downarrow = \Delta \times \{call(P) \ : \ P \in \Pi\}$, $\Sigma_\uparrow = \Delta \times \{ret(P) \ : \ P \in \Pi\}$ and $\Sigma_L = \Delta \times \{int\}$. The words over such an alphabet that constitute behaviours of VPDSs have a particular form and we describe that now. A word σ over $\Sigma_\downarrow \cup \Sigma_\uparrow \cup \Sigma_L$ is well-nested if (1) for each prefix σ' of σ, $|\sigma'{\downarrow}\Sigma_\downarrow| \geq |\sigma'{\downarrow}\Sigma_\uparrow|$, and (2) if $\sigma(i) = (c, call(P))$, $\sigma(j) = (c', ret(P'))$ with $|\sigma[i..j]{\downarrow}\Sigma_\downarrow| = |\sigma[i..j]{\downarrow}\Sigma_\uparrow|$ then $P = P'$. In addition if σ is finite and has the same number of letters from $\Sigma_\downarrow$ and $\Sigma_\uparrow$ then we say it is a *complete well-nested word*.

We shall overload the symbol $\curvearrowright$ and write $i \curvearrowright j$ for positions i, j in a well-nested word σ if $\sigma(i) = (c, call(P))$, $\sigma(j) = (c', ret(P))$ with $|\sigma[i..j]{\downarrow}\Sigma_\downarrow| = |\sigma[i..j]{\downarrow}\Sigma_\uparrow|$. It captures the call-return relationship. We shall also write $i \curvearrowright_P j$ to explicitly indicate the associated procedure. Clearly if $i \curvearrowright j$ and $i' \curvearrowright j'$ then either the intervals $[i, j]$ and $[i', j']$ are completely disjoint or one is contained in the other. If $i \curvearrowright_P j$ and $[i, j]$ is not contained in any interval $[i', j']$ with $i' \curvearrowright_P j'$ then we say that $[i, j]$ is an *outermost* call to P in σ.

For any well-nested word σ, we define the map $\pi_{Pr}(\sigma)$ from $\Pi \cup \{\bot\}$ to $\mathbb{N}$ as follows: (1) $\pi_{Pr}(\sigma)(\bot) = |\sigma|$, (2) $\pi_{Pr}(\sigma)(P) = \sum\{j - i + 1 \mid [i, j]$ is an outermost call of P in $\sigma\}$. The function $\pi_{Pr}(\sigma)(P)$ computes the total length of all the outermost calls to the procedure P while $\pi_{Pr}(\sigma)(\bot)$ reports the length of σ. Notice that any word read by a VPDS along a run will be a well-nested word.

5 A Quantitative Extension of CaReT

We introduce in this section an extension of CaRet [6] which permits us to reason about quantitative properties of VPDSs using constraint formulas.

Let AP be a set of atomic propositions, and let Π be a finite set of procedure names. Then, we let $Prop = 2^{AP} \cup \{call(P), ret(P) \ : \ P \in \Pi\}$. We use $p, p_1, p_2, \ldots$ to refer to

elements of $Prop$ and $P, P', \ldots$ to refer to procedures in Π. We use $f, f_1, f_2, \ldots$ to refer to constraint formulas in $\mathcal{BC}(\Pi \cup \{\bot\})$. Formulas of *QCaRet* are given by the following syntax.

$$\varphi \ ::= \ p \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \bigcirc^g \varphi \,|\, \bigcirc^a \varphi \,|\, \bigcirc^c \varphi \,|\, \varphi \mathcal{U}^g \varphi \,|\, \varphi \mathcal{U}^a \varphi \,|\, \varphi \mathcal{U}^c \varphi \,|\, W_f(\varphi)$$

The logic CaReT is the sub-logic without the W_f operator. As with CaReT, the formulas are interpreted over well-nested words, both finite and infinite, over a visible alphabet, in this case $\Sigma(AP, \Pi)$ where $\Sigma_\downarrow = 2^{AP} \times \{call(P) \ : \ P \in \Pi\}$, $\Sigma_\uparrow = 2^{AP} \times \{ret(P) \ : \ P \in \Pi\}$ and $\Sigma_L = 2^{AP} \times \{int\}$. For any well-nested word σ and position i, we define three different notions of successors as follows: (1) $suc_g(i)$ is $i+1$ if $|\sigma| > i$ and $\bot$ (to denote undefined) otherwise. (2) $suc_a(i) = suc_g(i)$ if $\sigma(i) \notin \Sigma_\downarrow$, $suc_a(i) = j$ if $\sigma(i) \in \Sigma_\downarrow$ and $i \curvearrowright j$, and $suc_a(i) = \bot$ otherwise. (3) $suc_c(i) = j$ if j is the largest number less than i for which there is a k with $j \curvearrowright k$ and $i \leq k$. $suc_c(i) = \bot$ if no such j exists. With all this we can define the semantics of the formulas w.r.t any well-nested word σ and any position i in σ:

$$\begin{array}{rll}
(\sigma, i) \models p & \textbf{iff} & \sigma(i) = (c, C),\ p \in c \cup \{C\} \\
(\sigma, i) \models \neg\varphi & \textbf{iff} & (\sigma, i) \not\models \varphi \\
(\sigma, i) \models \varphi_1 \vee \varphi_2 & \textbf{iff} & (\sigma, i) \models \varphi_1 \textbf{ or } (\sigma, i) \models \varphi_2 \\
(\sigma, i) \models \bigcirc^x \varphi & \textbf{iff} & suc_x(i) \neq \bot \textbf{ and } (\sigma, suc_x(i)) \models \varphi \text{ for } x \in \{a, c, g\} \\
(\sigma, i) \models \varphi_1 \mathcal{U}^x \varphi_2 & \textbf{iff} & \exists n.\, \exists i_0, i_1, \cdots, i_n.\, i_0 = i \textbf{ and} \\
 & & \forall k.\, 0 \leq k < n \textbf{ implies } \big(suc_x(i_k) = i_{k+1} \textbf{ and } (\sigma, i_k) \models \varphi_1\big) \\
 & & \textbf{and } (\sigma, i_n) \models \varphi_2 \\
(\sigma, i) \models W_f(\varphi) & \textbf{iff} & \exists j.\, i \curvearrowright j \textbf{ and } [i, j] \text{ is an outermost call in } \sigma \textbf{ and} \\
 & & \pi_{Pr}(\sigma[i+1, j-1]) \models f \textbf{ and } (\sigma[i+1, j-1], 1) \models \varphi
\end{array}$$

We say that $\sigma \models \varphi$ if $(\sigma, 1) \models \varphi$. We define the finite and infinite word languages defined by φ: $\mathcal{L}(\varphi) = \{\sigma \mid \sigma \models \varphi, |\sigma| < \infty\}$ and $\mathcal{L}^\omega(\varphi) = \{\sigma \mid \sigma \models \varphi, |\sigma| = \infty\}$.

In addition to properties expressible in CaRet, QCaRet allows to express (nested) quantitative constraints. Below are few examples of such QCaRet formulas (here $\Diamond^x(\Psi) = \text{True } \mathcal{U}^x \ \Psi$ and $\Box^x(\Psi) = \neg\Diamond^x(\neg\Psi)$):

- *For every outermost invocation of P, the time spent in executing outermost invocations to Q is less than 20% of the total execution time.*

  $$\Box^g(call(P) \wedge W_{\text{True}} \Rightarrow W_{(5Q \leq \bot)}\text{True})$$

- *For every outermost procedure execution interval where the cumulated time of executing Q is lower than half of the total execution time, the execution time of Q is less than the cumulated execution time of P in that same procedure interval execution, and there must be one invocation to Q in that interval that takes more that 5 time units*

  $$\Box^g\Big(W_{2Q \leq \bot}\text{True} \Rightarrow \big(W_{Q \leq P}(\Diamond^g(call(Q) \wedge W_{\bot > 5}\text{True}))\big)\Big)$$

For the logic CaReT obtained by omitting the W_f operator, it is known from [6, 5] that these languages are languages of Visibly Pushdown Automata (VPA) and Büchi Visibly Pushdown Automata (BVPA) respectively.

We investigate in the next sections the translation of QCaRet to (a visible version of) CPDSs. In [5], "qualitative" extensions of CaRet have been defined. We can extend them to quantitative versions in the same way as we did above by adding the operator W_f. The approach we will present in the rest of the paper can be applied in the same way to these extensions, and the results concerning decidability of the satisfiability and model checking problems and their complexity can be obtained for them in a similar way.

6 Reachability/Emptiness for CPDAs

In this section we shall examine the reachability problem for CPDSs (equivalently, the language emptiness problem for CPDAs). While the general problem is undecidable, we identify an interesting decidable under-approximation which provides an important tool in proving the decidability of QCaReT.

6.1 Undecidability of Reachability

In this section, we show that the reachability problem for CPDSs is undecidable, infact it is possible to simulate the runs of a 2 Counter Machine (2CM) using a CPDA over the alphabet $\Sigma = \{inc_1, inc_2, dec_1, dec_2, z_1, z_2\}$. Σ also also serves as its stack alphabet. The simulation proceeds in two phases. In the first, the CPDA guesses the sequence of transitions used in an accepting run of the 2CM, in reverse order, and pushes corresponding counter operations on the stack. While doing so, it also conjoins constraints with the decrements and test for zeros as follows: to simulating a decrement transition by counter i it pushes $(dec_i, (inc_i >= dec_i))$ and to simulate a test for zero on counter i it pushes $(z_i, (inc_i = dec_i))$ where $i \in \{1, 2\}$. This entire phase is executed without reading any input. For the guessed sequence of transitions (in reverse) to constitute an accepting run, we need to verify that the counters remained positive throughout the run and that all zero tests were successful. This is done in the second phase, where it repeatedly pops its stack and reads the same letter from the input tape till it reaches the empty stack and accepts if it does. The second phase processes operations of guessed accepting run in the correct order, and it is easy to see that the constraints inserted into the stack ensure that every zero test was indeed successful and none of the decrements resulted in a negative value for the counter, thus verifying the validity of the guessed run. This gives us the following theorem.

▶ **Theorem 1.** *The language emptiness problem for CPDAs (reachability problem for CPDS) is undecidable.*

6.2 Technical Preliminaries

Before we proceed to the under-approximation we introduce some notations and prove an useful technical lemma. Let $A = (Q, \Gamma, \Sigma, \delta, s)$ be an CPDS. We say that a run $(q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ is a *weak X-run*, $X \in \Gamma_e \cup \{\bot\}$, if there is a γ such that $\gamma_0 = X\gamma$ and for each $0 \leq i \leq n$, $\gamma_i = \gamma_i' X\gamma$, that is, $X\gamma$ is a suffix of the stack contents of each configuration. In this case, for any γ', the following run $(q_0, X\gamma') \xrightarrow{\tau_1} (q_1, \gamma_1' X\gamma') \ldots, (q_{i-1}, \gamma_{i-1}' X\gamma') \xrightarrow{\tau_i} (q_i, \gamma_i' X\gamma') \ldots \xrightarrow{\tau_n} (q_n, \gamma_n' X\gamma')$ is also a weak X-run, where $\gamma_i' X\gamma = \gamma_i$, $1 \leq i \leq n$. Thus, we may say there is a weak X run from (q_0, X) to (q_n, X) to mean that there is such a run, without being specific about γ. We call such a run a *weak X-run from q_0 to q_n*. Further, if $\gamma_n = X\gamma = \gamma_0$ we say call it an X-run.

We let $\mathcal{L}^X_{q,q'}(A)$, $q, q' \in Q$, be the set of words w such that there is a weak X-run from q to q' on w. A CPDA for $\mathcal{L}^X_{q,q'}(A)$ can be constructed easily from A. The desired automaton $A^X_{q,q'}$ is $(Q, \Sigma, \Gamma, \delta^X_{q,q'}, q, \{q'\})$ where $\delta^X_{q,q'} = \delta \setminus \{(r, c, O, r') \mid O = (?\bot)\} \cup \{(r, c, (?\bot), r') \mid (r, c, (?X), r') \in \delta\}$. It treats $\bot$ as the symbol X for tests, never pops this "X" and never succeeds on a test for $\bot$. The size of the new automaton is linear in the size of A.

Another language of particular interest is the following: Suppose $X = (Y, \varphi)$ and further that $\tau = (p, a, push(X), q)$ and $\tau' = (p', b, pop(X), q')$ are transitions involving the push and pop of the same symbol. Then, we let $\mathcal{L}^X_{\tau,\tau'}(A)$ be $\{w = a.y.b \mid$ there is an X-run on y from q to $p'\}$. This identifies the languages of words recognized by runs consisting of τ, followed by

an X-run from q to p', followed by τ'. Please note that we use X-runs and not weak X-runs here. Finally, observe that this definition does not require that φ is satisfied by $\pi(ayb)$ (while the constraints along the run on y are enforced). This language is accepted by the CPDA $A^X_{\tau,\tau'} = (Q \cup \{s_a, t_b\}, \Sigma, \Gamma, \delta^X_{\tau,\tau'}, s_a, \{t_b\})$ where $\delta^X_{\tau,\tau'} = \{(s_a, a, int, q), (q', b, \bot?, t_b)\} \cup \delta \setminus \{(r, c, O, r') \mid O = (?\bot)\} \cup \{(r, c, (?\bot), r') \mid (r, c, (?X), r') \in \delta\}$. The size of this automaton is linear in the size of A. To summarize,

▶ **Lemma 2.** *Let $A =$ be a CPDS and let $X \in \Gamma_e$. Then for any $q, q' \in Q$, the language $\mathcal{L}^X_{q,q'}(A)$ is recognized by a CPDA $A^X_{q,q'}$ whose size is linear in A. Further, for any $X = (Y, \varphi) \in \Gamma_C$ and $\tau = (p, a, push(X), q), \tau' = (p', b, pop(X), q') \in \delta$, the language $\mathcal{L}^X_{\tau,\tau'}(A)$ is recognized by a CPDA $A^X_{\tau,\tau'}$ whose size is linear in A.*

6.3 Constraint height Bounded CPDAs

The *constraint height* of a configuration (q, γ) is defined by $|\gamma{\downarrow_{\Gamma \times \mathcal{C}(\Sigma)}}|$ (i.e., the number of constraint symbols in the stack). The constraint height of a finite run ρ is the maximum of the constraint heights of the configurations visited along ρ. The constraint height of an infinite run is defined similarly, with ∞ acting as the upper bound of the set of all integers.

For any CPDS A, we say that a state q is K *constraint height reachable* (or K*-reachable* for the sake of succinctness) if there is a run $(s, \bot) \xrightarrow{*} (q, \gamma)$ whose constraint height is bounded by K. The K-reachability problem is to determine if there is such a run. Similarly, for any CPDA A, $\mathcal{L}_K(A)$ is the set of all words accepted by runs with constraint height bounded by K. Note that all the example languages listed at the end of Section 3 are constraint height bounded.

When $K = 0$ we are effectively left with the pushdown system obtained by removing all the transitions involving constraints. Thus, 0-reachability is clearly decidable. Our main technical result is that the K-reachability problem for CPDS (or equivalently, the emptiness of $\mathcal{L}_K(A)$ for CPDAs) is decidable for any $K \geq 0$. Our proof of decidability establishes a stronger property as stated in the following theorem:

▶ **Theorem 3.** *Let A be a CPDA and let $K \in \mathbb{N}$. Then, $\pi(\mathcal{L}_K(A))$ is effectively semilinear and a finite-state automaton M with the same Parikh image can be computed in* 2-EXPTIME.

The rest of this section is devoted to the proof of this theorem. As a first step, we recall the Parikh's Theorem which states that the language of a pushdown automaton A can be simulated by a Nondeterministic Finite Automaton (NFA) upto Parikh-image equivalence.

▶ **Lemma 4** ([18])**.** *Let A be a pushdown automaton. We can construct an NFA M such that $\pi(\mathcal{L}(M)) = \{\pi(w) \mid w \in \mathcal{L}(A)\}$ and the size of M is bounded by $2^{p(|A|)}$ for a polynomial p.*

We now make use of a result from [8] to extend this to CPDAs.

▶ **Lemma 5.** *Let M be an NFA over Σ and φ be a formula in $\mathcal{BC}(\Sigma)$. Then we can construct an NFA M' with size bounded by $2^{|\varphi|}.(|M|.2^{|\varphi|})^{|\Sigma|^{d.k}}$ for some constant d, such that $\pi(\mathcal{L}(M')) = \pi(\{w \mid w \in \mathcal{L}(M)\ \&\ \pi(w) \models \varphi\})$, and where k is the depth of the formula and hence bounded by $|\varphi|$.*

The next lemma lists a couple of simple results about substitutions and Parikh-images.

▶ **Lemma 6.** *Let L be a language over $\Sigma \uplus \Sigma'$ and let σ assign a language $\sigma(a)$ over Σ for each $a \in \Sigma'$.*

1. *If L' is Parikh-equivalent to L and L'_a is Parikh-equivalent to $\sigma(a)$ for each $a \in \Sigma'$ then, $\sigma'(L')$, with $\sigma'(a) = L'_a$, is Parikh equivalent to $\sigma(L)$.*

2. *If M is an NFA for L and M_a is an NFA for $\sigma(a)$, $a \in \Sigma'$, then we can obtain an NFA for $\sigma(L)$ by replacing each transition on any letter $a \in \Sigma'$ by a copy of M_a. Thus, there is an NFA for $\sigma(L)$ whose size is bounded by $|M|(Max_{a\in\Sigma'}|M_a|)$.*

We now have the technical ingredients in place to address the proof of Theorem 3. We begin by observing that, given a CPDA $A = (Q, \Sigma, \Gamma, \delta, s, F)$ and a number K we can construct an CPDA $A[K]$ such that $\mathcal{L}_K(A) = \mathcal{L}_K(A[K]) = \mathcal{L}(A[K])$. Further, $A[K]$ faithfully records information regarding the constraint height of the configuration in its control state. This automaton is defined as follows: $A[K] = (Q\times\{0, 1, \ldots, K\}, \Sigma, \Gamma, \delta^K, (s, K), F\times\{0, 1, \ldots, K\})$. The transition relation δ^K is defined as follows:

- $((q, i), a, push((Y, \varphi)), (q', i-1)) \in \delta^K$ whenever $(q, a, push((Y, \varphi)), q') \in \delta$ and $1 \leq i \leq K$
- $((q, i), a, pop((Y, \varphi)), (q', i+1)) \in \delta^K$ whenever $(q, a, pop((Y, \varphi)), q') \in \delta$ and $0 \leq i < K$
- $((q, i), a, O, (q', i)) \in \delta^K$ whenever $(q, a, O, q') \in \delta$ and O does not involve constraints.

The transition relation δ^K faithfully simulates δ, moves from copy i to copy $i-1$ while pushing a constraint and moves from copy i to copy $i+1$ on popping a constraint. The constraint height of any reachable configuration with control state (q, i) is therefore $K - i$. A state of the form $(q, 0)$ does not permit pushing any constraint symbol. Also note that, for any $0 \leq j \leq K$, δ^j is just δ^K restricted to the state space of $A[j]$ $(Q \times \{0, 1, \ldots, j\})$.

Our strategy for the proof of Theorem 3 is the following: We will argue by induction on j, $0 \leq j \leq K$ that for any symbol $X = (Y, \varphi) \in \Gamma_C$ and any pair of transitions $\tau, \tau' \in \delta^j \setminus \delta^{j-1}$ which push and pop X respectively, we can construct an NFA Parikh-equivalent to $\mathcal{L}^X_{\tau,\tau'}(A[j])$ whose size is bounded by a function $f(j)$. We shall then derive an expression bounding $f(j)$, $j \leq K$. This will form a key ingredient in the proof of Theorem 3.

We observe that if $j = 0$ then there no transitions that push (or pop) symbols Γ_C and hence nothing is to be proved. We take $f(0) = 1$ (since $\mathcal{L}^X_{\tau,\tau'}(A[0]) = \emptyset$).

We proceed inductively as follows: We write B for $A[j]^X_{\tau,\tau'}$ to simplify the notation. We construct a simple pushdown automaton P from B. This automaton simulates B on all transitions other than those that push/pop elements of Γ_C. From any state $(p, j-1)$, instead of executing a push transition of the form $\mu = ((p, j-1), c, push((Z, \psi)), (p', j-2))$ it nondeterministically guesses a corresponding pop transition $\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1))$ (which must exist along any accepting run of $B = A[j]^X_{\tau,\tau'}$ – as the run must return to level j before acceptance) and simply *outputs* (i.e. reads from the input tape) a symbol $(\mu, (Z, \psi), \mu')$ to indicate this guess and changes state to $(q', j-1)$. Thus, this automaton does not need states of the form (p, i) for $p \in Q$ and $i < j-1$ and it never leaves "level j-1" (except when executing the transitions τ andτ').

Let $\Sigma_C[j] = \{(\mu, (Z, \psi), \mu') \mid \exists p, p', q, q'.\ \mu = ((p, j-1), c, push((Z, \psi)), (p', j-2))$ and $\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1))\}$. The alphabet of P is $\Sigma \cup \Sigma_C[j]$ and its stack alphabet Γ. Let s_a and t_b be the initial and final states of B (recall the definition of $A[j]^X_{\tau,\tau'}$ from Section 3), with a the letter read by τ and b the letter read by τ'. Then, $P = (\{s_a, t_b\} \cup (Q \times \{j\}), \Sigma \cup \Sigma_C[j], \Gamma, s_a, \Delta, \{t_b\})$ and Δ is given by

$$\delta[j]^X_{\tau,\tau'} \setminus \{(p, c, O, p') \mid \exists (Z, \psi).O = push((Z, \psi)) \text{ or } O = pop((Z, \psi))\} \bigcup$$
$$\{((p, j-1), (\mu, (Z, \psi), \mu'), (q', j-1)) \mid \mu = ((p, j-1), c, push((Z, \psi)), (p', j-2)),$$
$$\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1)),\ \mu, \mu' \in \delta[j]^X_{\tau,\tau'}\}$$

Fact 1. With $\sigma((\mu, (Z, \psi), \mu')) = \mathcal{L}^{(Z,\psi)}_{\mu,\mu'}(A[j-1])$, $\forall(\mu, (Z, \psi), \mu') \in \Sigma_C[j]$, $\mathcal{L}(B) = \sigma(\mathcal{L}(P))$.

We now construct an NFA M_P Parikh-equivalent to P using Lemma 4. Then using the inductive hypothesis we construct, for each pair of transitions μ,μ' that push and pop,

respectively, the same symbol $(Z, \psi) \in \Gamma_C$, an NFA $M'_{\mu,\mu'}$ Parikh-equivalent to $\mathcal{L}^{(Z,\psi)}_{\mu,\mu'}(A[j-1])$. Then, we apply Lemma 5 to obtain an NFA $M_{\mu,\mu'}$ a language Parikh-equivalent to $\{w \in L(M'_{\mu,\mu'}) \mid \pi(w) \models \psi\}$. We let σ' be the map assigning $\mathcal{L}(M_{\mu,\mu'})$ to $(\mu, (Z, \psi), \mu')$. Then, by Lemma 6, $\sigma(\mathcal{L}(P))$ is Parikh-equivalent to $\sigma'(\mathcal{L}(M_P))$. Thus, by Fact 1, $\sigma'(\mathcal{L}(M_P))$ is Parikh-equivalent to $\mathcal{L}(B)$.

The state size of M_P is bounded by $2^{p(|B|)}$ for some polynomial p. But the state space of B is linear in the state space of A, its alphabet is polynomial in the size of A and its number of transitions also polynomial in the size of A. Thus, the size of the state space of M_P is bounded by $2^{r(|A|)}$ for some polynomial r. The number of transitions is bounded by the product of the size of the alphabet and the number of pairs of states. The number of new letters is quadratic in the number of transitions of A (in $(\mu, (Z, \psi), \mu')$, the value of (Z, ψ) is determined by μ and μ'). Thus the number of transitions is also bounded by $2^{r(|A|)}$ for some polynomial r. Equivalently it is bounded by $2^{|A|^c}$ for some fixed constant c.

The size of each automaton of the from $M'_{\mu,\mu'}$, by the induction hypthesis, is bounded by $f(j-1)$. Then, by Lemma 5, the size of $M_{\mu,\mu'}$ is bounded by $2^{|\varphi|}.(f(j-1).2^{|\varphi|})^{|\Sigma|^{d|\varphi|}}$ for some constant d. Thus, by Lemma 6, we have an NFA Parikh-equilvalent to $\mathcal{L}(B)$ whose size is bounded by $2^{|A|^c}.2^{|\varphi|}.(f(j-1).2^{|\varphi|})^{|\Sigma|^{d|\varphi|}}$. Simplifing, we get the following recurrences for $f(j)$: (1) $f(0) = 1$, and (2) $f(j) = 2^{|A|^c}.2^{|\varphi||\Sigma|^{d|\varphi|}+1}.f(j-1)^{|\Sigma|^{d|\varphi|}}$.

This gives $f(j)$ an upperbound of the form $\mathcal{O}(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)}$. Thus, we have the following Lemma.

▶ **Lemma 7.** *There is an NFA Parikh-equivalent to $\mathcal{L}^X_{\tau,\tau'}(A[j])$, $0 \le j \le K$, whose size is bounded by $\mathcal{O}(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)})$.*

Next we observe that to compute the Parikh-image of $\mathcal{L}^X_{(q,j),(q',j)}(A[j])$ for any $X \in \Gamma_e$, $q \in Q$ and $0 \le j \le K$, we may proceed as follows: Any weak X run from (q, j) to (q', j) can be broken up as, a segment involving no pushing or popping of letters from Γ_C, followed by a segment from the push of a symbol from Γ_C all the way till corresponding pop, followed by another segment involving no push or pop of letters from Γ_C, followed by one beginning with a push of a constraint and ending with the corresponding pop, and so on. (Recall that (q, j) and (q', j) are at the same level j). In particular, any symbol from Γ_C that is pushed must also be popped along such a run. We can use the same idea as in the proof of Lemma 7 and summarize the segments between push and the corresponding pop of $(Z, \psi) \in \Gamma_C$ with a letter of the form $(p, j), (\mu, (Z, \psi), \mu'), (p', j))$ and construct a simple pushdown system with no constraints. We then compute the Parikh-image of this system. Finally, we substitute these letters with the language of the corresponding NFAs computed in Lemma 7, and use Lemma 6 to obtain the desired NFA. This gives us the following Lemma.

▶ **Lemma 8.** *For any $X \in \Gamma_e$ and any pair of states $(q, j), (q', j)$ in $A[j]$, there is an NFA Parikh-equivalent to $\mathcal{L}^X_{(q,j),(q',j)}(A[j])$, whose size is bounded by $O(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)})$.*

Now, we are in a position to complete the proof of Theorem 3. Suppose an accepting run of A reaches an accepting configuration (f, γ) where the constraint-height of γ is 0. Then, the corresponding run in $A[K]$ is a weak $\perp$-run from (s, K) to (f, K). Its emptiness can be checked using Lemma 8 by checking the emptiness of a double exponential sized NFA.

If the constraint-height of γ is j with $1 \le j \le K$ then, the corresponding run in $A[K]$ is a run from the state (s, K) to the state $(f, K - j)$.

We break up this run as follows (we let $\tau_\ell = ((q_\ell, \ell), a_\ell, push((Y_\ell, \varphi_\ell)), (p_{\ell-1}, \ell-1))$):

$$((s,K),\bot) \xrightarrow{w_1} ((q_K,K),\gamma_K) \xrightarrow{\tau_K} ((p_{K-1},K-1),(Y_K,\varphi_K)\gamma_K) \xrightarrow{w_2} ((q_{K-1},K-1),\gamma_{K-1})$$
$$\xrightarrow{\tau_{K-1}} \quad \ldots ((q_{j+1},j+1),\gamma_{j+1}) \xrightarrow{\tau_j} ((p_j,j),(Y_{j+1},\varphi_{j+1})\gamma_{j+1}) \xrightarrow{w_j} ((f_j,j),\gamma_j)$$

Here, we have identified that transitions that transfer a run from a state from at k to a state at level $k-1$ for the last time along the run, for each $K \geq k \geq j+1$. The existence of such a run is equivalent to firstly the existence of transitions $((q_k,k), a_k, push((Y_k,\varphi_k)), (p_{k-1}, k-1))$ for $K \geq k > j$ and secondly, the existence of a weak $\bot$-run from (s,K) to (q_K,K), (Y_{k+1},φ_{k+1})-run from (p_k,k) to (q_k,k) for $K > k > j$, and a weak (Y_{j+1},φ_{j+1})-run from (p_j,j) to (f_j,j). Once the transition sequence in the first part is fixed (and we cycle through there at most $|A|^j$ such sequences one by one), the existence of each of the weak runs in the second part can be determined using Lemma 8. Thus, we make at the most $|A|^j.j$ calls to the emptiness of NFAs of double exponential size and this can also be done in double exponential space. This completes the proof of Theorem 3. The following theorem provides a lower bound.

▶ **Theorem 9.** *The K-reachability problem for CPDS is* PSPACE*-hard.*

We end the section with the following theorem about decidability of repeated reachability.

▶ **Theorem 10.** *Let $A = (Q, \Gamma, \Sigma, \delta, s, F)$ be a CPDA let $K \in \mathbb{N}$. The problem of deciding if A has a K constraint height bounded infinite run that visits F infinitely often is decidable in* 2-EXPTIME.

7 Visible CPDS with procedural constraints

In this section, we develop a variant of our CPDS model with a view to establish the decidability of the logic QCaReT. The model by itself is interesting in its ability to model visible behaviours of recursive programs equipped with constraints. This model is a natural extension of the VPA model to our setting.

A *procedural CPDS (or PCPDS) A* is tuple $(Q, \Delta, \Pi, \Gamma, \delta, s)$. Its input tape alphabet is the visible alphabet $\Sigma(\Delta, \Pi)$. It is very similar to a CPDS over this alphabet, except for the language of constraints it uses (and their interpretation). The set of symbols that are pushed/popped is $\Gamma_P = \Pi \times (\Gamma_{PC} \cup \Gamma)$ where $\Gamma_{PC} = \Gamma \times \mathcal{BC}(\Pi \cup \bot)$. In particular, a push transition on an input letter $(c, call(P))$ must necessarily push a letter of the form (P, Z) for some $Z \in \Gamma_{PC} \cup \Gamma$. Similarly for pop transitions. Also note that the constraints only refer to the procedure in the input (and not to elements of the tape alphabet Δ).

The notions of configurations and unconstrained runs are defined as in the case of a CPDS. The key difference is in the interpretation of the constraints and thus in the definition of constrained runs. We note that any word (or prefix of a word) read by a PCPDS is necessarily well-nested.

An unconstrained run $\rho = (q_0,\gamma_0) \xrightarrow{\tau_1} (q_1,\gamma_1) \ldots, (q_{i-1},\gamma_{i-1}) \xrightarrow{\tau_i} (q_i,\gamma_i) \ldots \xrightarrow{\tau_n} (q_n,\gamma_n)$ with $\tau_k = (q_{k-1}, (c_k, P_k), o_k, q_k)$, $1 \leq k \leq n$, is a constrained run if for every transition τ_j with $o_j = pop((P,(Y,\varphi)))$ and $i \curvearrowright j$ (so that $P_i = call(P)$ and $P_j = ret(P)$ for some $P \in \Pi$) we have $\pi_{Pr}((c_{i+1}, C_{i+1}) \ldots (c_{j-1}, C_{j-1})) \models \varphi$. Observe here that the enclosing call and return points are omitted when checking the constraints, unlike CPDAs.

The reachability problem (as well as the associated language emptiness problem) for this model are defined as usual. These problems remain undecidable in general. We define the constraint height of configurations and runs of PCPDS analogous to those for CPDS. A configuration (or control state) is K *constraint height reachable* or K*-reachable*, if it can be

reached (from the initial configuration) through a run where the constraint height is bounded by K. The main result of this section is the following:

▶ **Theorem 11.** *The K-reachability for PCPDS is decidable and is in* 2-EXPTIME.

Proof-outline. Our main idea is the following: reduce the K-reachability in a PCPDS to K-reachability in a CPDS and use Theorem 3. Let $A = (Q, \Delta, \Pi, \Gamma, \delta, s)$ be the given PCPDS. The main difficulty is that, unlike in a CPDS, a constraint φ in a PCPDS is not expressed in terms of the tape letters read along the run, but instead it is expressed in terms of the number of transitions executed inside various procedures. We plan to handle this by using a more elaborate tape alphabet.

Consider a segment of a run of A of the form

$$\rho = (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$$

where $1 \curvearrowright_{(P,(Y,\varphi))} n$. We shall focus our attention on verifying the constraint φ on this run (and ignore the verification of the other constraints pushed and popped along the run). Let $a_1 \ldots a_n$, $a_i \in \Sigma(\Delta, \Pi)$, be the word read on the tape in this run. Suppose, φ refers to $R \in \Pi$. We need to "determine" the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$. Our idea is to replace each letter a_i by an enriched version b_i (from an extended alphabet Σ') so that we may determine the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$, for each R, from $\pi(b_1 b_2 b_3 \ldots b_{n-1} b_n)$ and also replace the formula φ over Π with an equivalent formula over Σ'. Once we perform this transformation, the satisfaction of the constraint depends on the (enriched) letters read along the run and thus we have a obtained a CPDS instead of a PCPDS.

Observe that the contents of the stack at each configuration along ρ can be written as: $\gamma_i = \gamma'_i(P, (Y, \varphi))\gamma_0$, for all $1 \leq i \leq n-1$ and $\gamma_0 = \gamma_n$. The value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$, for any R, is exactly the number of transitions taken from configurations where γ'_k includes an occurrence of an element of $\{R\} \times (\Gamma \cup \Gamma_C)$.

The automaton A' that we construct will simulate A and maintain additional information in its state and stack. Using this information, it outputs, in addition to a_i, the set $S_i \subseteq \Pi$ of the set of procedure symbols that appear in γ'_{i-1}. Thus, taking $b_i = (a_i, S_i)$, the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$ is the same as

$$\sum_{a \in \Sigma(\Delta,\Pi), R \in S \subseteq \Pi} \pi(b_1 b_2 b_3 \ldots b_{n-1} b_n)(a, S)$$

Using this equivalence we transform φ into a formula over the letters of the form (a, S).

This idea can easily be generalized to handle all constraints that are pushed/popped along a run by using the fact that the run is constraint height bounded. ◀

Furthermore, we can extend this result to the repeated reachability problem as follows: The CPDS A' constructed from the PCPDS A in Theorem-11 simulates A and in doing so maintains the current state of A as part of its state in each step of the simulation. The automaton A has a K constraint height bounded run visiting q infinite often if and only if A' has a K constraint height bounded run visiting some state in which q appears, infinitely often. By Theorem 10, this is decidable. Thus we have the following theorem.

▶ **Theorem 12.** *The K constraint bounded repeated reachability problem for PCPDS is decidable and is in* 2-EXPTIME.

8 Decidability of QCaReT

In the following, we show the decidability of the model-checking of our logic QCaReT. To that aim, we will need first to recall come algorithmic properties of the logic CaReT.

▶ **Theorem 13** ([6, 5]). *For any CaReT formula φ there is a VPA A_φ and a BVPA B_φ such that $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ and $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. Further, A_φ and B_φ are only exponentially larger than φ.*

VPAs and BVPAs are closed under intersection and have decidable emptiness problem [7]. This immediately gives decision procedures for checking the satisfiability of CaReT formulas as well as for model-checking VPAs/BVPAs w.r.t. CaReT formulas. Our aim is to lift these results to QCaReT and PCPDAs. We now utilize the theory of PCPDAs developed in the previous section to provide algorithms for deciding the satisfiability of QCaReT formulas as well their model checking w.r.t. PCPDAs (and hence VPAs as well).

For any formula φ in QCaReT, we may define its *depth*, denoting the maximum nesting of the operator W_f in it, as follows: $\mathbf{d}(p) = 0$, $\mathbf{d}(\neg\varphi) = \mathbf{d}(\varphi)$, $\mathbf{d}(\varphi_1 \vee \varphi_2) = max(\mathbf{d}(\varphi_1), \mathbf{d}(\varphi_2))$, $\mathbf{d}(\bigcirc^x \varphi) = \mathbf{d}(\varphi)$, $\mathbf{d}(\varphi_1 \mathcal{U}^x \varphi_2) = max(\mathbf{d}(\varphi_1), \mathbf{d}(\varphi_2))$ and $\mathbf{d}(W_f \varphi) = 1 + \mathbf{d}(\varphi)$.

We shall construct a PCPDS A_φ with $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ as well as a Büchi PCPDS B_φ with $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. We do so by proceeding inductively on the depth of formula φ.

If $\mathbf{d}(\varphi) = 0$, then φ is in CaReT and the associated automata are given by Theorem 13. Otherwise, we first turn φ into a CaReT formula as follows: Let $\mathcal{W} = \{W_{f_1}(\psi_1), \ldots, W_{f_k}(\psi_k)\}$ be the set of outer-most W_f formulas (that is, not within the scope of another W_f operator) in φ. We obtain φ' by replacing $W_{f_i}(\psi_i)$ by a new propositional variable $p(f_i, \psi_i)$. Let $AP' = \{p(f_i, \psi_i) \mid 1 \leq i \leq k\}$. Clearly, φ' is a CaReT formula over the set of propositions $AP \cup AP'$ and the set of procedures Π.

Let $\sigma\Uparrow$, for any well-nested word σ over $\Sigma(AP, \Pi)$, be the well-nested word over $\Sigma(AP \cup AP', \Pi)$ given by $\sigma\Uparrow(i) = (P', Y)$ where $\sigma(i) = (P, Y)$, $P' = P \cup \{p(f, \psi) \in AP' \mid (\sigma, i) \models W_f(\psi)\}$. It extends the labelling, interpreting $p(f, \psi)$ as the formula $W_f(\psi)$. Similarly $\sigma'\Downarrow$, for any well-nested word σ' over $\Sigma(AP \cup AP', \Pi)$, is the well-nested word over $\Sigma(AP, \Pi)$ given by $\sigma'\Downarrow(i) = (P, Y)$ where $\sigma'(i) = (P', Y)$ with $P = P' \cap AP$. It restricts the labels to the propositions in AP. Observe that $\sigma = \sigma\Uparrow\Downarrow$. The following lemma, whose proof is omitted, is an easy consequence of our construction:

▶ **Lemma 14.** *For any well-nested word σ over $\Sigma(AP, \Pi)$ $\sigma \models \varphi$ iff $\sigma\Uparrow \models \varphi'$.*

Now, with this Lemma in place, we proceed by constructing the VPA $A_{\varphi'}$ using Theorem 13 and use this in the construction of A_φ. The automaton A_φ does the following: It simulates $A_{\varphi'}$ by guessing a set of propositions from AP' at each step and verifies that its guess at each step is correct. That is, while reading a well-nested word σ over $\Sigma(AP, \Pi)$, (i) it simulates $A_{\varphi'}$ on a word σ' with $\sigma'\Downarrow = \sigma$ (ii) it verifies that $\sigma' = \sigma\Uparrow$. This would then mean, by Lemma 14, that A_φ accepts the language $\mathcal{L}(\varphi)$. We now describe the details of how to build an automaton satisfying (i) and (ii).

Clearly, (i) can be achieved by nondeterministically guessing a set of propositions from AP' at each step. The difficulty is in ensuring (ii), that is, for each i, $1 \leq i \leq |\sigma|$, if $C'_i \subseteq AP'$ is the set of propositions guessed in the ith step verify that $\sigma, i \models W_f(\psi)$ for each $p(f, \psi) \in C'_i$ and that $\sigma, i \not\models W_f(\psi)$ for each $p(f, \psi) \in AP' \setminus C'_i$. Let us examine the conditions under which $\sigma, i \models W_f(\psi)$. This requires the following properties:

1. $\sigma(i)$ must be in $\Sigma_\downarrow$. Say $\sigma(i) = (c, call(P))$.
2. This must be an outer-most call to P in σ.

3. There is a j with $i \curvearrowright_P j$ in σ.
4. $\sigma[i+1, j-1] \models \psi$
5. $\pi_{Pr}(\sigma[i+1, j-1]) \models f$.

Truth of item 1 is determined from the letter $\sigma(i)$ and so is easy to check. For item 2, we shall add a component to the state space of $A_{\varphi'}$ that keeps track of the list of procedures from Π that are currently active. To maintain this set correctly, we expand the stack alphabet of $A_{\varphi'}$ to tag the bottom-most occurrence of each procedure. With this modification we can determine, while reading an input letter $(c, call(P))$ whether it is an outer most call to P or not. Thus, w.l.o.g. we may assume this information is available with the simulation of $A_{\varphi'}$ and hence the truth of item 2 can be determined.

This leaves us with items 3,4 and 5. The truth of these items depends not only on the letter at i (and information about outer most calls stored in the state), but on the existence of a suitable j (as required by item 3) and the word read between positions i and j (to determine items 4 and 5). The automaton guesses whether such a j exists (and then ensures that along any accepting run, the guess is indeed correct).

If it guesses that such a j does not exist (it does so only if it also guessed $C'_i = \emptyset$, as implied by the semantics of the W_f operator) then instead of pushing the symbol, say Z, pushed by $A_{\varphi'}$, it pushes a symbol $Z_\perp$. This new symbol *feels* like Z (in that we allow a test for $Z_\perp$ to succeed whenever a test for Z succeeds) but there are no transitions that pop this symbol. This guarantees that we cannot read a return corresponding to the call at position i using any transition.

If it guesses that a j does exist (and in this case, it must ensure that a return corresponding to the call at position i is encountered along any accepting run in which such a guess is made. We shall return shortly to how this can be arranged.) then, for each $p(f, \psi) \in C'_i$, we must verify that (a) $\pi_{Pr}(\sigma[i+1, j-1]) \models f$ and (b) $\sigma[i+1, j-1] \models \varphi$. The former is dealt with the power of PCPDS to impose constraints. We simply push the constraint f onto the stack and the semantics of PCPDS ensures (a). We may have to push several such f, corresponding to different formulas in C'_i, but then it suffices to push the conjunction of these constraints. For (b), the idea is to start a copy of the automaton A_ψ to read the word until the position j where the matching return is encountered. Notice that $\mathbf{d}(\psi) < \mathbf{d}(\varphi)$ and by the induction hypothesis the existence of A_ψ is guaranteed. Observe that copies are started only at positions i that correspond to outer most calls and the copies terminate when the corresponding call returns. Thus, there are at most $|\Pi|.|AP'|$ such automata under simulation at any point.

We are not done yet. If the guess is that such a j exists, we are also obliged to show that for each $p(f, \psi) \notin C'_i$, either (a') $\pi_{Pr}(\sigma[i+1, j-1]) \not\models f$ or (b') $\sigma[i+1, j-1] \not\models \psi$. Again the automaton guesses which one to verify. To verify , $\pi_{Pr}(\sigma[i+1, j-1]) \not\models f$ observe that this is equivalent to $\pi_{Pr}(\sigma[i+1, j-1]) \models \neg f$. Thus, we simply do what we did in the previous paragraph. It suffices to push $\neg f$ as a constraint and let the semantics of PCPDS take care of the verification of $\pi_{Pr}(\sigma[i+1, j-1]) \models \neg f$. If we guess that $\sigma[i+1, j-1] \not\models \psi$ then we start a copy of the automaton for $A_{\neg\psi}$ (note that $\mathbf{d}(\varphi) > \mathbf{d}(\neg\psi)$), and verify that this automaton is in an accepting state when position j is reached.

Thus, the only thing left to explain is how we validate a guess that a j with $i \curvearrowright j$ exists. The visibility restriction prevents popping of the stack at the end to verify that there are no pending returns (also such a technique will not work in the construction of B_φ to deal with infinite words). The point is that, the number of constraints on the stack at any point during the run, due to the construction described here (and not counting those due to the automata A_ψ being simulated) is bounded by Π, one per outer most call that is currently

active. Let us call this height *outer constraint height*. Thus, we can keep track of the outer constraint height of the stack as part of the control state. Then, at step i, to ensure that the call at $\sigma(i)$ returns, we simply record the outer constraint height as a *target* in the control state. Whenever the current level falls to a target level we drop that from the target set. An accepting state verifies in addition that there are no pending targets to be reached. Actually it suffices to maintain the lowest target at any point and discharge it when it is visited.

In summary, we construct a PCPDS whose state has several components: a global component that tracks the state of $A_{\varphi'}$, records information to recover outer most calls, tracks the current outer constraint height and tracks the current target for the outer constraint height. It has also has one component for each pair $P \in \Pi$ and $W_f(\psi) \in \mathcal{W}$. This component maintains the state of the automaton A_ψ if a copy of this automaton has been started at the currently active outermost call to P, or else its value is $\perp$. Such a component gets reset to $\perp$ whenever the outer most call of P returns (after verifying that it had reached the accepting state). Finally, the accepting states verify that the simulation of $A_{\varphi'}$ is accepting, no target levels are pending and that all the additional components are in the $\perp$ state.

The changes needed to handle the Büchi automata construction in the case of B_φ are minor. The simulations still use A_ψ (since the calls are obliged to terminate at some j). The only issue is with tracking visits to accepting states of $A_{\varphi'}$ while ensuring that target levels are reached. This can be ensured as follows: we do not indicate visits to accepting states of $A_{\varphi'}$ when some target is pending. We simply record it in the local state and whenever we find that all targets have been attained we flag any visits to the accepting state in the intervening run. Since the setting and unsetting of target levels happens in a well-nested manner, we are guaranteed to indicate visits to accepting configurations infinitely often as long as the run met all its obligations (i.e. contains all the necessary returns) and visits accepting states infinitely often. This gives us the following theorem.

▶ **Theorem 15.** *For any QCaReT formula φ, we can construct a PCPDS A_φ and Büchi PCPDS B_φ such that $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ and $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. The resulting automata have $\mathcal{O}((2^{|\varphi|} \times |\Pi|^2)^{(|\Pi|.|\varphi|)^{\mathcal{O}(|\varphi|)}})$ states and they are $\mathcal{O}((|\Pi|.|\varphi|)^{\mathcal{O}(|\varphi|)})$ constraint height bounded.*

Closure under intersections and emptiness checking (via reachability/repeated reachability) of PCPDSs means that we may also model check VPAs (as well as constraint height bounded PCPDAs) against QCaReT specifications.

9 Conclusion

In this work, we provide a method to specify and verify the quantitative properties of procedural programs. For this purpose, we introduced an automaton model called the constrained pushdown system (CPDS). We showed that reachability on such systems in general is undecidable. We then showed that reachability and repeated reachability are decidable in 2-EXPTIME when the number of constraints in the stack remains bounded.

We also introduced the high level specification language called the QCaReT and an extension of visibly pushdown system called the procedural CPDS (PCPDS). Finally we provided an algorithm for satisfiability and model-checking QCaReT formulas against PCPDS (and hence a VPA) by a reduction to reachability/ repeated reachability on a CPDS.

One question that is left unanswered is whether the decision procedure for decidability of reachability in CPDS is optimal. While we provide a 2-EXPTIME procedure, we only have a PSPACE lower bound. As a future work, the language theoretic properties of the constraint height bounded CPDS is an interesting topic that can be explored.

References

1. P. A. Abdulla, M. F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *LATA*, volume 7183 of *LNCS*, 2012.
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Giorgio Delzanno, and Andreas Podelski. Push-down automata with gap-order constraints. In *Fundamentals of Software Engineering - 5th International Conference, FSEN 2013, Tehran, Iran, April 24-26, 2013, Revised Selected Papers*, volume 8161 of *Lecture Notes in Computer Science*. Springer, 2013.
3. Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 35–44. IEEE Computer Society, 2012.
4. S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
5. Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008. doi:10.2168/LMCS-4(4:11)2008.
6. Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, 2004. doi:10.1007/978-3-540-24730-2_35.
7. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, 2004. doi:10.1145/1007352.1007390.
8. Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8*, 2016. doi:10.1145/2933575.2934561.
9. Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model checking languages of data words. In *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*. Springer, 2012.
10. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, volume 999 of *LNCS*. Springer, 1994.
11. 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.
12. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of parikh automata and related models. In *Third Workshop on Non-Classical Models for Automata and Applications - NCMA 2011, Milan, Italy, July 18 - July 19, 2011. Proceedings*, 2011.
13. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded parikh automata. *Int. J. Found. Comput. Sci.*, 23(8):1691–1710, 2012.
14. X. Cai and M. Ogawa. Well-structured pushdown systems. In *CONCUR 2013*, volume 8052 of *LNCS*. Springer, 2013.

15 Krishnendu Chatterjee, Andreas Pavlogiannis, and Yaron Velner. Quantitative interprocedural analysis. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17*, 2015.

16 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 2015.

17 F. S. de Boer, M. M. Bonsangue, and J. Rot. It is pointless to point in bounded heaps. *Sci. Comput. Program.*, 112, 2015.

18 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12), 2011. doi:10.1016/j.ipl.2011.03.019.

19 Oscar H. Ibarra. Visibly pushdown automata and transducers with counters. *Fundam. Inform.*, 148(3-4):291–308, 2016.

20 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In JosC.M. Baeten, JanKarel Lenstra, Joachim Parrow, and GerhardJ. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003.

21 Christof Löding and Karianto Wong. On nondeterministic unranked tree automata with sibling constraints. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2328.

22 Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Counting in trees. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., 2008.

23 Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 2004.

Narrowing down the Hardness Barrier of Synthesizing Elementary Net Systems

Ronny Tredup
Universität Rostock, Germany
ronny.tredup@uni-rostock.de

Christian Rosenke
arivis AG, Munich, Germany
christian.rosenke@arivis.com

Abstract

Elementary net system feasibility is the problem to decide for a given automaton A if there is a certain boolean Petri net with a state graph isomorphic to A. This is equivalent to the conjunction of the state separation property (SSP) and the event state separation property (ESSP). Since feasibility, SSP and ESSP are known to be NP-complete in general, there was hope that the restriction of graph parameters for A can lead to tractable and practically relevant subclasses. In this paper, we analyze event manifoldness, the amount of occurrences that an event can have in A, and state degree, the number of allowed successors and predecessors of states in A, as natural input restrictions. Recently, it has been shown that all three decision problems, feasibility, SSP and ESSP, remain NP-complete for linear A where every event occurs at most three times. Here, we show that these problems remain hard even if every event occurs at most twice. Nevertheless, this has to be paid by relaxing the restriction on state degree, allowing every state to have two successor and two predecessor states. As we also show that SSP becomes tractable for linear A where every event occurs at most twice the only open cases left are ESSP and feasibilty for the same input restriction.

2012 ACM Subject Classification Software and its engineering → Petri nets, Theory of computation → Problems, reductions and completeness

Keywords and phrases Elementary net systems, Petri net synthesis, NP-completeness, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.16

Related Version A technical report covering all results of the paper is available at [11], `https://arxiv.org/abs/1711.00220`.

1 Introduction

In this paper we investigate the complexity of synthesizing elementary net systems (ENS), which are the most fundamental type of Petri nets [12]. ENSs are a powerful language for describing processes in digital hardware and provide lots of methods for specification, verification and synthesis of particularly asynchronous or self-timed circuits [5][15]. Moreover, equipped with basic concepts like choice and causality, ENSs are the formal foundation of business process modeling languages, as for instance the Business Process Modeling Notation (BPMN) [9], Event Driven Process Chains (EPC) [6] or activity diagrams in the UML standard [7]. Especially because of their simpleness ENSs are useful for the specifications of workflow management systems like MILANO [1].

ENS synthesis for a given automaton A, called transition system (TS) in this context, means to find a Petri net N with a state graph isomorphic to A. More precisely, N has to be a directed graph on place nodes P and transition nodes T linked by flow arcs F such that, starting from the given initial marking $M_0 \subseteq P$, the net can transform its current marking $M \subseteq P$ into $M' \subseteq P$ by a transition $t \in T$, if $(p,t) \in F$ for all deallocated places $p \in M \setminus M'$ and $(t,q) \in F$ for all occupied places $q \in M' \setminus M$. The reachable markings of N are required to exactly cover A's states S while the transitions T embody A's events E. Every t-transition from M to M' has to correspond to an A-arc $s \xrightarrow{e} s'$ from the state s of M to the state s' of M' and labeled by the event e standing for t.

To assess the complexity of ENS synthesis, this paper analysis the corresponding decision problem, called feasibility. For a given TS A, feasibility asks if there is an ENS N with a state graph isomorphic to A. As not every TS can be synthesized into an ENS, feasibility is a problem worth studying. Usually, it is approached by the *state separation property* (SPP) and the *event state separation property* (ESSP) as, according to [3], A is feasible if and only if it satisfies both properties.

This does not mean that the SSP and the ESSP are not of interest when considered alone. Synthesizing TSs A having only the ESSP leads to Petri nets implementing all event sequences of A by their transitions but with less states [3]. Being able to efficiently decide the SSP, on the other hand, could serve as a quick-fail preprocessing mechanism for synthesis.

Hiraishi [8] shows that both, SSP and ESSP, are NP-complete. Feasibility is NP-complete [2], too. Nevertheless, considerable efforts have been made to find practically relevant tractable cases. For example, feasibility becomes tractable for Flip-Flop nets, a superclass of ENSs [13]. Workflow net models as defined in [1] are a subclass of ENS that allow polynomial time feasibility, too.

Rather than generalizing or restricting the set of nets, this paper restricts the problem input to learn about synthesis complexity. We propose the following two natural and fundamental parameters of TSs that, when controlled, should have a resounding positive impact on synthesis complexity:

State degree of a TS A is the maximum amount g of incoming and, respectively, outgoing edges at the states of A. The decision problems where input is restricted to so called g-grade TSs are referred to as g-grade SSP, g-grade ESSP, and g-grade feasibility. If $g = 1$ and A is not a cycle we use the term *linear*.

Event manifoldness of a TS A is the maximum amount k of edges in A that can be labeled with the same event. Accordingly, we speak about k-fold TSs and the problems k-SSP, k-ESSP, and k-feasibility.

Benchmarks of the digital hardware design community show that practical TSs often have limited state-degree [4]. If restricted event manifoldness is practical relevant has not been evaluated, yet, but it is a straight forward TS parameter.

In [14], we already show that even simultaneous and extreme restrictions of event manifoldness and state degree do not help reducing complexity. In fact, SSP, ESSP, and feasibility remain NP-complete for linear 3-fold input TSs. In this paper, we draw a more precise picture of the problems' hardness. All three of them remain NP-complete for g-grade k-fold TSs if $g \geq 2$ and $k \geq 2$.

On the other hand, 1-SSP, 1-ESSP, and 1-feasibility, that is, when events occur only once, are trivially tractable for every state degree. As this paper also shows that linear 2-SSP can be solved in polynomial time the only remaining open questions concern linear 2-ESSP and linear 2-feasibility. Figure 1 shows an overview of our findings.

This paper is organized as follows: For a start, the following two sections introduce preliminary notions used throughout the paper. In Section 4, we introduce our main result, a polynomial time reduction of cubic monotone one-in-three 3-SAT to 2-grade 2-ESSP. Our

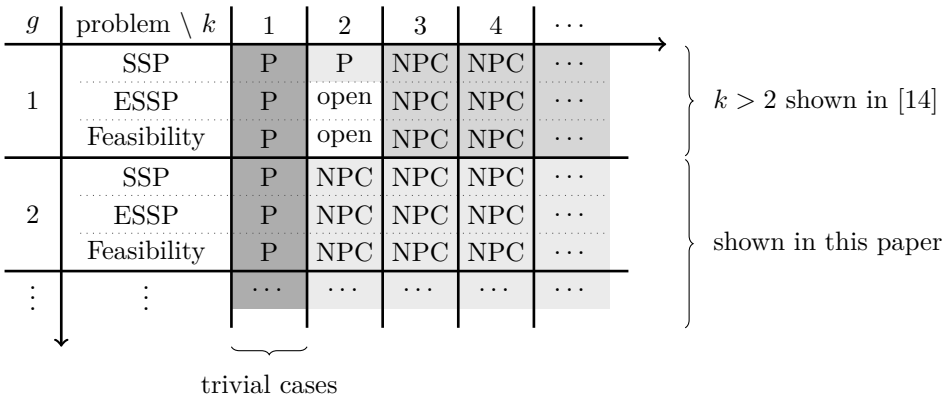


Figure 1 Overview of our results regarding the complexity of the SSP, the ESSP, and feasibility depending on the parameters state degree g and event manifoldness k. Considering the parameters individually, we have already determined the exact borderline between tractable and intractable cases.

reduction makes sure that the produced TS instances always have the SSP. In this way, ESSP and feasibility become the same problem with respect to the generated instances and, hence, we simultaneously show the NP-completeness of g-grade k-ESSP and g-grade k-feasibility for all $g \geq 2$ and $k \geq 2$.

That g-grade k-SSP is also hard to solve for $g, k \geq 2$ is provided in Section 5. Although usually perceived differently, we thereby imply that SSP is not easier than ESSP for TSs with limited state degree and event manifoldness.

Because of space limitations, some technical proofs have been omitted. For a complete presentation of all technical details, we refer to our technical report [11].

2 Preliminaries

This paper deals with (deterministic) *transition systems* (TS) $A = (S, E, \delta, s_0)$ which are determined by finite disjoint sets S of states and E of events, a partial transition function $\delta : S \times E \to S$, and an initial state $s_0 \in S$. Usually, we think of A as an edge-labeled directed graph with node set S where every triple $\delta(s, e) = s'$ is interpreted as an e-labeled edge $s \xrightarrow{e} s'$. For readability, we say that an event e *occurs* at state s if $\delta(s, e) = s'$ for some state s' and abbreviate this with $s \xrightarrow{e}$. Moreover, TSs are required to be *simple*, that is, there are no multi-edges $s \xrightarrow{e} s'$ and $s \xrightarrow{e'} s'$, *loop-free*, which rules out instant state recurrence like $s \xrightarrow{e} s$, *reachable*, where every state can be reached from s_0 by a directed path, and *reduced*, which means free of unused events in E.

Key concept of this paper are *g-grade* TSs A where both, the predecessor set $\{s' \mid \exists e \in E : \delta(s', e) = s\}$ and the successor set $\{s' \mid \exists e \in E : \delta(s, e) = s'\}$, contain at most g elements for every state $s \in S$. We use *linear* for 1-grade TSs that are not a cycle. Moreover, A is called *k-fold* if the set $\{(s, s') \mid \delta(s, e) = s'\}$ of e-connected states contains at most k pairs for every event $e \in E$.

Fundamental to the following notions are *regions* of TSs. A set $R \subseteq S$ is called a region of A if it permits a so-called *signature* $sig : E \to \{-1, 0, 1\}$. This means, all edges $s \xrightarrow{e} s'$ have to satisfy $R(s') = sig(e) + R(s)$, where, by a little abuse of notation, $R(s) = 1$ if $s \in R$ and otherwise $R(s) = 0$ for all $s \in S$. It is easy to see that every region R has a unique signature which is therefor called *the* signature sig_R of R. We say that an event e enters region R, respectively exits or obeys R, if $sig_R(e) = 1$, respectively $sig_R(e) = -1$ or $sig_R(e) = 0$.

Based on the previous definition, two states $s, s' \in S$ are *separable* in A if $R(s) \neq R(s')$ for some region R of A. Moreover, an event $e \in E$ is *inhibitable* at state $s \in S$ if there is a region R of A with either $R(s) = 0$ and $sig_R(e) = -1$ or $R(s) = 1$ and $sig_R(e) = 1$. Using this, a TS A has the *state separation property* (SSP), if all states of A are pairwise separable and it has the *event state separation property* (ESSP) if all events e of A are inhibitable at all states s where $s \xrightarrow{e}$ is not fulfilled. Then, A is *feasible* if and only if it has the SSP and the ESSP.

To study feasibility, ESSP and SSP for restricted TSs we define the g-grade (k-fold) problem for all naturals g (k) where the input is restricted to g-grade (k-fold) TSs. Notice that the set of g-grade k-fold TSs is a subclass of g'-grade k'-fold TSs in case $k \leq k'$ and $g \leq g'$. Hence, hardness results propagate up the problem hierarchy and efficient algorithms are legitimate for all lower classes.

As we approach feasibility by SSP and ESSP, which are defined on top of TSs, we omit a formal definition of ENSs and rather refer to, e.g., [3].

3 Unions, Transition System Containers

For our NP-completeness proofs this section introduces *unions*, a gadget concept to modularize our arguments. In a union, individual TSs are grouped together and treated as if being part of one TS. Moreover, we develop a *joining* operation to merge union parts and preserve their (E)SSP and feasibility.

Formally, if $A_0 = (S_0, E_0, \delta_0, s_0^0), \ldots, A_m = (S_m, E_m, \delta_m, s_0^m)$ are TSs with pairwise disjoint states then we say that $U(A_0, \ldots, A_m)$ is their *union*. By $S(U)$ we denote the entirety of all states in $A_0, \ldots, A_m$ and $E(U)$ is the aggregation of all events. The joint transition function $\Delta^U = \bigcup_{i=0}^{m} \delta_i$ of U is defined as

$$\Delta^U(s,e) = \begin{cases} \delta_i(s,e), & \text{if } s \in S_i \text{ and } e \in E_i, \\ \text{undefined}, & \text{else} \end{cases}$$

for all $s \in S(U)$ and all $e \in E(U)$. If every event in $E(U)$ occurs at most k times in U, not necessarily as part of the same TS, we say that U is k-fold.

For simplicity, we build unions recursively: Firstly, every TS A is identified with the union containing only A, that is, $A = U(A)$. Next, if $U_1 = U(A_0^1, \ldots, A_{m_1}^1), \ldots, U_n = (A_0^n, \ldots, A_{m_n}^n)$ are unions then $U(U_1, \ldots, U_n)$ is the union $U(A_0^1, \ldots, A_{m_1}^1, \ldots, A_0^n, \ldots, A_{m_n}^n)$ that flattens out the parent unions by cumulating all their TSs.

As we want to combine independent TSs $A_0, \ldots, A_m$ in a union $U = U(A_1, \ldots, A_m)$ and treat U as one TS, we need to lift regions, the SSP and the ESSP to U: We say that $R \subseteq S(U)$ is a region of U if it permits a signature $sig_R : E \to \{-1, 0, 1\}$. Hence, for all $i \in \{0, \ldots, m\}$ the subset $R_i = R \cap S_i$, coming from the states S_i of A_i, has to be a region of A_i with signature $sig_{R_i}(e) = sig_R(e)$ for all $e \in E_i$. Then, U has the SSP if for all states $s, s' \in S(U)$ coming from the same TS A_i there is a region R of U with $R(s) \neq R(s')$. Moreover, U has the ESSP if for all events $e \in E(U)$ and all states $s \in S(U)$ with $\neg(s \xrightarrow{e})$ there is a region R of U such that $R(s) = 0$ and $sig_R(e) = -1$ or $R(s) = 1$ and $sig_R(e) = 1$. Naturally, U is called feasible if it has both, the SSP and the ESSP.

To merge a union $U = U(A_0, \ldots, A_m)$ back into a single TS, we define the joining $A(U)$ as follows: If $s_0^0, \ldots, s_0^m$ are the initial states of U's TSs then $A(U) = (S(U) \cup Q, E(U) \cup Y \cup Z, \delta, q_0)$ is a TS with additional connector states $Q = \{q_0, \ldots, q_m\}$ and fresh events

$Y = \{y_0, \ldots, y_m\}, Z = \{z_0, \ldots, z_{m-1}\}$ joining the loose elements of U by

$$\delta(s,e) = \begin{cases} \Delta^U(s,e), & \text{if } s \in S(U) \text{ and } e \in E(U), \\ s_0^i, & \text{if } s = q_i \text{ and } e = y_i \\ q_{i+1}, & \text{if } s = q_i \text{ and } e = z_i. \end{cases}$$

Notice that $A(U)$ preserves k-foldness and, if every initial state s_0^i has at most one predecessor in A_i, it preserves g-gradeness for $g \geq 2$. The following lemma certifies the validity of joining (most) unions:

▶ **Lemma 1.** *Let $U = U(A_0, \ldots, A_m)$ be a union of TSs $A_0, \ldots, A_m$ which fulfill for every event e that there is at least one state s with $\neg(s \xrightarrow{e})$. Then U has the (E)SSP, respectively is feasible, if and only if the joining $A(U)$ has the (E)SSP, respectively is feasible.*

Proof. *If*: Projecting a region separating s and s', respectively inhibiting e at s, in $A(U)$ to the component TSs yields a region separating s and s', respectively inhibiting e at s in U. Hence, the (E)SSP of $A(U)$ trivially implies the (E)SSP of U.

Only if: A region R of U separating s and s', respectively inhibiting e at s, can be completed to become an equivalent region of $A(U)$ by setting

$$R(q_i) = 0, sig_R(z_j) = 0, \text{ and } sig_R(y_i) = R(s_0^i)$$

for all $i, j \in \{0, \ldots, m\}, j < m$.

Notice that R, defined as above, also inhibits e at all connector states. Hence, to inhibit an event $e \in E(U)$ at all connector states of $A(U)$, we choose any U-region R_e that inhibits e at any state $s \in S(U)$. As we require that every $e \in E(U)$ has $s \in S(U)$ with $\neg(s \xrightarrow{e})$, the ESSP of U implies the existence of R_e. Thus, in $A(U)$ every event of U can be inhibited at all required states.

For the (E)SSP of $A(U)$ it is subsequently sufficient to analyze (event) state separation concerning the connector states (events). By the uniqueness of the connector events $Y \cup Z$, it is easy to see that each connector state q_i on its own defines a region $R_i = \{q_i\}$ of $A(U)$ that inhibits y_i, z_i and separates q_i in $A(U)$. ◀

4 The Hardness of the ESSP and Feasibility for 2-grade 2-fold Transition Systems

This section presents our main result and answers the question if restricting the event manifoldness to $k = 2$ helps reducing the complexity of synthesizing ENS:

▶ **Theorem 2.** *Deciding the ESSP or feasibility is NP-complete on g-grade k-fold transition systems for all $g \geq 2$ and all $k \geq 2$.*

The rest of this section is devoted to the proof of this theorem.

That the g-grade k-fold versions of the ESSP and feasibility are contained in NP is clearly not a proof obligation here, as this already follows from the NP-completeness of the unrestricted problems [2][8].

For the proof of completeness in NP, we basically reduce cubic monotone one-in-three 3-SAT, which is NP-complete [10], to 2-grade 2-ESSP in polynomial time. Therefore, we start the reduction from a cubic monotone boolean CNF expression $\varphi = \{C_0, \ldots, C_{m-1}\}$, a set of negation-free 3-clauses where every variable occurs in exactly three clauses. The result is a union U^φ of gadget TSs that has the ESSP if and only if φ has a one-in-three model

M, that is, a subset of φ's variables $V(\varphi)$ with $|M \cap C_i| = 1$ for all $i \in \{0, \ldots, m-1\}$. This means that M exactly covers all clauses of φ. For example, the expression

$$\varphi_0 = \big\{\{x_0, x_1, x_2\}, \{x_0, x_1, x_4\}, \{x_0, x_2, x_3\}, \{x_1, x_4, x_5\}, \{x_2, x_3, x_5\}, \{x_3, x_4, x_5\}\big\}$$

has six clauses over the variables $V(\varphi_0) = \{x_0, \ldots, x_5\}$ which are satisfied by the one-in-three model $M_0 = \{x_0, x_5\}$. Unfortunately, we cannot use the expression φ_0 as a running example since the union U^{φ_0} resulting from our construction would already have 1500 states. Thus, its presentation as a whole would go far beyond the scope of this paper.

The construction of U^{φ} makes sure that even in the joining $A^{\varphi} = A(U^{\varphi})$ every event is used at most twice and has at most two predecessors and two successors. By design, the ESSP of U^{φ} implies the SSP, too. This makes ESSP and feasibility the same problem, even for A^{φ} as stated in Lemma 1. Consequently, our proof provides the NP-hardness for both problems on 2-grade 2-fold TSs.

In the following, we start with the details of constructing U^{φ}. The union consists of several functional components. Firstly, it installs a TS H, called the *head*, which initializes the connection between the satisfiability problem and the ESSP. It introduces the key event k that is supposed to be inhibitable at a certain key state if and only if φ has a one-in-three model. In order to achieve this behavior, U^{φ} adds a so-called translator T_i for every clause C_i. For a key region, one that inhibits k at the key state, the purpose of T_i is to implement one-in-three behavior for C_i. More precisely, T_i applies events to represent the three variables of C_i and assures that exactly one of them has a positive signature while the other two have to obey. This means for a key region that every gadget $T_0, \ldots, T_{m-1}$ has exactly one entering variable event which exactly translates into a one-in-three model for φ. Reversely, every one-in-three model tells us how we can define a key region by choosing exactly one entering variable in every translator.

The main problem so far is to get along with the restriction of using every event only twice. We solve this problem by adding more TSs to U^{φ} that, for a key region, generate helper and replacement events with predefined signatures, leaving, entering, or obeying. To create a better picture of our method, the following introduces the details of all applied gadget TSs. See Figure 2 to also visualize the technical details of the description.

Head. H is a TS having two responsibilities. Firstly, it introduces the key event k and the key state $h_{0,8}$. We add the name affix *key* to regions R_{key} of U^{φ} that inhibit k at $h_{0,8}$, or more precisely, where $sig_{R_{key}}(k) = -1$ and $h_{0,8} \notin R_{key}$. Secondly, H cooperates with the subsequent duplicator gadgets to prepare sufficient amounts of events with negative signature. The reason is that our reduction has to get along with applying k just twice. To duplicate the negative signature of k to other events, the so-called key copies, H works with a production line of $14m$ submodules H_j, each cooperating with a duplicator D_j to initialize one key copy. More precisely, for a key region, H_j prepares three events for D_j, two so-called *vice* events v_{2j}, v_{2j+1}, which have a positive signature (that is, vice with respect to the signature of k) and one obeying *wire* event w_{2j}. In return, the duplicator provides two key copies k_{3j}, k_{3j+1} and one obeying *accordance* event a_j. In H_{j+1} these three result events are used for the synchronization of the next vice and wire events. The main result of D_j, however, is k_{3j+2}, one of the $14m$ key copies that are free to be applied in the other reduction gadgets.

See Figure 2 (a) for a definition of H together with an illustration of H's part of a key region, R^H. Observe that there are *reachability* events $r_0, \ldots, r_{14m-2}$ which have the only purpose to make every state of H reachable from initial $h_{0,0}$. Moreover, for R^H, every module H_j receives key copies k_{3j-3}, k_{3j-2} and accordance event a_{j-1} from D_{j-1}. Thanks to a_{j-1},

the state $h_{j,8}$ behaves according to the key event $h_{0,8}$ and is excluded from R^H. Because of exiting k_{3j-3} the state $h_{j,1}$ is out of R^H, too. This imprints a zero signature on the *zero* events z_{2j}, z_{2j+1}. Exiting k_{3j-2} puts $h_{j,4}$ into R^H and excludes $h_{j,5}$ which, together with z_{2j}, z_{2j+1}, makes v_{2j}, v_{2j+1} entering and w_{2j}, w_{2j+1} obeying.

Duplicators. D_j are TSs that, for a key region, generate three key copies $k_{3j}, k_{3j+1}, k_{3j+2}$ and one obeying accordance event a_j using the vice events v_{2j}, v_{2j+1} with positive signature and the obeying wire event w_{2j}. Figure 2 (b) defines D_j and demonstrates the duplicator fraction R^{D_j} of a key region. The entering vice events force $d_{j,2}, d_{j,4}$ into R^{D_j} and exclude $d_{j,1}, d_{j,3}$. The obeying wire event signals the condition of $d_{j,4}$ to $d_{j,0}$ putting it into R^{D_j}. By design, $k_{3j}, k_{3j+1}, k_{3j+2}$ are exiting and a_j becomes obeying. As H consumes only k_{3j}, k_{3j+1} and a_j, we keep the remaining duplicate k_{3j+2} of k. Creating $14m$ duplicators in total, we get $14m$ free key copies.

Barters. B_q are TSs that, for a key region, barter key copies k_{q_1}, k_{q_2} for one obeying so-called *consistency* event c_q. The indices $q_1 = 18m + 6q + 1$ and $q_2 = q_1 + 3$ select two of the last $4 \cdot 2m$ items from the list of free key copies. The use of consistency events is to synchronize three events $x_i^\alpha, x_i^\beta, x_i^\gamma$ for every variable x_i. The reason is that we cannot represent the three occurrences of x_i in the expression φ by an event that can only be used twice. Consequently, we require three generated events of consistent signature to represent x_i. Figure 2 (c) introduces B_q and shows a respective key region part R^{B_q}. As both key copies are leaving, $b_{q,0}, b_{q,2}$ are in R^{B_i} and c_q is obeying. Altogether, we add $4m$ barters that consume $8m$ key copies to generate $4m$ consistency events.

Variable manifolders. X_i are TSs synchronizing three events for every variable $x_i \in V(\varphi)$. If $C_\alpha, C_\beta, C_\gamma$ are the three clauses containing x_i then X_i provides the events $x_i^\alpha, x_i^\beta, x_i^\gamma$. For a key region, they are supposed to have the same signature in order to treat them as manifestations of the same event representing x_i. The definition of X_i as well as an illustration of a possible key region fragment R^{X_i} are given in Figure 2 (d). To create the event equivalence, X_i applies four consistency events, c_{4i}, c_{4i+1}, c_{4i+2}, c_{4i+3}. Their obedience condemns the two state groups $x_{i,0}, x_{i,1}, x_{i,2}$ and $x_{i,3}, x_{i,4}, x_{i,5}$ to a consistent behavior with respect to R^{X_i}, that is, either all states of a group are part of the region or none of them. This brings $x_i^\alpha, x_i^\beta, x_i^\gamma$ into synchronicity.

Translators. T_i are unions $T_i = U(T_{i,0}, T_{i,1}, T_{i,2})$ of three TSs, each. For a key region, T_i implements $C_i = \{x_a, x_b, x_c\}$ by allowing a positive signature for exactly one variable representation, either x_a, x_b or x_c. Figure 2 (e-g) define the three TSs and introduces a possible key region fragment that assigns a positive signature to event x_b^i representing x_b. Apparently, $T_{i,1}$ parenthesizes an event for x_b and the proxy event p_i with two key copies while $T_{i,2}$ does the same for the locum event $\tilde{x}_b^i$ and p_i. For a key region, all key copies exit and the proxy event behaves equally in both TSs. This aligns the signature of x_b^i and $\tilde{x}_b^i$ and makes it non-negative. Furthermore, $T_{i,0}$ is a key copy delimited sequence of events that, for a key region, prevents a negative signature for x_a^i, x_c^i. As the limiting key copies are exiting and as none of $x_a^i, \tilde{x}_b^i, x_c^i$ can be exiting, exactly one of the events x_a^i, x_b^i, x_c^i has to enter.

Altogether, this construction results in the union $U^\varphi = U(U_1^\varphi, U_2^\varphi)$ with

$$U_1^\varphi = U(H, D_0, \ldots, D_{14m-1}),$$
$$U_2^\varphi = U(B_0, \ldots, B_{4m-1}, X_0, \ldots, X_{m-1}, T_0, \ldots, T_{m-1}).$$

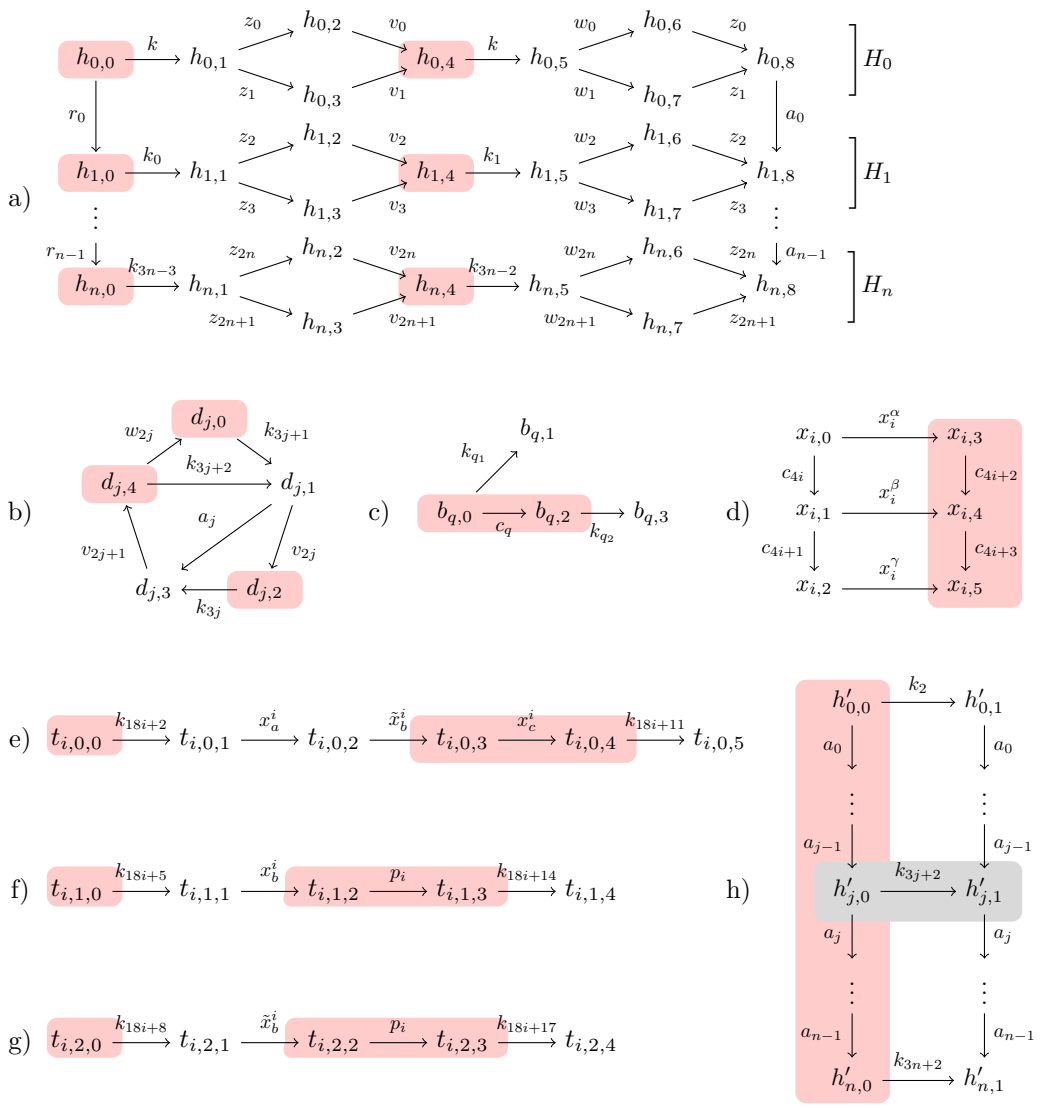


Figure 2 a-g) The gadgets of U^φ with their respective fractions of a key region. The red marked states are included in the key region and the unmarked are excluded. a) The head H with submodules $H_0, \ldots, H_n$ where $n = 14m - 1$. b) D_j, one of the $14m$ duplicators that provide the $14m$ key copies. c) B_q, one out of $4m$ barters trading $8m$ key copies for $4m$ consistency events. Here, $q_1 = 6q+18m+1$ and $q_2 = q_1 + 3$. d) The variable manifolder X_i using four consistency events to synchronize three variable events for x_i. Together, the m variable manifolders consume all $4m$ available consistency events. e-g) The translator T_i consisting of $T_{i,0}$ (e), $T_{i,1}$ (f), and $T_{i,2}$ (g). Using six key copies, T_i implements the clause C_i. All m translators together consume the remaining $6m$ key copies. h) The head H' of the union U^φ_{SSP} from Section 5. The red marked states describe the key region $R^{H'}$ and the gray marked states provide a region of H' that separates the states $h'_{j,0}, h'_{j,1}$ from the remaining states of H'.

The reason for separating U^φ into two sub unions U^φ_1 and U^φ_2 is that we want to reuse U^φ_2 in Section 5. Here, our last construction step is to join the TSs of U^φ in order to obtain the combined TS $A^\varphi = A(U^\varphi)$. Check Figure 2 to see that the initial states of the gadgets, that is, $h_{0,0}$, $d_{j,0}$, $b_{q,0}$, $x_{i,0}$, $t_{i,0,0}$, $t_{i,1,0}$, and $t_{i,2,0}$, have at most one predecessor state each.

Hence, the definition of joining guarantees that A^φ does not exceed the state degree of two. Moreover, as every event of U^φ occurs at most twice and A^φ just introduces additional unique events, A^φ is a 2-grade 2-fold TS.

Before we can show that A^φ has the ESSP if and only if φ has a one-in-three model, we need the next two lemmas to formalize the properties of key regions in U^φ:

▶ **Lemma 3.** *If R is a region of U_1^φ inhibiting k at $h_{0,8}$, that is, where, without loss of generality, $sig_R(k) = -1$ and $h_{0,8} \notin R$ then*

1. *for all $j \in \{0, \dots, 14m-1\}$ the region contains $h_{j,0}, h_{j,4}$ and excludes $h_{j,1}$, $h_{j,2}$, $h_{j,3}$, $h_{j,5}$, $h_{j,6}$, $h_{j,7}$, $h_{j,8}$,*
2. *for all $j \in \{0, \dots, 14m-1\}$ the region contains $d_{j,0}, d_{j,2}, d_{j,4}$ and excludes $d_{j,1}, d_{j,3}$,*
3. *all key copies exit, that is, for all $j \in \{0, \dots, 14m-1)$ the events k_{3j}, k_{3j+1}, k_{3j+2} have negative signature.*

Proof. Consider the individual gadget regions R^H and R^{D_j} demonstrated in Figure 2 (a) and (b). We show that, combined, they define the only region R of U_1^φ that inhibits k at $h_{0,8}$ with $sig_R(k) = -1$ and $h_{0,8} \notin R$. For this purpose, we use induction over j and simultaneously show that R fulfills (1-3):

For a start, let $j = 0$. We show that $sig_R(k) = -1$ and $R(h_{0,8}) = 0$ force R to coincide with R^H with respect to the part H_0 of H and with R^{D_0}. The requirement $sig_R(k) = -1$ immediately brings $R(h_{0,0}) = R(h_{0,4}) = 1$ and $R(h_{0,1}) = R(h_{0,5}) = 0$. Then, $R(h_{0,1}) = 0$ implies that $sig_R(z_0), sig_R(z_1) \in \{0, 1\}$. The second premise $R(h_{0,8}) = 0$ yields $sig_R(z_0), sig_R(z_1) \in \{-1, 0\}$, which consequently results in $sig_R(z_0) = sig_R(z_1) = 0$. This implies $R(h_{0,2}) = R(h_{0,3}) = 0$ and $R(h_{0,6}) = R(h_{0,7}) = 0$ making v_0, v_1 entering and w_0, w_1 obeying. The entering signature of v_0, v_1 makes R include $d_{0,2}, d_{0,4}$ and exclude $d_{0,1}, d_{0,3}$. As $R(d_{0,4}) = 1$ and w_0 obeys, we get $R(d_{0,0}) = 1$. By $R(d_{j,0}) = R(d_{j,4}) = 1$ and $R(d_{j,1}) = 0$ we obtain that k_1, k_2 exit. By $R(d_{j,1}) = R(d_{j,3}) = 0$ and $R(d_{j,2}) = 1$ we have that k_0 exits and a_0 obeys.

Now assume that R coincides with R^{D_i} and R^H on the parts H_i for all i less than j. Moreover, suppose that (1-3) hold for all indices less j. As $R(h_{j-1,8}) = 0$ and $sig_R(a_{j-1}) = 0$, we have $R(h_{j,8}) = 0$. Furthermore, we get the exiting $k_{3(j-1)}, k_{3(j-1)+1}$. Hence, we basically have the same situation as in H_0 and D_0. Consequently, a similar argumentation as for the induction start yields that R contains exactly the states $h_{j,0}, h_{j,4}$ of H_j and $d_{j,0}, d_{j,2}, d_{j,4}$ of D_j. This makes the vice events v_{2j}, v_{2j+1} enter and the wire events w_{2j}, w_{2j+1} obey. Moreover, D_j lets the key copies k_{3j}, k_{3j+1}, k_{3j+2} have a negative signature and a_j obey. ◀

▶ **Lemma 4.** *If R is a region of U_2^φ having an exiting signature for all key copies, that is, $sig_R(k_{3j+2}) = -1$ for all $j \in \{0, \dots, 14m-1\}$, then*

1. *for all $q \in \{0, \dots, 4m-1\}$ the region R contains $b_{q,0}, b_{q,2}$ and excludes $b_{q,1}, b_{q,3}$ and has $sig_R(c_q) = 0$,*
2. *for all $i \in \{0, \dots, m-1\}$ variable x_i, which occurs in clauses C_α, C_β, C_γ, is represented by events x_i^α, x_i^β, x_i^γ having the same signature*

   $$sig_R(x_i^\alpha) = sig_R(x_i^\beta) = sig_R(x_i^\gamma), \text{ and}$$

3. *for all $i \in \{0, \dots, m-1\}$ clause $C_i = \{x_a, x_b, x_c\}$ is realized in translator T_i making exactly one of the events x_a^i, x_b^i, x_c^i enter while the other two obey.*

Proof. Statement (1) means that R coincides with R^{B_q} for every barter B_q. As the key copies k_{q_1} and k_{q_2} are assumed to exit for $q_1 = 18m + 6q + 1$ and $q_2 = q_1 + 3$, this statement trivially follows. First and foremost, this implies that all consistency events $c_q, q \in$

$\{0,\dots,4m-1\}$ have an obeying signature. In statement (2), this obedience immediately fixes the states $x_{i,0}, x_{i,1}, x_{i,2}$ of variable manifolder X_i to behave consistently, that is, $R(x_{i,0}) = R(x_{i,1}) = R(x_{i,2})$. Analogously, we derive $R(x_{i,3}) = R(x_{i,4}) = R(x_{i,5})$. This implicates $R(x_{i,3}) - R(x_{i,0}) = R(x_{i,4}) - R(x_{i,1}) = R(x_{i,5}) - R(x_{i,2})$. Consequently, the variable events $x_i^\alpha, x_i^\beta, x_i^\gamma$ have the same signature for all $i \in \{0,\dots,m-1\}$.

Finally, statement (3) for the translator T_i can be seen as follows: For any region R' of a linear TS it is a simple observation that $\sum_{j=j_1}^{j_2-1} sig_R(e_j) = R(s_{j_2}) - R(s_{j_1})$ for any subsequence $s_{j_1} \xrightarrow{e_{j_1}} \dots \xrightarrow{e_{j_2-1}} s_{j_2}$ within the TS. As the TSs of T_i are linear, we get from $R(t_{i,1,3}) - R(t_{i,1,1}) = 1$ and $R(t_{i,2,3}) - R(t_{i,2,1}) = 1$ that $sig_R(x_b^i) + sig_R(p_i) = sig_R(\tilde{x}_b^i) + sig_R(p_i) = 1$. That means, $sig_R(x_b^i) = sig_R(\tilde{x}_b^i) = 1 - sig_R(p_i)$ which implies that x_b^i has a non-negative signature. As $R(t_{i,0,1}) = 0$ and $R(t_{i,0,4}) = 1$, we have non-negative signature of x_a^i, x_c^i, too. That $sig_R(x_a^i) + sig_R(\tilde{x}_b^i) + sig_R(x_c^i) = 1$ implies that exactly one of these events has a positive signature. ◀

Lemma 3 and Lemma 4 state that the structure of a key region defines a model of φ. That is why we can say that the existence of key region for U^φ implies the one-in-three satisfiability of φ:

▶ **Lemma 5.** *If there is a key region of U^φ then φ has a one-in-three model.*

Proof. Let R_{key} be a key region of U^φ, that is, $sig_{R_{key}}(k) = -1$ and $R_{key}(h_{0,8}) = 0$. First of all, Lemma 3 implies that $sig_{R_{key}}(k_{3j+2}) = -1$ for all $j \in \{0,\dots,14m-1\}$. As a consequence, we obtain from Lemma 4 for all variables x_i and their three occurrences in clauses $C_\alpha, C_\beta, C_\gamma$ that

$$sig_{R_{key}}(x_i^\alpha) = sig_{R_{key}}(x_i^\beta) = sig_{R_{key}}(x_i^\gamma).$$

Moreover, Lemma 4 means for every clause $C_i = \{x_a, x_b, x_c\}$ that exactly one of the events x_a^i, x_b^i, x_c^i has a positive signature while the other two obey. Hence, if we add a variable x_i to a set M if and only if the corresponding events have positive signature, then we clearly obtain for all clauses C_i that $|M \cap C_i| = 1$. This makes M a one-in-three model. ◀

The other way around, the required equivalence obliges us to derive a key region from any one-in-three model. We argue that working our way backwards through the construction ends up in a region that inhibits k at the key state.

▶ **Lemma 6.** *If φ has a one-in-three model then there is a key region of U^φ.*

Proof. Let $M \subseteq V(\varphi)$ be a one-in-three model of φ. We progressively build a region R by following the requirements of every individual gadget.

Firstly, for every variable x_i occurring in $C_\alpha, C_\beta, C_\gamma$ we take care that $sig_R(x_i^\alpha) = sig_R(x_i^\beta) = sig_R(x_i^\gamma) = M(x_i)$ where $M(x_i) = 1$ if $x_i \in M$ and $M(x_i) = 0$, otherwise. To this end, we let $x_{i,3}, x_{i,4}, x_{i,5} \in R^{X_i}$. Moreover, we set $x_{i,0}, x_{i,1}, x_{i,2} \in R^{X_i}$ if and only if x_i is not in M. This makes the consistency events $c_{4i}, c_{4i+1}, c_{4i+2}, c_{4i+3}$ obey. As different variable manifolders do not share events, the regions $R^{X_0},\dots,R^{X_{m-1}}$ are pairwise compatible.

For every clause $C_i = \{x_a, x_b, x_c\}$ the model M selects exactly one variable. By the M-conform construction of $R^{X_a}, R^{X_b}, R^{X_c}$ we get that exactly one of the events x_a^i, x_b^i, x_c^i enters and the others obey. Making the key copies of T_i exit, generates a sub region R^{T_i}. That T_i and T_j share events only for $i = j$ makes $R^{T_0},\dots,R^{T_{m-1}}$ pairwise compatible. As the variable events are selected in compliance with the variable manifolders and as translators and manifolders do not share further events, their sub regions are also compatible.

By the obeying consistency events, we can define a sub region R^{B_q} for every $q \in \{0, \ldots, 4m-1\}$. This also makes the used key copies exiting. As different barters have no event in common, share only consistency events with variable manifolders and no events at all with translators, the regions are all compatible.

Head and duplicators just share key copies with translators and barters. As all key copies are exiting and as the provided sub regions meet the conditions of Lemma 3, we can use this lemma as a construction manual for the sub regions $R^H, R^{D_0}, \ldots, R^{D_{14m-1}}$. Altogether, we get that the set R formed by

$$R^H \cup R^{D_0} \cup \cdots \cup R^{D_{14m-1}} \cup R^{B_0} \cup \cdots \cup R^{B_{4m-1}} \cup R^{X_0} \cup \cdots \cup R^{X_{m-1}} \cup R^{T_0} \cup \cdots \cup R^{T_{m-1}}$$

is a region of U^φ inhibiting k at $h_{0,8}$. ◀

At this point, the previous lemmas have established that φ has a one-in-three model M if and only if there is a key region for U^φ. Although this is basically the foundation of the proof for Theorem 2, it just delivers the *only-if* direction for the NP-completeness of 2-grade 2-fold ESSP by now: If A^φ has the ESSP then Lemma 1 lifts the ESSP to U^φ. By definition, there also has to be a region that inhibits k at $h_{0,8}$, a key region. Then Lemma 5 implies the existence of the one-in-three model M for φ.

Reversely, having M, Lemma 6 only inhibits k at the key state. For the remaining events e and states s of U^φ with $\neg(s \xrightarrow{e})$, we still have to show that e is inhibitable at s:

▶ **Lemma 7.** *If φ has a one-in-three model then $e \in E(U^\varphi)$ is inhibitable at $s \in S(U^\varphi)$ for every event e and state s of U^φ that fulfill $\neg(s \xrightarrow{e})$.*

Lemma 6 already shows the essential part of Lemma 7. But the proof for the remaining non-key event state combinations is very technical and does not lead to further insights. Therefore and for space limitations, we only go into one example here and refer to our technical report [11] for a full analysis:

▶ **Lemma 8.** *If φ has a one-in-three model then the key event k is inhibitable at all states $s \in S(U^\varphi)$ that fulfill $\neg(s \xrightarrow{k})$.*

Proof. Every relevant state s is subsequently provided with a region that inhibits k at s. For brevity however, we omit to repeat the key-region here, which already inhibits k at many states. To define the other regions, we just specify the signature of non-obeying events, as the majority of events are obedient. To this end, we present every required region as one entry in the following listing:

states	*exit*	*enter*	*affected TSs*
remaining states of H except $h_{1,0}$	a_0, k, k_0, r_1	v_1, w_1, z_0	H, D_0
$h_{1,0}$	a_1, k, k_1, k_2, k_3, k_8, r_0, r_2, w_4, w_5	a_2, k_4, k_5, k_6, r_1, v_1, w_1, z_0, z_2, z_3	H, D_0, D_1, D_2, $T_{0,0}$, $T_{0,1}$, $T_{0,2}$
$S(U^\varphi) \setminus S(H)$	k	z_0, z_1	H

The first column *states* lists the states s where k is inhibited by the respective region. The *exit* and *enter* columns provide the exiting, respectively entering, events of that region. To easily find the TSs that contain at least one of these non-obeying events, one can use the *affected TSs* column in the listing.

Notice that the first two lines of the listing show that k is inhibitable at all states of H. The third line presents a region of U^φ where k exits and all non-obeying events occur in H. Therefore, this region inhibits k at all remaining states of U^φ. ◀

The rest of the proof for Lemma 7 works just the same as demonstrated by Lemma 8. This leads to the *if* direction for the NP-completeness of 2-grade 2-fold ESSP: If φ has a one-in-three model M then Lemma 7 tells us that U^φ has the ESSP. Using Lemma 1, we can bring the ESSP down to A^φ, too. Altogether, we may now state that φ has a one-in-three model if and only if A^φ has the ESSP. As our construction is easily done in polynomial time, we have shown the NP-completeness of 2-grade 2-fold ESSP and by that, half of Theorem 2.

To complete the theorem's proof, it remains to establish that φ has a one-in-three model M if and only if A^φ is feasible. However, this is fairly easy reusing the work we have already done. In fact, the first direction is already there: If A^φ is feasible then it also has the ESSP, which we know implies the existence of M. Reversely, if M exists, then it is sufficient to show that, beside the already established ESSP, A^φ has the SSP, too:

▶ **Lemma 9.** *If φ has a one-in-three model then U^φ has the SSP.*

Proof. If s, s' are two states that do not belong to the same TS of U^φ then they are separable by definition. Hence, let s, s' be two distinct states of one TS $A \in U^\varphi$. If there is an event e that occurs at s, that is, $s \xrightarrow{e}$, but not s', that is, $\neg(s' \xrightarrow{e})$ then s, s' are separable. The reason for this comes from Lemma 7, which states that e is inhibitable at s' by a region R of U^φ. This means, that, without loss of generality, $R(s') = 0$ and $sig_R(e) = -1$, which implies $R(s) = 1$.

Using this condition, we get the separability for all state pairs s, s' that are in one of our TSs except for H as follows: As Figure 2 shows, every event of $A \neq H$ occurs only once within A and (ii) there is only one state of A without a successor. Hence, without loss of generality, there is an event e that occurs at s but not at s'.

Figure 2 also demonstrates that all the events $\{v_{2j}, v_{2j+1}, w_{2j}, w_{2j+1} \mid 0 \leq j < 14m\}$ and $\{a_j, k_{3j}, k_{3j+1}, r_j \mid 0 \leq j < 14m - 1\}$ occur only once in H and that $h_{14m-1,8}$ is the only state of H without a successor. Consequently, if $s, s' \in S(H)$ and s is neither in $\{h_{j,1}, h_{j,6}, h_{j,7} \mid 0 \leq j < 14m\}$ nor in $\{h_{0,4}, h_{14m-1,8}\}$ then the above condition makes s and s' separable, too. Moreover, notice that k is the only event occurring at $h_{0,4}$ and that no event occurs at $h_{14m-1,8}$ at all. Hence, our argument works to separate these two states from all states in $S(H)$, too.

As seen in Figure 2, z_{2j} and z_{2j+1} occur only within the part H_j of H. Applying the above condition again, we get for all $0 \leq j < 14m$ that $s \in \{h_{j,1}, h_{j,6}, h_{j,7}$ is separable from $s' \in S(H) \setminus \{h_{j,0}, \ldots, h_{j,8}\}$.

It remains to show that the states $\{h_{j,1}, h_{j,6}, h_{j,7}\}$ are pairwise separable for all $j \in \{0, \ldots, 14m - 1\}$. Notice that z_{2j} occurs at $h_{j,1}$ and $h_{j,6}$ but not at $h_{j,7}$. Hence, using the region inhibiting z_{2j} at $h_{j,7}$ coming from the above argumentation, separates both, $h_{j,1}$ and $h_{j,6}$ from $h_{j,7}$. Similarly, z_{2j+1} occurs at $h_{j,7}$ but not at $h_{j,6}$ which leads to their separability, too. ◀

As a last step, we can use Lemma 1 again, to bring the SSP of U^φ down to A^φ, too. This finally proves Theorem 2.

5 The Hardness of SSP for 2-grade 2-fold Transition Systems

This section completes our complexity analysis for the synthesis of TSs having event manifoldness less than three:

▶ **Theorem 10.** *Deciding the SSP is NP-complete on g-grade k-fold transition systems for all $g \geq 2$ and all $k \geq 2$.*

Proof. We reuse most of the reduction from Section 4 and create yet another union $U^{\varphi}_{\text{SSP}} = U(H', U^{\varphi}_2)$ by simply replacing U^{φ}_1 with the new head TS H' shown in Figure 2 (h). We prove that φ has a one-in-three model if and only if the two key states $h'_{0,0}, h'_{1,0}$ are separable by a key region R'_{key} if and only if U^{φ}_{SSP} has the SSP.

If U^{φ}_{SSP} has the SSP then there is a key region R'_{key} where, without loss of generality, $R'_{key}(h'_{0,0}) = 1$ and $R'_{key}(h'_{0,1}) = 0$. This implies $sig_{R'_{key}}(k_2) = -1$. Using this as a start, induction over j infers from $R'_{key}(h'_{j,0}) = 1$ and $R'_{key}(h'_{j,1}) = 0$ that a_j is obeying and that $R'_{key}(h'_{j+1,0}) = 1, R'_{key}(h'_{j+1,1}) = 0$ and that $sig_{R'_{key}}(k_{3j+2}) = -1$. Hence, on H' the key region is just $R^{H'}$ from Figure 2 (h). By Lemma 4, the exiting key copies $k_{3j+2}, j \in \{0, \ldots, 14m-1\}$ imply that every variable $x_i \in V(\varphi)$ is represented by three synchronized variable events and for every clause $C_j = x_a, x_b, x_c$ exactly one of x^j_a, x^j_b, x^j_c enters. Hence, taking just the variables of entering events, gets φ a one-in-three model.

Reversely, if φ has a one-in-three model, Lemma 6 provides a key region R_{key} for U^{φ}. As all key copies exit, we can easily transform R_{key} into a key region R'_{key} for U^{φ}_{SSP} by making the accordance event a_j obeying and defining $R'_{key}(h'_{j,0}) = 1, R'_{key}(h'_{j,1}) = 0$ for all $j \in \{0, \ldots, 14m-1\}$ as well as removing the region's definition on states of $S(U^{\varphi}_1)$. To argue the SSP of U^{φ}_{SSP} consider for all $i \in \{0, \ldots, m-1\}$ and all $j \in \{1, 2\}$ the region $R_{i,j}$, where all events obey but $sig_{R_{i,j}}(p_i) = -1$. This regions separates every state in $\{t_{i,j,0}, \ldots, t_{i,j,2}\}$ from every state in $\{t_{i,j,3}, t_{i,j,4}\}$. Analogously, let R_i be the region where just $sig_{R_i}(\tilde{x}^i_b) = -1$. This region separates states of $\{t_{i,0,0}, \ldots, t_{i,0,2}\}$ from states $\{t_{i,0,3}, \ldots, t_{i,0,5}\}$ as well as states of $\{t_{i,2,0}, t_{i,2,1}\}$ from $\{t_{i,2,2}, \ldots, t_{i,2,4}\}$. It is easy to see that the remaining state pairs of $S(U^{\varphi}_2)$ are either separated by the key region or by a region where all events obey except for one variable event or one consistency event. Finally, as no accordance event of H' occurs in U^{φ}_2, taking the key region and for all $j \in \{0, \ldots, 14m-1\}$ the region $\{h'_{j,0}, h'_{j,1}\}$ solves the remaining separation problems in H'.

Using Lemma 1, it is again possible to transfer the SSP from U^{φ}_{SSP} to its joined TS $A^{\varphi}_{\text{SSP}} = A(U^{\varphi}_{\text{SSP}})$ and back. As the polynomial time construction of A^{φ}_{SSP} is obvious just as its state degree and event manifoldness of two, the proof is complete. ◀

6 The Tractability of SSP for Linear 2-fold Transition Systems

This section shows that 2-fold SSP becomes tractable if we turn to linear TSs:

▶ **Theorem 11.** *Deciding the SSP can be done in polynomial time on linear 2-fold transition systems.*

To pove this theorem, we provide the following SSP-equivalent property for linear 2-fold TSs: If $A = s_0 \xrightarrow{e_1} \ldots \xrightarrow{e_t} s_t$ is a linear TS then $A^i_j = s_i \xrightarrow{e_{i+1}} \ldots \xrightarrow{e_j} s_j$ is called a *subsequence* of A for all $0 \leq i < j \leq t$ and A^i_j is *exactly 2-fold* if every contained event occurs exactly twice within A^i_j.

▶ **Lemma 12.** *A linear 2-fold TS A has the SSP if and only if A^i_j is not an exactly 2-fold subsequence for any $0 \leq i < j \leq t$.*

Proof. We reuse the simple observation that every region R of a linear TS A fulfills for all $0 \leq i < j \leq t$ that $\sum_{k=i}^{j} sig_R(e_k) = R(s_j) - R(s_i)$. Hence, if A has an exactly 2-fold subsequence A^i_j then every region R makes $R(s_j) - R(s_i)$ even, that is, $R(s_j) = R(s_i)$. This means, the two states are not separated by any region of A.

Reversely, assume A is free of exactly 2-fold subsequences and let $0 \leq i < j \leq t$. To see that s_i, s_j are separable, consider the three sequences A_i^0, A_j^i, A_t^j. If A_j^i contains an event that is unique in A then s_i, s_j are clearly separable. Otherwise, we select e_{min} from the events of A_j^i with first occurrence in A_i^0 such that the index i' of $s_{i'} \xrightarrow{e_{min}}$ is minimized. That is, we select the event e_{min} from A_j^i with leftmost first occurrence. If there is an event e in $A_i^{i'}$ that is unique in A or has its first occurrence in $A_{i'}^0$ or its second occurrence in A_t^j then a region R separating s_i, s_j is defined by $sig_R(e) = -sig_R(e_{min}) = 1$ while other events obey. If e does not exist, every event of $A_i^{i'}$ occurs twice in $A_j^{i'}$. In that case, we select e_{max} from the events of A_j^i with second occurrence in A_t^j such that the index j' of $s_{j'} \xrightarrow{e_{max}}$ is maximized. That is, we select the event e_{max} from A_j^i with rightmost second occurrence. If there is an event e in $A_{j'}^j$ that is unique in A or has its first occurrence in A_i^0 or its second occurrence in $A_t^{j'}$ then a region R separating s_i, s_j is defined by $sig_R(e) = -sig_R(e_{max}) = 1$ while other events obey. If e does not exist, every event of $A_{j'}^j$ occurs twice in $A_{j'}^i$.

But now, every event in A_j^i has a second occurrence in $A_{j'}^{i'}$ by the choice of e_{min} and e_{max}. Moreover, we have seen that every event in $A_i^{i'}$ and every event in $A_{j'}^j$ has its second occurrence in $A_{j'}^{i'}$. Hence, A has the exactly 2-fold subsequence $A_{j'}^{i'}$, a contradiction. ◀

For a proof of Theorem 11 it is now sufficient to understand that checking the linear 2-fold TS A for exactly 2-fold subsequences can be done by a straight forward algorithm in $O(t^3)$ time.

Moreover, the proof of Theorem 11 motivates an algorithm to efficiently compute separating regions of linear 2-fold TSs A. This algorithm uses a function $f(k)$ that, given an index $k \in \{0, \ldots, t-1\}$, returns the index of the second occurrence of event e_{k+1} that occurs at s_k or -1 if no second occurrence exists. Hence, for two states s_i and s_j we use only $O(t)$ calls to f to parse the sequence A_j^i for the indices i' and j' and to search the event e within $A_i^{i'}$, respectively $A_{j'}^j$. If e is not found, the algorithm denies the separability of s_i and s_j and otherwise, it returns the separating region given in the proof. The function f can be preprocessed as an array in at most $O(t \log t)$ time (depending on the representation of events). After the preprocessing, the algorithm runs in linear time $O(t)$.

7 Conclusion

With the present work on the SSP, the ESSP, and feasibility we consolidate the fact that ENS synthesis is a surprisingly difficult problem. While intractability has been known before when event manifoldness and state degree are limited to small constants, we show that even a tighter restriction of event manifoldness to $k = 2$ has no positive effect on the complexity. Bringing down intractability that close to trivial inputs, makes most considerations of restricting the TS graph structure futile and hampers other promising parameters. Consequently, our results rule out many straight forward approaches from *fixed parameter tractability*, too.

References

1 Alessandra Agostini and Giorgio De Michelis. Improving flexibility of workflow management systems. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management, Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2000. doi:10.1007/3-540-45594-9_14.

2 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. The synthesis problem for elementary net systems is np-complete. *Theor. Comput. Sci.*, 186(1-2):107–134, 1997. doi:10.1016/S0304-3975(96)00219-8.

3 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri Net Synthesis*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2015. doi:10.1007/978-3-662-47967-4.

4 Jordi Cortadella. Private correspondance, 2017.

5 Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Complete state encoding based on the theory of regions. In *2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '96), March 18-21, 1996, Aizu-Wakamatsu, Fukushima, Japan*, pages 36–47. IEEE Computer Society, 1996. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=3564, doi:10.1109/ASYNC.1996.494436.

6 August-Wilhelm Scheer. Business Process Engineering. Reference Models for Industrial Enterprises. Springer, 1994.

7 UML. Unified Modeling Language (UML). Object Management Group, 2018.

8 Kunihiko Hiraishi. Some complexity results on transition systems and elementary net systems. *Theor. Comput. Sci.*, 135(2):361–376, 1994. doi:10.1016/0304-3975(94)90112-0.

9 OMG. Business Process Model and Notation (BPMN). Object Management Group, 2018.

10 Cristopher Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete & Computational Geometry*, 26(4):573–590, 2001. doi:10.1007/s00454-001-0047-6.

11 Christian Rosenke and Ronny Tredup. The hardness of synthesizing elementary net systems from highly restricted inputs. *CoRR*, abs/1711.00220, 2017. arXiv:1711.00220.

12 Grzegorz Rozenberg and Joost Engelfriet. Elementary net systems. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 12–121. Springer, 1996. doi:10.1007/3-540-65306-6_14.

13 Vincent Schmitt. Flip-flop nets. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 517–528. Springer, 1996. doi:10.1007/3-540-60922-9_42.

14 Ronny Tredup, Christian Rosenke, and Karsten Wolf. Elementary net synthesis remains np-complete even for extremely simple inputs. In Victor Khomenko and Olivier H. Roux, editors, *Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings*, volume 10877 of *Lecture Notes in Computer Science*, pages 40–59. Springer, 2018. doi:10.1007/978-3-319-91268-4_3.

15 Alex Yakovlev and Albert Koelmans. Petri Nets and Digital Hardware Design. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*, pages 154–236. Springer, 1998. doi:10.1007/3-540-65307-4.

Up-To Techniques for Behavioural Metrics via Fibrations

Filippo Bonchi
Universitá di Pisa, Italy
filippo.bonchi@unipi.it

Barbara König
Universität Duisburg-Essen, Germany
barbara_koenig@uni-due.de

Daniela Petrişan
CNRS, IRIF, Université Paris Diderot, France
petrisan@irif.fr

Abstract

Up-to techniques are a well-known method for enhancing coinductive proofs of behavioural equivalences. We introduce up-to techniques for behavioural metrics between systems modelled as coalgebras and we provide abstract results to prove their soundness in a compositional way.

In order to obtain a general framework, we need a systematic way to lift functors: we show that the Wasserstein lifting of a functor, introduced in a previous work, corresponds to a change of base in a fibrational sense. This observation enables us to reuse existing results about soundness of up-to techniques in a fibrational setting. We focus on the fibrations of predicates and relations valued in a quantale, for which pseudo-metric spaces are an example. To illustrate our approach we provide an example on distances between regular languages.

2012 ACM Subject Classification Theory of computation → Concurrency, Theory of computation → Formal languages and automata theory, Theory of computation → Logic and verification

Keywords and phrases behavioural metrics, bisimilarity, up-to techniques, coalgebras, fibrations

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.17

Related Version A full version of the paper is available at [8], `https://arxiv.org/abs/1806.11064`.

Funding The first author acknowledges financial support from project ANR-16-CE25-0011 REPAS, the second author from DFG project BEMEGA, and the third author from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624).

Acknowledgements The authors are grateful to Shin-ya Katsumata, Henning Kerstan, Damien Pous and Paolo Baldan for precious suggestions and inspiring discussions.

1 Introduction

Checking whether two systems have an equivalent (or similar) behaviour is a crucial problem in computer science. In concurrency theory, one standard methodology for establishing behavioural equivalence of two systems is constructing a bisimulation relation between them. When the systems display a quantitative behaviour, the notion of behavioural equivalence is replaced with the more robust notion of behavioural metric [41, 14, 15].

Due to the sheer complexity of state-based systems, computing their behavioural equivalences and metrics can be very costly, therefore optimization techniques – the so called *up-to techniques* – have been developed to render these computations more efficient. These techniques found applications in various domains such as checking algorithms [10, 7], abstract interpretation [6] and proof assistants [13]. In the qualitative setting and in particular in concurrency, the theory of up-to techniques for bisimulations and various other coinductive predicates has been thoroughly studied [29, 33, 20]. On the other hand, in the quantitative setting, so far, only [12] has studied up-to techniques for behavioural metrics. However, the notion of up-to techniques therein and the accompanying theory of soundness are specific for probabilistic automata and are not instances of the standard lattice theoretic framework, which we will briefly recall next.

Suppose we want to verify whether two states in a system behave in the same way, (e.g. whether two states of an NFA accept the same language). The starting observation is that the relation of interest (e.g. behavioural equivalence or language equivalence) can be expressed as the greatest fixed point νb of a monotone function $b\colon \mathsf{Rel}_Q \to \mathsf{Rel}_Q$ on the complete lattice Rel_Q of relations on the state space Q of the system. Hence, in order to prove that two states x and y are behaviourally equivalent, i.e., $(x, y) \in \nu b$, it suffices to find a witness relation r which on one hand is a post-fixpoint of b, that is, $r \subseteq b(r)$ and on the other hand contains the pair (x, y). This is simply the coinduction proof principle. However, exhibiting such a witness relation r can be sometimes computationally expensive. In many situations this computation can be significantly optimized, if instead of computing a post-fixpoint of b one exhibits a relaxed invariant, that is a relation r such that $r \subseteq b(f(r))$ for a suitable function f. The function f is called a sound up-to technique when the proof principle

$$\frac{(x, y) \in r \quad r \subseteq b(f(r))}{(x, y) \in \nu b}$$

is valid. Establishing the soundness of up-to techniques on a case-by-case basis can be a tedious and sometimes delicate problem, see e.g. [28]. For this reason, several works [35, 31, 33, 20, 30, 32] have established a lattice-theoretic framework for proving soundness results in a modular fashion. The key notion is compatibility: for arbitrary monotone maps b and f on a complete lattice $(C, \leq)$, the up-to technique f is b-compatible iff $f \circ b \leq b \circ f$. Compatible techniques are sound and, most importantly, can be combined in several useful ways.

In this paper we develop a generic theory of up-to techniques for behavioural metrics applicable to different kinds of systems and metrics, which reuses established methodology. To achieve this we exploit the theory developed in [9] by modelling systems as *coalgebras* [34, 22] and behavioural metrics as coinductive predicates in a *fibration* [18]. In order to provide general soundness results, we need a principled way to lift functors from Set to metric spaces, a problem that has been studied in [19] and [3]. Our key observation is that these liftings arise from a change-of-base situation between $\mathcal{V}$-Rel and $\mathcal{V}$-Pred, namely the fibrations of relations, respectively predicates, valued over a quantale $\mathcal{V}$ (see Section 4 and 5).

In Section 6 we provide sufficient conditions ensuring the compatibility of basic quantitative up-to techniques, as well as proper ways to compose them. Interestingly enough, the conditions ensuring compatibility of the quantitative analogue of up-to reflexivity and up-to transitivity are subsumed by those used in [19] to extend monads to a bicategory of many-valued relations and generalize those in [3] (see the discussion after Theorem 21).

When the state space of a system is equipped with an algebraic structure, e.g. in process algebras, one can usually exploit this structure by reasoning up-to context. Assuming that the system forms a *bialgebra* [39, 26], i.e., that the algebraic structure distributes over the

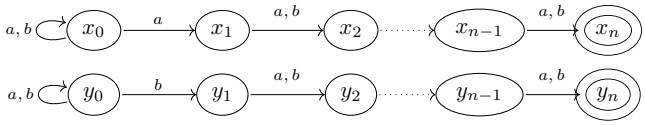


Figure 1 Example automaton.

coalgebraic behaviour as in GSOS specifications, we give sufficient conditions ensuring the compatibility of the quantitative version of contextual closure (Theorem 27).

In the qualitative setting, the sufficient conditions for compatibility are automatically met when taking as lifting the *canonical* relational one (see [9]). We show that the situation is similar in the quantitative setting for a certain notion of *quantitative canonical* lifting. In particular, up-to context is compatible for the canonical lifting under very mild assumptions (Theorem 30). As an immediate corollary we have that, in a bialgebra, syntactic contexts are *non-expansive* with respect to the behavioural metric induced by the canonical lifting. This property and weaker variants of it (such as non-extensiveness or uniform continuity), considered to be the quantitative analogue of behavioural equivalence being a congruence, have recently received considerable attention (see e.g. [15, 1, 38]).

To fix intuitions, Section 2 provides a motivating example, formally treated in Section 7. We conclude with a comparison to related work and a discussion of open problems in Section 8.

All proofs and additional material are provided in the full version of this paper [8].

2 Motivating example: distances between regular languages

Computing various distances (such as the edit-distance or Cantor metric) between strings, and more generally between regular languages or string distributions, has found various practical applications in various areas such as speech and handwriting recognition or computational biology. In this section we focus on a simple distance between regular languages, which we will call shortest-distinguishing-word-distance and is defined as $d_{\mathtt{sdw}}(L, K) = c^{|w|}$ – where w is the shortest word which belongs to exactly one of the languages L, K and c is a constant such that $0 < c < 1$.

As a running example, which will be formally explained in Section 7, we consider the non-deterministic finite automaton in Figure 1 and the languages accepted by the states x_0, respectively y_0. We can similarly define a distance on the states of an automaton as the aforementioned distance between the languages accepted by the two states. The inequality

$$d_{\mathtt{sdw}}(x_0, y_0) \leq c^n \qquad (\text{even } d_{\mathtt{sdw}}(x_0, y_0) = c^n) \tag{1}$$

holds in this example since no word of length smaller than n is accepted by either state. Note that computing this distance is PSPACE-hard since the language equivalence problem for non-deterministic automata can be reduced to it.

One way to show this is to determinize the automaton in Figure 1 and to use the fact that for deterministic automata the shortest-distinguishing-word-distance can be expressed as the greatest fixpoint for a monotone function. Indeed, for a finite deterministic automaton $(Q, (\delta_a \colon Q \to Q)_{a \in A}, F \subseteq Q)$ over a finite alphabet A, we have that $d_{\mathtt{sdw}} \colon Q \times Q \to [0, 1]$ is the greatest fixpoint of a function b defined on the complete lattice $[0, 1]^{Q \times Q}$ of functions ordered with the *reversed* pointwise order $\preceq$ and given by

$$b(d)(q_1, q_2) = \begin{cases} 1, \text{ if only one of } q_1, q_2 \text{ is in } F \\ \max\limits_{a \in A} c \cdot \{d(\delta_a(q_1), \delta_a(q_2))\}, \text{ otherwise .} \end{cases} \tag{2}$$

Notice that we use the reversed order on $[0,1]$, for technical reasons (see Section 4).

In order to prove (1) we can define a witness distance $\bar{d}$ on the states of the determinized automaton such that $\bar{d}(\{x_0\},\{y_0\}) \leq c^n$ and which is a post-fixpoint for b, i.e., $\bar{d} \preceq b(\bar{d})$. Notice that this would entail $\bar{d} \preceq d_{\mathtt{sdw}}$ and hence $d_{\mathtt{sdw}}(\{x_0\},\{y_0\}) \leq \bar{d}(\{x_0\},\{y_0\}) \leq c^n$.

This approach is problematic since the determinization of the automaton is of exponential size, so we have to define $\bar{d}$ for exponentially many pairs of sets of states. In order to mitigate the state space explosion we will use an up-to technique, which, just as up-to congruence in [10], exploits the join-semilattice structure of the state set $\mathcal{P}Q$ of the determinization of an NFA with state set Q. The crucial observation is the fact that given the states $Q_1, Q_2, Q_1', Q_2' \in \mathcal{P}Q$ in the determinization of an NFA, the following inference rule holds

$$\frac{d_{\mathtt{sdw}}(Q_1,Q_2) \leq r \qquad d_{\mathtt{sdw}}(Q_1',Q_2') \leq r}{d_{\mathtt{sdw}}(Q_1 \cup Q_1', Q_2 \cup Q_2') \leq r}$$

Based on this, we can define a monotone function f on $[0,1]^{\mathcal{P}Q \times \mathcal{P}Q}$ that closes a function d according to such proof rules, producing $f(d)$ such that $d \preceq f(d)$ (the formal definition of f is given in Section 7). The general theory developed in this paper allows us to show in Section 7 that f is a sound up-to technique, i.e., it is sufficient to prove $\bar{d} \preceq b(f(\bar{d}))$ in order to establish $\bar{d} \preceq d_{\mathtt{sdw}}$.

Using this technique it suffices to consider a quadratic number of pairs of sets of states in the example. In particular we define a function $\bar{d}\colon \mathcal{P}Q \times \mathcal{P}Q \to [0,1]$ as follows:

$$\bar{d}(\{x_i\},\{y_j\}) = c^{n-\max\{i,j\}}$$

and $\bar{d}(X_1,X_2) = 1$ for all other values. Note that this function is not a metric but rather, what we will call in Section 4, a relation valued in $[0,1]$.

It holds that $\bar{d}(\{x_0\},\{y_0\}) = c^n$. It remains to show that $\bar{d} \preceq b(f(\bar{d}))$. For this, it suffices to prove that

$$b(f(\bar{d}))(\{x_i\},\{y_j\}) \leq \bar{d}(\{x_i\},\{y_j\})\,.$$

For instance, when $i = j = 0$ we compute the sets of a-successors, which are $\{x_0,x_1\}$, $\{y_0\}$. We have that $\bar{d}(\{x_0\},\{y_0\}) = c^n \leq c^{n-1}$, $\bar{d}(\{x_0\},\{y_1\}) = c^{n-1}$ and using the up-to proof rule introduced above we obtain that $f(\bar{d})(\{x_0,x_1\},\{y_0\}) \leq c^{n-1}$. The same holds for the sets of b-successors and since x_0 and y_0 are both non-final we infer $b(f(\bar{d}))(\{x_0\},\{y_0\}) \leq c \cdot c^{n-1} = c^n = \bar{d}(\{x_0\},\{y_0\})$. The remaining cases (when $i \neq 0 \neq j$) are analogous.

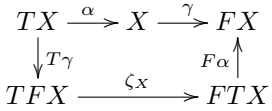
Our aim is to introduce such proof techniques for behavioural metrics, to make this kind of reasoning precise, not only for this specific example, but for coalgebras in general. Furthermore, we will not limit ourselves to metrics and distances, but we will consider more general relations valued in arbitrary quantales, of which the interval $[0,1]$ is an example.

3 Preliminaries

We recall here formal definitions for notions such as coalgebras, bialgebras or fibrations.

▶ **Definition 1.** A coalgebra for a functor $F\colon \mathcal{C} \to \mathcal{C}$, or an F-coalgebra is a morphism $\gamma\colon X \to FX$ for some object X of $\mathcal{C}$, referred to as the carrier of the coalgebra γ. A morphism between two coalgebras $\gamma\colon X \to FX$ and $\xi\colon Y \to FY$ is a morphism $f\colon X \to Y$ such that $\xi \circ f = Ff \circ \gamma$. Algebras for the functor F, or F-algebras, are defined dually as morphisms of the form $\alpha\colon FX \to X$.

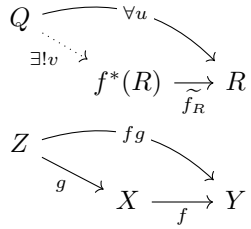
▶ **Definition 2.** Consider two functors F, T and a natural transformation $\zeta\colon TF \Rightarrow FT$. A bialgebra for ζ is a tuple (X, α, γ) such that $\alpha\colon TX \to X$ is a T-algebra, $\gamma\colon X \to FX$ is an F-coalgebra so that the diagram on the left commutes. We call ζ the distributive law of the bialgebra (X, α, γ), even when T is not a monad.



▶ **Example 3.** The determinization of an NFA can be seen as a bialgebra with $X = \mathcal{P}Q$, the algebra $\mu_Q\colon \mathcal{P}\mathcal{P}Q \to \mathcal{P}Q$ given by the multiplication of the powerset monad, a coalgebra for the functor $F(X) = 2 \times X^A$, and a distributive law $\zeta\colon \mathcal{P}F \to F\mathcal{P}$ defined for $M \subseteq 2 \times X^A$ by $\zeta_X(M) = (\bigvee_{(b,f)\in M} b, [a \mapsto \{f(a) \mid (b, f) \in M\}])$. See [37, 23] for more details.

We now introduce the notions of fibration and bifibration.

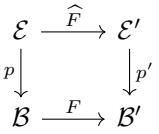
▶ **Definition 4.** A functor $p\colon \mathcal{E} \to \mathcal{B}$ is called a fibration when for every morphism $f\colon X \to Y$ in $\mathcal{B}$ and every R in $\mathcal{E}$ with $p(R) = Y$ there exists a map $\widetilde{f_R}\colon f^*(R) \to R$ such that $p(\widetilde{f_R}) = f$, satisfying the following universal property:
For all maps $g : Z \to X$ in $\mathcal{B}$ and $u\colon Q \to R$ in $\mathcal{E}$ sitting above fg (i.e., $p(u) = fg$) there is a unique map $v\colon Q \to f^*(R)$ such that $u = \widetilde{f_R}v$ and $p(v) = g$.
For X in $\mathcal{B}$ we denote by $\mathcal{E}_X$ the fibre above X, i.e., the subcategory of $\mathcal{E}$ with objects mapped by p to X and arrows sitting above the identity on X.



A map $\widetilde{f}$ as above is called a Cartesian lifting of f and is unique up to isomorphism. If we make a choice of Cartesian liftings, the association $R \mapsto f^*(R)$ gives rise to the so-called reindexing functor $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$. In what follows we will only consider split fibrations, that is, the Cartesian liftings are chosen such that we have $(fg)^* = g^*f^*$.

A functor $p\colon \mathcal{E} \to \mathcal{B}$ is called a bifibration if both $p\colon \mathcal{E} \to \mathcal{B}$ and $p^{op}\colon \mathcal{E}^{op} \to \mathcal{B}^{op}$ are fibrations. Interestingly, a fibration is a bifibration if and only if each reindexing functor $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$ has a left adjoint $\Sigma_f \dashv f^*$, see [21, Lemma 9.1.2]. We will call the functors Σ_f direct images along f.

Two important examples of bifibrations are those of relations over sets, $p\colon \mathsf{Rel} \to \mathsf{Set}$, and of predicates over sets, $p\colon \mathsf{Pred} \to \mathsf{Set}$, which played a crucial role in [9]. We do not recall their exact definitions here, as they arise as instances of the more general bifibrations of quantale-valued relations and predicates described in detail in the next section.



Given fibrations $p\colon \mathcal{E} \to \mathcal{B}$ and $p'\colon \mathcal{E}' \to \mathcal{B}'$ and a functor on the base categories $F\colon \mathcal{B} \to \mathcal{B}'$, we call $\widehat{F}\colon \mathcal{E} \to \mathcal{E}'$ a lifting of F when $p'\widehat{F} = Fp$. Notice that a lifting $\widehat{F}$ restricts to a functor between the fibres $\widehat{F}_X\colon \mathcal{E}_X \to \mathcal{E}'_{FX}$. We omit the subscript X when it is clear from the context.

Consider an arbitrary lifting $\widehat{F}$ of F and a morphism $f\colon X \to Y$ in $\mathcal{B}$. For any $R \in \mathcal{E}_Y$ the maps $\widetilde{Ff}_{\widehat{F}R}\colon (Ff)^*(\widehat{F}R) \to \widehat{F}R$ and $\widehat{F}(\widetilde{f_R})\colon \widehat{F}(f^*R) \to \widehat{F}R$ sit above Ff. Using the universal property in Definition 4, we obtain a canonical morphism

$$\widehat{F} \circ f^*(R) \to (Ff)^* \circ \widehat{F}(R)\,. \tag{3}$$

A lifting $\widehat{F}$ is called a fibred lifting when the natural transformation in (3) is an isomorphism.

4 Moving towards a quantitative setting

We start by introducing two fibrations which are the foundations for our quantitative reasoning: predicates and relations valued in a quantale.

▶ **Definition 5.** A quantale $\mathcal{V}$ is a complete lattice equipped with an associative operation $\otimes : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ which is distributive on both sides over arbitrary joins $\bigvee$.

This implies that for every $y \in \mathcal{V}$ the functor $- \otimes y$ has a right adjoint $[y, -]$. Similarly, for every $x \in \mathcal{V}$, the functor $x \otimes -$ has a right adjoint, denoted by $[\![x, -]\!]$. Thus, for every $x, y, z \in \mathcal{V}$, we have: $x \otimes y \leq z \iff x \leq [y, z] \iff y \leq [\![x, z]\!]$.

If $\otimes$ has an identity element or unit 1 for $\otimes$ the quantale is called unital. If $x \otimes y = y \otimes x$ for every $x, y \in \mathcal{V}$ the quantale is called commutative and we have $[x, -] = [\![x, -]\!]$. Hereafter, we only work with unital, commutative quantales.

▶ **Example 6.** The Boolean algebra 2 with $\otimes = \wedge$ is a unital and commutative quantale: the unit is 1 and $[y, z] = y \to z$. The complete lattice $[0, \infty]$ ordered by the reversed order[1] of the reals, i.e., $\leq = \geq_{\mathbb{R}}$ and with $\otimes = +$ is a unital commutative quantale: the unit is 0 and for every $y, z \in [0, \infty]$ we have $[y, z] = z \dot{-} y$ (truncated subtraction). Also $[0, 1]$ is a unital quantale where $r \otimes s = \min(r + s, 1)$ (truncated addition).

▶ **Definition 7.** Given a set X and a quantale $\mathcal{V}$, a $\mathcal{V}$-valued predicate on X is a map $p : X \to \mathcal{V}$. A $\mathcal{V}$-valued relation on X is a map $r : X \times X \to \mathcal{V}$.

Given two $\mathcal{V}$-valued predicates $p, q : X \to \mathcal{V}$, we say that $p \leq q \iff \forall x \in X.\ p(x) \leq q(x)$.

▶ **Definition 8.** A morphism between $\mathcal{V}$-valued predicates $p : X \to \mathcal{V}$ and $q : Y \to \mathcal{V}$ is a map $f : X \to Y$ such that $p \leq q \circ f$. We consider the category $\mathcal{V}\text{-}\mathsf{Pred}$ whose objects are $\mathcal{V}$-valued predicates and arrows are as above.

▶ **Definition 9.** A morphism between $\mathcal{V}$-valued relations $r : X \times X \to \mathcal{V}$ and $q : Y \times Y \to \mathcal{V}$ is a map $f : X \to Y$ such that $p \leq q \circ (f \times f)$. We consider the category $\mathcal{V}\text{-}\mathsf{Rel}$ whose objects are $\mathcal{V}$-valued relations and arrows are as above.

The bifibration of $\mathcal{V}$-valued predicates. The forgetful functor $\mathcal{V}\text{-}\mathsf{Pred} \to \mathsf{Set}$ mapping a predicate $p : X \to \mathcal{V}$ to X is a bifibration. The fibre $\mathcal{V}\text{-}\mathsf{Pred}_X$ is the lattice of $\mathcal{V}$-valued predicates on X. For $f : X \to Y$ in Set the reindexing and direct image functors on a predicate $p \in \mathcal{V}\text{-}\mathsf{Pred}_Y$ are given by

$$f^*(p) = p \circ f \qquad \text{and} \qquad \Sigma_f(p)(y) = \bigvee \{p(x) \mid x \in f^{-1}(y)\}\,.$$

The bifibration of $\mathcal{V}$-valued relations. Notice that we have the following pullback in Cat, where $\Delta X = X \times X$. This is a change-of-base situation and thus the functor $\mathcal{V}\text{-}\mathsf{Rel} \to \mathsf{Set}$ mapping each $\mathcal{V}$-valued relation to its underlying set is also a bifibration. We denote by $\mathcal{V}\text{-}\mathsf{Rel}_X$ the fibre above a set X. For each set X the functor ι restricts to an isomorphism $\iota_X : \mathcal{V}\text{-}\mathsf{Rel}_X \to \mathcal{V}\text{-}\mathsf{Pred}_{X \times X}$.

$$\begin{array}{ccc} \mathcal{V}\text{-}\mathsf{Rel} & \xrightarrow{\iota} & \mathcal{V}\text{-}\mathsf{Pred} \\ \downarrow & \lrcorner & \downarrow \\ \mathsf{Set} & \xrightarrow[\Delta]{} & \mathsf{Set} \end{array}$$

For $f : X \to Y$ in Set the reindexing and direct image on $p \in \mathcal{V}\text{-}\mathsf{Rel}_Y$ are given by

$$f^*(p) = p \circ (f \times f) \qquad \text{and} \qquad \Sigma_f(p)(y) = \bigvee \{p(x, x') \mid (x, x') \in (f \times f)^{-1}(y, y')\}\,.$$

For two relations $p, q \in \mathcal{V}\text{-}\mathsf{Rel}_X$, we define their composition $p \cdot q \colon X \times X \to \mathcal{V}$ by $p \cdot q(x, y) = \bigvee \{p(x, z) \otimes q(z, y) \mid z \in X\}$. We define the diagonal relation $diag_X \in \mathcal{V}\text{-}\mathsf{Rel}_X$ by $diag_X(x, y) = 1$ if $x = y$ and $\bot$ otherwise.

[1] To avoid confusion we use $\vee, \wedge$ in the quantale and $\inf, \sup$ in the reals.

▶ **Definition 10.** We say that a $\mathcal{V}$-valued relation $r : X \times X \to \mathcal{V}$ is

- reflexive if for all $x \in X$ we have $r(x, x) \geq 1$, (i.e., $r \geq diag_X$);
- transitive if $r \cdot r \leq r$;
- symmetric if $r = r \circ \mathsf{sym}_X$, where $\mathsf{sym}_X : X \times X \to X \times X$ is the symmetry isomorphism.

We denote by $\mathcal{V}$-Cat the full subcategory of $\mathcal{V}$-Rel consisting of reflexive, transitive relations and by $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ the full subcategory of $\mathcal{V}$-Rel that are additionally symmetric.

Note that $\mathcal{V}$-Cat is the category of small categories enriched over $\mathcal{V}$ in the sense of [25].

▶ **Example 11.** For $\mathcal{V} = 2$, $\mathcal{V}$-valued relations are just relations. Reflexivity, transitivity and symmetry coincide with the standard notions, so $\mathcal{V}$-Cat is the category of preorders, while $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ is the category of equivalence relations.

For $\mathcal{V} = [0, \infty]$, $\mathcal{V}$-Cat is the category of generalized metric spaces à la Lawvere [27] (i.e., directed pseudo-metrics and non-expansive maps), while $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ is the one of pseudo-metrics.

5 Lifting functors to $\mathcal{V}$-Pred and $\mathcal{V}$-Rel

In the previous section, we have introduced the fibrations of interest for quantitative reasoning. In order to deal with coinductive predicates in this setting, it is convenient to have a structured way to lift Set-functors to $\mathcal{V}$-valued predicates and relations, and eventually to $\mathcal{V}$-enriched categories. Our strategy is to first lift functors to $\mathcal{V}$-Pred and then, by exploiting the change of base, move these liftings to $\mathcal{V}$-Rel. A comparison with the extensions of Set-monads to the bicategory of $\mathcal{V}$-matrices [19] is provided in Section 8.

5.1 $\mathcal{V}$-predicate liftings

Liftings of Set-functors to the category Pred (for $\mathcal{V} = 2$) of predicates have been widely studied in the context of coalgebraic modal logic, as they correspond to modal operators (see e.g. [36]). For $\mathcal{V}$-Pred, we proceed in a similar way. Let us analyse what it means to have a fibred lifting $\widehat{F}$ to $\mathcal{V}$-Pred of an endofunctor F on Set. First, recall that the fibre $\mathcal{V}$-Pred_X is just the preorder $\mathcal{V}^X$. So the restriction $\widehat{F}_X$ to such a fibre corresponds to a *monotone* map $\mathcal{V}^X \to \mathcal{V}^{FX}$. The fact that $\widehat{F}$ is a fibred lifting essentially means that the maps $(\mathcal{V}^X \to \mathcal{V}^{FX})_X$ form a natural transformation between the contravariant functors $\mathcal{V}^-$ and $\mathcal{V}^{F-}$. Furthermore, by Yoneda lemma we know that natural transformations $\mathcal{V}^- \Rightarrow \mathcal{V}^{F-}$ are in one-to-one correspondence with maps $ev : F\mathcal{V} \to \mathcal{V}$, which we will call hereafter evaluation maps. One can characterise the evaluation maps which correspond to the *monotone* natural transformations. These are the monotone evaluation maps $ev : (F\mathcal{V}, \ll) \to (\mathcal{V}, \leq)$ with respect to the usual order $\leq$ on $\mathcal{V}$ and an order $\ll$ on $F\mathcal{V}$ defined by applying the standard canonical relation lifting of F to $\leq$.

▶ **Proposition 12.** *There is a one-to-one correspondence between*

- *fibred liftings* $\widehat{F}$ *of* F *to* $\mathcal{V}$-Pred,
- *monotone natural transformations* $\mathcal{V}^- \Rightarrow \mathcal{V}^{F-}$,
- *monotone evaluation maps* $ev : F\mathcal{V} \to \mathcal{V}$.

Notice that the correspondence between fibred liftings and monotone evaluation maps is given in one direction by $ev = \widehat{F}(id_{\mathcal{V}})$, and conversely, by $\widehat{F}(p\colon X \to \mathcal{V}) = ev \circ F(p)$.

Evaluation maps as Eilenberg-Moore algebras. Evaluation maps have also been extensively considered in the coalgebraic approach to modal logics [36]. A special kind of evaluation map arises when the truth values $\mathcal{V}$ have an algebraic structure for a given monad (T, μ, η), that is, we have $\mathcal{V} = T\Omega$ for some object Ω and the evaluation map $T\mathcal{V} \to \mathcal{V}$ is an Eilenberg-Moore algebra for T. This notion of monadic modality has been studied in [17] where the category of free algebras for T was assumed to be order enriched. Under reasonable assumptions the evaluation map obtained as the free Eilenberg-Moore algebra on Ω (i.e., $ev\colon T\mathcal{V} \to \mathcal{V}$ is just $\mu_\Omega\colon T^2\Omega \to T\Omega$) is a monotone evaluation map, and hence gives rise to a fibred lifting of T (see [8] for more details.)

We provide next several examples of monotone evaluation maps which arise in this fashion.

▶ **Example 13.** When T is the powerset monad $\mathcal{P}$ and $\Omega = 1$ we obtain $\mathcal{V} = 2$ and $\mu_1\colon \mathcal{P}2 \to 2$ corresponds to the $\Diamond$ modality, i.e., to an existential predicate transformer, see [17].

▶ **Example 14.** When T is the probability distribution functor $\mathcal{D}$ on Set and $\Omega = 2 = \{0, 1\}$ equipped with the order $1 \sqsubseteq 0$ we obtain $\mathcal{V} = \mathcal{D}\{0, 1\} \cong [0, 1]$ with the reversed order of the reals, i.e., $\leq = \geq_{\mathbb{R}}$. In this case $ev_{\mathcal{D}}(f) = \sum_{r\in[0,1]} r \cdot f(r)$ for $f\colon [0, 1] \to [0, 1]$ a probability distribution (expectation of the identity random variable).

The canonical evaluation map. In the case $\mathcal{V} = 2$, there exists a simple way of lifting a functor $F\colon \mathsf{Set} \to \mathsf{Set}$: given a predicate $p\colon U \rightarrowtail X$, one defines the canonical predicate lifting $\widehat{F}_{\mathsf{can}}(U)$ of F as the epi-mono factorization of $Fp\colon FU \to FX$. This lifting corresponds to a canonical evaluation map $\mathsf{true}\colon 1 \to 2$ which maps the unique element of 1 into the element 1 of the quantale 2. For $\mathcal{V}$-relations, a generalized notion of canonical evaluation map was introduced in [19]. For $r \in \mathcal{V}$ consider the subset $\uparrow r = \{v \in \mathcal{V} \mid v \geq r\}$ and write $\mathsf{true}_r\colon \uparrow r \to \mathcal{V}$ for the inclusion. Given $u \in F\mathcal{V}$ we write $u \in F(\uparrow r)$ when u is in the image of the injective function $F(\mathsf{true}_r)$. Following [19], we define $ev_{\mathsf{can}} : F\mathcal{V} \to \mathcal{V}$ as follows:

$$ev_{\mathsf{can}}(u) = \bigvee\{r \mid u \in F(\uparrow r)\}.$$

▶ **Example 15.** Assume F is the powerset functor $\mathcal{P}$ and let $u \in \mathcal{P}(\mathcal{V})$. We obtain that

$$ev_{\mathsf{can}}(u) = \bigvee\{r \mid u \subseteq \uparrow r\}, \text{ or equivalently, } ev_{\mathsf{can}}(u) = \bigwedge u\,.$$

When $\mathcal{V} = 2$ we obtain $ev_{\mathsf{can}}\colon \mathcal{P}2 \to 2$ given by $ev_{\mathsf{can}}(u) = 1$ iff $u = \emptyset$ or $u = \{1\}$. This corresponds to the $\Box$ operator from modal logic. If $\mathcal{V} = [0, \infty]$ we have $ev_{\mathsf{can}}(u) = \sup u$.

▶ **Example 16.** The canonical evaluation map for the distribution monad $\mathcal{D}$ and $\mathcal{V} = [0, 1]$ is $ev_{\mathsf{can}}(f) = \sup_{r\in[0,1]} f(r)$, which is not the monad multiplication.

The canonical evaluation map ev_{can} is monotone whenever the functor F preserves weak pullbacks (see [8]). For such functors, by Proposition 12, the map ev_{can} induces a fibred lifting $\widehat{F}_{\mathsf{can}}$ of F, called the canonical $\mathcal{V}$-Pred-lifting of F and defined by

$$\widehat{F}_{\mathsf{can}}(p)(u) = \bigvee\{r \mid F(p)(u) \in F(\uparrow r)\} \quad \text{for } p \in \mathcal{V}\text{-}\mathsf{Pred}_X \text{ and } u \in FX\,.$$

5.2 From predicates to relations via Wasserstein

We describe next how functor liftings to $\mathcal{V}$-Rel can be systematically obtained using the change-of-base situation described above. In particular, we see how the Wasserstein metric between probability distributions (defined in terms of couplings of distributions) can be naturally modelled in the fibrational setting.

Consider a $\mathcal{V}$-predicate lifting $\widehat{F}$ of a Set-functor F. A natural way to lift F to $\mathcal{V}$-relations using $\widehat{F}$ is to regard a $\mathcal{V}$-relation $r\colon X \times X \to \mathcal{V}$ as a $\mathcal{V}$-predicate on the product $X \times X$. Formally, we will use the isomorphism ι_X described in Section 4. We can apply the functor $\widehat{F}$ to the predicate $\iota_X(r)$ in order to obtain the predicate $\widehat{F} \circ \iota_X(r)$ on the set $F(X \times X)$. Ideally, we would want to transform this predicate into a relation on FX. So first, we have to transform it into a predicate on $FX \times FX$. To this end, we use the natural transformation

$$\lambda^F\colon F \circ \Delta \Rightarrow \Delta \circ F \text{ defined by } \lambda^F_X = \langle F\pi_1, F\pi_2\rangle\colon F(X \times X) \to FX \times FX\,. \tag{4}$$

We drop the superscript and simply write λ when the functor F is clear from the context. Additionally, the bifibrational structure of $\mathcal{V}$-Rel plays a crucial role, as we can use the direct image functor Σ_{λ_X} to transform $\widehat{F} \circ \iota_X(r)$ into a predicate on $FX \times FX$. Putting all the pieces together, we define a lifting of F on the fibre $\mathcal{V}$-Rel_X as the composite $\overline{F}_X$ given by:

$$\overline{F}_X\colon \mathcal{V}\text{-}\mathsf{Rel}_X \xrightarrow{\iota_X} \mathcal{V}\text{-}\mathsf{Pred}_{\Delta X} \xrightarrow{\widehat{F}_{\Delta X}} \mathcal{V}\text{-}\mathsf{Pred}_{F\Delta X} \xrightarrow{\Sigma_{\lambda_X}} \mathcal{V}\text{-}\mathsf{Pred}_{\Delta FX} \xrightarrow{{\iota_{FX}}^{-1}} \mathcal{V}\text{-}\mathsf{Rel}_{FX} \tag{5}$$

The aim is to define a lifting $\overline{F}$ of F to $\mathcal{V}$-Rel. The above construction provides the definition of $\overline{F}$ on the fibres and, in particular, on the objects of $\mathcal{V}$-Rel. For a morphism between $\mathcal{V}$-relations $p \in \mathcal{V}\text{-}\mathsf{Rel}_X$ and $q \in \mathcal{V}\text{-}\mathsf{Rel}_Y$, i.e., a map $f\colon X \to Y$ such that $p \leq f^*(q)$, we define $\overline{F}(f)$ as the map $Ff\colon FX \to FY$. To see that this is well defined it remains to show that $\overline{F}p \leq (Ff)^*(\overline{F}q)$. This is the first part of the next proposition.

▶ **Proposition 17.** *The functor $\overline{F}$ defined above is a well defined lifting of F to $\mathcal{V}$-Rel. Furthermore, when F preserves weak pullbacks and $\widehat{F}$ is a fibred lifting of F to $\mathcal{V}$-Pred, then $\overline{F}$ is a fibred lifting of F to $\mathcal{V}$-Rel.*

Spelling out the concrete description of the direct image functor and of λ_X, we obtain for a relation $p \in \mathcal{V}\text{-}\mathsf{Rel}_X$ and $t_1, t_2 \in FX$, that

$$\overline{F}(p)(t_1, t_2) = \bigvee\{\widehat{F}(p)(t) \mid t \in F(X \times X), F\pi_i(t) = t_i\} \tag{6}$$

Unravelling the definition of $\widehat{F}(p)(t) = ev \circ F(p)$, we obtain for $\overline{F}(p)$ the same formula as for the extension of F on $\mathcal{V}$-matrices, as given in [19, Definition 3.4]. This definition in [19] is obtained by a direct generalisation of the Barr extensions of Set-functors to the bicategory of relations. In contrast, we obtained (6) by exploiting the fibrational change-of-base situation and by first considering a $\mathcal{V}$-Pred-lifting.

We call a lifting of the form $\overline{F}$ the Wasserstein lifting of F corresponding to $\widehat{F}$. This terminology is motivated by the next example.

▶ **Example 18.** When $F = \mathcal{D}$ (the distribution functor), $\mathcal{V} = [0, 1]$ and ev_F is as in Example 14 then $\overline{F}$ is the original Wasserstein metric from transportation theory [42], which – by the Kantorovich-Rubinstein duality – is the same as the Kantorovich metric. Here we compare two probability distributions $t_1, t_2 \in \mathcal{D}X$ and obtain as a result the coupling $t \in \mathcal{D}(X \times X)$ with marginal distributions t_1, t_2, giving us the optimal plan to transport the "supply" t_1 to the "demand" t_2. More concretely, given a metric $d\colon X \times X \to \mathcal{V}$, the (discrete) Wasserstein metric is defined as

$$d^W(t_1, t_2) = \inf\{\sum_{x,y \in X} d(x, y) \cdot t(x, y) \mid \sum_y t(x, y) = t_1(x), \sum_x t(x, y) = t_2(y)\}.$$

On the other hand, when ev_F is the canonical evaluation map of Example 16 the corresponding $\mathcal{V}$-Rel-lifting $\overline{F}$ minimizes the longest distance (and hence the required time) rather than the total cost of transport.

▶ **Example 19.** Let us spell out the definition when $F = \mathcal{P}$ (powerset functor), $\mathcal{V} = [0,1]$ and $ev_F \colon \mathcal{P}[0,1] \to [0,1]$ corresponds to sup, which is clearly monotone and is the canonical evaluation map as in Example 15.

Then, given a metric $d \colon X \times X \to [0,1]$ and $X_1, X_2 \subseteq X$, the lifted metric is defined as follows (remember that the order is reversed on $[0,1]$):

$$\overline{F}(d)(X_1, X_2) = \inf\{\sup d[Y] \mid Y \subseteq X \times X, \pi_i[Y] = X_i\}$$

As explained in [5], this is the same as the Hausdorff metric d^H defined by:

$$d^H(X_1, X_2) = \sup\{\sup_{x_1 \in X_1} \inf_{x_2 \in X_2} d(x_1, x_2), \sup_{x_2 \in X_2} \inf_{x_1 \in X_1} d(x_1, x_2)\}$$

The next lemma establishes that this construction is functorial: liftings of natural transformations to $\mathcal{V}$-Pred can be converted into liftings of natural transformations between the corresponding Wasserstein liftings on $\mathcal{V}$-Rel.

▶ **Lemma 20.** *If there exists a lifting $\widehat{\zeta} \colon \widehat{F} \Rightarrow \widehat{G}$ of a natural transformation $\zeta \colon F \Rightarrow G$, then there exists a lifting $\overline{\zeta} \colon \overline{F} \Rightarrow \overline{G}$ between the corresponding Wasserstein liftings. Furthermore, when $\widehat{F}$ and $\widehat{G}$ correspond to monotone evaluation maps ev_F and ev_G, then the lifting $\widehat{\zeta}$ exists and is unique if and only if $ev_F \leq ev_G \circ \zeta_{\mathcal{V}}$.*

For $\mathcal{V} = [0, \infty]$, one is also interested in lifting functors to the category of (generalized) pseudo-metric spaces, not just of $[0,\infty]$-valued relations. This motivates the next question: when does the lifting $\overline{F}$ restrict to a functor on $\mathcal{V}$-Cat and $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$? We have the following characterization theorem, where $\kappa_X \colon X \to \mathcal{V}$ is the constant function $x \mapsto 1$ and $u \otimes v \colon X \to \mathcal{V}$ denotes the pointwise tensor of two predicates $u, v \colon X \to \mathcal{V}$, i.e., $(u \otimes v)(x) = u(x) \otimes v(x)$.

▶ **Theorem 21.** *Assume $\widehat{F}$ is a lifting of F to $\mathcal{V}$-Pred and $\overline{F}$ is the corresponding $\mathcal{V}$-Rel Wasserstein lifting. Then*

- *If $\widehat{F}(\kappa_X) \geq \kappa_{FX}$ then $\overline{F}(diag_X) \geq diag_{FX}$, hence $\overline{F}$ preserves reflexive relations;*
- *If $\widehat{F}$ is a fibred lifting, F preserves weak pullbacks and $\widehat{F}(p \otimes q) \geq \widehat{F}(p) \otimes \widehat{F}(q)$ then $\overline{F}(p \cdot q) \geq \overline{F}(p) \cdot \overline{F}(q)$, hence $\overline{F}$ preserves transitive relations;*
- *$\overline{F}$ preserves symmetric relations.*

Consequently, when all the above hypotheses are satisfied, then the corresponding $\mathcal{V}$-Rel Wasserstein lifting $\overline{F}$ restricts to a lifting of F to both $\mathcal{V}$-Cat and $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$.

For $\mathcal{V} = [0, \infty]$, the first condition of Theorem 21 is a relaxed version of a condition in [5, Definition 5.14] used to guarantee reflexivity. The second condition (for transitivity) is equivalent to a non-symmetric variant of a condition in [5] (see [8]).

We can establish generic sufficient conditions on a monotone evaluation map ev so that the corresponding $\mathcal{V}$-Pred-lifting $\widehat{F}$ satisfies the conditions of Theorem 21. In [8] we show that $\widehat{F}(p \otimes q) \geq \widehat{F}(p) \otimes \widehat{F}(q)$ holds whenever the map $\otimes \colon \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ is the carrier of a lax morphism in the category of F-algebras between $(\mathcal{V}, ev)^2 \to (\mathcal{V}, ev)$, i.e., $\otimes \circ (ev \times ev) \circ \lambda_{\mathcal{V}} \leq ev \circ F(\otimes)$. Furthermore, $\widehat{F}(\kappa_X) \geq \kappa_X$ holds whenever the map $\kappa_{\mathbb{1}} \colon \mathbb{1} \to \mathcal{V}$ is the carrier of a lax morphism from the one-element F-algebra $! \colon F\mathbb{1} \to \mathbb{1}$ to $(\mathcal{V}, ev)$, i.e., $\kappa_{\mathbb{1}} \circ ! \leq ev \circ F\kappa_{\mathbb{1}}$. These two requirements correspond to the conditions $(Q_\otimes)$, respectively (Q_k) satisfied by a topological theory in the sense of [19, Definition 3.1]. Since these two are satisfied by the canonical evaluation map ev_{can},[2] we immediately obtain

[2] The same observation is present in [19, Theorem 3.3(b)] but in a slightly different setting.

▶ **Proposition 22.** *Whenever F preserves weak pullbacks the canonical lifting $\widehat{F}_{\mathsf{can}}$ satisfies the conditions in Theorem 21:*
1. $\widehat{F}_{\mathsf{can}}(p \otimes q) \geq \widehat{F}_{\mathsf{can}}(p) \otimes \widehat{F}_{\mathsf{can}}(q)$, *for all* $p, q \in \mathcal{V}\text{-}\mathsf{Pred}_X$,
2. $\widehat{F}_{\mathsf{can}}(\kappa_X) \geq \kappa_X$.

An immediate consequence of Proposition 22 and of Theorem 21 is that the Wasserstein lifting $\overline{F}_{\mathsf{can}}$ that corresponds to $\widehat{F}_{\mathsf{can}}$ restricts to a lifting of F to both $\mathcal{V}\text{-}\mathsf{Cat}$ and $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$.

6 Quantitative up-to techniques

The fibrational constructions of the previous section provides a convenient setting to develop an abstract theory of quantitative up-to techniques. The coinductive object of interest is the greatest fixpoint of a monotone map b on $\mathcal{V}\text{-}\mathsf{Rel}$, hereafter denoted by νb. Recall that an up-to technique, namely a monotone map f on $\mathcal{V}\text{-}\mathsf{Rel}$, is sound whenever $d \leq b(f(d))$ implies $d \leq \nu b$, for all $d \in \mathcal{V}\text{-}\mathsf{Rel}_X$; it is compatible if $f \circ b \leq b \circ f$ in the pointwise order. It is well-known that compatibility entails soundness. Another useful property is:

$$\text{if } f \text{ is compatible, then } f(\nu b) \leq \nu b\,. \tag{7}$$

Following [9], we assume hereafter that b can be seen as the composite

$$b\colon \mathcal{V}\text{-}\mathsf{Rel}_X \xrightarrow{\overline{F}} \mathcal{V}\text{-}\mathsf{Rel}_{FX} \xrightarrow{\xi^*} \mathcal{V}\text{-}\mathsf{Rel}_X\,. \tag{8}$$

where $\xi\colon X \to FX$ is some coalgebra for $F\colon \mathsf{Set} \to \mathsf{Set}$. When F admits a final coalgebra $\omega\colon \Omega \to F\Omega$, the unique morphism $!\colon X \to \Omega$ induces the behavioural closure up-to technique

$$bhv\colon \mathcal{V}\text{-}\mathsf{Rel}_X \xrightarrow{\Sigma_!} \mathcal{V}\text{-}\mathsf{Rel}_\Omega \xrightarrow{!^*} \mathcal{V}\text{-}\mathsf{Rel}_X \tag{9}$$

where $bhv(p)(x,y) = \bigvee\{p(x',y') \mid !(x) = !(x') \text{ and } !(y) = !(y')\}$. For $\mathcal{V} = 2$, behavioural closure corresponds to the usual up-to behavioural equivalence (bisimilarity). Other immediate generalisations are the up-to reflexivity (*ref*), up-to transitivity (*trn*) and up-to symmetry (*sym*) techniques. Whenever $\overline{F}$ is obtained through the Wasserstein construction of some $\widehat{F}$ satisfying the conditions of Theorem 21, these techniques are compatible (see [8] for more details).

As usual, compatible techniques can be combined either by function composition ($\circ$) or by arbitrary joins ($\bigvee$). For instance compatibility of up-to metric closure, defined as the composite $mtr = trn \circ sym \circ ref$ follows immediately from compatibility of *trn*, *sym* and *ref*. In $\mathcal{V}\text{-}\mathsf{Rel}$ there is yet another useful way to combine up-to techniques – called *chaining* in [12] – and defined as the composition ($\cdot$) of relations.

▶ **Proposition 23.** *Let $f_1, f_2\colon \mathcal{V}\text{-}\mathsf{Rel}_X \to \mathcal{V}\text{-}\mathsf{Rel}_X$ be compatible with respect to $b\colon \mathcal{V}\text{-}\mathsf{Rel}_X \to \mathcal{V}\text{-}\mathsf{Rel}_X$. If $\overline{F}(p \cdot q) \geq \overline{F}(p) \cdot \overline{F}(q)$ for all $p, q \in \mathcal{V}\text{-}\mathsf{Rel}_X$, then $f_1 \cdot f_2$ is b-compatible.*

In the reminder of this section, we focus on quantitative generalizations of the up-to contextual closure technique, which given an algebra $\alpha\colon TX \to X$, is seen as the composite:

$$f\colon \mathcal{V}\text{-}\mathsf{Rel}_X \xrightarrow{\overline{T}} \mathcal{V}\text{-}\mathsf{Rel}_{TX} \xrightarrow{\Sigma_\alpha} \mathcal{V}\text{-}\mathsf{Rel}_X\,. \tag{10}$$

▶ **Example 24.** Consider a signature Σ and the algebra of Σ-terms with variables in X $\mu_X\colon T_\Sigma T_\Sigma X \to T_\Sigma X$. The contextual closure $ctx\colon \mathcal{V}\text{-}\mathsf{Rel}_{T_\Sigma X} \to \mathcal{V}\text{-}\mathsf{Rel}_{T_\Sigma X}$ is defined as in (10) by taking the canonical lifting of the functor T_Σ. For all $t_1, t_2 \in T_\Sigma X$ and $d \in \mathcal{V}\text{-}\mathsf{Rel}_{T_\Sigma X}$

$$ctx(d)(t_1,t_2) = \bigvee_C \{\bigwedge_j d(s^1_j, s^2_j) \mid t_i = C(s^i_0, \ldots, s^i_n)\}$$

where C ranges over arbitrary contexts and s^i_j over terms. Notice that for $\mathcal{V} = 2$, this boils down to the usual notion of contextual closure of a relation. Details can be found in [8].

▶ **Example 25.** Let $\mathcal{V} = [0,1]$. In [12], the convex closure of $d \in \mathcal{V}\text{-}\mathsf{Rel}_{\mathcal{D}(X)}$ is defined for $\Delta, \Theta \in \mathcal{D}(X)$ as

$$cvx(d)(\Delta,\Theta) = \inf\{\sum_i p_i \cdot d(\Delta_i,\Theta_i) \mid \Delta = \sum_i p_i \cdot \Delta_i, \Theta = \sum_i p_i \cdot \Theta_i\}$$

where $\Delta_i, \Theta_i \in \mathcal{D}(X)$, $p_i \in [0,1]$. This can be obtained as in (10) by taking the lifting of $\mathcal{D}$ from Example 18 and the algebra given by the multiplication $\mu_X \colon \mathcal{D}\mathcal{D}X \to \mathcal{D}X$. Details can be found in [8].

We consider next systems modelled as bialgebras $(X, \alpha \colon TX \to X, \xi \colon X \to FX)$ for a natural transformation $\zeta \colon T \circ F \Rightarrow F \circ T$. When b and f are as in (8), respectively (10), we use [9, Theorem 2] to obtain

▶ **Proposition 26.** *If there exists a lifting $\overline{\zeta} \colon \overline{T} \circ \overline{F} \Rightarrow \overline{F} \circ \overline{T}$ of ζ, then f is b-compatible.*

The next theorem establishes sufficient conditions for the existence of a lifting of ζ.

▶ **Theorem 27.** *Assume the natural transformation $\zeta \colon T \circ F \Rightarrow F \circ T$ lifts to a natural transformation $\widehat{\zeta} \colon \widehat{T} \circ \widehat{F} \Rightarrow \widehat{F} \circ \widehat{T}$ and that we have $\widehat{T} \circ \Sigma_{\lambda^F_X} \leq \Sigma_{T\lambda^F_X} \circ \widehat{T}$. Then ζ lifts to a distributive law $\overline{\zeta} \colon \overline{T} \circ \overline{F} \Rightarrow \overline{F} \circ \overline{T}$.*

Proof Sketch. Notice that $\widehat{T \circ F} := \widehat{T} \circ \widehat{F}$ and $\widehat{F \circ T} := \widehat{F} \circ \widehat{T}$ are liftings of the composite functors $T \circ F$, respectively $F \circ T$. We will denote by $\overline{T \circ F}$ and $\overline{F \circ T}$ the corresponding Wasserstein liftings obtained from $\widehat{T \circ F}$, respectively $\widehat{F \circ T}$ as in Section 5. We split the proof obligation into three parts:

$$\overline{T} \circ \overline{F} \underset{(1)}{\Rightarrow} \overline{T \circ F} \overset{\tilde{\zeta}}{\underset{(2)}{\Rightarrow}} \overline{F \circ T} \underset{(3)}{\Rightarrow} \overline{F} \circ \overline{T}\,.$$

(1) lifts the identity natural transformation on $T \circ F$. Its existence is proved using the hypothesis $\widehat{T} \circ \Sigma_{\lambda^F_X} \leq \Sigma_{T\lambda^F_X} \circ \widehat{T}$.
(2) is obtained by applying Lemma 20 to $\widehat{\zeta}$. Such liftings have already been studied in [4].
(3) lifts the identity natural transformation on $F \circ T$. ◀

The first requirement of the previous theorem holds for the canonical $\mathcal{V}$-Pred-liftings under mild assumptions on F and T.

▶ **Proposition 28.** *Assume that $\zeta \colon T \circ F \Rightarrow F \circ T$ is a natural transformation and that, furthermore, T preserves weak pullbacks and F preserves intersections. Then ζ lifts to a natural transformation $\widehat{\zeta} \colon \widehat{T}_{\mathsf{can}} \circ \widehat{F}_{\mathsf{can}} \Rightarrow \widehat{F}_{\mathsf{can}} \circ \widehat{T}_{\mathsf{can}}$.*

The next proposition establishes sufficient conditions for the second hypothesis of Theorem 27. We need a property on $\mathcal{V}$ that holds for the quantales in Example 6 and was also assumed in [19]. Given $u, v \in \mathcal{V}$ we write $u \lll v$ (u is totally below v) if for every $W \subseteq \mathcal{V}$, $v \leq \bigvee W$ implies that there exists $w \in W$ with $u \leq w$. The quantale $\mathcal{V}$ is constructively completely distributive iff for all $v \in \mathcal{V}$ it holds that $v = \bigvee\{u \in \mathcal{V} \mid u \lll v\}$. In [8] we prove a more general statement in which the lifting of T is not assumed to be the canonical one, that is useful to guarantee the result for interesting liftings, such as the one in Example 18.

▶ **Proposition 29.** *Assume that T preserves weak pullbacks and that $\mathcal{V}$ is constructively completely distributive. Then $\widehat{T}_{\mathsf{can}} \circ \Sigma_f \leq \Sigma_{Tf} \circ \widehat{T}_{\mathsf{can}}$.*

Combining Theorem 27 and Propositions 26, 28 and 29 we conclude:

▶ **Theorem 30.** *Let $(X, \alpha\colon TX \to X, \xi\colon X \to FX)$ be a bialgebra for a natural transformation $\zeta\colon T \circ F \Rightarrow F \circ T$. If $\mathcal{V}$ is constructively completely distributive, T preserves weak pullbacks and F preserves intersections, then $f = \overline{T}_{\mathsf{can}} \circ \Sigma_\alpha$ is compatible with respect to $b = \overline{F}_{\mathsf{can}} \circ \xi^*$.*

When α is the free algebra for a signature $\mu_X\colon T_\Sigma T_\Sigma X \to T_\Sigma X$ (as in Example 24), the above theorem guarantees that up-to contextual closure is compatible with respect to b. By (7), the following holds.

▶ **Corollary 31.** *For all terms t_1, t_2 and unary contexts C, $\nu b(t_1, t_2) \leq \nu b(C(t_1), C(t_2))$.*

For $\mathcal{V} = 2$, since the canonical quantitative lifting coincides with the canonical relational one, then νb is exactly the standard coalgebraic notion of behavioural equivalence [18]. Therefore the above corollary just means that behavioural equivalence is a congruence.

For $\mathcal{V} = [0, \infty]$ instead, this property boils down to *non-expansiveness* of contexts with respect to the behavioural metric. It is worth to mention that this property often fails in probabilistic process algebras when taking the standard Wasserstein lifting which, as shown in Example 18, is *not* the canonical one. We leave as future work to explore the implications of this insight.

7 Example: distance between regular languages

We will now work out the quantitative version of the up-to congruence technique for non-deterministic automata. We consider the shortest-distinguishing-word-distance $d_{\mathtt{sdw}}$, proposed in Section 2. As explained, we will assume an on-the-fly determinization of the non-deterministic automaton, i.e., formally we will work with a coalgebra that corresponds to a deterministic automaton on which we have a join-semilattice structure.

We explain next the various ingredients of the example:

Coalgebra and algebra. As outlined in Section 2 and Example 3 the determinization of an NFA with state space Q is a bialgebra (X, α, ξ) for the distributive law $\zeta_X\colon \mathcal{P}(2 \times X^A) \to 2 \times (\mathcal{P}X)^A$, where $X = \mathcal{P}Q$, $\alpha\colon \mathcal{P}X \to X$ is given by union and $\xi\colon X \to 2 \times X^A$ specifies the DFA structure of the determinization. Hence, we instantiate the generic results in the previous section with $TX = \mathcal{P}X$, $FX = 2 \times X^A$ and ζ as defined in Example 3.

Lifting the functors. We take the quantale $\mathcal{V} = [0, 1]$ (Example 6) and consider the Wasserstein liftings of the endofunctors F and T to $\mathcal{V}$-Rel corresponding to the following evaluation maps:

- $ev_F(b, f) := c \cdot \max_{a \in A} f(a)$, where $b \in \{0, 1\}$, $f\colon A \to [0, 1]$ and c is the constant used in $d_{\mathtt{sdw}}$, and,
- $ev_T := ev^{\mathcal{P}}_{\mathsf{can}} = \sup$, the canonical evaluation map as in Example 15.

These are monotone evaluation maps that satisfy the hypothesis of Theorem 21. Hence the corresponding Wasserstein liftings restrict to $\mathcal{V}$-Cat. We computed the Wasserstein lifting of $T = \mathcal{P}$ in Example 19: applying the lifted functor $\overline{T}$ to a map $d\colon X \times X \to [0, 1]$, gives us the Hausdorff distance, i.e., $\overline{T}(d)(X_1, X_2) = d^H(X_1, X_2)$, where $X_1, X_2 \subseteq X$ and d^H denotes the Hausdorff metric based on d. On the other hand, the Wasserstein lifting of F corresponding to ev_F associates to a metric $d\colon X \times X \to [0, 1]$ the metric $\overline{F}(d)\colon FX \times FX \to [0, 1]$ given by

$$((b_1, f_1), (b_2, f_2)) \mapsto \begin{cases} 1 & \text{if } b_1 \neq b_2 \\ \max\limits_{a \in A} c \cdot \{d(f_1(a), f_2(a))\} & \text{otherwise} \end{cases}$$

Fixpoint equation. The map b for the fixpoint equation was defined in Section 6 as the composite $\xi^* \circ \overline{F}$. Using the above lifting $\overline{F}$, this computation yields exactly the map b defined in (2), whose largest fixpoint (smallest with respect to the natural order on the reals) is the shortest-distinguishing-word-distance introduced in Section 2.

Up-to technique. The next step is to determine the map f introduced in Section 6 for the up-to technique and defined as the composite $\Sigma_\alpha \circ \overline{T}$ on $\mathcal{V}$-Rel. Combining the definition of the direct image functors on $\mathcal{V}$-Rel with the lifting $\overline{T}$, we obtain for a given a map $d\colon X \times X \to [0,1]$ that

$$f(d)(x_1, x_2) = \inf\{d^H(X_1, X_2) \mid X_1, X_2 \subseteq X, \alpha(X_i) = x_i\}$$

To show that $f(d)(Q_1, Q_2) \leq_{\mathbb{R}} r$ for two sets $Q_1, Q_2 \subseteq Q$ (i.e., $Q_1, Q_2 \in X$) and a constant r we use the following rules:

$$f(d)(\emptyset,\emptyset)\leq_{\mathbb{R}} r \qquad \frac{d(Q_1,Q_2)\leq_{\mathbb{R}} r}{f(d)(Q_1,Q_2)\leq_{\mathbb{R}} r} \qquad \frac{f(d)(Q_1,Q_2)\leq_{\mathbb{R}} r \quad f(d)(Q_1',Q_2')\leq_{\mathbb{R}} r}{f(d)(Q_1\cup Q_1', Q_2\cup Q_2')\leq_{\mathbb{R}} r}$$

Lifting of distributive law. In order to prove that the distributive law lifts to $\mathcal{V}$-Rel and hence that the up-to technique is sound by virtue of Proposition 26, we can prove that the two conditions of Theorem 27 are met by the $\mathcal{V}$-Pred liftings of F and T corresponding to the evaluation maps ev_F and ev_T, see [8].

Everything combined, we obtain a sound up-to technique, which implies that the reasoning in Section 2 is valid. Furthermore, as the example shows, the up-to technique can significantly simplify behavioural distance arguments and speed up computations.

8 Related and future work

Up-to techniques for behavioural metrics in a probabilistic setting have been considered in [12] using a generalization of the Kantorovich lifting [11]. In Section 6, we have shown that the basic techniques introduced in [12] (e.g., metric closure, convex closure and contextual closure) as well as the ways to combine them (composition, join and chaining) naturally fit within our framework. The main difference with our approach – beyond the fact that we consider arbitrary coalgebras while the results in [12] just cover coalgebras for a fixed functor – is that the definition of up-to techniques and the criteria to prove their soundness do not fit within the standard framework of [33]. Nevertheless, as illustrated by a detailed comparison in [8], the techniques of [12] can be reformulated within the standard theory and thus proved sound by means of our framework. An important observation brought to light by compositional methodology inherent to the fibrational approach, is that for probabilistic automata a bisimulation metric up-to convexity in the sense of [12] is just a bisimulation metric, see [8]. Nevertheless, the up-to convex closure technique can find meaningful applications in linear, trace-based behavioural metrics (see [4]).

The Wasserstein (respectively Kantorovich) lifting of the distribution functor involving couplings was first used for defining behavioural pseudometrics using final coalgebras in [40]. Our work is based instead on liftings for arbitrary functors, a problem that has been considered in several works (see e.g. [19, 2, 5, 24]), despite with different shades. The closest to our approach are [19] and [5] that we discuss next.

In [19] Hofmann introduces a generalization of the Barr extension (of Set-functors to Rel), namely he defines extensions of Set-monads to the bicategory of $\mathcal{V}$-matrices, in which 0-cells are sets and the $\mathcal{V}$-relations are 1-cells. Some of the definitions and techniques do overlap

between the developments in [19] and the results we presented in Section 5. However, there are also some (subtle) differences which would not allow us to use off the shelf his results.

First, in order to reuse the results in [9], we need to recast the theory in a fibrational setting, rather than the bicategorical setting of [19]. The definition of *topological theory* [19, Definition 3.1] comprises what we call an evaluation map, but which additionally has to satisfy various conditions. An important difference with what we do is that the condition $(Q_{\bigvee})$ in the aforementioned definition entails that the predicate lifting one would obtain from such an evaluation map would be an *opfibred lifting*, rather than a fibred lifting as in our setting. Indeed, the condition $(Q_{\bigvee})$ can be equivalently expressed in terms of a natural transformation involving the *covariant* functor $P_{\mathcal{V}}$, as opposed to the *contravariant* one $\mathcal{V}^{-}$ that we used in Section 5.1. Lastly, in our framework we need to work with arbitrary functors, not necessarily carrying a monad structure.

In [5] we provided a generic construction for the Wasserstein lifting of a functor to the category of pseudo-metric spaces, rather than on arbitrary quantale-valued relations. The realisation that this construction is an instance as a change-of-base situation between $\mathcal{V}$-Rel and $\mathcal{V}$-Pred allows us to exploit the theory in [9] for up-to techniques and, as a side result, provides simpler (and cleaner) conditions for the restriction $\mathcal{V}$-Cat (Theorem 21).

We leave for future work several open problems. What is a universal property for the canonical Wasserstein lifting? Secondly, can the Wasserstein liftings presented here be captured in the framework of [2] or [24]? We also leave for future work the development of up-to techniques for other quantales than 2 and $[0, 1]$. We are particularly interested in weighted automata [16] over quantales and in conditional transition systems, a variant of featured transition systems.

References

1. G. Bacci, G. Bacci, K.G. Larsen, and R. Mardare. Computing behavioral distances, compositionally. In *Proc. of MFCS '13*, pages 74–85. Springer, 2013. LNCS 8087.
2. A. Balan, A. Kurz, and J. Velebil. Extensions of functors from Set to $\mathcal{V}$-cat. In *CALCO*, volume 35 of *LIPIcs*, pages 17–34, 2015.
3. P. Baldan, F. Bonchi, H. Kerstan, and B. König. Behavioral metrics via functor lifting. In *FSTTCS*, volume 29 of *LIPIcs*, 2014.
4. P. Baldan, F. Bonchi, H. Kerstan, and B. König. Towards trace metrics via functor lifting. In *CALCO*, volume 35 of *LIPIcs*, pages 35–49, 2015.
5. P. Baldan, F. Bonchi, H. Kerstan, and B. König. Coalgebraic behavioral metrics. *LMCS*, to appear. arXiv:1712.07511.
6. F. Bonchi, P. Ganty, R. Giacobazzi, and D. Pavlovic. Sound up-to techniques and complete abstract domains. In *Proc. of LICS '18*, 2018.
7. F. Bonchi, B. König, and S. Küpper. Up-to techniques for weighted systems. In *Proc. of TACAS '17, Part I*, pages 535–552. Springer, 2017. LNCS 10205.
8. F. Bonchi, B. König, and D. Petrişan. Up-to techniques for behavioural metrics via fibrations, 2018. arXiv:1806.11064. `arXiv:1806.11064`.
9. F. Bonchi, D. Petrişan, D. Pous, and J. Rot. Coinduction up-to in a fibrational setting. In *CSL-LICS*. ACM, 2014. Paper No. 20.
10. F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013.
11. K. Chatzikokolakis, D. Gebler, C. Palamidessi, and L. Xu. Generalized bisimulation metrics. In *Proc. of CONCUR '14*. Springer, 2014. LNCS/ARCoSS 8704.

12 K. Chatzikokolakis, C. Palamidessi, and V. Vignudelli. Up-to techniques for generalized bisimulation metrics. In *CONCUR*, volume 59 of *LIPIcs*, pages 35:1–35:14, 2016.

13 N.A. Danielsson. Up-to techniques using sized types. *Proc. ACM Program. Lang.*, 2(POPL):43:1–43:28, 2017.

14 L. de Alfaro, M. Faella, and M. Stoelinga. Linear and branching metrics for quantitative transition systems. In *ICALP*, pages 97–109. Springer, 2004. LNCS 3142.

15 J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004.

16 M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009.

17 I. Hasuo. Generic weakest precondition semantics from monads enriched with order. *Theor. Comput. Sci.*, 604:2–29, 2015.

18 C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. and Comp.*, 145:107–152, 1998.

19 D. Hofmann. Topological theories and closed objects. *Advances in Mathematics*, 215(2):789–824, 2007.

20 C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *POPL*, pages 193–206. ACM, 2013.

21 B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999.

22 B. Jacobs. *Introduction to coalgebra. Towards mathematics of states and observations*. Cambridge University Press, 2016.

23 B. Jacobs, A. Silva, and A. Sokolova. Trace semantics via determinization. *J. Comput. Syst. Sci.*, 81(5):859–879, 2015. doi:10.1016/j.jcss.2014.12.005.

24 S. Katsumata and T. Sato. Codensity liftings of monads. In *CALCO*, volume 35 of *LIPIcs*, pages 156–170, 2015.

25 M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *Lecture Notes in Mathematics*. Cambridge University Press, 1982.

26 B. Klin. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011. doi:10.1016/j.tcs.2011.03.023.

27 F.W. Lawvere. Metric spaces, generalized logic, and closed categories. *Reprints in Theory and Applications of Categories*, pages 1–37, 2002.

28 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

29 R. Milner and D. Sangiorgi. Techniques of weak bisimulation up-to. In *CONCUR*. Springer-Verlag, 1992. LNCS 630.

30 J. Parrow and T. Weber. The largest respectful function, 2016. arXiv:1605.04136.

31 D. Pous. Complete lattices and up-to techniques. In *APLAS*, volume 4807 of *LNCS*, pages 351–366. Springer, 2007.

32 D. Pous. Coinduction all the way up. In *Proc. of LICS '16*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

33 D. Pous and D. Sangiorgi. Enhancements of the coinductive proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2011.

34 J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical computer science*, 249(1):3–80, 2000.

35 D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

36 L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theor. Comp. Sci.*, 390:230–247, 2008.

37 A. Silva, F. Bonchi, M. M. Bonsangue, and J.J.M.M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.

38 S. Tini, K.G. Larsen, and D. Gebler. Compositional bisimulation metric reasoning with probabilistic process calculi. *Logical Methods in Computer Science*, 12, 2017.

39 D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proc. of LICS '97*, pages 280–291. IEEE, 1997.

40 F. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In *ICALP*, volume 2076 of *LNCS*, pages 421–432. Springer, 2001.

41 F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comp. Sci.*, 331:115–142, 2005.

42 C. Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.

Completeness for Identity-free Kleene Lattices

Amina Doumane
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
amina.doumane@ens-lyon.fr

Damien Pous
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
damien.pous@ens-lyon.fr

Abstract

We provide a finite set of axioms for identity-free Kleene lattices, which we prove sound and complete for the equational theory of their relational models. Our proof builds on the completeness theorem for Kleene algebra, and on a novel automata construction that makes it possible to extract axiomatic proofs using a Kleene-like algorithm.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Kleene algebra, Graph languages, Petri Automata, Kleene theorem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.18

Related Version Long version at [13], `https://hal.archives-ouvertes.fr/hal-01780845`.

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157). This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

Relation algebra is an efficient tool to reason about imperative programs. In this approach, the bigstep semantics of a program P is a binary relation $[P]$ between memory states [20, 21, 6, 16, 1]. This relation is built from the elementary relations corresponding to the atomic instructions of P, which are combined using standard operations on relations, for instance composition and transitive closure, that respectively encode sequential composition of programs, and iteration (while loops). Abstracting over the concrete behaviour of atomic instructions, one can compare two programs P, Q by checking whether the expressions $[P]$ and $[Q]$ are equivalent in the model of binary relations, which we write as $\mathcal{R}el \models [P] = [Q]$.

To enable such an approach, one should obtain two properties: decidability of the predicate $\mathcal{R}el \models e = f$, given two expressions e and f as input, and axiomatisability of this relation. Decidability makes it possible to automate the verification process, thus alleviating the burden for the end-user [17, 14, 9, 25, 28]. Axiomatisation offers a better way of understanding the equational theory of relations and provides a certificate for programs verification. Indeed, an axiomatic proof of $e = f$ can be seen as a certificate, which can be exchanged, proofread, and combined in a modular way. Axiomatisations also make it possible to solve hard instances manually, when the existing decision procedures have high complexity and/or when considered instances are large [22, 17, 7].

Depending on the class of programs under consideration, several sets of operations on relations can be considered. In this paper we focus on the following set of operations: composition ($\cdot$), transitive closure ($\_^+$), union ($+$), intersection ($\cap$) and the empty relation (0). The expressions generated by this signature are called KL^--expressions. An example of an inequality in the corresponding theory is $\mathcal{R}el \models (a \cap c) \cdot (b \cap d) \leq (a \cdot b)^+ \cap (c \cdot d)$: when a, b, c, d are interpreted as arbitrary binary relations, we have $(a \cap c) \cdot (b \cap d) \subseteq (a \cdot b)^+ \cap (c \cdot d)$. The operations of composition, union and transitive closure arise naturally when defining the bigstep semantics of sequential programs. In contrast, intersection, which is the operation of interest in the present paper, is not a standard operation on programs. This operation is however useful when it comes to specifications: it allows one to express local conjunctions of specifications. For instance, a specification of the shape $(a \cap b)^+$ expresses the fact that execution traces must consist of sequences of smaller traces satisfying both a and b.

The operations of KL^- contain those of identity-free regular expressions, whose equational theory inherits the good properties of *Kleene algebra* (KA).We summarise them below.

First recall that each regular expression e can be associated with a set of words $\mathcal{L}(e)$ called its language. Valid inequations between regular expressions inequalities can be characterised by language inclusions [29]:

$$\mathcal{R}el \models e \leq f \quad \text{iff} \quad \mathcal{L}(e) \subseteq \mathcal{L}(f) \tag{1}$$

Second, we have the celebrated equivalence between regular expressions and non-deterministic finite automata (NFA) via a *Kleene theorem*: for every regular expression e, there is an NFA such that $\mathcal{L}(e)$ is the language of A, and conversely. Decidability follows (in fact, PSpace-completeness). Lastly, although every purely equational axiomatisation of this theory must be infinite [30], Kozen has proved that Conway's finite quasi-equational axiomatisation [12] is sound and complete [19]. (There is also an independent proof of this result by Boffa [8], based on the extensive work of Krob [26].)

Those three results nicely restrict to identity-free Kleene algebra (KA^-), which form a proper fragment of Kleene algebra [23]. It suffices to consider languages of non-empty words: Equation (1) remains, Kleene's theorem still holds, and we have the following characterisation, where we write $\mathsf{KA}^- \vdash e \leq f$ when $e \leq f$ is derivable from the axioms of KA^-:

$$\mathcal{L}(e) \subseteq \mathcal{L}(f) \quad \text{iff} \quad \mathsf{KA}^- \vdash e \leq f \tag{2}$$

There are counterparts to the first two points for KL^--expressions. Each KL^--expression e can be associated with a set of graphs $\mathcal{G}(e)$ called its graph language, and valid inequations of KL^--expressions can be characterised through these languages of graphs. A subtlety here is that we have to consider graphs modulo homomorphisms; writing ${}^{\triangleleft}\mathcal{G}$ for the closure of a set of graphs $\mathcal{G}$ under graph homomorphisms, we have [10]:

$$\mathcal{R}el \models e \leq f \quad \text{iff} \quad {}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f) \tag{3}$$

KL^--expressions are equivalent to a model of automata over graphs called Petri automata [10]. As for KA^--expressions, a Kleene-like theorem holds [11]: for every KL^--expression e, there is a Petri automaton whose language is $\mathcal{G}(e)$, and conversely. Decidability (in fact, ExpSpace-completeness) of the equational theory follows [10, 11].

What is missing to this picture is an axiomatisation of the corresponding equational theory. In the present paper, we provide such an axiomatisation, which we call KL^-, and which comprises the axioms for identity-free Kleene algebra (KA^-) and the axioms of *distributive lattices* for $\{+, \cap\}$. Completeness of this axiomatisation is the difficult result we prove here:

$${}^{\triangleleft}\mathcal{G}\,(e) \subseteq {}^{\triangleleft}\mathcal{G}\,(f) \quad \text{entails} \quad \mathsf{KL}^- \vdash e \leq f \tag{4}$$

We proceed in two main steps. First we show that $\mathcal{G}(e) \subseteq \mathcal{G}(f)$ entails $\mathsf{KL}^- \vdash e \leq f$, using a technique inspired from [24], this is what we call *completeness for strict language inclusion*.The second step is much more involved. There we exploit the Kleene theorem for Petri automata [11]: starting from expressions e, f such that ${}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f)$, we build two Petri automata $\mathscr{A}, \mathscr{B}$ respectively recognising $\mathcal{G}(e)$ and $\mathcal{G}(f)$. Then we design a product construction to synchronise $\mathscr{A}$ and $\mathscr{B}$, and a Kleene-like algorithm to extract from this construction two expressions e', f' such that $\mathcal{G}(e) = \mathcal{G}(e')$, $\mathsf{KL}^- \vdash e' \leq f'$, and $\mathcal{G}(f') \subseteq \mathcal{G}(f)$. This *synchronised Kleene theorem* suffices to conclude using the first step.

To our knowledge, this is the first completeness result for a theory involving Kleene iteration and intersection. Identity-free Kleene lattices were studied in depth by Andréka, Mikulás and Németi [2]; they have in particular shown that over this syntax, the equational theories generated by binary relations and formal languages coincide. But axiomatisability remained opened. The restriction to the identity-free fragment is important for several reasons. First of all, it makes it possible to rely on the technique used in [10] to compare Petri automata, which does not scale in the presence of identity. Second, this is the fragment for which the Kleene theorem for Petri automata is proved the most naturally [11]. Third, 'strange' laws appear in the presence of 1 [3], *e.g.*, $1 \cap (b \cdot a) \leq a \cdot (1 \cap (b \cdot a)) \cdot b$, and axiomatisability is still open even in the finitary case where Kleene iteration is absent – see the erratum about [3].

Proofs of completeness for other extensions of Kleene algebra include Kleene algebra with tests (KAT) [20], nominal Kleene algebra [24], and Concurrent Kleene algebra [27, 18]. The latter extension is the closest to our work since the parallel operator of concurrent Kleene algebra shares some properties of the intersection operation considered in the present work (*e.g.*, it is commutative and it satisfies a weak interchange law with sequential composition).

The paper is organised as follows. In Sect. 2, we recall KL^--expressions, their graph language and the corresponding model of Petri automata. In Sect. 3 we give our axiomatisation and state the completeness result. Then we show it following the proof scheme presented earlier: in Sect. 4 we show completeness for strict language inclusions, we recall in Sect. 5 the Kleene theorem of KL^- expressions, on which we build to show our synchronised Kleene theorem in Sect. 6.

2 Expressions, graph languages and Petri automata

2.1 Expressions and their relational semantics

We let $a, b \ldots$ range over the letters of a fixed alphabet X. We consider the following syntax of KL^--*expressions*, which we simply call expressions if there is no ambiguity:

$$e, f ::= e \cdot f \;\mid\; e + f \;\mid\; e \cap f \;\mid\; e^+ \;\mid\; 0 \;\mid\; a \qquad (a \in X)$$

We denote their set by Exp_X and we often write ef for $e \cdot f$. When we remove intersection ($\cap$) from the syntax of KL^--expressions we get KA^--*expressions*, which are the identity-free regular expressions.

If $\sigma : X \to \mathcal{P}(S \times S)$ is an interpretation of the letters into some space of relations, we write $\widehat{\sigma}$ for the unique homomorphism extending σ into a function from Exp_X to $\mathcal{P}(S \times S)$. An inequation between two expressions e and f is *valid*, written $\mathcal{R}el \models e \leq f$, if for every such interpretation σ we have $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$.

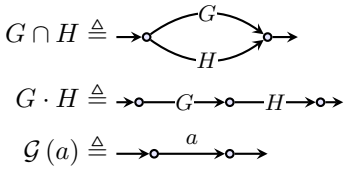


Figure 1 Operations on graphs.

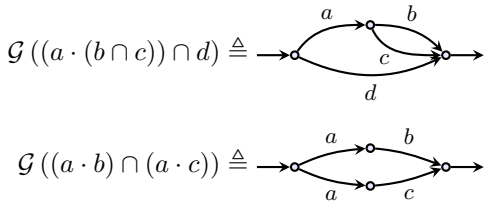


Figure 2 Graphs associated with some terms.

2.2 Terms, graphs, and homomorphisms

We let $u, v \ldots$ range over expressions built using only letters, $\cap$ and $\cdot$, which we call *terms*. (Terms thus form a subset of expressions: they are those expressions not using 0, $+$ and $\_^+$.) We will use 2-pointed labelled directed graphs, simply called *graphs* in the sequel. Those are tuples $\langle V, E, s, t, l, \iota, o\rangle$ with V (resp. E) a finite set of vertices (resp. edges), $s, t : E \to V$ the *source* and *target* functions, $l : E \to X$ the *labelling* function, and $\iota, o \in V$ two distinguished vertices, respectively called *input* and *output*.

As depicted in Fig. 1, graphs can be composed in series or in parallel, and a letter can be seen as a graph with a single edge labelled by that letter. One can thus recursively associate to every term u a graph $\mathcal{G}(u)$ called the *graph of* u. Two examples are given in Fig. 2; graphs of terms are *series-parallel* [31].

▶ **Definition 1** (Graph homomorphism). A *homomorphism* from $G = \langle V, E, s, t, l, \iota, o\rangle$ to $G' = \langle V', E', s', t', l', \iota', o'\rangle$ is a pair $h = \langle f, g\rangle$ of functions $f : V \to V'$ and $g : E \to E'$ that respect the various components: $s' \circ g = f \circ s$, $t' \circ g = f \circ t$, $l = l' \circ g$, $\iota' = f(\iota)$, and $o' = f(o)$.

We write $G' \lhd G$ if there exists a graph homomorphism from G to G'.

Such a homomorphism is depicted in Fig. 3. A pleasant way to think about graph homomorphisms is the following: we have $G \lhd H$ if G is obtained from H by merging (or identifying) some nodes, and by adding some extra nodes and edges. For instance, the graph G in Fig. 3 is obtained from H by merging the nodes $1, 2$ and by adding an edge between the input and the output labelled by d.

The starting point of the present work is the following characterisation:

▶ **Theorem 2** ([5, Thm. 1], [15, p. 208]). *For all terms* u, v, $\mathcal{R}el \models u \leq v$ *iff* $\mathcal{G}(u) \lhd \mathcal{G}(v)$.

2.3 Graph language of an expression

To generalise the previous characterisation to KL^--expressions, one interprets expressions by sets (languages) of graphs: graphs play the role of words for KA-expressions.

▶ **Definition 3** (Term and graph languages of expressions). The *term language* of an expression e, written $[\![e]\!]$, is the set of terms defined recursively as follows:

$$[\![e \cdot f]\!] \triangleq \{u \cdot v \mid u \in [\![e]\!] \text{ and } v \in [\![f]\!]\} \qquad [\![0]\!] \triangleq \emptyset$$
$$[\![e \cap f]\!] \triangleq \{u \cap v \mid u \in [\![e]\!] \text{ and } v \in [\![f]\!]\} \qquad [\![a]\!] \triangleq \{a\}$$
$$[\![e + f]\!] \triangleq [\![e]\!] \cup [\![f]\!] \qquad [\![e^+]\!] \triangleq \bigcup_{n>0} \{u_1 \cdot \cdots \cdot u_n \mid \forall i, u_i \in [\![e]\!]\}$$

The *graph language* of e is the set of graphs $\mathcal{G}(e) \triangleq \{\mathcal{G}(u) \mid u \in [\![e]\!]\}$.

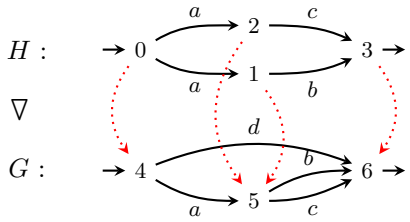


Figure 3 A graph homomorphism.

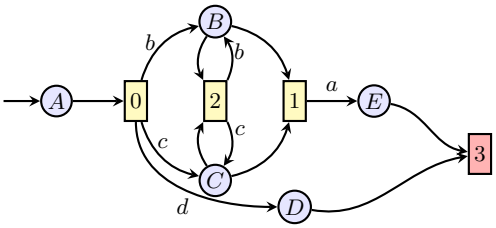


Figure 4 A Petri automaton.

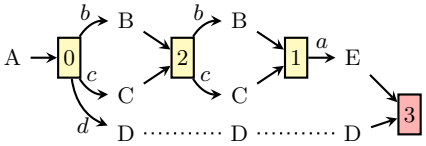


Figure 5 Run of a Petri automaton.

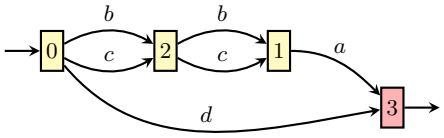


Figure 6 Graph of a run.

Note that for every term u, $[\![u]\!] = \{u\}$, so that the graph language of u thus contains just the graph of u. This justifies the overloaded notation $\mathcal{G}(u)$. Given a set S of graphs, we write ${}^{\lhd}S$ for its downward closure w.r.t. $\lhd$: ${}^{\lhd}S \triangleq \{G \mid G \lhd G', G' \in S\}$. We obtain:

▶ **Theorem 4** ([10, Thm. 6]). *For all expressions e, f, $\mathcal{R}el \models e \leq f$ iff ${}^{\lhd}\mathcal{G}(e) \subseteq {}^{\lhd}\mathcal{G}(f)$.*

2.4 Petri automata

We recall the notion of Petri automata [10, 11], an automata model that recognises precisely the graph languages of our expressions.

▶ **Definition 5** (Petri Automaton). A *Petri automaton (PA)* over the alphabet X is a tuple $\mathscr{A} = \langle P, \mathcal{T}, \iota \rangle$ where:

- P is a finite set of *places*,
- $\mathcal{T} \subseteq \mathcal{P}(P) \times \mathcal{P}(X \times P)$ is a set of *transitions*,
- $\iota \in P$ is the *initial place* of the automaton.

For each transition $t = \langle {}^{\triangleleft}t, t^{\triangleright} \rangle \in \mathcal{T}$, ${}^{\triangleleft}t$ is assumed to be non-empty; ${}^{\triangleleft}t \subseteq P$ is the *input of* t; and $t^{\triangleright} \subseteq X \times P$ is the *output of* t. We write $\pi_2(t^{\triangleright}) \triangleq \{p \mid \exists a, \langle a, p \rangle \in t^{\triangleright}\}$ for the set of the output places of t. Transitions with empty outputs are called *final*.

A PA is depicted in Fig. 4: places are represented by circles and transitions by squares.

Let us now recall the operational semantics of PA. Fix a PA $\mathscr{A} = \langle P, \mathcal{T}, \iota \rangle$ for the remainder of this section. A *state* of this automaton is a set of places. In a given state $S \subseteq P$, a transition $t = \langle {}^{\triangleleft}t, t^{\triangleright} \rangle$ is *enabled* if ${}^{\triangleleft}t \subseteq S$. In that case, we may fire t, leading to a new state $S' = (S \setminus {}^{\triangleleft}t) \cup \pi_2(t^{\triangleright})$. We write $S \xrightarrow{t}_{\mathscr{A}} S'$ in this case.

▶ **Definition 6** (Run of a PA). A *run* is a sequence $\langle S_1, t_1, S_2, \ldots, t_{n-1}, S_n \rangle$, where S_i are states, t_i are transitions such that $S_i \xrightarrow{t_i}_{\mathscr{A}} S_{i+1}$ for every $i \in [1, n-1]$, $S_1 = \{\iota\}$ and $S_n = \emptyset$.

A run of the PA from Fig. 4 is depicted in Fig. 5; this run gives rise to a graph, depicted in Fig. 6; see [11, Def. 3] for a formal definition in the general case.

▶ **Definition 7** (Graph language of a PA). The *graph language of a PA* $\mathscr{A}$, written $\mathcal{G}(\mathscr{A})$, consists of the graphs of its runs.

$$e \cap (f \cap g) = (e \cap f) \cap g \qquad e \cap f = f \cap e \qquad e \cap e = e$$

$$e \cap (f + g) = (e \cap f) + (e \cap g) \qquad e \cap (e + f) = e \qquad e + (e \cap f) = e$$

$$e + (f + g) = (e + f) + g \qquad e + f = f + e \qquad e + e = e$$

$$e{\cdot}(f{\cdot}g) = (e{\cdot}f){\cdot}g \quad e{\cdot}(f{+}g) = e{\cdot}f{+}e{\cdot}g \quad (e{+}f){\cdot}g = e{\cdot}g{+}f{\cdot}g \quad e{+}0 = e \quad e{\cdot}0 = 0 = 0{\cdot}e$$

$$e{+}e{\cdot}e^{+} = e^{+} = e{+}e^{+}{\cdot}e \quad e{\cdot}f{+}f = f \;\Rightarrow\; e^{+}{\cdot}f{+}f = f \quad f{\cdot}e{+}f = f \;\Rightarrow\; f{\cdot}e^{+}{+}f = f$$

Figure 7 KL^{-}: the first three lines correspond to distributive lattices, the last three to KA^{-}.

PA are assumed to be *safe* (in standard Petri net terminology, places contain at most one *token* at any time – whence the definition of states as sets rather than multisets) and to accept only series-parallel graphs. These two conditions are decidable [11]. Here we moreover assume that all PA have the same set of places P.

PA and KL^{-}-expressions denote the same class of graph languages:

▶ **Theorem 8** (Kleene theorem [11, Thm. 18])**.**

(i) *For every expression e, there is a Petri automaton $\mathscr{A}$ such that $\mathcal{G}(e) = \mathcal{G}(\mathscr{A})$.*

(ii) *Conversely, for every Petri automaton $\mathscr{A}$, there is an expression e such that $\mathcal{G}(e) = \mathcal{G}(\mathscr{A})$.*

3 Axiomatisation and structure of completeness proof

Let us introduce now our axiomatisation.

▶ **Definition 9.** The axioms of KL^{-} are the union of

- the axioms of identity-free Kleene algebra (KA^{-}) [23], and
- the axioms of a distributive lattice for $\{+, \cap\}$.

It is easy to check that those axioms are valid for binary relations, whence soundness of KL^{-}:

▶ **Theorem 10** (Soundness)**.** *If $\mathsf{KL}^{-} \vdash e \leq f$ then $\mathcal{R}el \models e \leq f$.*

The rest the paper is devoted the converse implication, which thanks to Thm. 4 amounts to:

▶ **Theorem 11** (Completeness)**.** *If ${}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f)$ then $\mathsf{KL}^{-} \vdash e \leq f$.*

The following very weak form of Thm. 11 is easy to obtain from the results in the literature:

▶ **Proposition 1.** *For all terms u, v, $\mathcal{G}(u) \lhd \mathcal{G}(v)$ entails $\mathsf{KL}^{-} \vdash u \leq v$.*

Proof. Follows from Thm. 4, completeness of semilattice-ordered semigroups [4] for relational models, and the fact the the axioms of KL^{-} entail those of semilattice-ordered semigroups. ◀

As explained in the introduction, our first step consists in proving KL^{-} completeness w.r.t. strict graph language inclusions, *i.e.*, not modulo homomorphisms:

▶ **Theorem 12** (Completeness for strict language inclusions)**.** *If $\mathcal{G}(e) \subseteq \mathcal{G}(f)$ then $\mathsf{KL}^{-} \vdash e \leq f$.*

The proof is given in Sect. 4. Our second step is to get the following theorem (Sect. 6):

▶ **Theorem 13** (Synchronised Kleene Theorem). *If $\mathscr{A}, \mathscr{B}$ are PA such that ${}^{\triangleleft}\mathcal{G}(\mathscr{A}) \subseteq {}^{\triangleleft}\mathcal{G}(\mathscr{B})$, then there are expressions e, f such that:*

$$\mathcal{G}(\mathscr{A}) = \mathcal{G}(e), \quad \mathsf{KL}^- \vdash e \leq f, \quad \text{and} \quad \mathcal{G}(f) \subseteq \mathcal{G}(\mathscr{B}).$$

The key observation for the proof is that the state-removal procedure used to transform a PA into a KL^- expression is highly non-deterministic. When considering two PA at a time, one can use this flexibility in order to synchronise the computation of the two expressions, so that they become easier to compare axiomatically. The concrete proof is quite technical and requires us to first recall many concepts from the proof [11] of Thm. 8(ii) (Sect. 5); it heavily relies on both Thm. 12 and Prop. 1.

Completeness of KL^- follows using Thm. 8(i) and Thm. 12 as explained in the introduction.

4 Completeness for strict language inclusion

Recall that the graph language of an expression e, $\mathcal{G}(e)$, is defined as the set of graphs of the term language of e, $[\![e]\!]$. We first prove that KL^- is complete for term language inclusions:

▶ **Proposition 2.** *If $[\![e]\!] \subseteq [\![f]\!]$ then $\mathsf{KL}^- \vdash e \leq f$.*

Proof. We follow a technique similar to the one recently used in [24]. We consider the maximal KA^--subexpressions, and we compute the atoms of the Boolean algebra of word languages generated by those expressions. By KA^- completeness [19, 23], we get KA^- (and thus KL^-) proofs that those are equal to appropriate sums of atoms. We distribute the surrounding intersections over those sums and replace the resulting intersections of atoms by fresh letters. This allows us to proceed recursively (on the intersection-depth of the terms), using substitutivity to recover a KL^- proof of the starting inequality. ◀

The difference between the term language and the graph language is that intersection is interpreted as an associative and commutative operation in the latter. We bury this difference by defining a 'saturation' function s on KL^--expressions such that for all e,

$$(\dagger) \quad \mathsf{KL}^- \vdash s(e) = e, \qquad \text{and} \qquad (\ddagger) \quad [\![s(e)]\!] = \{u \mid \mathcal{G}(u) \in \mathcal{G}(e)\} \quad .$$

Intuitively, this function uses distributivity and idempotency of sum to replace all intersections appearing in the expression by the sum of all their equivalent presentations modulo associativity and commutativity. For instance, $s(a \cap (b \cap c))$ is a sum of twelve terms (six choices for the ordering times two choices for the parenthesing). Technically, one should be careful to expand the expression first by maximally distributing sums, in order to make all potential n-ary intersections apparent. For instance, $((a \cap b) + d) \cap c$ expands to $((a \cap b) \cap c) + (d \cap c)$ so that its saturation is a sum of twelve plus two terms. For the same reason, all iterations should be unfolded once: we unfold and expand $(a \cap b)^+ \cap c$ into $((a \cap b) \cap c) + ((a \cap b) \cdot (a \cap b)^+ \cap c)$ before saturating it. We finally obtain Thm. 12 using (‡), Prop. 2, and (†):

$$\mathcal{G}(e) \subseteq \mathcal{G}(f) \quad \Rightarrow \quad [\![s(e)]\!] \subseteq [\![s(f)]\!] \quad \Rightarrow \quad \mathsf{KL}^- \vdash s(e) \leq s(f) \quad \Rightarrow \quad \mathsf{KL}^- \vdash e \leq f$$

5 Kleene theorem for Petri automata

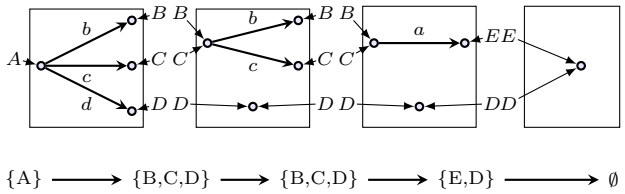
To prove the synchronised Kleene theorem (Thm. 13), we cannot use the Kleene theorem for PA (Thm. 8) as a black box: we use in a fine way the algorithm underlying the proof of the second item. We thus explain how it works [11] in details.

Figure 8 Rewriting rules for state-removal procedure.

Recall that to transform an NFA $\mathscr{A}$ to a regular expression e, one rewrites it using the rules of Fig. 8 until one reaches an automaton where there is a unique transition from the initial state to the final one, labelled by an expression e. While doing so, one goes through generalised NFA, whose transitions are labelled by regular expressions instead of letters.

We use the same technique for PA: we start by converting the PA into a NFA over a richer alphabet, which we call a *Template Automaton (TA)*, then we reduce this automaton using the rules of Fig. 8 until we get a single transition labelled by the desired expression.

To get some intuitions about the way we convert a PA into an NFA, consider the run in Fig. 5 and its graph in Fig. 6. One can decompose the run and the graph as follows:



The graph can thus be seen as a word over an alphabet of 'boxes', and the run as a path in an NFA whose states are sets of places of the PA. The letters of the alphabet, the above boxes, can be seen as 'slices of graphs'; they arise naturally from the transitions of the starting PA (Fig. 4 in this example).

5.1 Template automata

In order to make everything work, we need to refine both this notion of states and this notion of boxes to define template automata:

- states (sets of places) are refined into *types*. We let σ, τ range over types. A type is a tree whose leaves are labelled by places. When we forget the tree structure of a type τ, we get a a state $\overline{\tau}$. See [11, Def. 10] for a formal definition of types, which is not needed here. We call *singleton types* those types whose associated state is a singleton.
- letters will be *templates*: finite sets of boxes like depicted above but with edges labelled with arbitrary KL^--expressions; we define those formally below.

Given a directed acyclic graph (DAG) G, we write $\min G$ (resp. $\max G$) for the set of its sources (resp. sinks). A DAG is non-trivial when it contains at least one edge.

▶ **Definition 14** (Boxes). Let σ, τ be types. A *box* from σ to τ is a triple $\langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle$ where G is a non-trivial DAG with edges labelled in Exp_X, $\overrightarrow{\mathfrak{p}}$ is a map from $\overline{\sigma}$, the *input ports*, to the vertices of G, and $\overleftarrow{\mathfrak{p}}$ is a bijective map from $\overline{\tau}$, the *output ports*, to $\max G$, and where an additional condition relative to types holds [11, Def. 11]. (This condition can be kept abstract here.) A *basic* box is a box labelled with letters rather than arbitrary expressions. A *1-1* box is a box between singleton types.

We let α, β range over boxes and we write $\beta : \sigma \to \tau$ when β is a box from σ to τ.

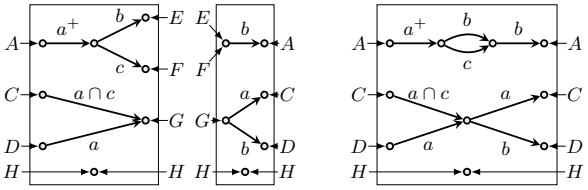


Figure 9 Two boxes and their composition.

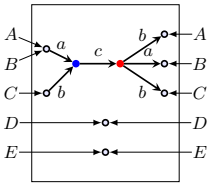


Figure 10 An atomic box.

We represent boxes graphically as in Fig. 9. Inside the rectangle is the DAG, with the input ports on the left-hand side and the output ports on the right-hand side. The maps $\overrightarrow{\mathfrak{p}}$ and $\overleftarrow{\mathfrak{p}}$ are represented by the arrows going from the ports to vertices inside the rectangle. Note that unlike $\overleftarrow{\mathfrak{p}}$, the map $\overrightarrow{\mathfrak{p}}$ may reach inner nodes of the DAG. 1-1 boxes are those with exactly one input port and one output port.

Boxes compose like in a category: if $\alpha : \sigma \to \tau$ and $\beta : \tau \to \rho$ then we get a box $\alpha \cdot \beta : \sigma \to \rho$ by putting the graph of α to the left of the graph of β, and for every port $p \in \overline{\tau}$, we identify the node $\overleftarrow{\mathfrak{p}_1}(p)$ with the node $\overrightarrow{\mathfrak{p}_2}(p)$. For instance the third box in Fig. 9 is obtained by composing the first two.

The key property enforced by the condition on types (kept abstract here) is the following:

▶ **Lemma 15.** *A 1-1 box is just a series-parallel 2-pointed graph labelled in* Exp_X.

Accordingly, one can extract a KL^--expression from any 1-1 box β, which we write $e(\beta)$ and call its *expression*.

▶ **Definition 16** (Templates). A *template* $\Gamma : \sigma \to \tau$ is a finite set of boxes from σ to τ. A *1-1 template* is a template of 1-1 boxes. The *expression* of a 1-1 template, written $e(\Gamma)$, is the sum of the expressions of its boxes.

Templates can be composed like boxes, by computing all pairwise box compositions.

▶ **Definition 17** (Box language of a template). A basic box is *generated* by a box β if it can be obtained by replacing each edge $x \xrightarrow{e} y$ of its DAG by a graph $G' \in \mathcal{G}(e)$ with input vertex x and output vertex y. The *box language* of a template Γ, written $\mathcal{B}(\Gamma)$, is the set of basic boxes generated by its boxes.

As expected, the box language of a template $\Gamma : \sigma \to \tau$ only contains boxes from σ to τ. Thanks to Lem. 15, when Γ is a 1-1 template, its box language can actually be seen as a set of graphs, and we have:

▶ **Proposition 3.** *For every 1-1 template* Γ*, we have* $\mathcal{B}(\Gamma) = \mathcal{G}(e(\Gamma))$.

We can finally define template automata:

▶ **Definition 18** (Template automaton (TA)). A *template automaton* is an NFA whose states are types, whose alphabet is the set of templates, whose transitions are of the form $\langle \sigma, \Gamma, \tau \rangle$ where $\Gamma : \sigma \to \tau$, and with a single initial state and a single accepting state which are singleton types. A *basic TA* is a TA whose all transitions are labelled by basic boxes.

By definition, a word accepted by a TA is a sequence of templates that can be composed into a single 1-1 template Γ, and thus gives rise to a set of graphs $\mathcal{B}(\Gamma)$. The *graph language of a TA* $\mathscr{E}$, written $\mathcal{G}(\mathscr{E})$, is the union of all those sets of graphs.

An important result of [11] is that we can translate every PA into a TA:

▶ **Proposition 4.** *For every PA $\mathscr{A}$, there exists a basic TA $\mathscr{E}$ such that $\mathcal{G}(\mathscr{A}) = \mathcal{G}(\mathscr{E})$.*

TA were defined so that they can be reduced using the state-removal procedure from Fig. 8. Templates can be composed sequentially and are closed under unions, so that now we only miss an operation $\_^*$ on templates to implement the first rule. Since we work in an identity-free (and thus star-free) setting, it suffices to define a strict iteration operation $\_^+$; and to rely on the following shorthands $\Delta \cdot \Gamma^* = \Delta \cup \Delta \cdot \Gamma^+$ and $\Gamma^* \cdot \Delta = \Delta \cup \Gamma^+ \cdot \Delta$.

Such an operation is provided in [11]:

▶ **Proposition 5.** *There exists a function $\_^+$ on templates such that if the TA obtained from a PA $\mathscr{A}$ through Prop. 4 reduces to a TA $\mathscr{E}$ by the rules in Fig. 8, then $\mathcal{G}(\mathscr{A}) = \mathcal{G}(\mathscr{E})$.* [1]

One finally obtains the Kleene theorem for PA by reducing the TA until it consists of a single transition labelled by a 1-1 template Γ: at this point, $e(\Gamma)$ is the desired KL^--expression.

5.2 Computing the iteration of a template

We need to know how the above template iteration can be defined to obtain our synchronised Kleene theorem, so that we explain it in this section. This section is required only to understand how we define a synchronised iteration operation in Sect. 6.

First notice that templates on which we need to compute $\_^+$ are of type $\sigma \to \sigma$. We first define this operation for a restricted class of templates, which we call *atomic*.

▶ **Definition 19** (Atomic boxes and templates, Support)**.** A box $\beta = \langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle : \sigma \to \sigma$ is *atomic* if its graph has a single non-trivial connected component C, and if for every vertex v outside C, there is a unique port $p \in \overline{\sigma}$ such that $\overrightarrow{\mathfrak{p}}(p) = \overleftarrow{\mathfrak{p}}(p) = v$. An *atomic template* is a template composed of atomic boxes.

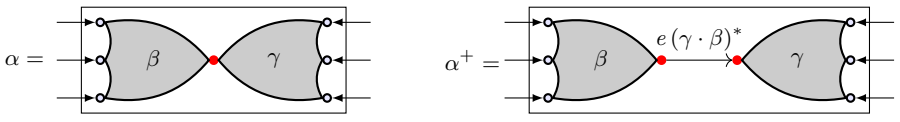
The *support* of a box $\beta : \sigma \to \sigma$ is the set $supp(\beta) \triangleq \{p \mid \overrightarrow{\mathfrak{p}}(p) \neq \overleftarrow{\mathfrak{p}}(p)\}$. The support of a template is the union of the supports of its boxes.

The following property of atomic boxes, makes it possible to compute their iteration:

▶ **Lemma 20** ([11, Lem. 7.18])**.** *The non-trivial connected component of an atomic box $\beta : \sigma \to \sigma$ always contains a vertex c, s.t. for every port p mapped inside that component, all paths from $\overrightarrow{\mathfrak{p}}(p)$ to a maximal vertex visit c. We call such a vertex a* bowtie *for β.*

Notice that the bowtie of a box is not unique. For instance, the atomic box in Fig. 10 contains two bowties: the blue and the red nodes.

We compute the iteration of an atomic box as follows. First choose a bowtie for this box, then split it at the level of this node into the product $\alpha = \beta \cdot \gamma$. The box $\gamma \cdot \beta$ is 1-1, we can thus extract from it a term $e(\gamma \cdot \beta)$. We set α^+ to be the template consisting of α and the box obtained from α by replacing the bowtie by an edge labelled $e(\gamma \cdot \beta)^+$. For the sake of conciseness, we denote this two-box template as on the right below, with an edge labelled with a starred expression.



[1] This statement is not simpler because, unfortunately, there is no function $\_^+$ on templates such that $\mathcal{B}(\Gamma^+) = \mathcal{B}(\Gamma)^+$).

Data: Atomic template Γ
Result: A template Γ^+ s.t.
$\mathcal{B}(\Gamma^+) = \mathcal{B}(\Gamma)^+$
if $\Gamma = \emptyset$ **then**
  Return $\emptyset$
else
  Write $\Gamma = \Delta \cup \{\alpha\} \cup \Sigma$ such that
  $supp(\Delta) \subseteq supp(\alpha)$ and
  $supp(\Sigma) \cap supp(\alpha) = \emptyset$;
  Choose a bowtie for α;
  Split α into $\beta \cdot \gamma$ at the level of this bowtie;
  Return
  $(\Delta^+ \cdot \Sigma^*) \cup (\Delta^* \cdot \Sigma^+) \cup (\Delta^* \cdot \delta \cdot \Delta^* \cdot \Sigma^*)$,
  where δ is the two-box template depicted on the right.
end

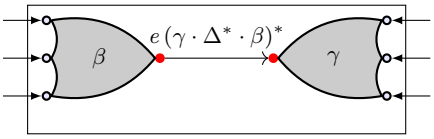


Figure 11 Iteration of an atomic template.

It is not difficult to see that $\mathcal{B}(\alpha^+) = \mathcal{B}(\alpha)^+$. Depending on the bowtie we have chosen, the box α^+ will be different. This is why we will write $\alpha^+_{\bowtie}$ to say that the bowtie $\bowtie$ has been selected for the computation of the iteration.

Now we need to generalise this construction to compute the iteration of an atomic template. For this, we need the following property, saying that the supports of atomic boxes of the same type are either disjoint or comparable:

▶ **Lemma 21.** *For all atomic boxes $\beta, \gamma : \sigma \to \sigma$, we have either 1) $supp(\beta) \subseteq supp(\gamma)$, or 2) $supp(\gamma) \subseteq supp(\beta)$, or 3) $supp(\beta) \cap supp(\gamma) = \emptyset$.*

We can compute the iteration of an atomic template by the algorithm in Fig. 11; intuitively, atomic boxes with disjoint support can be iterated in any order: they cannot interfere; in contrast, atomic boxes with small support must be computed before atomic boxes with strictly larger support: the iteration of the latter depends on that of the former. (Also note that since $supp(\Delta) \subseteq supp(\alpha)$ we have also $supp(\Delta^+) \subseteq supp(\alpha)$ thus the template $\gamma \cdot \Delta^* \cdot \beta$ is 1-1 and it gives rise to an expression $e(\gamma \cdot \Delta^* \cdot \beta)$.)

We finally compute the iteration of an arbitrary template $\Gamma : \sigma \to \sigma$ as follows: from each connected component of the graph of each box in Γ stems an atomic box; let $At(\Gamma)$ be the set of all these atomic boxes; we set $\Gamma^+ = At(\Gamma)^+$.

The overall algorithm contains two sources of non-determinism. First, one can partially choose in which order to process the atomic boxes. This is reflected by the choice of the box α, which we will call the *pivot*. For instance if $\Gamma = \{\alpha_1, \alpha_2, \beta\}$ such that $supp(\alpha_1) = supp(\alpha_2)$ and $supp(\beta) \cap supp(\alpha_1) = \emptyset$, then we can choose either α_1 or α_2 as the pivot, and the computation will respectively start with the computation of α_2^+ or that of α_1^+, yielding two distinct expressions. (In contrast, choices about boxes with disjoint support do not change the final result.) Second, every box of the template is eventually processed, and one must thus choose a bowtie for all of them. We write $\Gamma^+_{\bowtie,\leq}$ to make explicit the choice of the bowties and the computation order.

6 Synchronised Kleene theorem for PA

We can now prove Thm. 13. To synchronise the computation of two expressions e, f for two PA $\mathscr{A}, \mathscr{B}$ respectively, we construct a *synchronised product automaton* $\mathscr{E} \times \mathscr{F}$ between a TA $\mathscr{E}$ for $\mathscr{A}$ and a TA $\mathscr{F}$ for $\mathscr{B}$.

The states of this automaton are triples $\langle \sigma, \eta, \tau \rangle$ where σ and τ are types, *i.e.*, states from the TA $\mathscr{E}$ and $\mathscr{F}$, and $\eta : \overline{\tau} \to \overline{\sigma}$ is a function used to enforce coherence conditions. Its transitions have the form $\langle \langle \sigma, \eta, \tau \rangle, \langle \Gamma, \Delta \rangle, \langle \sigma', \eta', \tau' \rangle \rangle$ where $\langle \sigma, \Gamma, \sigma' \rangle$ is a transition of $\mathscr{E}$, $\langle \tau, \Delta, \tau' \rangle$ is a transition of $\mathscr{F}$, and Γ and Δ satisfy a certain condition which we call *refinement*, written $\Gamma \leq \Delta$.

The overall strategy is as follows. We reduce $\mathscr{E} \times \mathscr{F}$ using the rules of Fig. 8, where the operations $\cdot$ and $\cup$ are computed pairwise. The operation $\_^*$ is also computed pairwise, but in a careful way, exploiting the non-determinism of this operation to ensure that we maintain the refinement relation. We eventually get a single transition labelled by a pair of 1-1 templates Γ and Δ such that $\mathcal{B}(\Gamma) = \mathcal{G}(\mathscr{A})$, $\mathcal{B}(\Delta) = \mathcal{G}(\mathscr{B})$, and $\Gamma \leq \Delta$. To conclude, it suffices to deduce $\mathsf{KL}^- \vdash e(\Gamma) \leq e(\Delta)$ from the latter property. To sum-up, what we need to do now is:

- **Refinement**: define the refinement relation $\leq$ on templates;
- **Initialisation**: define $\mathscr{E} \times \mathscr{F}$ so that refinement holds;
- **Stability**: show that the refinement relation is maintained during the rewriting process;
- **Finalisation**: show that refinement between 1-1 templates entails KL^- provability.

6.1 Refinement relation

We first generalise graph homomorphisms to templates; this involves dealing with multiple ports, with finite sets, and with edge labels which are now arbitrary KL^--expressions. For the latter, we do not require strict equality but KL^--derivable inequalities.

▶ **Definition 22** (Box and template homomorphisms)**.** Let $\sigma, \tau, \sigma', \tau'$ be four types with two functions $\eta : \overline{\sigma} \to \overline{\tau}$ and $\eta' : \overline{\sigma'} \to \overline{\tau'}$. Let $\beta = \left\langle \overrightarrow{\mathfrak{p}}_\beta, \langle V_\beta, E_\beta, s_\beta, t_\beta, l_\beta \rangle, \overleftarrow{\mathfrak{p}}_\beta \right\rangle$ be a box of type $\tau \to \tau'$ and let $\alpha = \left\langle \overrightarrow{\mathfrak{p}}_\alpha, \langle V_\alpha, E_\alpha, s_\alpha, t_\alpha, l_\alpha \rangle, \overleftarrow{\mathfrak{p}}_\alpha \right\rangle$ be a box of type $\sigma \to \sigma'$. A homomorphism from α to β is a pair $\langle f, g \rangle$ of functions $f : V_\alpha \to V_\beta$ and $g : E_\alpha \to E_\beta$ s.t.:

- $s_\beta \circ g = f \circ s_\alpha$, $t_\beta \circ g = f \circ t_\alpha$,
- $\forall e \in E_\alpha, \quad \mathsf{KL}^- \vdash l_\beta \circ g(e) \leq l_\alpha(e)$,
- If $\{v\} \subseteq V_\alpha$ is a trivial connected component, so is $f(v)$.
- $\overrightarrow{\mathfrak{p}}_\beta \circ \eta = f \circ \overrightarrow{\mathfrak{p}}_\alpha$ and $\overleftarrow{\mathfrak{p}}_\beta \circ \eta' = f \circ \overleftarrow{\mathfrak{p}}_\alpha$. (We call this condition (η, η')-*compatibility*.)

We write $\beta \lhd_{\eta,\eta'} \alpha$ when there exists such a homomorphism. For two templates $\Gamma : \tau \to \tau'$ and $\Delta : \sigma \to \sigma'$, we write $\Gamma \lhd_{\eta,\eta'} \Delta$ if for all $\beta \in \Gamma$, there exists $\alpha \in \Delta$ such that $\beta \lhd_{\eta,\eta'} \alpha$.

We abbreviate $\Gamma \lhd_{\eta,\eta'} \Delta$ as $\Gamma \lhd \Delta$ when Γ, Δ are 1-1 templates, or when $\sigma = \tau$, $\sigma' = \tau'$ and η, η' are the identity function id. A box homomorphism is depicted in Fig. 12.

The above relation on templates is not enough for our needs; we have to extend it so that it is preserved during the rewriting process. We first write $\Gamma \sqsubseteq \Delta$ when $\mathcal{B}(\Gamma) \subseteq \mathcal{B}(\Delta)$, for two templates Γ, Δ of the same type. Refinement is defined as follows:

▶ **Definition 23** (Refinement)**.** We call *refinement* the relation on templates defined by $\leq_{\eta,\eta'} \triangleq \lhd_{\eta,\eta'} \cdot (\lhd_{\mathrm{id},\mathrm{id}} \cup \sqsubseteq)^*$, where $\_^*$ is reflexive transitive closure.

The following proposition shows that refinement implies provability of the expressions extracted from 1-1 templates. This gives us the finalisation step.

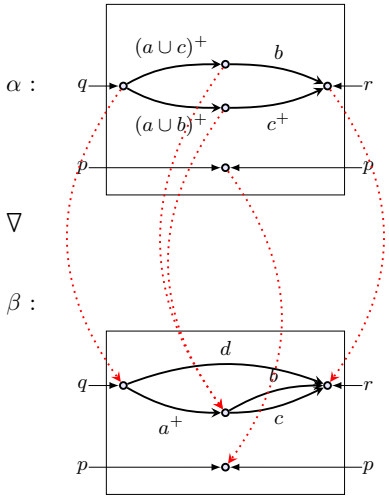


Figure 12 A box homomorphism.

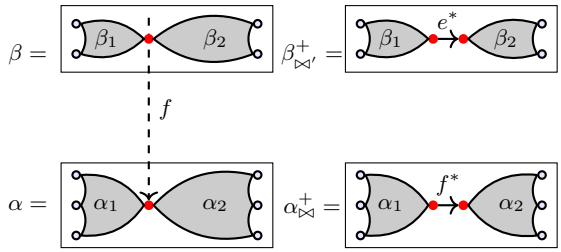


Figure 13 Bowtie compatible boxes.

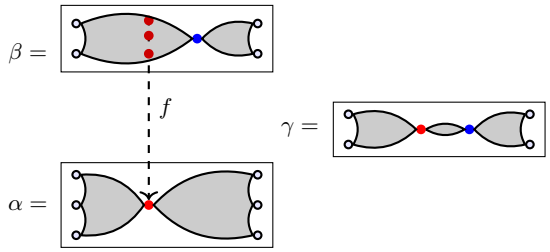


Figure 14 Case of bowtie incompatible boxes.

▶ **Proposition 6.** *If* Δ, Γ *are 1-1 templates such that* $\Delta \leq \Gamma$*, then* $\mathsf{KL}^- \vdash e(\Delta) \leq e(\Gamma)$.

Proof. When $\Delta \subseteq \Gamma$, it follows from Prop. 3 and Thm. 12; when $\Delta \lhd \Gamma$, it follows from Prop. 1. We conclude by transitivity. ◀

6.2 Synchronised product automaton (initialisation)

▶ **Definition 24** (2-Template automata (2-TA))**.** A *2-template automaton* is an NFA whose states are tuples of the form $\langle \tau, \eta, \sigma \rangle$ where τ, σ are types and $\eta : \overline{\sigma} \to \overline{\tau}$, whose alphabet is the set of pairs of templates, whose transitions are of the form $\langle \langle \sigma, \eta, \tau \rangle, \langle \Gamma, \Delta \rangle, \langle \sigma', \eta', \tau' \rangle \rangle$ where $\Gamma : \sigma \to \sigma'$, $\Delta : \tau \to \tau'$, and $\Gamma \leq_{\eta,\eta'} \Delta$, and with a single initial state and a single accepting state which consist of singleton types.

If $\mathcal{T}$ is a 2-TA, we denote by $\pi_1(\mathcal{T})$ (resp. $\pi_2(\mathcal{T})$) the automaton obtained by projecting the alphabet, the states and the transitions of $\mathcal{T}$ on the first (resp. last) component. Note that $\pi_1(\mathcal{T})$ and $\pi_2(\mathcal{T})$ are TA.

▶ **Definition 25** (Synchronised product of TA)**.** Let $\mathscr{E}, \mathscr{F}$ be two TA. The *synchronised product* of $\mathscr{E}$ and $\mathscr{F}$, written $\mathscr{E} \times \mathscr{F}$ is the 2-TA where $\langle \langle \tau, \eta, \sigma \rangle, \langle \Gamma, \Delta \rangle, \langle \tau', \eta', \sigma' \rangle \rangle$ is a transition of $\mathscr{E} \times \mathscr{F}$ iff $\langle \tau, \Gamma, \tau' \rangle$ is a transition of $\mathscr{E}$, $\langle \sigma, \Delta, \sigma' \rangle$ is a transition of $\mathscr{F}$ and $\Gamma \leq_{\eta,\eta'} \Delta$. (And with initial and accepting states defined from those of of $\mathscr{E}$ and $\mathscr{F}$.)

Note that we enforce refinement in the definition of this product, so that $\pi_1(\mathscr{E} \times \mathscr{F})$ is a sub-automaton of $\mathscr{E}$ and $\pi_2(\mathscr{E} \times \mathscr{F})$ is a sub-automaton of $\mathscr{F}$. Thus $\mathcal{G}(\pi_1(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{E})$ and $\mathcal{G}(\pi_2(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{F})$. When $\mathscr{E}, \mathscr{F}$ are TA coming from PA $\mathscr{A}, \mathscr{B}$ such that ${}^{\lhd}\mathcal{G}(\mathscr{A}) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{B})$, we can use the results from [11] about simulations to strengthen the first inclusion into an equality:

▶ **Theorem 26.** *Let* $\mathscr{A}, \mathscr{B}$ *be two PA,* $\mathscr{E}, \mathscr{F}$ *be basic TA such that* $\mathcal{G}(\mathscr{A}) = \mathcal{L}(\mathscr{E})$ *and* $\mathcal{G}(\mathscr{B}) = \mathcal{L}(\mathscr{F})$ *(given by Prop. 4). If* ${}^{\lhd}\mathcal{G}(\mathscr{A}_1) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{A}_2)$ *then:*

- $\mathcal{G}(\pi_1(\mathscr{E} \times \mathscr{F})) = \mathcal{G}(\mathscr{A})$;

- $\mathcal{G}(\pi_2(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{B})$.

Proof. The second point follows from the observation above. The first one comes from the simulation result ([11, Prop. 9.10]) for PA. Indeed, if ${}^{\lhd}\mathcal{G}(\mathscr{A}) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{B})$, then there is a simulation ([11, Def. 9.2]) between $\mathscr{A}$ and $\mathscr{B}$. This implies that for every run $\langle \tau_1, \Gamma_1, \tau_2, \dots, \Gamma_{n-1}, \tau_n \rangle$ of $\mathscr{E}$, there is a run $\langle \sigma_1, \Delta_1, \sigma_2, \dots, \Delta_{n-1}, \sigma_n \rangle$ of $\mathscr{F}$ and a set of mapping $\eta_i : \overline{\sigma_i} \to \overline{\tau_i}$, $i \in [1, n]$ such that $\Gamma_i \lhd_{\eta_i, \eta_{i+1}} \Delta_i$ for every $i \in [1, n-1]$. ◀

6.3 Maintaining refinement during reductions

Let us finally show that refinement is stable by composition, union, and iteration.

▶ **Theorem 27** (Stability of refinement by $\cdot$ and $\cup$)**.**
- *If* $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ *and* $\Delta_1 \leq_{\eta',\eta"} \Delta_2$ *then* $\Gamma_1 \cdot \Delta_1 \leq_{\eta,\eta"} \Gamma_2 \cdot \Delta_2$.
- *If* $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ *and* $\Delta_1 \leq_{\eta,\eta'} \Delta_2$ *then* $\Gamma_1 \cup \Delta_1 \leq_{\eta,\eta'} \Gamma_2 \cup \Delta_2$.

Proof. To show the first property it suffices to show the following results:

$$\text{If} \quad \Gamma_1 \lhd_{\eta,\eta'} \Gamma_2 \quad \text{and} \quad \Delta_1 \lhd_{\eta',\eta''} \Gamma_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \lhd_{\eta',\eta''} \Gamma_2 \cdot \Delta_2. \qquad (L_1)$$

$$\text{If} \quad \Gamma_1 \sqsubseteq \Gamma_2 \quad \text{and} \quad \Delta_1 \sqsubseteq \Delta_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \sqsubseteq \Gamma_2 \cdot \Delta_2. \qquad (L_2)$$

$$\text{If} \quad \Gamma_1 \lhd \Gamma_2 \quad \text{and} \quad \Delta_1 \sqsubseteq \Delta_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \, (\lhd \cdot \sqsubseteq)^* \, \Gamma_2 \cup \Delta_2. \qquad (L_3)$$

To show (L_1), consider a box $\alpha_1 \in \Gamma_1$ and $\beta_1 \in \Delta_1$. By hypothesis, there is a box $\alpha_2 \in \Gamma_2$ and an (η, η')-compatible homomorphism $h = \langle f, g \rangle$ from α_2 to α_1 and a box $\beta_2 \in \Delta_2$ and an (η', η'')-compatible homomorphism $h' = \langle f', g' \rangle$ from β_2 to β_1. Let $h'' = \langle f'', g'' \rangle$, where f'' equals f in $\mathrm{dom}(f)$ and f' in $\mathrm{dom}(f')$, and g'' equals g in $\mathrm{dom}(g)$ and g' in $\mathrm{dom}(g')$. Using (η, η')-compatibility of h and (η', η'')-compatibility of h', it is easy to show that h'' is an (η, η'')-compatible homomorphism from $\alpha_2 \cdot \beta_2$ to $\alpha_1 \cdot \beta_1$, which concludes the proof of (L_1). (L_2) follows easily from the definition of $\sqsubseteq$. For (L_3), note that $\Delta_1 \lhd \Delta_1$ (we choose the identity homomorphism), thus by (L_1), we have that $\Gamma_1 \cdot \Delta_1 \lhd \Gamma_2 \cdot \Delta_1$. By (L_2), we have that $\Gamma_2 \cdot \Delta_1 \sqsubseteq \Gamma_2 \cdot \Delta_2$, which concludes the proof.

To show the first property, we proceed by induction on the length of the sequences justifying that $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ and $\Delta_1 \leq_{\eta',\eta"} \Delta_2$, using $(L_1), (L_2)$ and (L_3) for the base cases.

To show the second property, we follow the same proof schema, showing results similar to $(L_1) - (L_3)$ where $\cdot$ is replaced by $\cup$. ◀

▶ Remark. Thm. 27 justifies our definition of $\leq_{\eta,\eta'}$. Indeed, a more permissive definition would seem natural, but the first property of Thm 27 would fail. For instance, if $\Gamma_1 \sqsubseteq \Gamma_2$ and $\Delta_1 \lhd_{\eta,\eta'} \Delta_2$, we do not have in general that $\Gamma_1 \cdot \Delta_1 \leq_{\eta,\eta'} \Gamma_2 \cdot \Delta_2$.

The main theorem of this section is Thm 28, stating that the refinement relation is stable under iteration. As its proof is very technical, we give only a proof sketch here, and leave the technical details to [13, App. B].

▶ **Theorem 28** (Stability of refinement by $\_^+$)**.** *If* $\Gamma \leq_{\eta,\eta} \Delta$ *then there are bowtie choices* $\bowtie, \bowtie'$ *and computation orders* $\preceq, \preceq'$*, for* Γ *and* Δ *respectively, such that:* $\Gamma^+_{\bowtie,\preceq} \leq_{\eta,\eta} \Delta^+_{\bowtie',\preceq'}$.

Proof sketch. To prove Thm. 28, it is enough to show the following properties.
- If $\Gamma \sqsubseteq \Delta$ then, for every bowtie choices $\bowtie, \bowtie'$, and every computation orders $\preceq, \preceq'$ for Γ and Δ respectively, we have that $\Gamma^+_{\bowtie,\preceq} \sqsubseteq \Delta^+_{\bowtie',\preceq'}$.
- If $\Gamma \lhd_{\eta,\eta} \Delta$ then there are two bowtie choices $\bowtie, \bowtie'$ and two computation orders $\preceq, \preceq'$, for Γ and Δ respectively, such that $\Gamma^+_{\bowtie,\preceq} \leq_{\eta,\eta} \Delta^+_{\bowtie',\preceq'}$.

The first property follows from $\mathcal{B}(\Gamma^+_{\bowtie,\preceq}) = \mathcal{B}(\Gamma)^+$ for every bowtie choice $\bowtie$ and order $\preceq$.

For the sake of clarity, we give here the proof of the second proposition in the case where Γ and Δ are singletons of atomic boxes $\{\alpha\}$ and $\{\beta\}$ respectively. The general case is treated in [13, App. B]. Let $\bowtie, \bowtie'$ be bowtie choices for α and β respectively, and let $h = \langle f, g\rangle$ be a homomorphism from β to α.

Let us first treat the case where $f^{-1}(\bowtie) = \{\bowtie'\}$ (we say that α, β are bowtie compatible). This is illustrated by the boxes α, β of Fig. 13, where the bowties are the red nodes. If we decompose α and β at the level of their bowties, we get $\alpha = \alpha_1 \cdot \alpha_2$ and $\beta = \beta_1 \cdot \beta_2$, where $\alpha_2 \cdot \alpha_1$ and $\beta_2 \cdot \beta_1$ are 1-1 boxes. We write $e = e\,(\alpha_2 \cdot \alpha_1)$ and $f = e\,(\beta_2 \cdot \beta_1)$. The boxes $\alpha^+_{\bowtie}$ and $\beta^+_{\bowtie'}$ are depicted in Fig. 13. Let us show that there is a homomorphism from $\beta^+_{\bowtie'}$ to $\alpha^+_{\bowtie}$. The homomorphism h induces a homomorphism h_1 from β_1 to α_1 and a homomorphism h_2 from β_2 to α_2 ([13, App. B, Lem. 42]). Combining h_1 and h_1, we get almost a homomorphism from $\beta^+_{\bowtie'}$ to $\alpha^+_{\bowtie}$ (See Fig. 13), we need only to show that $\mathsf{KL}^- \vdash e \leq f$. But this follows from Prop. 6: indeed, we can combine h_1 and h_2 to get a homomorphism from $\beta_2 \cdot \beta_1$ to $\alpha_2 \cdot \alpha_1$. We have thus that $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \beta^+_{\bowtie'}$ ((η,η)-compatibility is easy).

Let us now treat the case where $N := f^{-1}(\bowtie)$ is not necessarily $\{\bowtie'\}$ (N is illustrated by the red node of β in Fig. 14). Let γ be the box obtained from β by merging the nodes N (see Fig. 14). There are two bowtie choices for γ: a bowtie $\bowtie_b$ inherited from β (blue in Fig. 14) and a bowtie $\bowtie_r$ coming from the nodes of N (red in Fig. 14).

Let h' be the homomorphism from β to γ that maps each node (and each edge) to itself, except for the nodes of N which are mapped to $\bowtie_r$. If we consider the bowtie $\bowtie_b$ for γ, then β and γ are bowtie compatible w.r.t. to h', thus $\gamma^+_{\bowtie_b} \lhd \beta^+_{\bowtie'}$ using the previous case.

Let h'' be the homomorphism from γ to α, which is exactly h except that it maps the node $\bowtie_r$ to the bowtie $\bowtie$ of α. If we consider the bowtie $\bowtie_r$ for γ, then γ and α are bowtie compatible w.r.t. h'', thus $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \gamma^+_{\bowtie_r}$ using the previous case again.

Notice finally that $\gamma^+_{\bowtie_r} \sqsubseteq \gamma^+_{\bowtie_b}$. To sum up, we have: $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \gamma^+_{\bowtie_r} \sqsubseteq \gamma^+_{\bowtie_b} \lhd \beta^+_{\bowtie'}$. ◀

The last case in this proof explains the need to work with refinement ($\leq$) rather than just homomorphisms ($\lhd$): when starting from templates that are related by homomorphism and iterating them, the templates we obtain are not necessarily related by a single homomorphism, only by a sequence of homomorphisms and inclusions.

7 Future work

We have proven that KL^- axioms are sound and complete w.r.t. the relational models of identity-free Kleene lattices, and thus also w.r.t. their language theoretic models, by the results from [2].

Whether one can obtain a finite axiomatisation in presence of identity remains open. This question is important since handling the identity relation is the very first step towards handling *tests*, which are crucial in order to model the control flow of sequential programs precisely (*e.g.*, as in Kleene algebra with tests [20]).

An intermediate problem, which is still open to the best of our knowledge, consists in finding an axiomatisation for the fragment with composition, intersection and identity (not including transitive closure) [3, see errata available online].

References

1 C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: semantic foundations for networks. In *POPL*, pages 113–126. ACM, 2014. doi:10.1145/2535838.2535862.

2 H. Andréka, S. Mikulás, and I. Németi. The equational theory of Kleene lattices. *Theoretical Computer Science*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.

3 H. Andréka and Sz. Mikulás. Axiomatizability of positive algebras of binary relations. *Algebra Universalis*, 66(1):7–34, 2011. doi:10.1007/s00012-011-0142-3.

4 H. Andréka. Representation of distributive lattice-ordered semigroups with binary relations. *Algebra Universalis*, 28:12–25, 1991.

5 H. Andréka and D.A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.

6 A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, CS Dpt., Cornell University, July 2001. URL: http://hdl.handle.net/1813/5831.

7 A. Armstrong, G. Struth, and T. Weber. Programming and automating mathematics in the Tarski-Kleene hierarchy. *Journal of Logical and Algebraic Methods in Programming*, 83(2):87–102, 2014. doi:10.1016/j.jlap.2014.02.001.

8 Maurice Boffa. Une condition impliquant toutes les identités rationnelles. *Informatique Théorique et Applications*, 29(6):515–518, 1995. URL: http://archive.numdam.org/.../ITA_1995__29_6_515_0.pdf.

9 Thomas Braibant and Damien Pous. Deciding Kleene algebras in Coq. *Logical Methods in Computer Science*, 8(1):1–16, 2012. doi:10.2168/LMCS-8(1:16)2012.

10 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS*, pages 68–79. ACM, 2015. doi:10.1109/LICS.2015.17.

11 Paul Brunet and Damien Pous. Petri automata. *Logical Methods in Computer Science*, Volume 13, Issue 3, 2017. doi:10.23638/LMCS-13(3:33)2017.

12 J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.

13 Amina Doumane and Damien Pous. Completeness for identity-free kleene lattices. Full version of this extended abstract, available at https://hal.archives-ouvertes.fr/hal-01780845, 2018. URL: https://hal.archives-ouvertes.fr/hal-01780845.

14 S. Foster, G. Struth, and T. Weber. Automated engineering of relational and algebraic methods in Isabelle/HOL - (invited tutorial). In *RAMiCS*, volume 6663 of *LNCS*, pages 52–67. Springer, 2011. doi:10.1007/978-3-642-21070-9_5.

15 P.J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland. Elsevier, 1990.

16 C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In *CONCUR*, volume 5710 of *LNCS*, pages 399–414. Springer, 2009. doi:10.1007/978-3-642-04081-8_27.

17 P. Höfner and G. Struth. On automating the calculus of relations. In *IJCAR*, volume 5195 of *LNCS*, pages 50–66. Springer, 2008. doi:10.1007/978-3-540-71070-7_5.

18 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent kleene algebra: Free model and completeness. In *ESOP*, volume 10801 of *LNCS*, pages 856–882. Springer, 2018. doi:10.1007/978-3-319-89884-1_30.

19 D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

20 D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.

21 D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

22 D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *CL2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582. Springer, 2000. doi:10.1007/3-540-44957-4_38.

23 Dexter Kozen. Typed Kleene algebra. Technical Report TR98-1669, CS Dpt., Cornell University, 1998. URL: http://www.cs.cornell.edu/~kozen/papers/typed.pdf.

24 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in nominal kleene algebra. *J. Log. Algebr. Meth. Program.*, 91:17–32, 2017. doi:10.1016/j.jlamp.2017.06.002.

25 A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Algebraic Reasoning*, 49(1):95–106, 2012. doi:10.1007/s10817-011-9223-4.

26 D. Krob. Complete systems of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.

27 Michael R. Laurence and Georg Struth. Completeness theorems for pomset languages and concurrent kleene algebras. *CoRR*, abs/1705.05896, 2017. arXiv:1705.05896.

28 Damien Pous. Kleene Algebra with Tests and Coq tools for while programs. In *ITP*, volume 7998 of *LNCS*, pages 180–196. Springer, 2013. doi:10.1007/978-3-642-39634-2_15.

29 V. R. Pratt. Dynamic algebras and the nature of induction. In *STOC*, pages 22–28. ACM, 1980. doi:10.1145/800141.804649.

30 Volodimir Nikiforovych Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964.

31 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *STOC*, pages 1–12. ACM, 1979. doi:10.1145/800135.804393.

Reachability in Parameterized Systems: All Flavors of Threshold Automata

Jure Kukovec
TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria
jkukovec@forsyte.at

Igor Konnov
University of Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
igor.konnov@inria.fr
https://orcid.org/0000-0001-6629-3377

Josef Widder
TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria
widder@forsyte.at
https://orcid.org/0000-0003-2795-611X

Abstract

Threshold automata, and the counter systems they define, were introduced as a framework for parameterized model checking of fault-tolerant distributed algorithms. This application domain suggested natural constraints on the automata structure, and a specific form of acceleration, called single-rule acceleration: consecutive occurrences of the same automaton rule are executed as a single transition in the counter system. These accelerated systems have bounded diameter, and can be verified in a complete manner with bounded model checking.

We go beyond the original domain, and investigate extensions of threshold automata: non-linear guards, increments and decrements of shared variables, increments of shared variables within loops, etc., and show that the bounded diameter property holds for several extensions. Finally, we put single-rule acceleration in the scope of flat counter automata: although increments in loops may break the bounded diameter property, the corresponding counter automaton is flattable, and reachability can be verified using more permissive forms of acceleration.

2012 ACM Subject Classification Software and its engineering → Software verification, Theory of computation → Logic and verification, Software and its engineering → Software fault tolerance

Keywords and phrases threshold & counter automata, parameterized verification, reachability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.19

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); and by the Vienna Science and Technology Fund (WWTF) via project APALACHE (ICT15-103).

Acknowledgements We thank the anonymous reviewers for spotting a few technical omissions in the preliminary version of this paper.

1 Introduction

Threshold automata were introduced as a framework for modeling and verification [23, 25, 24, 22] and recently for synthesis [29] of fault-tolerant distributed algorithms. These algorithms typically wait for a quorum of messages, e.g., in replication services, the primary replica may block until it received acknowledgments from a majority of the back-up replicas [28, 33, 14, 34].

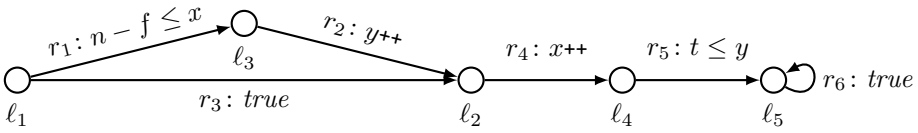


Figure 1 A threshold automaton.

Moreover, these algorithms are parameterized by design, i.e., the number of processes n is a parameter, and consequently, the primary in our example contains a guard that waits for more than $n/2$ messages, a so-called threshold guard. As a result, the local transition relation is parameterized, and therefore these systems are out of reach of classic work in parameterized model checking [15, 7].

We recall the necessary notions of threshold automata by the example in Figure 1. It operates on parameters n, t, and f, and shared variables x and y. The vertices are called locations, and the edges are called rules, which can be guarded, and can increase a shared variable. For instance, the threshold guard in rule r_1 compares the value of variable x to a linear expression over parameters $n - f$. The semantics of threshold automata is defined via counter systems, where a configuration contains the values of shared variables, and a counter value κ_i for each location ℓ_i. The transition relation then is defined by operations on the counters and shared variables. For instance, for some c, if $\kappa_2 \geq c$, then there is a transition defined by rule r_4 that increases κ_4 by c, decreases κ_2 by c and increases x by c.

By allowing arbitrary values of factor c, one obtains a transition relation with a specific form of acceleration (single-rule acceleration), built-in by construction. Then, the transition system is a graph with vertices being configurations, and edges being transitions. By defining paths in this graph, and distances between vertices, one can define the diameter of a transition system. If the diameter d is bounded, then every state is reachable in d steps, and bounded model checking of executions of lengths up to d is a complete verification method for reachability [6]. It was shown [25] that the diameter of transition systems defined by threshold automata is bounded, and in particular, it does not depend on the values of the parameters such as n, t, and f. However, several restrictions on threshold automata were used in [25] to bound the diameter. While these restrictions are well-justified for the original domain of fault-tolerant algorithms, two questions remain open: (i) which of these restrictions were actually necessary to prove the results under single-rule acceleration, and (ii) which restrictions could be avoided by allowing a more permissive form of acceleration?

The purpose of this paper is to explore various extensions of threshold automata, and understand which of them maintain a bounded diameter. We study extensions of the following properties of threshold automata as defined in [25]:

Increments in loops. Canonical threshold automata defined in [25] do not allow updates of shared variables within loops.

Guards. In [25], threshold guards compare shared variables to a threshold, that is, a linear combination of parameters. Since parameter values are fixed in a run, thresholds are effectively constant. As shared variables can only increase in [25], the guards are monotonic; for instance, once the shared variable is greater than the threshold it stays greater, and the evaluation of the guards stays unchanged after that. We consider more general guards: we replace the shared variables (e.g., x) by a function over shared variables, and consider the special case of a difference $(x - y)$, as well as piecewise monotone functions.

Table 1 Summary of results. "p.m. $f(x)$" means a piecewise monotone function of x.

Level	Reversals	Canonical	Bounded diameter?	Flattable?	Decidable reachability?	Class name
x	0	✓	[25, Thm. 8] ✓	✓	✓	TA
p.m. $f(x)$	0	✓	Cor. 18 ✓	✓	✓	PMTA
x	$\leq k$	✓	[27, Thm. 4] ✓	✓	✓	rbTA
x	0	✗	Thm. 9 ✗	Thm. 24 ✓	✓	NCTA
$x - y$	0	✓	✗	✗	Thm. 11 ✗	BDTA
x	∞	✓	✗	✗	Thm. 10 ✗	rTA

Reversibility. In [25], only increments on shared variables are considered because increments are sufficient to model sending a message. As a result, threshold guards were monotone. In this paper, we also consider decrements, which produce schedules that have alternating periods of increasing a variable and decreasing it.

For these extensions, we show that under certain conditions these automata entail bounded diameter results as well. Thus, the diameter result of [25] can be seen as a special case of the results of this paper.

Finally, we consider threshold automata in the scope of counter automata, a modeling framework for infinite-state systems [10, 30, 4]. We consider the concepts of (i) a *flat* counter automaton, whose control graph does not contain nested loops, and (ii) a *flattable* counter automaton, for which a flat counter automaton with the same reachability relation exists. For these automata, there are procedures and tools (FAST) for reachability analysis [30, 4]. We will discuss that the results of [25, 21] imply that canonical threshold automata (no increments in loops) entail flattable counter automata – which explains why FAST verified some benchmarks in the experiments of [25]. Moreover, we show that we can get rid of the canonicity restriction and still prove that the resulting counter automaton is flattable. That is, while non-canonical threshold automata do not fall into the fragment that can be verified with the methods from [25, 21], one can still analyze these automata with more permissive forms of acceleration as implemented in FAST.

An overview of our results is in Table 1, where the simpler classes are at the top; these classes are defined in Section 2.3. The bounded diameter property implies flattability, as we show in Proposition 21, which can be seen in the first three lines. For completeness, in line 3, we mention results on reversal-bounded threshold automata rbTA, which consider the structure of runs rather than threshold automata [27]. Note that flattability of a counter automaton implies that reachability for this automaton is decidable [30].

2 System model

This section generalizes the definitions of [25]. We use the following sets: integers $\mathbb{Z}$ and their extension $\mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, +\infty\}$, non-negative integers $\mathbb{N}_0$, reals $\mathbb{R}$. We denote a vector of integers by $\vec{x}$. When the vector dimension is clear, we write $\vec{1}_k$ to denote the unit vector that has 1 at position k and 0 everywhere else, and $\vec{0}$ is the vector filled with zeroes.

2.1 Unrestricted threshold automata

An *unrestricted threshold automaton* (UTA) is a tuple $(\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$ where $\mathcal{L}$ is a finite set of *local states* (locations), $\mathcal{I} \subseteq \mathcal{L}$ is a set of *initial local states*, Γ is a finite set of *shared variables*, Π is a finite set of *parameter variables*, and $\mathcal{R}$ is a finite set of *rules*, which are defined below.

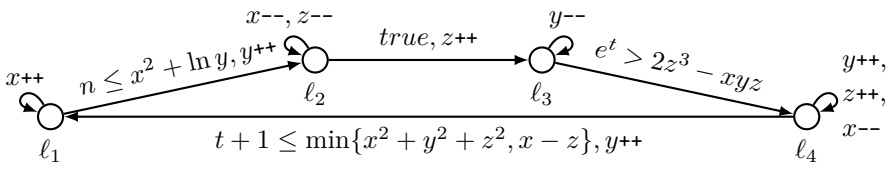


Figure 2 An unrestricted threshold automaton.

Guards. A *nonlinear guard* of a UTA is a formula: $thd(\vec{p}) \bowtie lvl(\vec{x})$, where $\vec{p} = [p_1, \ldots, p_{|\Pi|}]$, $\vec{x} = [x_1, \ldots, x_{|\Gamma|}]$, $lvl\colon \mathbb{Z}^{|\Gamma|} \to \mathbb{R}$ is the *level function*, $thd\colon \mathbb{Z}^{|\Pi|} \to \mathbb{R}$ is the *threshold function*, and $\bowtie$ is one of $\{<, \leq, >, \geq\}$. When $\bowtie$ is either $<$ or $\leq$, the guard is called a *lower guard*, otherwise it is an *upper guard*. For $x \in \Gamma$ and $a_0, a_1, \ldots, a_{|\Pi|} \in \mathbb{Z}$, a guard of the following form is called *affine*: $a_0 + \sum_{i=1}^{|\Pi|} a_i p_i \bowtie x$. (Affine guards have only one shared variable.)

Rules. A *rule* is a tuple $(from, to, \Phi, \vec{u})$ where $to, from \in \mathcal{L}$ are two local states, Φ is a set of nonlinear guards and $\vec{u} \in \mathbb{Z}^{|\Gamma|}$ is an update vector.

▶ **Example 1.** Consider the automaton in Figure 2, demonstrating the nonlinear guards and rules that are *not* considered in [25]. ◁

2.2 Semantics of UTA: counter systems

Configurations. For a UTA $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$, a triple of vectors $(\vec{\kappa}, \vec{g}, \vec{p}) \in \mathbb{N}_0^{|\mathcal{L}|} \times \mathbb{Z}^{|\Gamma|} \times \mathbb{N}_0^{|\Pi|}$ is called a *configuration*. The vectors have the following meaning: vector $\vec{\kappa} \in \mathbb{N}_0^{|\mathcal{L}|}$ stores the values of the location counters, vector $\vec{g} \in \mathbb{Z}^{|\Gamma|}$ stores the values of the shared variables, and vector $\mathbb{N}_0^{|\Pi|}$ stores the parameters.

Transitions. Given a UTA $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$, a *transition* is a pair $(rule, factor)$ where $rule \in \mathcal{R}$ and $factor \in \mathbb{N}_0$. Note that the single-rule acceleration is built into to the definition of a transition, by allowing $factor > 1$. We use the notation $t.rule$ and $t.factor$ to refer to the tuple elements of the same name. Additionally, for any tuple field e of $t.rule$ we shorten $t.rule.e$ to $t.e$ for brevity (e.g., $t.rule.from$ becomes $t.from$).

Given a configuration σ and a formula φ over the shared variables Γ and parameters Π, we will use the notation $(\sigma.\vec{g}, \sigma.\vec{p}) \models \varphi$, or just $\sigma \models \varphi$, to mean that the formula φ holds true when the shared variables and the parameters are substituted with their respective values from $\sigma.\vec{g}$ and $\sigma.\vec{p}$.

We say that a *rule* r *is unlocked* in a configuration σ if $(\sigma.\vec{g}, \sigma.\vec{p}) \models \bigwedge_{\varphi \in r.\Phi} \varphi$. Further, a transition $t = (r, a)$ is unlocked in a configuration σ if r remains unlocked after at least $a - 1$ updates imposed by $r.\vec{u}$, that is, for each $k \in \{0, 1, \ldots, a-1\}$, it holds that $(\sigma.\vec{g} + k \cdot r.\vec{u}, \sigma.\vec{p}) \models \bigwedge_{\varphi \in r.\Phi} \varphi$.

▶ **Definition 2.** A transition $t = (r, a)$ is *applicable to* a configuration σ if t is unlocked in σ and $\sigma.\vec{\kappa}[r.from] \geq a$. When t is applicable to σ, we call σ' the result of applying t to σ – denoted as $t(\sigma)$ – if the requirements 1–3 are met:

1. the location counters are changed by a, that is, $\sigma'.\vec{\kappa} = \sigma.\vec{\kappa} + a \cdot (\vec{1}_{r.to} - \vec{1}_{r.from})$,
2. the update vector is added a times to the shared variables: $\sigma'.\vec{g} = \sigma.\vec{g} + a \cdot r.\vec{u}$,
3. the parameters do not change: $\sigma'.\vec{p} = \sigma.\vec{p}$.

Definition 2 explicitly allows successive applications of the same rule to be compressed into a single transition. This kind of acceleration was introduced in [25], and we call it *single-rule* acceleration.

▶ **Example 3.** Consider the automaton in Figure 1. The following table shows a configuration σ_0, and configurations σ_1 and σ_2 after applying one and two transitions, respectively, to the configuration σ_0:

configuration	counters $\vec{\kappa}$	shared variables $\vec{g}$	parameters $\vec{p}$
σ_0	$(\mathbf{4}, 0, 0, 0, 0)$	$(0, 0)$	$(4, 1, 1)$
σ_1	$(\mathbf{2}, \mathbf{2}, 0, 0, 0)$	$(0, 0)$	$(4, 1, 1)$
σ_2	$(2, \mathbf{1}, 0, \mathbf{1}, 0)$	$(\mathbf{1}, 0)$	$(4, 1, 1)$

First, the parameters are initialized to $n = 4, t = f = 1$, and the counter of location ℓ_1 equals to n (configuration σ_0). Then, transition $(r_3, 2)$ is applied to σ_0, resulting in the counter of ℓ_2 increasing by 2 and the counter of ℓ_1 decreasing by 2 in configuration σ_1. Finally, rule r_4 is executed once, incrementing x to obtain σ_2. ◁

Number of instances. As in [25], we assume that a threshold automaton is equipped with a function $N : \mathbb{N}_0^{|\Pi|} \to \mathbb{N}_0$. Intuitively, every configuration σ captures a state of $N(\sigma.\vec{p})$ instances of the threshold automaton. The authors of [25] did not restrict function N, as they were concerned only with the length of the shortest sequences of transitions connecting any two configurations. In this paper, we assume that the relation $\{(\vec{p}, N(\vec{p})) \colon \vec{p} \in \mathbb{N}_0^{|\Pi|}\}$ can be defined with a formula in Presburger arithmetic. In Example 3, we define N with the following formula over the parameters n, t, and f as well as the outcome of the function N: $(n > 3t \to N = n - f \wedge f \geq 0 \wedge t \geq 0) \wedge (n \leq 3t \to N = 0)$.

In our example, the number N is positive only if $n > 3t$, and equals to zero otherwise. This allows us to prune "irrelevant" parameter values. (In distributed computing, this is achieved by writing a so-called *resilience condition*.)

Counter systems. Having defined the configurations and transitions, we define a counter system of a threshold automaton:

▶ **Definition 4.** Given a UTA $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, R)$, we define its *counter system* $CS(A)$ as a transition system (Σ, I, R), where:

- Σ is the set of all possible *configurations*.
- $I \subseteq \Sigma$ is the set of *initial configurations*; their counter values in the initial locations sum up to $N(\vec{p})$. Formally, a configuration $\sigma_0 \in \Sigma$ belongs to I if and only if the following conditions hold: $\sigma_0.\vec{\kappa}[\ell] = 0$ for $\ell \in \mathcal{L} \setminus \mathcal{I}$ and $N(\sigma_0.\vec{p}) = \sum_{\ell \in \mathcal{I}} \sigma_0.\vec{\kappa}[\ell]$, as well as, $\sigma_0.\vec{g} = \vec{0}$.
- $R \subseteq \Sigma \times \Sigma$ is the *transition relation*. A pair of configurations (σ, σ') belongs to R if and only if there is a transition t that is applicable to σ, and $\sigma' = t(\sigma)$.

A *schedule* is a finite sequence of transitions. A schedule $\tau = t_1, \ldots, t_m$ is *applicable* to a configuration σ_0 if there exists a sequence of configurations $\sigma_1, \ldots, \sigma_m$ where $\sigma_i = t_i(\sigma_{i-1})$ for all $0 < i \leq m$. We define $\tau(\sigma_0)$ to be σ_m. We denote the concatenation of schedules τ and τ' by $\tau \cdot \tau'$ and the length of a schedule $\tau = t_1, \ldots, t_m$ as $|\tau| = m$. By ϵ, we refer to the empty schedule, which has length 0 and satisfies $\epsilon(\sigma) = \sigma$ for all σ in Σ.

For a schedule $\tau = t_1, \ldots, t_n$ and two indices $i, j \in \mathbb{Z}$, we define the subschedule $\tau_{[i,j]}$ as follows ($\tau_{[i,j)}$, $\tau_{(i,j]}$, and $\tau_{(i,j)}$ are obtained by choosing the intervals accordingly):

$$\tau_{[i,j]} = \begin{cases} t_{\max(1,i)}, \ldots, t_{\min(n,j)}, & \text{when } i \leq j, \\ \epsilon, & \text{when } i > j \end{cases}$$

We say that a configuration σ' is *reachable* from a configuration σ, if there is a schedule τ with the following properties: (1) τ is applicable to σ, and (2) $\tau(\sigma) = \sigma'$.

Bounded diameters. The central result of [25] is that for counter systems of threshold automata one can check, whether one configuration is reachable from another. It is sufficient to inspect the schedules of length within a precomputed bound on the diameter:

▶ **Definition 5.** Given a UTA A and its counter system $CS(A) = (\Sigma, I, R)$, a number $d \in \mathbb{N}_0 \cup \{\infty\}$ is the *diameter* of $CS(A)$ if d is the smallest number with the following property:

For every pair of configurations $\sigma, \sigma' \in \Sigma$, if σ' is reachable from σ, then there is a schedule τ such that: (a) τ is applicable to σ, (b) $\tau'(\sigma) = \sigma'$, and (c) $|\tau'| \leq d$.

One of our contributions is in finding fragments of unrestricted threshold automata whose counter systems have a bounded diameter. In Section 4, we give examples of UTA whose counter systems have unbounded diameters. Moreover, we show that there are classes of UTA, for which the following problem – which generalizes the problem from [21] – is undecidable:

Parameterized reachability. Given a UTA $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, R)$, a *state property* B is a Boolean combination of formulas that have the form $\vec{\kappa}[\ell] = 0$, for some $\ell \in \mathcal{L}$. The *parameterized reachability* problem is to decide whether there are parameter values $\vec{p} \in \mathbb{N}_0^{|\Pi|}$, an initial configuration $\sigma_0 \in I$, with $\sigma_0.\vec{p} = \vec{p}$, and a schedule τ, such that τ is applicable to σ_0, and property B holds in the final state: $\tau(\sigma_0) \models B$.

2.3 Fragments of unrestricted threshold automata

In order to prove the bounded diameter property, we consider various restrictions on the guards, updates, the transition relation, and other aspects of UTA. The first restriction prohibits modifications of shared variables in loops [25]:

▶ **Definition 6.** A rule r lies on a cycle, if there is a sequence of rules $r_0, \ldots, r_k$, where $r = r_0$ and $r_i.to = r_j.from$ for $0 \leq i \leq k$ and $j = i + 1 \mod (k+1)$.
A UTA is *canonical* if $r.\vec{u} = \vec{0}$ for every rule $r \in \mathcal{R}$ that lies on a cycle.

Canonical Threshold Automata (TA). This class contains UTAs with the following properties: (1) they are canonical, (2) all guards are affine, and (3) the update vectors in all rules are non-negative. This is the class of automata considered in [25, 24], which is known to have a bounded diameter:

▶ **Theorem 7** ([25]). *For every TA A, there exists a constant $\mathcal{C}$, such that the diameter of the associated counter system is less than or equal to $d(CS(A)) = (\mathcal{C}+1) \cdot |\mathcal{R}| + \mathcal{C}$ (independently of the parameters).*

Piecewise Monotone Threshold Automata (PMTA). This class contains UTAs with the following properties: (1) they are canonical, (2) all level functions in the guards are piecewise monotone[1], and (3) the update vectors in all rules are non-negative.

Bounded Difference Threshold Automata (BDTA). This class contains UTAs with the following properties: (1) they are canonical, (2) all level functions in the guards are of the form x_i or $x_i - x_j$ for some $x_i, x_j \in \Gamma$, and (3) the update vectors in all rules are non-negative.

[1] The domain of a piecewise monotone function can be decomposed into finitely many intervals where the function is monotone.

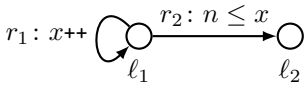


Figure 3 A simple NCTA.

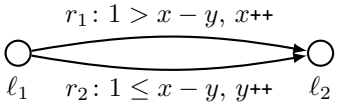


Figure 4 A BDTA with unbounded diameter.

Non-canonical generalizations of TA, PMTA, and BDTA. For the mentioned classes, we omit the requirement of the automaton being canonical, and denote these classes as: NCTA, NCPMTA, and NCBDTA.

Reversible of TA, PMTA, and BDTA. For the mentioned classes, we allow shared variables to be both increased and decreased, and denote these classes as: rTA, rPMTA, and rBDTA.

Reversal-bounded extensions of TA, PMTA, and BDTA. To introduce reversal-bounded automata, we need the following definition.

▶ **Definition 8.** A schedule $t_1 \cdot \tau \cdot t_2$ is an x-reversal if: (a) one of the transitions t_1 or t_2 increases x and the other decreases x, that is, $t_1.\vec{u}[x] \cdot t_2.\vec{u}[x] < 0$, and (b) every transition t in τ does not update x, that is, $t.\vec{u}[x] = 0$. If for every shared variable x, the number of x-reversals in a schedule is at most N, the schedule is called N-*reversible*.

Similar to reversal-bounded counter machines [20], we define the classes rbTA, rbPMTA, and rbBDTA by restricting the counter systems of the respective reversible automata to N-reversible schedules (where N is fixed).

3 Negative results: unbounded diameters and undecidability

We give examples of NCTA and BDTA whose counter systems have unbounded diameters. Then, we show that reachability is undecidable for counter systems of BDTA and rTA.

3.1 Unbounded diameters of non-canonical threshold automata

When we permit shared variables to be updated within loops, the diameter of the counter system becomes unbounded:

▶ **Theorem 9.** *There is an NCTA whose counter system has unbounded diameter.*

Proof. Figure 3 shows such an NCTA, where x is the only shared variable, and n the only parameter. To prove the theorem, take the configuration σ with $\sigma.\vec{\kappa} = (1, 0)$, $\sigma.\vec{g} = (0)$, and $\sigma.\vec{p} = (n)$ for $n > 0$. We show that the following configuration σ' can be reached from σ in no less than $n + 1$ transitions: $\sigma'.\vec{\kappa} = (0, 1)$, $\sigma'.\vec{g} = (n)$, and $\sigma'.\vec{p} = (n)$. In σ, rule r_2 is locked, and rule r_1 is not, so r_1 must be used at least n times to unlock r_2. Since the sum of the values of location counters initially is 1 and is invariant, we can only use transitions with a factor of at most 1. Thus, to reach σ' from σ, we have to execute n copies of the transition $(r_1, 1)$ and then the transition $(r_2, 1)$. Hence, the diameter must be at least $n + 1$, and thus grows with the unbounded parameter n. ◀

The automaton in Figure 3 encodes the simple loop "`while (n <= x) x++;`" One can argue that this automaton can be accelerated by compressing self-loops into one transition; which requires another form of acceleration. Figure 5 shows an example that cannot be easily fixed by this. This example can be treated with more general acceleration techniques, as demonstrated in Section 6.

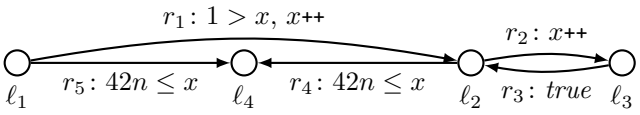


Figure 5 A non-canonical automaton with unbounded diameter.

3.2 Undecidability for reversible and bounded-difference automata

We show even stronger results for rTA and BDTA: reachability is undecidable and thus counter systems of such automata cannot be analyzed with any form of acceleration.

▶ **Theorem 10.** *Parameterized reachability for counter systems of loop-free rTA is undecidable.*

Proof. We use rTA to encode two-counter machines 2CM, for which the halting problem is undecidable [32]. A command of a 2CM is a triple (`from`, `cmd`, `to`) where `from` and `to` are labels from the set $\{1, \ldots, m\}$ for some m, and `cmd` is one of the operations: `inc` x, `dec` x, `inc` y, `dec` y, `zero?` x, `zero?` y. The label m designates the halting command. For the two counters we use two shared variables x and y. For each label i we also add a shared variable at_i, that we use as a Boolean flag to indicate whether the 2CM currently is at label i. There is also a shared variable *init*, which is used for initialization.

It remains to encode the control structure of a 2CM (which may contain loops) in a threshold automaton without loops. Our rTA has three locations ℓ_0, ℓ_1, ℓ_2, where ℓ_0 is the initial one. First, we introduce a special initialization rule from ℓ_0 to ℓ_1 that is guarded with $init < 1$ and increments *init*. Second, for each command we introduce a rule from ℓ_0 to ℓ_1. For command (i, cmd, j), the rule is guarded with $at_i > 0 \land init \geq 1 \land init < 2$, and e.g., $0 \geq x \land 0 \leq x$, if the test for zero is needed. The update of the rule contains at_i`--` and at_j`++` (goto label j from label i), and the required increment/decrement of a counter as e.g., x`++` or y`--`. Third, the last rule detects that the 2CM halted: it goes from ℓ_0 to ℓ_2 and is guarded with $at_m \geq 1$.

The number of instances is $N(n) = n + 2$ for the only parameter n. Thus, n steps of the 2CM are modeled by $n + 1$ transitions of the constructed counter system; the $(n + 2)$th transition may move at least one automaton to the location ℓ_2. Hence, the counter system simulates arbitrarily many steps of the 2CM. We ask the parameterized reachability question of whether the counter system reaches a configuration σ with $\sigma.\vec{\kappa}[\ell_2] \neq 0$ (for some value of n). A positive answer is given *if and only if* the 2CM halts; undecidability follows. ◀

Now we consider BDTA. Figure 4 shows an example of a BDTA whose counter system has unbounded diameter: Every schedule allowed by this threshold automaton is an alternating sequence of the transitions $(r_1, 1)$ and $(r_2, 1)$. Thus to increase the counter κ_2 to n, we require a schedule of length n, which is an unbounded parameter. This shows that sinlge-rule acceleration does not help us to analyze BDTA. In fact, no form of acceleration helps:

▶ **Theorem 11.** *Parameterized reachability for counter systems of loop-free BDTA is undecidable.*

The proof goes along the same lines as the proof of Theorem 10. The only complication is to encode a decrement of a shared variable by using increments and bounded differences. To this end, for each variable x, we introduce two shared variables x_1 and x_2. The difference $x_1 - x_2$ simulates a counter x. Whenever x has to be decremented, we increment x_2, and when x has to be incremented, we increment x_1. A test $x = 0$ is simulated as a conjunction $0 \geq x_1 - x_2 \land 0 \leq x_1 - x_2$.

4 Positive results: bounding the diameter

We extend the framework and the proofs of [25] to prove the bounded diameter property for certain fragments of UTAs. A key observation in [25] is that if shared variables are only increased, then the evaluation of every (affine) threshold guard changes at most once in a schedule. This argument obviously applies even if increments occur in loops:

▶ **Proposition 12** (Monotonicity of affine guards). *For an NCTA configuration σ, if a transition t is applicable to σ, then the following holds:*

1. *For a lower affine guard φ:*
   (a) If $\sigma \models \varphi$, then $t(\sigma) \models \varphi$, and (b) if $t(\sigma) \not\models \varphi$, then $\sigma \not\models \varphi$.
2. *For an upper affine guard φ:*
   (c) If $\sigma \not\models \varphi$ then $t(\sigma) \not\models \varphi$, and (d) if $t(\sigma) \models \varphi$, then $\sigma \models \varphi$.

4.1 A sufficient condition for diameter boundedness

Proposition 12 does not apply to unrestricted threshold automata for two reasons: First, NCTA only allow shared variables to be incremented, whereas UTA allow both increments and decrements. Obviously, an affine threshold guard such as $n \leq x$ can change its evaluation arbitrary many times, if increments and decrements of x are alternated (as parameter n is constant in a schedule). Second, even if we restrict updates of shared variables to non-negative vectors, guards such as $0 \leq x - y$ can change their evaluations arbitrarily often in a single schedule (cf. Theorem 11).

Proposition 12 implies that for every (affine) guard φ, when a schedule τ is applied to a configuration σ, schedule τ can be split into two intervals: $\tau_{[1,k)}$ and $\tau_{[k,|\tau|]}$ with the following property: $\tau_{[1,i)}(\sigma) \models \varphi$ iff $\sigma \models \varphi$ for $1 \leq i \leq k$, and $\tau_{[1,j]}(\sigma) \not\models \varphi$ iff $\tau(\sigma) \not\models \varphi$ for $k \leq j \leq |\tau|$. In other words, the evaluation of φ may only change in the transition from $\tau_{[1,k-1]}(\sigma)$ to $\tau_{[1,k]}(\sigma)$. We extend this idea to non-linear guards by requiring the guards to preserve their evaluations in a bounded number of intervals. In face of Theorem 11, we thus impose two restrictions on UTA: (1) we allow only non-negative updates of shared variables, and (2) we allow level functions to change evaluation of the guards a bounded number of times.

Consider a guard φ, a configuration σ, and a schedule τ applicable to σ. We say that τ is *steady* with respect to (φ, σ), if it has the following property: $\tau_{[1,i]}(\sigma) \models \varphi$ if and only if $\sigma \models \varphi$ for $1 \leq i \leq |\tau|$.

▶ **Definition 13** (Bounded steadiness). We say that a guard φ of a UTA A is *bounded-steady* w.r.t A, if there exists a number $N \geq 0$, called the *flip bound* of φ, with the following property:

For every configuration σ of the counter system of A and every schedule $\tau = t_1, \ldots, t_n$ applicable to σ, there is a sequence of indices $0 = i_0 \leq i_1 \leq \cdots \leq i_N \leq i_{N+1} = n + 1$ such that $\tau_{(i_j, i_{j+1})}$ is steady with respect to φ and $\tau_{[1,i_j]}(\sigma)$ for $0 \leq j \leq N$.

Bounded-steadiness is central in proving the bounded diameter property:

▶ **Theorem 14** (Bounded diameter criterion). *Every canonical UTA A with non-negative updates of shared variables satisfies the following:*

If every guard is bounded-steady w.r.t. A, then the diameter of the counter system $CS(A)$ is bounded by a constant.

In the context of TA, constructions are introduced in [25] to remove cycles and reorder transitions (to apply acceleration), in order to shorten subschedules in which evaluations of guards do not change, i.e., steady subschedules. The results of [25] can be summarized in the following lemma.

▶ **Lemma 15.** *There exists an total order of rules $\prec$ such that for every schedule τ, there exists a unique schedule,* $\mathrm{short}(\tau)$*, with the following properties:*

1. *If transition (r, a) appears in* $\mathrm{short}(\tau)$*, then τ contains a transition (r, a') for some a'.*
2. *If transition (r, a) appears before (r', a') in* $\mathrm{short}(\tau)$*, then $r \prec r'$.*
3. *If for a configuration σ, an applicable schedule τ is steady with respect to all guards and σ, then* $\mathrm{short}(\tau)$ *is applicable to σ and $\tau(\sigma) = \mathrm{short}(\tau)(\sigma)$.*

One can prove the above lemma independently of the shape of the guards. For the proof one only uses that in a steady schedule the evaluation of guards does not change. As a result, one can directly apply the proofs from [25] to generalize Lemma 15 to UTA. This allows us to replace a steady schedule τ by $\mathrm{short}(\tau)$, which reaches the same state and whose length is bounded by Lemma 15(2), because threshold automata have a fixed number of rules and $\prec$ is a total order. What remains to be proven for Theorem 14 is that every schedule of a threshold automaton with bounded-steady guards can be decomposed into a bounded number of steady subschedules.

Proof of Theorem 14. Let $\varphi_1, \ldots, \varphi_m$ be the bounded-steady guards. Let σ be a configuration and $\tau = t_1, \ldots, t_n$ a schedule applicable to it. Since each φ_j is bounded-steady it has a flip bound N_j, for which there exist $i^j_1, \ldots, i^j_{N_j}$ with the property that $\tau_{(i^j_k, i^j_{k+1})}$ is steady with respect to φ_j and $\tau_{[1,i^j_k]}(\sigma)$ for $0 \leq k \leq N_j$. We denote by S^i_j the set of critical indices $\{i^j_1, \ldots, i^j_{N_j}\}$.

We denote by S the set $\bigcup_{j=1}^m S_j$, and by $i_1, \ldots, i_l$ its elements. Additionally, denote $i_0 = 0$ and $i_{l+1} = n + 1$. The set S partitions τ into finer subschedules than each S_j, that is, for every $0 \leq k \leq l$ and for every $1 \leq j \leq m$ there is an index $i^j_p \in S_j$ such that the schedule $\tau_{(i_k, i_{k+1})}$ is a subschedule of the steady schedule $\tau_{(i^j_p, i^j_{p+1})}$. Because a subschedule of a steady schedule is also steady by definition (w.r.t. its initial configuration and the same guard), we can conclude that the schedules $\tau_{(i_k, i_{k+1})}$ are steady with respect to all guards φ_j and $\tau_{[1,i_k]}(\sigma)$.

We can therefore apply Lemma 15 to each $\tau_{(i_k, i_{k+1})}$ and replace it with a shortened schedule. By property (2) of Lemma 15 and because $\prec$ is a total order, the length of the shortened schedules is at most $|\mathcal{R}|$. After replacing every $\tau_{(i_k, i_{k+1})}$ with $\mathrm{short}(\tau_{(i_k, i_{k+1})})$, we obtain a schedule τ', which is applicable to σ, has the property that $\tau'(\sigma) = \tau(\sigma)$ and $|\tau'| \leq (|S|+1) \cdot |\mathcal{R}| + |S|$. By the definition of S, it holds that $|S| \leq \sum_{j=1}^m |S_j| \leq \sum_{j=1}^m N_j$. ◀

4.2 Two fragments with bounded-steady guards

Theorem 14 gives us a sufficient condition for a function to be used in a guard so that the resulting counter system has a bounded diameter. The condition applies to threshold automata with non-negative updates to shared variables. Thus, we can characterize bounded-steady guards by the shape of their level functions.

▶ **Proposition 16.** *Every canonical UTA A with non-negative updates of shared variables has the following property: If a threshold guard φ has the shape $thd(\vec{p}) \bowtie F(y)$ for a shared variable $y \in \Gamma$, a comparison $\bowtie \in \{<, \leq, >, \geq\}$, and a piecewise-monotone function $F \colon \mathbb{Z} \to \mathbb{R}$, then the guard φ is bounded-steady w.r.t. A.*

▶ **Example 17.** Consider piecewise-monotone functions $f_1(x)$, $f_2(x)$ and reals $a_n, \ldots a_0, b \in \mathbb{R}$ with $b > 0$. Then, $a_n \cdot x^n + \cdots + a_1 \cdot x + a_0$, b^x, $\ln x$, and $\min\{f_1(x), f_2(x)\}$ are piecewise-monotone functions of $x \in \mathbb{Z}$. Each of them can be used as $F(x)$ in Proposition 16, and thus they produce bounded-steady guards. ◁

As a corollary of Proposition 16 and Theorem 14, the threshold automata with piecewise-monotone functions in the guards have the bounded diameter property:

▶ **Corollary 18.** *For every PMTA, the diameter of its counter system is bounded.*

Note that the affine threshold guards of [25] have the shape required in Proposition 16, and thus are just a special case.

We generalize Proposition 16 to guards over multiple shared variables. Recall that an m-dimensional integer box is a product of m intervals, that is, $B = \mathbb{Z}^m \cap [a_1, b_1] \times \cdots \times [a_m, b_m]$ for some boundaries $a_1, b_1, \ldots, a_m, b_m \in \mathbb{Z}_\infty$.

▶ **Proposition 19.** *Consider a UTA with non-negative updates of shared variables. A non-linear guard $thd(\vec{p}) \bowtie lvl(\vec{x})$, for $\bowtie \in \{<, \leq, >, \geq\}$, is bounded-steady, if:*
For every level $C \in \mathbb{R}$, the function domain $\mathbb{Z}^{|\Gamma|}$ of the level function can be partitioned into a finite set of disjoint $|\Gamma|$-dimensional boxes $B_1, \ldots, B_k$ that satisfy $\{\vec{x} \in B_i \mid C \bowtie lvl(\vec{x})\}$ is equal to either B_i or $\emptyset$ for $1 \leq i \leq k$.

As a result, the following two-variable functions give us bounded-steady guards:

$$x + y, \quad x \cdot y, \quad \min(f_1(x), f_2(y)) \text{ or } \max(f_1(x), f_2(y)) \text{ for piecewise-monotone } f_1 \text{ and } f_2$$

5 Relation to flattable counter automata

Counter automata model infinite-state systems and have acceleration procedures and tools for reachability analysis [10, 30, 4]. Threshold automata give rise to accelerated counter systems. In this section, we establish a link between these two frameworks. In particular, from a threshold automaton A, we construct two kinds of counter automata: $\mathsf{CA}^0(A)$ is a counter automaton that executes a single UTA rule without any built-in acceleration, and $\mathsf{CA}^1(A)$ is a counter automaton that executes one UTA rule several times in one step. The automaton $\mathsf{CA}^1(A)$ corresponds to our counter system $CS(A)$ in Section 2.2. In our analysis, single-rule acceleration plays a central role in finding diameter bounds, whereas the procedures for counter automata employ more general forms of acceleration. In fact, $\mathsf{CA}^0(A)$ and $\mathsf{CA}^1(A)$ have the same reachability relation, and any of them can in principle be used as the input to the techniques for counter automata.

We recall the definitions of counter automata from [30], operating on m counters.

▶ **Definition 20.** An m-dimensional counter automaton CA is defined as a tuple $(Q, T, \mathsf{src}, \mathsf{tgt}, \{G_t\}_{t \in T})$ with the following properties:

- Q and T are finite, non-empty sets of CA-*locations* and CA-*transitions* respectively,
- $\mathsf{src} : T \to Q$ and $\mathsf{tgt} : T \to Q$ are the *source* and *target* mappings respectively, and
- $\{G_t\}_{t \in T}$ is a finite family of binary relations on $\mathbb{N}^m$ called *flow guards*.

The semantics of the counter automaton CA is defined as a transition system $(\mathcal{C}_{\mathsf{CA}}, \to_{\mathsf{CA}})$ with the following properties:

1. The set $\mathcal{C}_{\mathsf{CA}} = Q \times \mathbb{N}_0^m$ captures CA-configurations, and
2. the relation $\to_{\mathsf{CA}} \subseteq \mathcal{C}_{\mathsf{CA}} \times \mathcal{C}_{\mathsf{CA}}$ captures CA-steps. CA makes a step from a configuration $(q, \vec{x}) \in \mathcal{C}_{\mathsf{CA}}$ to a configuration $(q', \vec{x}') \in \mathcal{C}_{\mathsf{CA}}$ via a transition $t \in T$ – formally written as $(q, \vec{x}) \to_{\mathsf{CA}} (q', \vec{x}')$ – if the following holds:

$$q = \mathsf{src}(t) \text{ and } q' = \mathsf{tgt}(t) \text{ and } (\vec{x}, \vec{x}') \in G_t$$

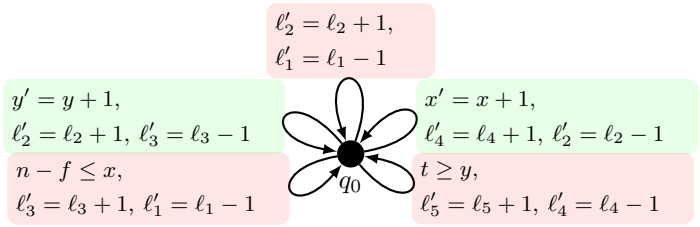


Figure 6 A counter automaton for the threshold automaton in Figure 1.

A sequence $(q_1, \vec{x}_1), \ldots, (q_k, \vec{x}_k)$ of CA-configurations is called a CA-*path*, if $(q_i, \vec{x}_i) \to_{\mathsf{CA}} (q_{i+1}, \vec{x}_{i+1})$ for $1 \leq i < k$. Then, the *reachability relation* $\to^*_{\mathsf{CA}} \subseteq \mathbb{N}_0^{|m|} \times \mathbb{N}_0^{|m|}$ contains all the pairs of vectors that are connected with a path for some control locations, that is, $\vec{x} \to^*_{\mathsf{CA}} \vec{x}'$ if and only if there is a CA-path $(q_1, \vec{x}_1), \ldots, (q_k, \vec{x}_k)$ with $\vec{x} = \vec{x}_1$ and $\vec{x}' = \vec{x}_k$.

A counter automaton without acceleration. Fix an unrestricted threshold automaton $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$, and let P be the set of variables $\mathcal{L} \cup \Gamma \cup \Pi$. To represent the configurations of the UTA counter system, we use vectors $\vec{x} = (x_1, \ldots, x_{|P|}) \in \mathbb{N}_0^{|P|}$, where each element x_i stores the value of a variable from the set P (there is a bijection). For a vector $\vec{x} \in \mathbb{N}_0^{|P|}$ and a set $U \subseteq P$, with $x|_U$, we denote the projection of $\vec{x}$ on the variables from U.

A $|P|$-dimensional counter automaton $\mathsf{CA}^0(A) = (Q, T, \mathsf{src}, \mathsf{tgt}, \{G_t\}_{t \in T})$ is constructed as follows:

- The automaton has only one CA-location, that is, $Q = \{q_0\}$ for some q_0,
- The CA-transitions are identical to the UTA rules, that is, $T = \mathcal{R}$,
- Every transition $t \in T$ originates from the location q_0 and ends in q_0; formally, $\mathsf{src}(t) = \mathsf{tgt}(t) = q_0$,
- For every rule $r \in \mathcal{R}$, the flow relation $G_r \subseteq \mathbb{N}_0^{|P|} \times \mathbb{N}_0^{|P|}$ is the intersection of two relations Guard_r and Update_r that are defined as:

$$(\vec{x}, \vec{x}') \in \mathsf{Guard}_r \text{ if and only if } (\vec{x}|_\Gamma, \vec{x}|_\Pi) \models \bigwedge_{\varphi \in r.\Phi} \varphi$$

$$(\vec{x}, \vec{x}') \in \mathsf{Update}_r \text{ if and only if } \vec{x}'|_\Pi = \vec{x}|_\Pi, \tag{1}$$

$$\vec{x}'|_\Gamma = \vec{x}|_\Gamma + r.\vec{u}, \text{ and } \vec{x}'|_\mathcal{L} = \vec{x}|_\mathcal{L} + \vec{1}_{r.to} - \vec{1}_{r.from} \tag{2}$$

Given the threshold automaton in Figure 1, we construct the respective counter automaton in Figure 6. Apart from the shared variables and parameters, the counter automaton explicitly maintains a counter for each location of UTA, whereas in threshold automata these counters are implicit.

A counter automaton with single-rule acceleration. Given a UTA A, we define its counter automaton with single-rule acceleration $\mathsf{CA}^1(A)$. This automaton is structurally the same as $\mathsf{CA}^0(A)$, except that the flow relation G_r for $r \in \mathcal{R}$ accounts for a non-negative acceleration factor a:

$$(\vec{x}, \vec{x}') \in G_r \text{ if and only if } \exists a \geq 0.\ \forall k : 0 \leq k < a.\ (\vec{x} + k \cdot r.\vec{u}, \vec{x}') \in \mathsf{Guard}_r,$$
$$\vec{x}'|_\Pi = \vec{x}|_\Pi, \vec{x}'|_\Gamma = \vec{x}|_\Gamma + a \cdot r.\vec{u}, \text{ and } \vec{x}'|_\mathcal{L} = \vec{x}|_\mathcal{L} + a \cdot (\vec{1}_{r.to} - \vec{1}_{r.from})$$

Discussions. If we ignore the location q_0, the counter automaton $\mathsf{CA}^1(A)$ has the same transition relation as the counter system of A; as defined in Section 2.2 or in [25], that is, with built-in single-rule acceleration. General acceleration procedures for reachability analysis were developed for counter automata [30, 4]. These techniques terminate on flat and flattable counter automata. A counter automaton is *flat*, if its control graph – built of locations and transitions – does not contain nested loops [10]. A counter automaton A is *flattable*, if there is a flat counter automaton F with the same reachability relation, that is, $\rightarrow_F^* = \rightarrow_A^*$. The counter automata $\mathsf{CA}^0(A)$ and $\mathsf{CA}^1(A)$ are obviously not flat, as can be seen from Figure 6, the question is whether they are flattable.

As can be seen from the definition of $\mathsf{CA}^1(A)$, single-rule acceleration has a special form: it merges successive occurrences of a rule of $\mathsf{CA}^0(A)$ into one transition, provided that the counter values are sufficiently large. The motivation behind this acceleration is to perform transitions of many processes in a distributed system in *parallel* [25], in contrast to compressing *sequential* steps.

The bounded diameter property for a threshold automaton A implies flattability of the counter automaton $\mathsf{CA}^0(A)$. It is sufficient to unroll $\mathsf{CA}^0(A)$ up to the diameter bound and add self-loops to model single-rule acceleration:

▶ **Proposition 21.** *For every unrestricted threshold automaton A, if the diameter of the counter system $CS(A)$ is bounded, then the counter automaton $\mathsf{CA}^0(A)$ is flattable.*

6 Flattability for non-canonical threshold automata

It is easy to see that the counter systems of non-canonical threshold automata do not have bounded diameter, when applying single-rule acceleration. Interestingly, we show that the respective counter automata for NCTA are flattable. Hence, they can be thus analyzed with general acceleration tools such as FAST [4].

Additional definitions. To prove flattability, we adapt a few definitions from [24]. Let $\mathcal{G} = \bigcup_{r\in\mathcal{R}} r.\Phi$. Then, $\Phi^{\mathrm{R}} = \{g \in \mathcal{G} \mid g \text{ is an upper guard}\}$ and $\Phi^{\mathrm{F}} = \{g \in \mathcal{G} \mid g \text{ is a lower guard}\}$. A *context* is a pair $(\Omega^{\mathrm{R}}, \Omega^{\mathrm{F}})$, where $\Omega^{\mathrm{R}} \subseteq \Phi^{\mathrm{R}}$ and $\Omega^{\mathrm{F}} \subseteq \Phi^{\mathrm{F}}$. The set Ω^{R} keeps track of unlocked guards from Φ^{R}, and the set Ω^{F} keeps track of locked guards from Φ^{F}. We usually denote a context with Ω, and refer to its first and second component by writing Ω^{R} and Ω^{F} respectively. For contexts Ω_1 and Ω_2, we say that $\Omega_1 \sqsubseteq \Omega_2$ if and only if $\Omega_1^{\mathrm{R}} \cup \Omega_1^{\mathrm{F}} \subseteq \Omega_2^{\mathrm{R}} \cup \Omega_2^{\mathrm{F}}$.

Finally, for a context Ω, we define a formula $form(\Omega)$ that summarizes the constraints of the guards that are locked/unlocked in the context: $\bigwedge_{\psi\in\Psi^+} \psi \wedge \bigwedge_{\psi\in\Psi^-} \neg\psi$ for $\Psi^+ = \Omega^{\mathrm{R}} \cup (\Phi^{\mathrm{F}} \setminus \Omega^{\mathrm{F}})$ and $\Psi^- = (\Phi^{\mathrm{R}} \setminus \Omega^{\mathrm{R}}) \cup \Omega^{\mathrm{F}}$. We write $[\![form(\Omega)]\!]$ to denote the set of vectors that satisfy $form(\Omega)$, that is, $\vec{x} \in [\![form(\Omega)]\!]$ if and only if $(\vec{x}|_\Gamma, \vec{x}|_\Pi) \models form(\Omega)$ holds true.

▶ **Definition 22.** For a NCTA $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$ and a context Ω, we define the *slice* of A with context Ω as a threshold automaton $A|_\Omega = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}|_\Omega)$, where a rule $r \in \mathcal{R}$ belongs to $\mathcal{R}|_\Omega$ if and only if $form(\Omega) \rightarrow \bigwedge_{\varphi\in r.\Phi} \varphi$.

Overview of the proof. We start with an NCTA. The relation $\sqsubset$ is a partial order on the contexts. We construct a flat counter automaton as a composition of flat counter automata, one per context, that are then connected according to the partial order $\sqsubset$. Figure 7 sketches the construction. In more detail, for each context Ω of A we construct the slice. We show that when one removes the threshold guards from the slice, its counter automaton becomes structurally a BPP-net [16, 18], which are known to be flattable [30]. Thus, there is a

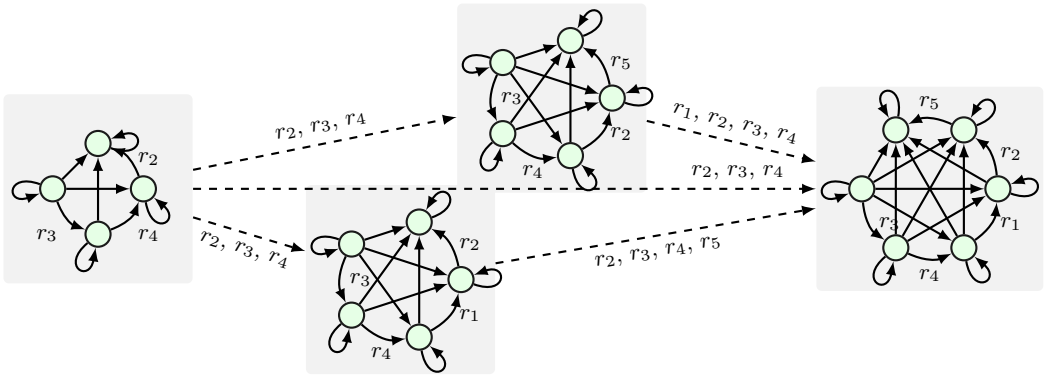


Figure 7 An example of the flattened threshold automaton from Figure 1. The edges connecting the gray blocks connect all the states inside the blocks.

flattened counter automaton $F(\Omega)$ for Ω. However, as $F(\Omega)$ does not have threshold guards, it allows transitions to leave the context Ω earlier than in the original counter system. Thus, we add additional constraints to $F(\Omega)$ to keep the transitions in the context, and form a "flat slice". Then, we combine flat slices for each context according to the partial order between the contexts, and obtain a flat counter automaton whose reachability relation is the same as of $\mathsf{CA}^0(A)$.

▶ **Proposition 23.** *For every non-canonical threshold automaton A and context Ω, there is a flat counter automaton $Flat(A|_\Omega)$ that has the same reachability relation when restricted to the* CA*-configurations that match the context, that is, $\rightarrow^*_{Flat(A|_\Omega)} \cap [\![form(\Omega)]\!]^2$ equals to $\rightarrow^*_{\mathsf{CA}^0(A)} \cap [\![form(\Omega)]\!]^2$.*

Assembling the flat counter automata for the slices. Fix a non-canonical threshold automaton $A = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R})$. Proposition 23 allows us to flatten a single slice. To flatten $\mathsf{CA}^0(A)$, we flatten slices and connect them with context changing transitions.

As a first step, we enumerate all contexts $\Omega_1, \ldots, \Omega_K$, where $K = |\Phi^{\mathrm{R}} \times \Phi^{\mathrm{F}}|$. For each context $i \in \{1, \ldots, K\}$, we apply Proposition 23, to construct a flat counter automaton $Flat(i) = (Q_i, T_i, \mathsf{src}_i, \mathsf{tgt}_i, \{G^i_t\}_{t \in T_i})$. We assume that the sets $Q_1, \ldots, Q_K$ and $T_1, \ldots, T_K$ are all disjoint. We use $Flat(1), \ldots, Flat(K)$ to construct two sets of counter automata:

1. An automaton $FlatSlice(i)$ produces paths of $\mathsf{CA}^0(A)$ in the context Ω_i. Formally, $FlatSlice(i) = (Q_i, T_i, \mathsf{src}_i, \mathsf{tgt}_i, \{G^i_t \cap [\![form(\Omega)]\!]^2\}_{t \in T_i})$.
2. An automaton $Branch(i, j)$, for $1 \le i, j \le K$ such that $\Omega_i \sqsubseteq \Omega_j$ and $i \neq j$, produces the context-changing transitions from $FlatSlice(i)$ to $FlatSlice(j)$. Formally,

$$Branch(i,j) = (Q_i \cup Q_j, T_{i,j}, \mathsf{src}_{i,j}, \mathsf{tgt}_{i,j}, \{G^{i,j}_t\}_{t \in T_{i,j}}),$$

where the components of $Branch(i, j)$ are defined as follows for $t \in T_i$:

- There is a transition for each ith slice transition and jth slice state: $T_{i,j} = T_i \times Q_j$,
- The mappings are $\mathsf{src}_{i,j}((t, q)) = \mathsf{src}_i(t)$ and $\mathsf{tgt}_{i,j}((t, q)) = q$ for $q \in Q_j$, and
- We restrict the guards to the two contexts: $G^{i,j}_t = G^i_t \cap ([\![form(\Omega_i)]\!] \times [\![form(\Omega_j)]\!])$.

A flat version of $\mathsf{CA}^0(A)$ is the union of all flat slices and branches:

$$Flattened(A) = \bigcup_{1 \le i \le K} FlatSlice(i) \cup \bigcup_{(i,j) \in E} Branch(i,j) \text{ for } E = \{(i,j) \mid \Omega_i \sqsubseteq \Omega_j, i \neq j\} \quad (3)$$

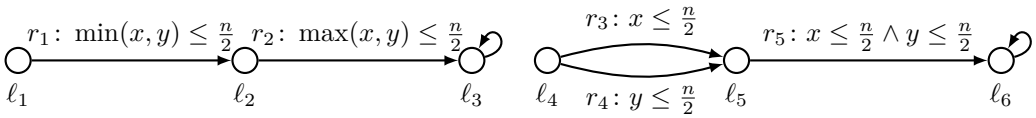


Figure 8 An unrestricted TA (left) and an equivalent threshold automaton (right).

We define the union $A \cup B$ as usual: The states, transitions, and flows of $A \cup B$ are the unions of the A's and B's states, transitions, and flows respectively. The source and target mappings are identical to the A's and B's mappings on their domains.

▶ **Theorem 24.** *For every non-canonical threshold automaton A, its flattened version has the same reachability relation:* $\rightarrow^*_{Flattened(A)} = \rightarrow^*_{CA^0(A)}$.

7 Conclusions

Verification of infinite-state systems and parameterized concurrent systems is a lively research area, e.g., see some recent results [19, 13, 1, 11, 17, 12, 31, 8, 2]. There are many different modeling frameworks, and it is not easy to understand relations between them. However, this understanding is of paramount importance for reusing existing tools. In this paper, on the one hand, we give reachability results for new classes of systems, and on the other hand, establish the relation of the model in [25, 21] to counter automata [10, 30]. We clarify the relation between the single rule acceleration introduced in [25] to acceleration in (flattable) counter automata [4, 30]. The single-rule acceleration in [25] is very simple compared to the general acceleration techniques [30, 4]. Still, it was demonstrated to be effective in parameterized verification of fault-tolerant distributed algorithms [22, 21].

The benefits of our extended framework are two-fold. On one hand, we can use our results to optimize threshold automata. Figure 8 shows an unrestricted threshold automaton that uses minimum and maximum. This UTA can be expressed as an equivalent threshold automaton by introducing more rules and guards (see Figure 8), which makes it harder to reason about. On the other hand, our framework permits some new guards, which have no corresponding encoding in threshold automata. For instance, a threshold $x < \sqrt{n/\log n}$ in [3] gives us such an example (though they are using the synchronous model of computation).

Some open questions still remain. Regarding application to distributed algorithms, we observe that in the pseudo code of several distributed consensus algorithms, processes pick the "*most often received value*" from a set of received values [5, 9]. A shared variable encoding – such as the one in [26] – maintains the number of messages with value 0 in a shared variable x_0, and the number of messages with value 1 in a shared variable x_1. The pseudo code statement about the "most often received value" needs a bounded difference guard "$x_1 - x_0 > 0$", which leads to undecidability as we show. This calls for further insights on modeling of such algorithms.

While we focused on reachability in this paper, as future work, we plan to lift the results of this paper to safety and liveness, following the ideas of [22].

References

1 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. Parameterized verification through view abstraction. *STTT*, 18(5):495–516, 2016.

2 Benjamin Aminof, Sasha Rubin, Ilina Stoilkovska, Josef Widder, and Florian Zuleger. Parameterized model checking of synchronous distributed algorithms by abstraction. In *VMCAI*, volume 10747 of *LNCS*, pages 1–24, 2018.

3 Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 193–199. ACM, 1998.

4 Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. Fast: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

5 Martin Biely, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, André Schiper, and Josef Widder. Tolerating corrupted communication. In *PODC*, pages 244–253, 2007.

6 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, volume 1579 of *LNCS*, pages 193–207, 1999.

7 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.

8 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. On reducing linearizability to state reachability. In *ICALP*, pages 95–107, 2015.

9 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

10 Hubert Comon and Yan Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV*, pages 268–279. Springer, 1998.

11 Giorgio Delzanno. A unified view of parameterized verification of abstract models of broadcast communication. *STTT*, 18(5):475–493, 2016.

12 Cezara Drăgoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A logic-based framework for verifying consensus algorithms. In *VMCAI*, volume 8318 of *LNCS*, pages 161–181, 2014.

13 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods in System Design*, 50(2-3):140–167, 2017.

14 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

15 E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995.

16 Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.*, 31(1):13–25, 1997.

17 Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proving liveness of parameterized programs. In *LICS*, pages 185–196, 2016.

18 Laurent Fribourg and Hans Olsén. A decompositional approach for computing least fixed-points of datalog programs with z-counters. *Constraints*, 2(3/4):305–335, 1997.

19 Zeinab Ganjei, Ahmed Rezine, Petru Eles, and Zebo Peng. Counting dynamically synchronizing processes. *STTT*, 18(5):517–534, 2016.

20 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

21 Igor Konnov, Marijana Lazić, 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*, 2017.

22 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.

23 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704, pages 125–140. Elsevier, 2014.

24 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In *CAV (Part I)*, volume 9206 of *LNCS*, pages 85–102, 2015.

25 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.

26 Igor Konnov, Josef Widder, Francesco Spegni, and Luca Spalazzi. Accuracy of message counting abstraction in fault-tolerant distributed algorithms. In *VMCAI*, pages 347–366, 2017.

27 Jure Kukovec. Generalizing threshold automata for reachability in parameterized systems. Master's thesis, University of Ljubljana, 2016. URL: `http://forsyte.at/wp-content/uploads/Kukovec-27142109-2016.pdf`.

28 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

29 Marijana Lazić, Igor Konnov, Josef Widder, and Roderick Bloem. Synthesis of distributed algorithms with parameterized threshold guards. In *OPODIS*, volume 95 of *LIPIcs*, pages 32:1–32:20, 2017. `doi:10.4230/LIPIcs.OPODIS.2017.32`.

30 Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 5, pages 489–503. Springer, 2005.

31 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In *CAV, Part II*, pages 217–237, 2017.

32 Marvin L Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

33 Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *SOSP*, pages 358–372, 2013.

34 Brian M. Oki and Barbara Liskov. Viewstamped replication: A general primary copy. In *PODC*, pages 8–17, 1988.

Selective Monitoring

Radu Grigore[1]
University of Kent, UK
https://orcid.org/0000-0003-1128-0311

Stefan Kiefer[2]
University of Oxford, UK

Abstract

We study selective monitors for labelled Markov chains. Monitors observe the outputs that are generated by a Markov chain during its run, with the goal of identifying runs as correct or faulty. A monitor is selective if it skips observations in order to reduce monitoring overhead. We are interested in monitors that minimize the expected number of observations. We establish an undecidability result for selectively monitoring general Markov chains. On the other hand, we show for non-hidden Markov chains (where any output identifies the state the Markov chain is in) that simple optimal monitors exist and can be computed efficiently, based on DFA language equivalence. These monitors do not depend on the precise transition probabilities in the Markov chain. We report on experiments where we compute these monitors for several open-source Java projects.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases runtime monitoring, probabilistic systems, Markov chains, automata, language equivalence

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/1806.06143`.

1 Introduction

Consider an MC (Markov chain) whose transitions are labelled with letters, and a finite automaton that accepts languages of infinite words. Computing the probability that the random word emitted by the MC is accepted by the automaton is a classical problem at the heart of probabilistic verification. A finite prefix may already determine whether the random infinite word is accepted, and computing the probability that such a *deciding* finite prefix is produced is a nontrivial *diagnosability* problem. The theoretical problem we study in this paper is how to catch deciding prefixes without observing the whole prefix; i.e., we want to minimize the expected number of observations and still catch all deciding prefixes.

Motivation. In runtime verification a program sends messages to a monitor, which decides if the program run is faulty. Usually, runtime verification is turned off in production code because monitoring overhead is prohibitive. QVM (quality virtual machine) and ARV (adaptive runtime verification) are existing pragmatic solutions to the overhead problem,

[1] Work supported by EPSRC grant EP/R012261/1.
[2] Work supported by a Royal Society University Research Fellowship.

which perform best-effort monitoring within a specified overhead budget [1, 3]. ARV relies on RVSE (runtime verification with state estimation) to also compute a probability that the program run is faulty [21, 15]. We take the opposite approach: we ask for the smallest overhead achievable without compromising precision at all.

Previous Work. Before worrying about the performance of a monitor, one might want to check if faults in a given system can be diagnosed at all. This problem has been studied under the term *diagnosability*, first for non-stochastic finite discrete event systems [19], which are labelled transition systems. It was shown in [14] that diagnosability can be checked in polynomial time, although the associated monitors may have exponential size. Later the notion of diagnosability was extended to stochastic discrete-event systems, which are labelled Markov chains [22]. Several notions of diagnosability in stochastic systems exist, and some of them have several names, see, e.g., [20, 4] and the references therein. Bertrand et al. [4] also compare the notions. For instance, they show that for one variant of the problem (referred to as A-diagnosability or SS-diagnosability or IF-diagnosability) a previously proposed polynomial-time algorithm is incorrect, and prove that this notion of diagnosability is PSPACE-complete. Indeed, most variants of diagnosability for stochastic systems are PSPACE-complete [4], with the notable exception of AA-diagnosability (where the monitor is allowed to diagnose wrongly with arbitrarily small probability), which can be solved in polynomial time [5].

Selective Monitoring. In this paper, we seem to make the problem harder: since observations by a monitor come with a performance overhead, we allow the monitor to skip observations. In order to decide how many observations to skip, the monitor employs an *observation policy*. Skipping observations might decrease the probability of deciding (whether the current run of the system is faulty or correct). We do not study this tradeoff: we require policies to be *feasible*, i.e., the probability of deciding must be as high as under the policy that observes everything. We do not require the system to be diagnosable; i.e., the probability of deciding may be less than 1. Checking whether the system is diagnosable is PSPACE-complete ([4], Theorem 8).

The Cost of Decision in General Markov Chains. The *cost* (of decision) is the number of observations that the policy makes during a run of the system. We are interested in minimizing the expected cost among all feasible policies. We show that if the system is diagnosable then there exists a policy with finite expected cost, i.e., the policy may stop observing after finite expected time. (The converse is not true.) Whether the infimum cost (among feasible policies) is finite is also PSPACE-complete (Theorem 14). Whether there is a feasible policy whose expected cost is smaller than a given threshold is undecidable (Theorem 15), even for diagnosable systems.

Non-Hidden Markov Chains. We identify a class of MCs, namely non-hidden MCs, where the picture is much brighter. An MC is called *non-hidden* when each label identifies the state. Non-hidden MCs are always diagnosable. Moreover, we show that *maximally procrastinating* policies are (almost) optimal (Theorem 27). A policy is called maximally procrastinating when it skips observations up to the point where one further skip would put a decision on the current run in question. We also show that one can construct an (almost) optimal maximally procrastinating policy in polynomial time. This policy *does not depend* on the exact probabilities in the MC, although the expected cost under that policy does. That is, we efficiently construct a policy that is (almost) optimal regardless of the transition probabilities

on the MC transitions. We also show that the infimum cost (among all feasible policies) can be computed in polynomial time (Theorem 28). Underlying these results is a theory based on automata, in particular, checking language equivalence of DFAs.

Experiments. We evaluated the algorithms presented in this paper by implementing them in Facebook Infer, and trying them on 11 of the most forked Java projects on GitHub. We found that, on average, selective monitoring can reduce the number of observations to a half.

2 Preliminaries

Let S be a finite set. We view elements of $\mathbb{R}^S$ as *vectors*, more specifically as row vectors. We write $\mathbf{1}$ for the all-1 vector, i.e., the element of $\{1\}^S$. For a vector $\mu \in \mathbb{R}^S$, we denote by μ^T its transpose, a column vector. A vector $\mu \in [0,1]^S$ is a *distribution over* S if $\mu\mathbf{1}^\mathsf{T} = 1$. For $s \in S$ we write e_s for the (*Dirac*) distribution over S with $e_s(s) = 1$ and $e_s(t) = 0$ for $t \in S \setminus \{s\}$. We view elements of $\mathbb{R}^{S\times S}$ as *matrices*. A matrix $M \in [0,1]^{S\times S}$ is called *stochastic* if each row sums up to one, i.e., $M\mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}$.

For a finite alphabet Σ, we write Σ^* and Σ^ω for the finite and infinite words over Σ, respectively. We write ε for the empty word. We represent languages $L \subseteq \Sigma^\omega$ using deterministic finite automata, and we represent probability measures Pr over Σ^ω using Markov chains.

A (discrete-time, finite-state, labelled) *Markov chain (MC)* is a quadruple (S, Σ, M, s_0) where S is a finite set of states, Σ a finite alphabet, s_0 an initial state, and $M : \Sigma \to [0,1]^{S\times S}$ specifies the transitions, such that $\sum_{a\in\Sigma} M(a)$ is a stochastic matrix. Intuitively, if the MC is in state s, then with probability $M(a)(s,s')$ it emits a and moves to state s'. For the complexity results in this paper, we assume that all numbers in the matrices $M(a)$ for $a \in \Sigma$ are rationals given as fractions of integers represented in binary. We extend M to the mapping $M : \Sigma^* \to [0,1]^{S\times S}$ with $M(a_1 \cdots a_k) = M(a_1) \cdots M(a_k)$ for $a_1, \ldots, a_k \in \Sigma$. Intuitively, if the MC is in state s then with probability $M(u)(s,s')$ it emits the word $u \in \Sigma^*$ and moves (in $|u|$ steps) to state s'. An MC is called *non-hidden* if for each $a \in \Sigma$ all non-zero entries of $M(a)$ are in the same column. Intuitively, in a non-hidden MC, the emitted letter identifies the next state. An MC (S, Σ, M, s_0) defines the standard probability measure Pr over Σ^ω, uniquely defined by assigning probabilities to cylinder sets $\{u\}\Sigma^\omega$, with $u \in \Sigma^*$, as follows:

$$\Pr(\{u\}\Sigma^\omega) \;\coloneqq\; e_{s_0} M(u) \mathbf{1}^\mathsf{T}$$

A *deterministic finite automaton (DFA)* is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ a finite alphabet, $\delta : Q \times \Sigma \to Q$ a transition function, q_0 an initial state, and $F \subseteq Q$ a set of accepting states. We extend δ to $\delta : Q \times \Sigma^* \to Q$ as usual. A DFA defines a language $L \subseteq \Sigma^\omega$ as follows:

$$L \;\coloneqq\; \{\, w \in \Sigma^\omega \mid \delta(q_0, u) \in F \text{ for some prefix } u \text{ of } w \,\}$$

Note that we do not require accepting states to be visited infinitely often: just once suffices. Therefore we can and will assume without loss of generality that there is f with $F = \{f\}$ and $\delta(f, a) = f$ for all $a \in \Sigma$.

For the rest of the paper we fix an MC $\mathcal{M} = (S, \Sigma, M, s_0)$ and a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. We define their composition as the MC $\mathcal{M} \times \mathcal{A} \coloneqq (S \times Q, \Sigma, M', (s_0, q_0))$ where $M'(a)((s,q), (s',q'))$ equals $M(a)(s,s')$ if $q' = \delta(q,a)$ and 0 otherwise. Thus, $\mathcal{M}$ and $\mathcal{M} \times \mathcal{A}$ induce the same probability measure Pr.

An *observation* $o \in \Sigma_\perp$ is either a letter or the special symbol $\perp \notin \Sigma$, which stands for "not seen". An *observation policy* $\rho : \Sigma_\perp^* \to \{0,1\}$ is a (not necessarily computable) function that, given the observations made so far, says whether we should observe the next letter. An observation policy ρ determines a projection $\pi_\rho : \Sigma^\omega \to \Sigma_\perp^\omega$: we have $\pi_\rho(a_1 a_2 \dots) = o_1 o_2 \dots$ when

$$o_{n+1} \;=\; \begin{cases} a_{n+1} & \text{if } \rho(o_1 \dots o_n) = 1 \\ \perp & \text{if } \rho(o_1 \dots o_n) = 0 \end{cases} \qquad \text{for all } n \geq 0$$

We denote the *see-all policy* by $\bullet$; thus, $\pi_\bullet(w) = w$.

In the rest of the paper we reserve a for letters, o for observations, u for finite words, w for infinite words, v for finite observation prefixes, s for states from an MC, and q for states from a DFA. We write $o_1 \sim o_2$ when o_1 and o_2 are the same or at least one of them is $\perp$. We lift this relation to (finite and infinite) sequences of observations (of the same length). We write $w \gtrsim v$ when $u \sim v$ holds for the length-$|v|$ prefix u of w.

We say that v is *negatively deciding* when $\Pr(\{w \gtrsim v \mid w \in L\}) = 0$. Intuitively, v is negatively deciding when v is incompatible (up to a null set) with L. Similarly, we say that v is *positively deciding* when $\Pr(\{w \gtrsim v \mid w \notin L\}) = 0$. An observation prefix v is *deciding* when it is positively or negatively deciding. An observation policy ρ *decides* w when $\pi_\rho(w)$ has a deciding prefix. A *monitor* is an interactive algorithm that implements an observation policy: it processes a stream of letters and, after each letter, it replies with one of "yes", "no", or "skip n letters", where $n \in \mathbb{N} \cup \{\infty\}$.

▶ **Lemma 1.** *For any w, if some policy decides w then $\bullet$ decides w.*

Proof. Let ρ decide w. Then there is a deciding prefix v of $\pi_\rho(w)$. Suppose v is positively deciding, i.e., $\Pr(\{w' \gtrsim v \mid w' \notin L\}) = 0$. Let u be the length-$|v|$ prefix of w. Then $\Pr(\{w \gtrsim u \mid w' \notin L\}) = 0$, since v can be obtained from u by possibly replacing some letters with $\perp$. Hence u is also positively deciding. Since u is a prefix of $w = \pi_\bullet(w)$, we have that $\bullet$ decides w. The case where v is negatively deciding is similar. ◀

It follows that $\max_\rho \Pr(\{w \mid \rho \text{ decides } w\}) = \Pr(\{w \mid \bullet \text{ decides } w\})$. We say that a policy ρ is *feasible* when it also attains the maximum, i.e., when

$$\Pr(\{w \mid \rho \text{ decides } w\}) \;=\; \Pr(\{w \mid \bullet \text{ decides } w\})\,.$$

Equivalently, ρ is feasible when $\Pr(\{w \mid \bullet \text{ decides } w \text{ implies } \rho \text{ decides } w\}) = 1$, i.e., almost all words that are decided by the see-all policy are also decided by ρ. If $v = o_1 o_2 \dots$ is the shortest prefix of $\pi_\rho(w)$ that is deciding, then the *cost of decision* $C_\rho(w)$ is $\sum_{k=0}^{|v|-1} \rho(o_1 \dots o_k)$. This paper is about finding feasible observation policies ρ that minimize $\mathrm{Ex}(C_\rho)$, the expectation of the cost of decision with respect to Pr.

3 Qualitative Analysis of Observation Policies

In this section we study properties of observation policies that are qualitative, i.e., not directly related to the cost of decision. We focus on properties of observation prefixes that a policy may produce.

Observation Prefixes

We have already defined deciding observation prefixes. We now define several other types of prefixes: enabled, confused, very confused, and finitary. A prefix v is *enabled* if it occurs with positive probability, $\Pr(\{w \gtrsim v\}) > 0$. Intuitively, the other types of prefixes v are defined

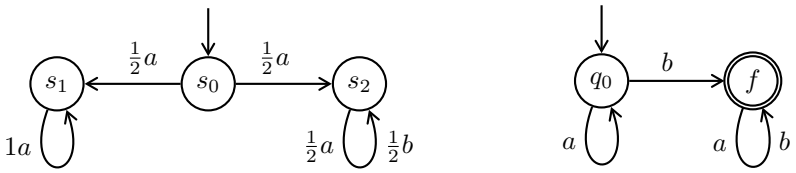
in terms of what would happen if we were to observe all from now on: if it is not almost sure that eventually a deciding prefix is reached, then we say v is confused; if it is almost sure that a deciding prefix will not be reached, then we say v is very confused; if it is almost sure that eventually a deciding or very confused prefix is reached, then we say v is finitary. To say this formally, let us make a few notational conventions: for an observation prefix v, we write $\Pr(v)$ as a shorthand for $\Pr(\{\, uw \mid u \sim v \,\})$; for a set Υ of observation prefixes, we write $\Pr(\Upsilon)$ as a shorthand for $\Pr\big(\bigcup_{v\in\Upsilon}\{\, uw \mid u \sim v \,\}\big)$. With these conventions, we define:

1. v is *confused* when $\Pr(\{\, vu \mid vu \text{ deciding} \,\}) < \Pr(v)$
2. v is *very confused* when $\Pr(\{\, vu \mid vu \text{ deciding} \,\}) = 0$
3. v is *finitary* when $\Pr(\{\, vu \mid vu \text{ deciding or very confused} \,\}) = \Pr(v)$

Observe that (a) confused implies enabled, (b) deciding implies not confused, and (c) enabled and very confused implies confused. The following are alternative equivalent definitions:

1. v is *confused* when $\Pr(\{\, uw \mid u \sim v,\ \text{no prefix of } vw \text{ is deciding} \,\}) > 0$
2. v is *very confused* when vu' is non-deciding for all enabled vu'
3. v is *finitary* when $\Pr(\{uw \mid u \sim v,\ \text{no prefix of } vw \text{ is deciding or very confused}\}) = 0$

▶ **Example 2.** Consider the MC and the DFA depicted here:



All observation prefixes that do not start with b are enabled. The observation prefixes ab and $\bot b$ and, in fact, all observation prefixes that contain b, are positively deciding. For all $n \in \mathbb{N}$ we have $\Pr(\{w \gtrsim a^n \mid w \in L\}) > 0$ and $\Pr(\{w \gtrsim a^n \mid w \notin L\}) > 0$, so a^n is not deciding. If the MC takes the right transition first then almost surely it emits b at some point. Thus $\Pr(\{aaa\cdots\}) = \frac{1}{2}$. Hence ε is confused. In this example only non-enabled observation prefixes are very confused. It follows that ε is not finitary.

Beliefs

For any s we write $\Pr_s$ for the probability measure of the MC $\mathcal{M}_s$ obtained from $\mathcal{M}$ by making s the initial state. For any q we write $L_q \subseteq \Sigma^\omega$ for the language of the DFA $\mathcal{A}_q$ obtained from $\mathcal{A}$ by making q the initial state. We call a pair (s,q) *negatively deciding* when $\Pr_s(L_q) = 0$; similarly, we call (s,q) *positively deciding* when $\Pr_s(L_q) = 1$. A subset of $S \times Q$ is called *belief*. We call a belief *negatively* (*positively*, respectively) *deciding* when all its elements are. We fix the notation $B_0 \coloneqq \{(s_0, q_0)\}$ (for the *initial belief*) for the remainder of the paper. Define the *belief NFA* as the NFA $\mathcal{B} = (S \times Q, \Sigma_\bot, \Delta, B_0, \emptyset)$ with:

$$\begin{aligned} \Delta((s,q),a) &= \{(s',q') \mid\ M(a)(s,s') > 0,\ \delta(q,a) = q'\} \quad \text{for } a \in \Sigma \\ \Delta((s,q),\bot) &= \bigcup_{a\in\Sigma} \Delta((s,q),a) \end{aligned}$$

We extend the transition function $\Delta : (S \times Q) \times \Sigma_\bot \to 2^{S\times Q}$ to $\Delta : 2^{S\times Q} \times \Sigma_\bot^* \to 2^{S\times Q}$ in the way that is usual for NFAs. Intuitively, if belief B is the set of states where the product $\mathcal{M} \times \mathcal{A}$ could be now, then $\Delta(B, v)$ is the belief adjusted by additionally observing v. To reason about observation prefixes v algorithmically, it will be convenient to reason about the belief $\Delta(B_0, v)$.

We define confused, very confused, and finitary beliefs as follows:

1. B is *confused* when $\Pr_s(\{\, uw \mid \Delta(B,u) \text{ deciding} \,\}) < 1$ for some $(s,q) \in B$
2. B is *very confused* when $\Delta(B,u)$ is empty or not deciding for all u
3. B is *finitary* when $\Pr_s(\{\, uw \mid \Delta(B,u) \text{ deciding or very confused} \,\}) = 1$ for all $(s,q) \in B$

▶ **Example 3.** In Example 2 we have $B_0 = \{(s_0,q_0)\}$, and $\Delta(B_0,a^n) = \{(s_1,q_0),(s_2,q_0)\}$ for all $n \geq 1$, and $\Delta(B_0,b) = \emptyset$, and $\Delta(B_0,a\bot) = \{(s_1,q_0),(s_2,q_0),(s_2,f)\}$, and $\Delta(B_0,\bot\upsilon) = \{(s_2,f)\}$ for all υ that contain b. The latter belief $\{(s_2,f)\}$ is positively deciding. We have $\Pr_{s_1}(\{uw \mid \Delta(\{(s_1,q_0)\},u) \text{ is deciding}\}) = 0$, so any belief that contains (s_1,q_0) is confused. Also, B_0 is confused as $\Pr_{s_0}(\{uw \mid \Delta(\{(s_0,q_0)\},u) \text{ is deciding}\}) = \frac{1}{2}$.

Relation Between Observation Prefixes and Beliefs

By the following lemma, the corresponding properties of observation prefixes and beliefs are closely related.

▶ **Lemma 4.** *Let υ be an observation prefix.*

1. *υ is enabled if and only if $\Delta(B_0,\upsilon) \neq \emptyset$.*
2. *υ is negatively deciding if and only if $\Delta(B_0,\upsilon)$ is negatively deciding.*
3. *υ is positively deciding if and only if $\Delta(B_0,\upsilon)$ is positively deciding.*
4. *υ is confused if and only if $\Delta(B_0,\upsilon)$ is confused.*
5. *υ is very confused if and only if $\Delta(B_0,\upsilon)$ is very confused.*
6. *υ is finitary if and only if $\Delta(B_0,\upsilon)$ is finitary.*

The following lemma gives complexity bounds for computing these properties.

▶ **Lemma 5.** *Let υ be an observation prefix, and B a belief.*

1. *Whether υ is enabled can be decided in P.*
2. *Whether υ (or B) is negatively deciding can be decided in P.*
3. *Whether υ (or B) is positively deciding can be decided in P.*
4. *Whether υ (or B) is confused can be decided in PSPACE.*
5. *Whether υ (or B) is very confused can be decided in PSPACE.*
6. *Whether υ (or B) is finitary can be decided in PSPACE.*

Proof sketch. The belief NFA $\mathcal{B}$ and the MC $\mathcal{M} \times \mathcal{A}$ can be computed in polynomial time (even in deterministic logspace). For items 1–3, there are efficient graph algorithms that search these product structures. For instance, to show that a given pair (s_1,q_1) is not negatively deciding, it suffices to show that $\mathcal{B}$ has a path from (s_1,q_1) to a state (s_2,f) for some s_2. This can be checked in polynomial time (even in NL).

For items 4–6, one searches the (exponential-sized) product of $\mathcal{M}$ and the *determinization* of $\mathcal{B}$. This can be done in PSPACE. For instance, to show that a given belief B is confused, it suffices to show that there are $(s_1,q_1) \in B$ and u_1 and s_2 such that $\mathcal{M}$ has a u_1-labelled path from s_1 to s_2 such that there do *not* exist u_2 and s_3 such that $\mathcal{M}$ has a u_2-labelled path from s_2 to s_3 such that $\Delta(B,u_1u_2)$ is deciding. This can be checked in NPSPACE = PSPACE by nondeterministically guessing paths in the product of $\mathcal{M}$ and the determinization of $\mathcal{B}$. ◀

Diagnosability

We call a policy a *diagnoser* when it decides almost surely.

▶ **Example 6.** In Example 2 a diagnoser does not exist. Indeed, the policy $\bullet$ does not decide when the MC takes the left transition, and decides (positively) almost surely when the MC takes the right transition in the first step. Hence $\Pr(\{w \mid \bullet \text{ decides } w\}) = \Pr(\Sigma^*\{b\}\Sigma^\omega) = \frac{1}{2}$. So $\bullet$ is not a diagnoser. By Lemma 1, it follows that there is no diagnoser.

Diagnosability can be characterized by the notion of confusion:

▶ **Proposition 7.** *There exists a diagnoser if and only if ε is not confused.*

The following proposition shows that diagnosability is hard to check.

▶ **Theorem 8** (cf. [4, Theorem 6])**.** *Given an MC $\mathcal{M}$ and a DFA $\mathcal{A}$, it is PSPACE-complete to check if there exists a diagnoser.*

Theorem 8 essentially follows from a result by Bertrand et al. [4]. They study several different notions of diagnosability; one of them (*FA-diagnosability*) is very similar to our notion of diagnosability. There are several small differences; e.g., their systems are not necessarily products of an MC and a DFA. Therefore we give a self-contained proof of Theorem 8.

Proof sketch. By Proposition 7 it suffices to show PSPACE-completeness of checking whether ε is confused. Membership in PSPACE follows from Lemma 5.4. For hardness we reduce from the following problem: given an NFA $\mathcal{U}$ over $\Sigma = \{a, b\}$ where all states are initial and accepting, does $\mathcal{U}$ accept all (finite) words? This problem is PSPACE-complete [16, Lemma 6]. ◀

Allowing Confusion

We say an observation policy *allows confusion* when, with positive probability, it produces an observation prefix $v\bot$ such that $v\bot$ is confused but v is not.

▶ **Proposition 9.** *A feasible observation policy does not allow confusion.*

Hence, in order to be feasible, a policy must observe when it would get confused otherwise. In § 5 we show that in the non-hidden case there is almost a converse of Proposition 9; i.e., in order to be feasible, a policy need not do much more than not allow confusion.

4 Analyzing the Cost of Decision

In this section we study the computational complexity of finding feasible policies that minimize the expected cost of decision. We focus on the decision version of the problem: *Is there a feasible policy whose expected cost is smaller than a given threshold?* Define:

$$c_{inf} \coloneqq \inf_{\text{feasible } \rho} \mathrm{Ex}(C_\rho)$$

Since the see-all policy $\bullet$ never stops observing, we have $\Pr(C_\bullet = \infty) = 1$, so $\mathrm{Ex}(C_\bullet) = \infty$. However, once an observation prefix v is deciding or very confused, there is no point in continuing observation. Hence, we define a *light see-all* policy $\circ$, which observes until the observation prefix u is deciding or very confused; formally, $\circ(v) = 0$ if and only if v is deciding or very confused. It follows from the definition of very confused that the policy $\circ$ is feasible. Concerning the cost $C_\circ$ we have for all w

$$C_\circ(w) = \sum_{n=0}^{\infty} \big(1 - D_n(w)\big)\,, \tag{1}$$

where $D_n(w) = 1$ if the length-n prefix of w is deciding or very confused, and $D_n(w) = 0$ otherwise. The following results are proved in the full version of the paper, on arXiv:

▶ **Lemma 10.** *If ε is finitary then $\mathrm{Ex}(C_\circ)$ is finite.*

▶ **Lemma 11.** *Let ρ be a feasible observation policy. If $\Pr(C_\rho < \infty) = 1$ then ε is finitary.*

▶ **Proposition 12.** *c_{inf} is finite if and only if ε is finitary.*

▶ **Proposition 13.** *If a diagnoser exists then c_{inf} is finite.*

▶ **Theorem 14.** *It is PSPACE-complete to check if $c_{inf} < \infty$.*

Lemma 10 holds because, in $\mathcal{M} \times \mathcal{A}$, a bottom strongly connected component is reached in expected finite time. Lemma 11 says that a kind of converse holds for feasible policies. Proposition 12 follows from Lemmas 10 and 11. Proposition 13 follows from Propositions 7 and 12. To show Theorem 14, we use Proposition 12 and adapt the proof of Theorem 8.

The main negative result of the paper is that one cannot compute c_{inf}:

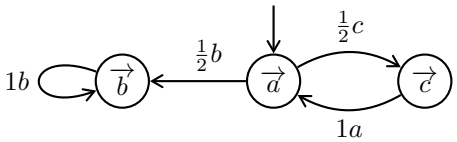
▶ **Theorem 15.** *It is undecidable to check if $c_{inf} < 3$, even when a diagnoser exists.*

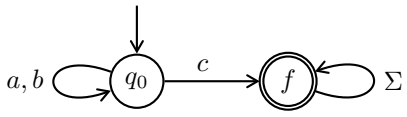
Proof sketch. By a reduction from the undecidable problem whether a given probabilistic automaton accepts some word with probability $> \frac{1}{2}$. The proof is somewhat complicated. In fact, in the full version of the paper (arXiv) we give two versions of the proof: a short incorrect one (with the correct main idea) and a long correct one. ◀

5 The Non-Hidden Case

Now we turn to positive results. In the rest of the paper we assume that the MC $\mathcal{M}$ is non-hidden, i.e., there exists a function $\overrightarrow{\cdot} : \Sigma \to S$ such that $M(a)(s, s') > 0$ implies $s' = \overrightarrow{a}$. We extend $\overrightarrow{\cdot}$ to finite words so that $\overrightarrow{ua} = \overrightarrow{a}$. We write $s \xrightarrow{u}$ to indicate that there is s' with $M(u)(s, s') > 0$.

▶ **Example 16.** Consider the following non-hidden MC and DFA:





$B_0 \coloneqq \{(\overrightarrow{a}, q_0)\}$
$B_1 \coloneqq \Delta(B_0, \bot) = \{(\overrightarrow{b}, q_0), (\overrightarrow{c}, f)\}$
$B_2 \coloneqq \Delta(B_0, \bot^2) = \{(\overrightarrow{b}, q_0), (\overrightarrow{a}, f)\}$
$B_3 \coloneqq \Delta(B_0, \bot^2 b) = \{(\overrightarrow{b}, q_0), (\overrightarrow{b}, f)\}$

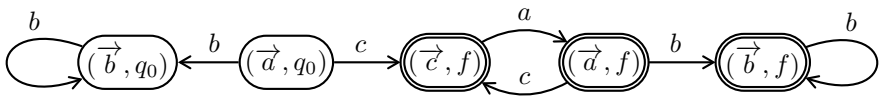
B_0 is the initial belief. The beliefs B_0 and B_1 are not confused: indeed, $\Delta(B_1, b) = \{(\overrightarrow{b}, q_0)\}$ is negatively deciding, and $\Delta(B_1, a) = \{(\overrightarrow{a}, f)\}$ is positively deciding. The belief B_2 is confused, as there is no $i \in \mathbb{N}$ for which $\Delta(B_2, b^i)$ is deciding. Finally, B_3 is very confused.

We will show that in the non-hidden case there always exists a diagnoser (Lemma 23). It follows that feasible policies need to decide almost surely and, by Proposition 13, that c_{inf} is finite. We have seen in Proposition 9 that feasible policies do not allow confusion. In this section we construct policies that procrastinate so much that they avoid confusion just barely. We will see that such policies have an expected cost that comes arbitrarily close to c_{inf}.

Language Equivalence

We characterize confusion by language equivalence in a certain DFA. Consider the belief NFA $\mathcal{B}$. In the non-hidden case, if we disallow $\bot$-transitions then $\mathcal{B}$ becomes a DFA $\mathcal{B}'$. For $\mathcal{B}'$ we define a set of accepting states by $F_{\mathcal{B}'} := \{(s,q) \mid \Pr_s(L_q) = 1\}$.

▶ **Example 17.** For the previous example, a part of the DFA $\mathcal{B}'$ looks as follows:



States that are unreachable from $(\overrightarrow{a}, q_0)$ are not drawn here.

We associate with each (s,q) the language $L_{s,q} \subseteq \Sigma^*$ that $\mathcal{B}'$ accepts starting from initial state (s,q). We call $(s,q), (s',q')$ *language equivalent*, denoted by $(s,q) \approx (s',q')$, when $L_{s,q} = L_{s',q'}$.

▶ **Lemma 18.** *One can compute the relation $\approx$ in polynomial time.*

Proof. For any (s,q) one can use standard MC algorithms to check in polynomial time if $\Pr_s(L_q) = 1$ (using a graph search in the composition $\mathcal{M} \times \mathcal{A}$, as in the proof of Lemma 5.3). Language equivalence in the DFA $\mathcal{B}'$ can be computed in polynomial time by minimization. ◀

We call a belief $B \subseteq S \times Q$ *settled* when all $(s,q) \in B$ are language equivalent.

▶ **Lemma 19.** *A belief $B \subseteq S \times Q$ is confused if and only if there is $a \in \Sigma$ such that $\Delta(B,a)$ is not settled.*

It follows that one can check in polynomial time whether a given belief is confused. We generalize this fact in Lemma 22 below.

▶ **Example 20.** In Example 16 the belief B_3 is not settled. Indeed, from the DFA in Example 17 we see that $L_{\overrightarrow{b},q_0} = \emptyset \neq \{b\}^* = L_{\overrightarrow{b},f}$. Since $B_3 = \Delta(B_2, b)$, by Lemma 19, the belief B_2 is confused.

Procrastination

For a belief $B \subseteq S \times Q$ and $k \in \mathbb{N}$, if $\Delta(B, \bot^k)$ is confused then so is $\Delta(B, \bot^{k+1})$. We define:

$$\mathit{cras}(B) \;:=\; \sup\{\, k \in \mathbb{N} \mid \Delta(B,\bot^k) \text{ is not confused} \,\} \;\in\; \mathbb{N} \cup \{-1, \infty\}$$

We set $\mathit{cras}(B) := -1$ if B is confused. We may write $\mathit{cras}(s,q)$ for $\mathit{cras}(\{(s,q)\})$.

▶ **Example 21.** In Example 16 we have $\mathit{cras}(B_0) = \mathit{cras}(\overrightarrow{a}, q_0) = 1$ and $\mathit{cras}(B_1) = 0$ and $\mathit{cras}(B_2) = \mathit{cras}(B_3) = -1$ and $\mathit{cras}(\overrightarrow{b}, q_0) = \mathit{cras}(\overrightarrow{a}, f) = \infty$.

▶ **Lemma 22.** *Given a belief B, one can compute $\mathit{cras}(B)$ in polynomial time. Further, if $\mathit{cras}(B)$ is finite then $\mathit{cras}(B) < |S|^2 \cdot |Q|^2$.*

Proof. Let $k \in \mathbb{N}$. By Lemma 19, $\Delta(B, \bot^k)$ is confused if and only if:

$$\exists\, a.\, \exists\, (s,q),(t,r) \in \Delta(B,\bot^k) : s \xrightarrow{a},\ t \xrightarrow{a},\ (\overrightarrow{a}, \delta(q,a)) \not\approx (\overrightarrow{a}, \delta(r,a))$$

This holds if and only if there is $B_2 \subseteq B$ with $|B_2| \leq 2$ such that:

$$\exists\, a.\, \exists\, (s,q),(t,r) \in \Delta(B_2,\bot^k) : s \xrightarrow{a},\ t \xrightarrow{a},\ (\overrightarrow{a}, \delta(q,a)) \not\approx (\overrightarrow{a}, \delta(r,a))$$

Let G be the directed graph with nodes in $S \times Q \times S \times Q$ and edges

$$((s,q,t,r),(s',q',t',r')) \qquad \iff \qquad \Delta(\{(s,q),(t,r)\},\bot) \;\supseteq\; \{(s',q'),(t',r')\}\,.$$

Also define the following set of nodes:

$$U \;\coloneqq\; \{(s,q,t,r) \mid \exists\, a : s \xrightarrow{a},\; t \xrightarrow{a},\; (\overrightarrow{a},\delta(q,a)) \not\approx (\overrightarrow{a},\delta(r,a))\}$$

By Lemma 18 one can compute U in polynomial time. It follows from the argument above that $\Delta(B,\bot^k)$ is confused if and only if there are $(s,q),(t,r) \in B$ such that there is a length-k path in G from (s,q,t,r) to a node in U. Let $k \le |S \times Q \times S \times Q|$ be the length of the shortest such path, and set $k \coloneqq \infty$ if no such path exists. Then k can be computed in polynomial time by a search of the graph G, and we have $\mathit{cras}(B) = k - 1$. ◀

The Procrastination Policy

For any belief B and any observation prefix υ, the language equivalence classes represented in $\Delta(B,\upsilon)$ depend only on υ and the language equivalence classes in B. Therefore, when tracking beliefs along observations, we may restrict B to a single representative of each equivalence class. We denote this operation by $B{\downarrow}$. A belief B is settled if and only if $|B{\downarrow}| \le 1$.

A *procrastination policy* $\rho_{pro}(K)$ is parameterized with (a large) $K \in \mathbb{N}$. Define (and precompute) $k(s,q) \coloneqq \min\{K, \mathit{cras}(s,q)\}$ for all (s,q). We define $\rho_{pro}(K)$ by the following monitor that implements it:

1. $i \coloneqq 0$
2. while (s_i,q_i) is not deciding:
   a. skip $k(s_i,q_i)$ observations, then observe a letter a_i
   b. $\{(s_{i+1},q_{i+1})\} \coloneqq \Delta((s_i,q_i),\bot^{k(s_i,q_i)}a_i){\downarrow}$;
   c. $i \coloneqq i+1$;
3. output yes/no decision

It follows from the definition of *cras* and Lemma 19 that $\Delta((s_i,q_i),\upsilon_i){\downarrow}$ is indeed a singleton for all i. We have:

▶ **Lemma 23.** *For all $K \in \mathbb{N}$ the procrastination policy $\rho_{pro}(K)$ is a diagnoser.*

Proof. For a non-hidden MC $\mathcal{M}$ and a DFA $\mathcal{A}$, there is at most one successor for (s,q) on letter a in the belief NFA $\mathcal{B}$, for all s,q,a. Then, by Lemma 19, singleton beliefs are not confused, and in particular the initial belief B_0 is not confused. By Lemma 4.4, ε is not confused, which means that $\Pr(\{\, u \mid u \text{ deciding}\,\}) = \Pr(\varepsilon) = 1$. Since almost surely a deciding word u is produced and since $\Delta(B_0,u) \subseteq \Delta(B_0,\upsilon)$ whenever $u \sim \upsilon$, it follows that eventually an observation prefix υ is produced such that $\Delta(B_0,\upsilon)$ contains a deciding pair (s,q). But, as remarked above, $\Delta(B_0,\upsilon)$ is settled, so it is deciding. ◀

The Procrastination MC $\mathcal{M}_{pro}(K)$

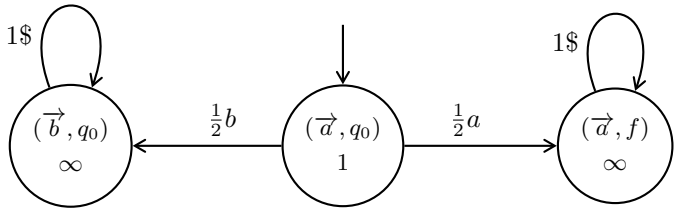
The policy $\rho_{pro}(K)$ produces a (random, almost surely finite) word $a_1a_2\cdots a_n$ with $n = C_{\rho_{pro}(K)}$. Indeed, the observations that $\rho_{pro}(K)$ makes can be described by an MC. Recall that we have previously defined a composition MC $\mathcal{M} \times \mathcal{A} = (S \times Q, \Sigma, M', (s_0,q_0))$. Now define an MC $\mathcal{M}_{pro}(K) \coloneqq (S \times Q, \Sigma \cup \{\$\}, M_{pro}(K), (s_0,q_0))$ where $\$ \notin \Sigma$ is a fresh letter

and the transitions are as follows: when (s,q) is deciding then $M_{pro}(K)(\$)\big((s,q),(s,q)\big) \coloneqq 1$, and when (s,q) is not deciding then

$$M_{pro}(K)(a)\big((s,q),(\overrightarrow{a},q')\big) \;\coloneqq\; \Big(M'(\bot)^{k(s,q)}M'(a)\Big)\big((s,q),(\overrightarrow{a},q')\big)\,,$$

where the matrix $M'(\bot) \coloneqq \sum_a M'(a)$ is powered by $k(s,q)$. The MC $\mathcal{M}_{pro}(K)$ may not be non-hidden, but could be made non-hidden by (i) collapsing all language equivalent $(s,q_1),(s,q_2)$ in the natural way, and (ii) redirecting all \$-labelled transition to a new state $\overrightarrow{\$}$ that has a self-loop. In the understanding that $\$\$\$\cdots$ indicates 'decision made', the probability distribution defined by the MC $\mathcal{M}_{pro}(K)$ coincides with the probability distribution on sequences of non-$\bot$ observations made by $\rho_{pro}(K)$.

▶ **Example 24.** For Example 16 the MC $\mathcal{M}_{pro}(K)$ for $K \geq 1$ is as follows:



Here the lower number in a state indicate the *cras* number. The left state is negatively deciding, and the right state is positively deciding. The policy $\rho_{pro}(K)$ skips the first observation and then observes either b or a, each with probability $\frac{1}{2}$, each leading to a deciding belief.

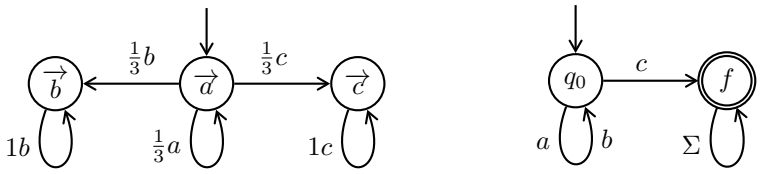
Maximal Procrastination is Optimal

The following lemma states, loosely speaking, that when a belief $\{(s,q)\}$ with $cras(s,q)=\infty$ is reached and K is large, then a single further observation is expected to suffice for a decision.

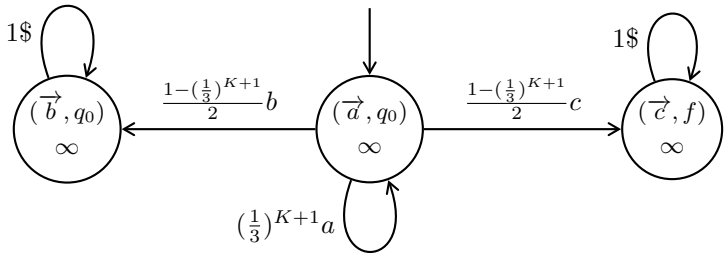
▶ **Lemma 25.** *Let $c(K,s,q)$ denote the expected cost of decision under $\rho_{pro}(K)$ starting in (s,q). For each $\varepsilon > 0$ there exists $K \in \mathbb{N}$ such that for all (s,q) with $cras(s,q)=\infty$ we have $c(K,s,q) \leq 1+\varepsilon$.*

Proof sketch. The proof is a quantitative version of the proof of Lemma 23. The singleton belief $\{(s,q)\}$ is not confused. Thus, if K is large then with high probability the belief $B \coloneqq \Delta(\{(s,q)\}, \bot^K a)$ (for the observed next letter a) contains a deciding pair (s',q'). But if $cras(s,q)=\infty$ then, by Lemma 19, B is settled, so if B contains a deciding pair then B is deciding. ◀

▶ **Example 26.** Consider the following variant of the previous example:



The MC $\mathcal{M}_{pro}(K)$ for $K \geq 0$ is as follows:



The left state is negatively deciding, and the right state is positively deciding. We have $c(K, \overrightarrow{b}, q_0) = c(K, \overrightarrow{c}, f) = 0$ and $c(K, \overrightarrow{a}, q_0) = 1/(1 - (\frac{1}{3})^{K+1})$.

Now we can prove the main positive result of the paper:

▶ **Theorem 27.** *For any feasible policy ρ there is $K \in \mathbb{N}$ such that:*

$$\mathrm{Ex}(C_{\rho_{pro}(K)}) \;\leq\; \mathrm{Ex}(C_\rho)$$

Proof sketch. Let ρ be a feasible policy. We choose $K > |S|^2 \cdot |Q|^2$, so, by Lemma 22, $\rho_{pro}(K)$ coincides with $\rho_{pro}(\infty)$ until time, say, n_∞ when $\rho_{pro}(K)$ encounters a pair (s, q) with $cras(s, q) = \infty$. (The time n_∞ may, with positive probability, never come.) Let us compare $\rho_{pro}(K)$ with ρ up to time n_∞. For $n \in \{0, \ldots, n_\infty\}$, define $\upsilon_{pro}(n)$ and $\upsilon_\rho(n)$ as the observation prefixes obtained by ρ_{pro} and ρ, respectively, after n steps. Write $\ell_{pro}(n)$ and $\ell_\rho(n)$ for the number of non-$\perp$ observations in $\upsilon_{pro}(n)$ and $\upsilon_\rho(n)$, respectively. For beliefs B, B' we write $B \preceq B'$ when for all $(s, q) \in B$ there is $(s', q') \in B'$ with $(s, q) \approx (s', q')$. One can show by induction that we have for all $n \in \{0, \ldots, n_\infty\}$:

$$\ell_{pro}(n) \leq \ell_\rho(n) \quad \text{and} \quad \big(\Delta(B_0, \upsilon_{pro}(n)) \preceq \Delta(B_0, \upsilon_\rho(n)) \quad \text{or} \quad \ell_{pro}(n) < \ell_\rho(n)\big)$$

If time n_∞ does not come then the inequality $\ell_{pro}(n) \leq \ell_\rho(n)$ from above suffices. Similarly, if at time n_∞ the pair (s, q) is deciding, we are also done. If after time n_∞ the procrastination policy $\rho_{pro}(K)$ observes at least one more letter then ρ also observes at least one more letter. By Lemma 25, one can choose K large so that for $\rho_{pro}(K)$ one additional observation probably suffices. If it is the case that ρ almost surely observes only one letter after n_∞, then $\rho_{pro}(K)$ also needs only one more observation, since it has observed at time n_∞. ◀

It follows that, in order to compute c_{inf}, it suffices to analyze $\mathrm{Ex}(C_{\rho_{pro}(K)})$ for large K. This leads to the following theorem:

▶ **Theorem 28.** *Given a non-hidden MC $\mathcal{M}$ and a DFA $\mathcal{A}$, one can compute c_{inf} in polynomial time.*

Proof. For each (s, q) define $c(K, s, q)$ as in Lemma 25, and define $c(s, q) := \lim_{K \to \infty} c(K, s, q)$. By Lemma 25, for each non-deciding (s, q) with $cras(s, q) = \infty$ we have $c(s, q) = 1$. Hence the $c(s, q)$ satisfy the following system of linear equations where some coefficients come from the procrastination MC $\mathcal{M}_{pro}(\infty)$:

$$c(s,q) \;=\; \begin{cases} 0 & \text{if } (s,q) \text{ is deciding} \\ 1 & \text{if } (s,q) \text{ is not deciding and } cras(s,q) = \infty \\ 1 + c'(s,q) & \text{otherwise} \end{cases}$$

$$c'(s,q) \;=\; \sum_a \sum_{q'} M_{pro}(\infty)\big((s,q), (\overrightarrow{a}, q')\big) \cdot c(\overrightarrow{a}, q') \qquad \text{if } cras(s,q) < \infty$$

By solving the system one can compute $c(s_0, q_0)$ in polynomial time. We have:

$$c_{inf} = \inf_{\text{feasible } \rho} \mathrm{Ex}(C_\rho) \overset{\text{Thm27}}{=} \lim_{K \to \infty} \mathrm{Ex}(C_{\rho_{pro}(K)}) = c(s_0, q_0)$$

Hence one can compute c_{inf} in polynomial time. ◀

6 Empirical Evaluation of the Expected Optimal Cost

We have shown that maximal procrastination is optimal in the non-hidden case (Theorem 27). However, we have not shown *how much* better the optimal policy is than the see-all baseline. It appears difficult to answer this question analytically, so we address it empirically. We implemented our algorithms in a fork of the Facebook Infer static analyzer [8], and applied them to 11 open-source projects, totaling 80 thousand Java methods. We found that in $> 90\%$ of cases the maximally procrastinating monitor is trivial and thus the optimal cost is 0, because Infer decides statically if the property is violated. In the remaining cases, we found that the optimal cost is roughly half of the see-all cost, but the variance is high.

Design. Our setting requires a DFA and an MC representing, respectively, a program property and a program. For this empirical estimation of the expected optimal cost, the DFA is fixed, the MC shape is the symbolic flowgraph of a real program, and the MC probabilities are sampled from Dirichlet distributions.

The DFA represents the following property: 'there are no two calls to *next* without an intervening call to *hasNext*'. To understand how the MC shape is extracted from programs, some background is needed. Infer [8, 9] is a static analyzer that, for each method, infers several preconditions and, attached to each precondition, a symbolic path. For a simple example, consider a method whose body is '$\mathsf{if}\ (b)\ x.next();\ \mathsf{if}\ (!b)\ x.next()$'. Infer would generate two preconditions for it, b and $\neg b$. In each of the two attached symbolic paths, we can see that *next* is not called twice, which we would not notice with a control flowgraph. The symbolic paths are inter-procedural. If a method f calls a method g, then the path of f will link to a path of g and, moreover, it will pick one of the paths of g that corresponds to what is currently known at the call site. For example, if $g(b)$ is called from a state in which $\neg b$ holds, then Infer will select a path of g compatible with the condition $\neg b$.

The symbolic paths are finite because abstraction is applied, including across mutually recursive calls. But, still, multiple vertices of the symbolic path correspond to the same vertex of the control flowgraph. For example, Infer may go around a for-loop five times before noticing the invariant. By coalescing those vertices of the symbolic path that correspond to the same vertex of the control flowgraph we obtain an *SFG (symbolic flowgraph)*. We use such SFGs as the skeleton of MCs. Intuitively, one can think of SFGs as inter-procedural control flowgraphs restricted based on semantic information. Vertices correspond to locations in the program text, and transitions correspond to method calls or returns. Transition probabilities should then be interpreted as a form of static branch prediction. One could learn these probabilities by observing many runs of the program on typical input data, for example by using the Baum–Welch algorithm [17]. Instead, we opt to show that the improvement in expected observation cost is robust over a wide range of possible transition probabilities, which we do by drawing several samples from Dirichlet distributions. Besides, recall that the (optimal) procrastination policy does not depend on transition probabilities.

Once we have a DFA and an MC we compute their product. In some cases, it is clear that the product is empty or universal. These are the cases in which we can give the verdict right away, because no observation is necessary. We then focus on the non-trivial cases.

Table 1 Reduction in expected observation cost, on real-world data. Each SFG (symbolic flowgraph) corresponds to one inferred precondition of a method. The size of monitors is measured in number of language equivalence classes. (LOC = lines of code; GAvg = geometric average.)

	Project Size			Monitors			$c_{inf}/\mathrm{Ex}(C_\circ)$	
Name	Methods	SFGs	LOC	Count	Avg-Size	Max-Size	Med	GAvg
tomcat	26K	52K	946K	343	69	304	0.53	0.50
okhttp	3K	6K	49K	110	263	842	0.46	0.42
dubbo	8K	16K	176K	91	111	385	0.53	0.51
jadx	4K	9K	48K	204	96	615	0.58	0.50
RxJava	12K	45K	192K	83	41	285	0.52	0.53
guava	22K	43K	1218K	1126	134	926	0.41	0.41
clojure	5K	19K	66K	219	120	767	0.44	0.44
AndroidUtilCode	3K	7K	436K	39	89	288	0.66	0.58
leakcanary	1K	1K	11K	12	79	268	0.66	0.59
deeplearning4j	21K	40K	408K	262	51	341	0.58	0.58
fastjson	2K	7K	47K	204	63	597	0.59	0.53

For non-trivial MC $\times$ DFA products, we compute the expected cost of the light see-all policy $\mathrm{Ex}(C_\circ)$, which observes all letters until a decision is made and then stops. We can do so by using standard algorithms [2, Chapter 10.5]. Then, we compute $\mathcal{M}_{pro}$, which we use to compute the expected observation cost c_{inf} of the procrastination policy (Theorem 28). Recall that in order to compute $\mathcal{M}_{pro}$, one needs to compute the *cras* function, and also to find language equivalence classes. Thus, computing $\mathcal{M}_{pro}$ entails computing all the information necessary for implementing a procrastinating monitor.

Methodology. We selected 11 Java projects among those that are most forked on GitHub. We ran Infer on each of these projects. From the inferred specifications, we built SFGs and monitors that employ light see-all policies and maximal procrastination policies. From these monitors, we computed the respective expected costs, solving the linear systems using Gurobi [12]. Our implementation is in a fork of Infer, on GitHub.

Results. The results are given in Table 1. We first note that the number of monitors is much smaller than the number of methods, by a factor of 10 or 100. This is because in most cases we are able to determine the answer statically, by analyzing the symbolic paths produced by Infer. The large factor should not be too surprising: we are considering a fixed property about iterators, not all Java methods use iterators, and, when they do, it is usually easy to tell that they do so correctly. Still, each project has a few hundred monitors, which handle the cases that are not so obvious.

We note that $\frac{c_{inf}}{\mathrm{Ex}(C_\circ)} \approx 0.5$. The table supports this by presenting the median and the geometric average, which are close to each-other; the arithmetic average is also close. There is, however, quite a bit of variation from monitor to monitor, as shown in Figure 1. We conclude that selective monitoring has the potential to significantly reduce the overhead of runtime monitoring.

7 Future Work

In this paper we required policies to be feasible, which means that our selective monitors are as precise as non-selective monitors. One may relax this and study the tradeoff between efficiency (skipping even more observations) and precision (probability of making a decision).

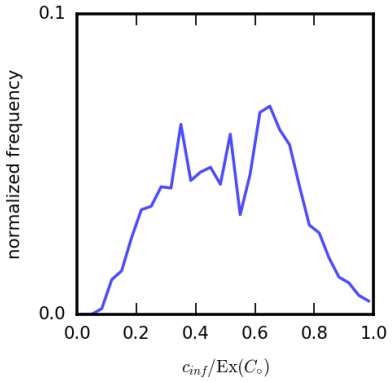


Figure 1 Empirical distribution of $c_{inf}/\mathrm{Ex}(C_\circ)$, across all projects.

Further, one could replace the diagnosability notion of this paper by other notions from the literature; one could investigate how to compute c_{inf} for other classes of MCs, such as acyclic MCs; one could study the sensitivity of c_{inf} to changes in transition probabilities; and one could identify classes of MCs for which selective monitoring helps and classes of MCs for which selective monitoring does not help.

A nontrivial extension to the formal model would be to include some notion of data, which is pervasive in practical specification languages used in runtime verification [13]. This would entail replacing the DFA with a more expressive device, such as a nominal automaton [7], a symbolic automaton [10], or a logic with data (e.g., [11]). Alternatively, one could side-step the problem by using the slicing idea [18], which separates the concern of handling data at the expense of a mild loss of expressive power. Finally, the monitors we computed could be used in a runtime verifier, or even in session type monitoring where the setting is similar [6].

References

1 Matthew Arnold, Martin T. Vechev, and Eran Yahav. QVM: an efficient runtime for detecting defects in deployed systems. In *OOPSLA*, 2008.

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

3 Ezio Bartocci, Radu Grosu, Atul Karmarkar, Scott A. Smolka, Scott D. Stoller, Erez Zadok, and Justin Seyster. Adaptive runtime verification. In *RV*, 2012.

4 N. Bertrand, S. Haddad, and E. Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. In *Proceedings of FSTTCS*, volume 29 of *LIPIcs*, pages 417–429, 2014.

5 N. Bertrand, S. Haddad, and E. Lefaucheux. Accurate approximate diagnosability of stochastic systems. In *Proceedings of LATA*, pages 549–561. Springer, 2016.

6 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *TCS*, 2017.

7 Mikołaj Bojánczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *LMCS*, 2014.

8 C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *NASA Formal Methods Symposium*, 2015.

9 C. Calcagno, D. Distefano, P.W. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *JACM*, 2011.

10 Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *CAV*, 2017.

11 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *TOCL*, 2009.

12 Gurobi Optimization, Inc. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2017.

13 Klaus Havelund, Martin Leucker, Giles Reger, and Volker Stolz. A Shared Challenge in Behavioural Specification (Dagstuhl Seminar 17462). *Dagstuhl Reports*, 2018. `doi:10.4230/DagRep.7.11.59`.

14 S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

15 K. Kalajdzic, E. Bartocci, S.A. Smolka, S.D. Stoller, and R. Grosu. Runtime verification with particle filtering. In *RV*, 2013.

16 J.-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47):5010–5021, 2009.

17 Brian G. Leroux. Maximum-likelihood estimation for hidden markov models. *Stochastic Processes and Their Applications*, 1992.

18 Grigore Roşu and Feng Chen. Semantics and algorithms for parametric monitoring. *LMCS*, 2012.

19 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

20 A. Prasad Sistla, Miloš Žefran, and Yao Feng. Monitorability of stochastic dynamical systems. In *Proceedings of CAV*, pages 720–736. Springer, 2011.

21 S.D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S.A. Smolka, and E. Zadok. Runtime verification with state estimation. In *RV*, 2011.

22 D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4):476–492, 2005.

Synchronizing the Asynchronous

Bernhard Kragl
IST Austria
https://orcid.org/0000-0001-7745-9117

Shaz Qadeer
Microsoft

Thomas A. Henzinger
IST Austria

Abstract

Synchronous programs are easy to specify because the side effects of an operation are finished by the time the invocation of the operation returns to the caller. Asynchronous programs, on the other hand, are difficult to specify because there are side effects due to pending computation scheduled as a result of the invocation of an operation. They are also difficult to verify because of the large number of possible interleavings of concurrent computation threads. We present *synchronization*, a new proof rule that simplifies the verification of asynchronous programs by introducing the fiction, for proof purposes, that asynchronous operations complete synchronously. Synchronization summarizes an asynchronous computation as immediate atomic effect. Modular verification is enabled via *pending asynchronous calls* in atomic summaries, and a complementary proof rule that eliminates pending asynchronous calls when components and their specifications are composed. We evaluate synchronization in the context of a multi-layer refinement verification methodology on a collection of benchmark programs.

2012 ACM Subject Classification Software and its engineering → Formal software verification

Keywords and phrases concurrent programs, asynchronous programs, deductive verification, refinement, synchronization, mover types, atomic action, commutativity, Lipton reduction

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.21

Funding This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

1 Introduction

This paper focuses on the deductive verification of *asynchronous concurrent programs*, an important class that includes distributed fault-tolerant protocols, message-passing programs, client-server applications, event-driven mobile applications, workflows, device drivers, and many embedded and cyber-physical systems. A key aspect of such programs is that (long-running) operations complete asynchronously. A process that invokes an operation does not block for the operation to finish. Instead, the result from the completion of the operation is communicated later, e.g., via a callback message. Asynchronous completion not only introduces concurrency and nondeterminism into the program semantics, but also makes the task of specifying the correct behavior of operations difficult. The behavior of a *synchronous* operation can be specified with a precondition and a postcondition because there is no ambiguity about the state just before and just after the operation executes. The behavior of an *asynchronous* operation is harder to specify because multiple operations can be in flight at the same time and partial results from other operations may have already affected the state before the operation finishes.

In this paper, we propose that reasoning about asynchronous computation can be simplified via *synchronization*, a program transformation that generalizes reduction [15, 6]. While reduction allows the creation of a coarse-grained atomic action from a sequence of fine-grained atomic actions performed by a single thread, synchronization allows the creation of a coarse-grained atomic action from an asynchronous computation executed by a potentially unbounded number of concurrent threads. Synchronization reduces the number of interleavings; it allows us to pretend, for the purposes of proof, that asynchronous calls complete synchronously and atomically, which leads to significantly simpler invariants.

Synchronization, similar to reduction, relies on commutativity properties of low-level atomic actions. Establishing commutativity may be difficult if these atomic actions access shared state that is also accessed by other, interfering concurrent computations. To enable synchronization in the presence of interference, we leverage the observation that commutativity properties among a set of atomic actions can be established by abstracting these actions [5]. In particular, we incorporate synchronization as a program transformation in the verification methodology of *program layers* [12], which allows the programmer to chain together a sequence of increasingly abstract concurrent programs containing atomic actions that are increasingly coarse-grained. Since program layers allow history variables to be introduced, history variables are sufficient for converting an arbitrary safety property into assertions, and the synchronization transformation preserves all assertion failures, our technique is applicable to the proof of arbitrary safety properties of asynchronous programs.

Synchronization, if used naively, leads to summaries that are not modular and hence not reusable. Consider a scenario where a client invokes an operation S of a service, upon whose completion a callback function C is invoked asynchronously. If the code of C is synchronized into S, the summary of S will be cluttered by the effects of C, making reuse across a different client impossible. To solve this problem, we generalize atomic summaries to support *pending asynchronous calls* (*pending asyncs* in short). Using pending asyncs, we can synchronize asynchrony internal to the service, while leaving the asynchronous callback to C as pending in the summary of S, thus enabling the reuse across different clients. Once the summary of S has been absorbed into the client, we need a mechanism to replace the pending async with the effect of the concrete implementation of C. For that we provide a second proof rule to eliminate pending asyncs from specifications.

We integrated our proof rules in the CIVL verifier [9] which provided a baseline framework of program layers. We report on our experience verifying a collection of benchmark programs, showing that our technique enables elegant specifications and proofs of asynchronous programs.

2 Overview

We start with an overview of our new verification technique based on the two concepts *synchronization* and *pending asyncs*. In our examples we follow the convention of writing procedure names capitalized (e.g., `Acquire`), and atomic action names in all caps (e.g., `ACQUIRE`). We use the notation `[...]` to denote unnamed atomic actions, i.e., the statements inside square brackets are considered to execute indivisibly.

2.1 Asynchronous Increments and Decrements

Consider the program in Figure 1 (a). The program comprises a single procedure `Main` that uses a global variable `x` and a local variable `i`. Every iteration of the while loop in `Main` creates two new threads, one executing an atomic increment `[x := x + 1]`, and one

(a)

```
global var x

proc Main(n):
  i := 0
  while i < n:
    async [x := x + 1]
    async [x := x - 1]
    i := i + 1
```

(b)

```
proc Main:
  async Foo
  assert false

proc Foo:
  call Foo
```

(c)

```
global var x

proc Main(n):
  i := 0
  while i < n:
    async [x := x + 1]
    async [x := x - 1]
    if *: i := i + 1
```

Figure 1 Asynchronous increments and decrements.

executing an atomic decrement `[x := x - 1]`. Due to asynchronous thread creation, the execution of individual increments and decrements can be interleaved arbitrarily. However, once all threads finish, the value in variable `x` is equal to its initial value. Thus, `Main` *refines* the atomic action `[skip]`, which does nothing.

A standard noninterference-based proof of this program requires an invariant that states that "`x` is equal to its original value, plus the number of finished increment threads, minus the number of finished decrement threads". Stating this invariant requires ghost code that tracks the progress of each thread. In contrast, our *synchronization* proof rule (Section 4) allows us to consider both asynchronous calls in `Main` as regular synchronous calls. Then sequential reasoning suffices to prove that the procedure leaves the variable `x` unchanged. Synchronization is justified by the commutativity of atomic actions on shared state. Specifically, both increment and decrement are *left movers* in the context of our program. Thus the asynchronous computation steps in an interleaved execution can be rearranged to obtain a corresponding synchronous execution that preserves final states.

However, commutativity alone is not sufficient! We also need to ensure that synchronization preserves failing behaviors. Consider the program in Figure 1 (b) where `Main` asynchronously calls a procedure `Foo` (which calls itself recursively) followed by a failing assertion. The program has failing executions; the assertion can be scheduled any time between steps of `Foo`. If we synchronize the call to `Foo`, however, the nontermination of `Foo` makes the assertion unreachable and thus synchronization must not be allowed. We could require termination of the synchronized program, but this would be unnecessarily restrictive. We propose a weaker condition called *cooperation*, which only requires the possibility to terminate. In other words, it must be impossible for the synchronized program to reach a state where nontermination is inevitable. To illustrate cooperation, consider Figure 1 (c), a modification of (a) which nondeterministically increments the loop counter `i`. The program does not terminate because it *may* loop forever, but it cooperates because it *can* always increment `i`. By synchronization we can show analogously to (a) that (c) also refines `[skip]`.

2.2 Lock Service

Figure 2 (a) shows a simple lock service implementation. A client requests the lock by asynchronously invoking `Acquire`, which is implemented as spinlock using the atomic compare-and-swap (CAS) operation on the global variable `l`. Once successful, the client of the lock service is notified via an asynchronous callback. Summarizing `Acquire` as atomic action via synchronization of the callback is not desirable, because it would drag in the effect of the client into the specification of `Acquire`. Instead, we propose the modular, reusable, and client-independent atomic action specifications `ACQUIRE` and `RELEASE` shown in (b). Notice how we represent guarded atomic transitions as program code. But more importantly, observe that the atomic action specification `ACQUIRE` contains a *pending async* to `Callback`. That is, we allow the effect of asynchronous thread creation as part of atomic actions. Now, to

(a)
```
global var l

proc Acquire (tid):
  b := false
  while !b:
    call b := CAS(l, nil, tid)
  async Callback(tid)

proc Release (tid):
  call [l := nil]
```

(b)
```
global var l

action ACQUIRE (tid):
  assert tid != nil
  assume l == nil
  l := tid
  async Callback(tid)

action RELEASE (tid):
  assert tid != nil && l == tid
  l := nil
```

(c)
```
global var x

proc Callback(tid):
  call [t := x]
  call [x := t + 1]
  async Release(tid)
```

(d)
```
global var x, l

action CALLBACK(tid):
  assert tid != nil
  assert l == tid
  x := x + 1
  l := nil
```

(e)
```
global var x, l

action ACQUIRE' (tid):
  assert tid != nil
  assume l == nil
  l := tid
  x := x + 1
  l := nil
```

(f)
```
global var x

action ACQUIRE'' (tid):
  x := x + 1
```

Figure 2 Lock service.

make use of such specifications, our technique is complemented with a proof rule to eliminate pending asyncs (Section 6), once an atomic action specification for the target is available. For example, consider the callback implementation in (c) that reads and writes a shared variable `x`, and then releases the lock. Since the callback is only supposed to be invoked with the lock held, we strengthen `[t := x]` and `[x := t + 1]` with the gate `assert tid != nil && l == tid`, which makes the operations commutative. Together with `RELEASE` being a left mover, we use synchronization to show that `Callback` refines the atomic action `CALLBACK` in (d). Now that we have an atomic action specification for `Callback`, we use it to eliminate the pending async in `ACQUIRE` and obtain the atomic action `ACQUIRE'` in (e). Notice how the gates of `CALLBACK` are discharged by the code preceding the pending async in `ACQUIRE`. Finally, we can abstract away the lock acquire and release, such that the client of the lock service only sees the atomic action `ACQUIRE''` in (f).

2.3 Layered Refinement Proofs

Our proof rules connect a lower-level, more fine-grained program with a higher-level, more coarse-grained program (both a bottom-up and top-down interpretation is possible), and repeated applications lead to a hierarchy of connected programs. However, due to the structure-preserving nature of our rules, in practice (Section 7) the programmer only writes a single program with *layer annotations* [12] that encode the program on multiple layers of abstraction. Our verifier automatically extracts the hierarchy of programs and generates the necessary verification conditions to justify their connection.

3 An Asynchronous Programming Language

In this section we define a core asynchronous programming language on which we formalize our verification technique, and recall the notion of mover types and reduction.

Variables and stores. Let $\mathcal{V}$ be a set of *variables* partitioned into *global variables* $\mathcal{V}_G$ and *local variables* $\mathcal{V}_L$, and $\mathcal{V}_R \subseteq \mathcal{V}_L$ is a set of *return variables*. A *store* is a mapping $\sigma : \mathcal{V} \to \mathcal{D}$ that assigns a *value* from a domain $\mathcal{D}$ to every variable. Similarly, $g : \mathcal{V}_G \to \mathcal{D}$ is a *global store* and $\ell : \mathcal{V}_L \to \mathcal{D}$ is a *local store*. Let $g{\cdot}\ell$ denote the combination of g and ℓ into a store. To model return values from a procedure with local store ℓ_1 to a caller procedure with local store ℓ_2, we define the resulting store at the caller as $\ell_1 \rhd \ell_2 = \lambda v.\,\text{if } v \in \mathcal{V}_R \text{ then } \ell_1(v) \text{ else } \ell_2(v)$.

$$(g, TC[\ell][\texttt{skip}; s] \uplus \mathcal{T}) \Rightarrow (g, TC[\ell][s] \uplus \mathcal{T}) \quad \textsc{Seq}$$

$$\frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \in \rho \quad (g{\cdot}\ell, g'{\cdot}\ell', \Omega) \in \alpha \quad \mathcal{T}' = \{(\ell'', \texttt{call } P) \mid (\ell'', P) \in \Omega\}}{(g, TC[\ell][\texttt{call } A] \uplus \mathcal{T}) \Rightarrow (g', TC[\ell'][\texttt{skip}] \uplus \mathcal{T}' \uplus \mathcal{T})} \quad \textsc{ActionStep}$$

$$\frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \notin \rho}{(g, TC[\ell][\texttt{call } A] \uplus \mathcal{T}) \Rightarrow \lightning} \quad \textsc{ActionFail}$$

$$\frac{s' = \text{if } (\ell \in le) \text{ then } s_1 \text{ else } s_2}{(g, TC[\ell][\texttt{if } le \texttt{ then } s_1 \texttt{ else } s_2] \uplus \mathcal{T}) \Rightarrow (g, TC[\ell][s'] \uplus \mathcal{T})} \quad \textsc{If}$$

$$(g, TC[\ell][\texttt{call } P] \uplus \mathcal{T}) \Rightarrow (g, (\ell, \mathcal{P}.P){\cdot}TC[\ell][\texttt{skip}] \uplus \mathcal{T}) \quad \textsc{Call}$$

$$(g, (\ell_1, \texttt{skip}){\cdot}TC[\ell_2][s] \uplus \mathcal{T}) \Rightarrow (g, TC[\ell_1 \triangleright \ell_2][s] \uplus \mathcal{T}) \quad \textsc{Return}$$

$$(g, TC[\ell][\texttt{async } P] \uplus \mathcal{T}) \Rightarrow (g, TC[\ell][\texttt{skip}] \uplus (\ell, \texttt{call } P) \uplus \mathcal{T}) \quad \textsc{Async} \qquad (g, (\ell, \texttt{skip}) \uplus \mathcal{T}) \Rightarrow (g, \mathcal{T}) \quad \textsc{End}$$

Figure 3 Small-step operational semantics.

Atomic actions. We generalize gated actions introduced in [5] with the idea of pending asyncs. An *atomic action* is a pair (ρ, α), where the *gate* ρ is a set of stores and the *update* α is a set of *transitions* $(\sigma, \sigma', \Omega)$ where σ,σ' are stores and Ω is a finite multiset of *pending asyncs* (ℓ, P) consisting of a local store and a procedure name. If an atomic action is executed in a store σ with $\sigma \notin \rho$, the program "fails"; otherwise, if $\sigma \in \rho$, a transition $(\sigma, \sigma', \Omega) \in \alpha$ atomically updates the store to σ' and creates new threads according to Ω.

Atomic actions subsume many standard programming language statements. In particular, (nondeterministic) assignments, assertions, and assumptions. The table on the right shows some examples ranging over variables x and y.

command	gate	update
$x := x + y$	*true*	$x' = x + y \wedge y' = y$
havoc x	*true*	$y' = y$
assert $x < y$	$x < y$	$x' = x \wedge y' = y$
assume $x < y$	*true*	$x < y \wedge x' = x \wedge y' = y$

Syntax. A program $\mathcal{P}$ is a finite mapping from *atomic action names* A to atomic actions, and *procedure names* P to *statements* s of the form

$$s ::= \texttt{skip} \mid s; s \mid \texttt{if } le \texttt{ then } s \texttt{ else } s \mid \texttt{call } A \mid \texttt{call } P \mid \texttt{async } P.$$

A program contains a dedicated procedure *Main* that serves as an entry point for executions, and every atomic action name respectively procedure name appearing in a call statement must be properly mapped to an atomic action respectively statement. We will write $\mathcal{P}.A$ and $\mathcal{P}.P$ for $\mathcal{P}(A)$ and $\mathcal{P}(P)$, and $A, P \in \mathcal{P}$ for $A, P \in dom(\mathcal{P})$. We identify the conditional expression le with the set of local stores that satisfy it.

Semantics. A *frame* f is a pair (ℓ, s) of local store ℓ and statement s. A *thread* t is a sequence of frames $\vec{f}$, denoting a call stack. A *state* $(g, \mathcal{T})$ is a pair of global store g and a finite multiset of threads $\mathcal{T}$. By slight abuse of notation we will identify a thread t with the singleton multiset $\{t\}$, and thus write $\mathcal{T} \uplus t$ for adding t to $\mathcal{T}$. Let *statement contexts* SC, *frame contexts* FC, and *thread contexts* TC be defined as follows:

$$SC ::= \bullet_{Stmt} \mid SC; s \qquad FC ::= (\bullet_{LStore}, SC) \qquad TC ::= FC{\cdot}\vec{f}$$

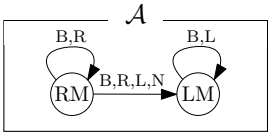
$TC[\ell][s]$ denotes the thread obtained by filling the two unique holes $\bullet_{Stmt}$ and $\bullet_{LStore}$ in TC with statement s and local store ℓ, respectively. Thus, $TC[\ell][s]$ executes s from ℓ as next step. The operational semantics is formalized in Figure 3 as a transition relation $\Rightarrow$ between states. An *execution* π is a sequence of states $x_0 \Rightarrow x_1 \Rightarrow \ldots$, and we write $\pi : x_0 \Rightarrow^* x_n$ to denote that π is an execution that starts in x_0 and ends in x_n.

Refinement. Given a program $\mathcal{P}$, we are interested in executions that start with a single thread executing *Main* from some initial store $\sigma = g{\cdot}\ell$, i.e., executions that start in a state $(g, (\ell, \texttt{call}\ \mathit{Main}))$. In particular, we are interested in executions that either fail or terminate. We define $\mathit{Bad}(\mathcal{P})$ to be the set of initial stores associated with *failing executions*, and $\mathit{Good}(\mathcal{P})$ to be the relation between initial and final stores associated with *terminating executions*:

$$\mathit{Bad}(\mathcal{P}) = \left\{g{\cdot}\ell \mid \left(g, (\ell, \texttt{call}\ \mathit{Main})\right) \Rightarrow^* \lightning\right\};\quad \mathit{Good}(\mathcal{P}) = \left\{(g{\cdot}\ell, g') \mid \left(g, (\ell, \texttt{call}\ \mathit{Main})\right) \Rightarrow^* (g', \varnothing)\right\}.$$

A program $\mathcal{P}_1$ *refines* a program $\mathcal{P}_2$, denoted $\mathcal{P}_1 \preccurlyeq \mathcal{P}_2$, if (1) $\mathit{Bad}(\mathcal{P}_1) \subseteq \mathit{Bad}(\mathcal{P}_2)$ and (2) $\overline{\mathit{Bad}(\mathcal{P}_2)} \circ \mathit{Good}(\mathcal{P}_1) \subseteq \mathit{Good}(\mathcal{P}_2)$; $\overline{\cdot}$ is set complement, $\circ$ is relation composition. The first condition states that $\mathcal{P}_2$ has to preserve failing executions of $\mathcal{P}_1$. The second condition states that $\mathcal{P}_2$ has to preserve terminating executions of $\mathcal{P}_1$ for initial states that cannot fail. That is, $\mathcal{P}_2$ can fail more often than $\mathcal{P}_1$.

Reduction. Let M be a mapping from atomic action names to *mover types* [6]: B (*both mover*), L (*left mover*), R (*right mover*), N (*non-mover*). Intuitively, an atomic action is a right mover, if it commutes to the right (i.e., later in time) with respect to all other atomic actions in $\mathcal{P}$. A left mover is symmetric, and an atomic action can be both a left and right mover. Reduction has traditionally been applied to multithreaded programs to convert a sequence of atomic actions performed by a single thread into an atomic block. The sequence of mover types of the atomic actions in this block must be a valid run of the nondeterministic *atomicity automaton* $\mathcal{A}$ on the right. In this paper, we exploit and extend this work to synchronize asynchronous computation spanning multiple threads.



We define the predicate $\mathit{MoverValid}(\mathcal{P}, M)$ which holds whenever the atomic actions in $\mathcal{P}$ satisfy the mover types indicated by M. Formally, $\mathit{MoverValid}(\mathcal{P}, M)$ holds if for all $A_1, A_2 \in \mathcal{P}$ with $\mathcal{P}.A_1 = (\rho_1, \alpha_1)$ and $\mathcal{P}.A_2 = (\rho_2, \alpha_2)$, the following conditions hold (generalizing [10] to support pending asyncs).

- **Commutativity:** If $M(A_1) \in \{R, B\}$ or $M(A_2) \in \{L, B\}$, then the effect of executing A_1 followed by A_2 in two different threads can also be achieved by A_2 followed by A_1.

$$\begin{array}{l}\forall g, \bar{g}, g', \ell_1, \ell_1',\\ \quad \ell_2, \ell_2', \Omega_1, \Omega_2\\ \exists \hat{g}, \Omega_1', \Omega_2'\end{array} : \left(\begin{array}{ll} \wedge & g{\cdot}\ell_1 \in \rho_1\\ \wedge & g{\cdot}\ell_2 \in \rho_2\\ \wedge & (g{\cdot}\ell_1, \bar{g}{\cdot}\ell_1', \Omega_1) \in \alpha_1\\ \wedge & (\bar{g}{\cdot}\ell_2, g'{\cdot}\ell_2', \Omega_2) \in \alpha_2\end{array}\right) \implies \left(\begin{array}{ll} \wedge & (g{\cdot}\ell_2, \hat{g}{\cdot}\ell_2', \Omega_2') \in \alpha_2\\ \wedge & (\hat{g}{\cdot}\ell_1, g'{\cdot}\ell_1', \Omega_1') \in \alpha_1\\ \wedge & \Omega_1 \uplus \Omega_2 = \Omega_1' \uplus \Omega_2'\end{array}\right)$$

- **Forward preservation:** If $M(A_1) \in \{R, B\}$ or $M(A_2) \in \{L, B\}$, then the failure of A_2 after the execution of A_1 implies that A_2 must also fail before the execution of A_1.
  $\forall g, g', \ell_1, \ell_1', \ell_2, \Omega_1 : (g{\cdot}\ell_1 \in \rho_1 \wedge g{\cdot}\ell_2 \in \rho_2 \wedge (g{\cdot}\ell_1, g'{\cdot}\ell_1', \Omega_1) \in \alpha_1) \implies g'{\cdot}\ell_2 \in \rho_2$
- **Backward preservation:** If $M(A_2) \in \{L, B\}$, then the failure of A_1 before the execution of A_2 implies that A_1 must also fail after the execution of A_2. $\forall g, g', \ell_1, \ell_2, \ell_2', \Omega_2 :$ $(g{\cdot}\ell_2 \in \rho_2 \wedge (g{\cdot}\ell_2, g'{\cdot}\ell_2', \Omega_2) \in \alpha_2 \wedge g'{\cdot}\ell_1 \in \rho_1) \implies g{\cdot}\ell_1 \in \rho_1$
- **Nonblocking:** If $M(A_2) \in \{L, B\}$, then A_2 must be nonblocking. $\forall \sigma \in \rho_2\ \exists \sigma', \Omega :$ $(\sigma, \sigma', \Omega) \in \alpha_2$
- **Async freedom:** If $M(A_1) \in \{R, B\}$, then A_1 cannot have pending asynchronous calls.
  $\forall \sigma, \sigma', \Omega : \sigma \in \rho_1 \wedge (\sigma, \sigma', \Omega) \in \alpha_1 \implies \Omega = \varnothing$

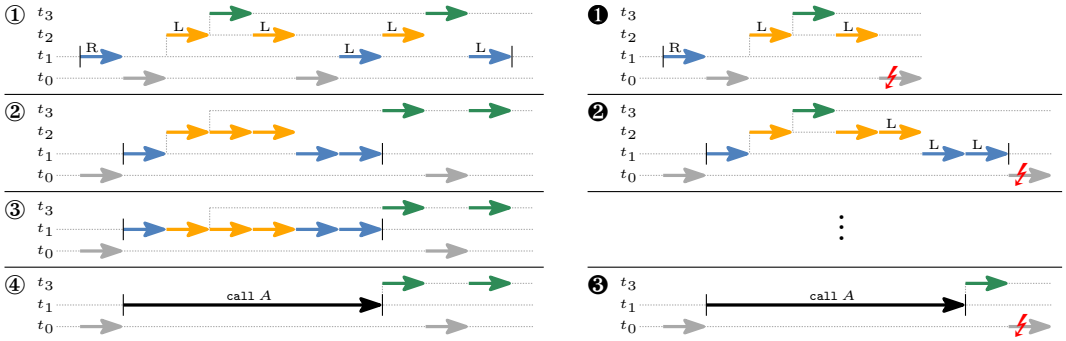


Figure 4 Synchronizing asynchronous executions.

4 Synchronizing Asynchrony

In this section, we formalize the synchronization proof rule which allows us to transform a procedure into an atomic action that summarizes asynchronous effects, either directly or via pending asyncs. Synchronization requires two technical innovations. First, we extend the *commutativity* conditions required for reduction to account for asynchronous thread creation. Second, we impose a new *cooperation* condition necessary for the soundness of our transformation.

Given a procedure Q, a mover typing M, and a set of procedures Σ to synchronize in Q (asynchronous calls to procedures not in Σ are treated as pending asyncs), the SYNCHRONIZE rule transforms procedure Q into an atomic action (ρ, α) with fresh name A:

SYNCHRONIZE

$$\frac{\mathit{MoverValid}(\mathcal{P}, M) \quad \mathit{Sync}(\mathcal{P}, M, Q, \Sigma) \quad \mathit{Refinement}(\mathcal{P}, Q, \Sigma, \rho, \alpha)}{\mathcal{P} \rightsquigarrow \mathcal{P}[Q \mapsto \texttt{call } A] \cup [A \mapsto (\rho, \alpha)]} \quad \begin{array}{l} Q \in \mathcal{P} \\ A \notin \mathcal{P} \end{array}$$

We already defined *MoverValid* in the previous section. Now we informally discuss the soundness of SYNCHRONIZE, and formally defined the other two premises *Sync* and *Refinement*. In the next section we show how all premises can be efficiently checked in practice.

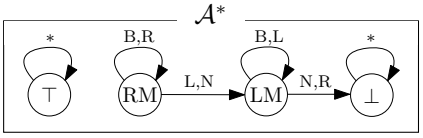
▶ **Theorem 1.** *If* $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2$ *using the* SYNCHRONIZE *rule, then* $\mathcal{P}_1 \preccurlyeq \mathcal{P}_2$.

Intuition. The core idea of Theorem 1 is the rewriting of a $\mathcal{P}_1$-execution π_1 into an equivalent $\mathcal{P}_2$-execution π_2. Concretely, (1) if π_1 fails then π_2 must fail, and (2) if π_1 terminates then π_2 must either terminate with the same final state or fail. We illustrate this transformation in Figure 4. On the left, ① shows part of an asynchronous execution, initially comprising two threads t_0 and t_1. Thread t_1 executes the transformed procedure Q (the call and return are indicated with black bars), which makes an asynchronous call to spawn t_2, and t_2 asynchronously spawns t_3. Notice that t_2 terminates after three steps. We consider the procedure of t_2 to be in Σ (i.e., to be synchronized), while the procedure of t_3 is not in Σ (i.e., to be treated as pending async). Our goal is to transform execution ① into execution ②, which has the following properties: (1) Q executes without interruption from t_0, (2) t_2 terminates without interruption before t_1 continues, and (3) t_3 only starts after Q returns. To permit this transformation, *Sync* requires that Q, *including* asynchronous calls to procedures in Σ, executes a sequence of right movers, followed by at most one non-mover, followed by a sequence of left movers. Furthermore, the asynchronous calls to procedures in Σ must only execute left movers. The steps of t_1 and t_2 in ① are labeled with mover types that satisfy

this conditions. When moving the right mover to the right and the left movers to the left to obtain ②, the commutativity, forward preservation, and backward preservation properties of *MoverValid* guarantee that the executions stay equivalent. Now, as shown in ③, the steps of t_2 can be considered to execute synchronously in its parent t_1. Finally, *Refinement* ensures that the synchronized behavior of Q is summarized by the atomic action A in ④, which captures the creation of t_3 as pending async.

On the right of Figure 4, ❶ shows an execution where Q started, but then t_0 failed. Notice that, if all steps of Q before the failure are right movers, these steps can be removed from the execution (by moving them to the right, "past" the failure), and the failure occurs before Q even starts. In ❶, however, Q already executed a left mover. Even if we move the steps of Q together, the partial execution of Q is not summarized by A. However, we know that only left movers can follow in t_1 and t_2. Since left movers are non-blocking and backward preserving, they can be inserted at the end of the execution, right before the failure. The *cooperation* condition (part of *Sync*) ensures that this can be done so that Q is completed, as shown in ❷. Then we can again arrive at an execution where Q is replaced by A (see ❸).

Concurrent tracking semantics. The execution in ① is labeled with mover types that allowed us to rearrange the steps of Q to obtain the execution in ②. To characterize the executions for which such a rearrangement is possible in general, we define the *concurrent tracking semantics* $\xrightarrow{M,Q,\Sigma}$ (Figure 5) that is similar to $\Rightarrow$, except that we additionally track a *mover phase* m in frames, which is one of the states of the *tracking automaton* $\mathcal{A}^*$ on the right: $\top$ (*no tracking*), RM (*right-mover phase*), LM (*left-mover phase*), $\bot$ (*violation*). Call transitions from $\top$ to RM on a top-level call to Q, or otherwise propagates the mover phase of the caller to the callee. Conversely, Return transitions back to $\top$ when returning from a top-level call to Q, or otherwise propagates the mover phase of the callee to the caller. ActionStep follows a transition in $\mathcal{A}^*$ according to the mover type of the invoked atomic action. In particular, if we are tracking ($m \neq \top$), we stay in RM until a non-right mover (L or N) causes a transition to LM. In LM only left movers should follow, and thus the occurrence of a non-left mover (N or R) causes a transition to the violation state $\bot$. Notice that the async freedom condition of *MoverValid* forces a thread that executes an atomic action with pending asyncs to LM. This is important to ensure that only left movers can follow, which can be moved before the steps of any pending async. Similarly, Async transitions the parent thread of an asynchronous call to LM. The child thread is set to LM if we want to synchronize the call, otherwise it is not tracked. In both ActionStep and Async, if an untracked child thread executes `call` Q, the subsequent application of Call will start to track the child tread separately.



Sequential synchronized semantics. In ③ we are concerned with the sequential execution of Q, with asynchronous calls to procedures in Σ being synchronized. We formally define the *sequential synchronized semantics* $\xrightarrow{\Sigma}$ (Figure 5) that executes a single thread and stores a multiset of pending asyncs. In ActionStep, the pending asyncs of an atomic action are added to the already existing pending asyncs. For an asynchronous call to P, Async records a pending thread creation if $P \notin \Sigma$, and synchronizes the call if $P \in \Sigma$. The synchronized stack frame is marked with $\sharp$ such that it is popped in AsyncReturn without writing return variables to the caller. This technicality is necessary in our formalization. In practice, asynchronously called procedures simply cannot have return parameters.

$$\xrightarrow{M,Q,\Sigma}$$

$$(g, TC[\ell][\texttt{skip}; s][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, TC[\ell][s][m] \uplus \mathcal{T}) \ \textsc{Seq}$$

$$\frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \in \rho \quad (g{\cdot}\ell, g'{\cdot}\ell', \Omega) \in \alpha \quad m' = \mathcal{A}^*(m, M(A)) \quad \mathcal{T}' = \{(\ell'', \texttt{call}\ P, \top) \mid (\ell'', P) \in \Omega\}}{(g, TC[\ell][\texttt{call}\ A][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g', TC[\ell'][\texttt{skip}][m'] \uplus \mathcal{T}' \uplus \mathcal{T})} \ \textsc{ActionStep}$$

$$\frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \notin \rho}{(g, TC[\ell][\texttt{call}\ A][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} \lightning} \ \textsc{ActionFail}$$

$$\frac{s' = \text{if } (\ell \in le) \text{ then } s_1 \text{ else } s_2}{(g, TC[\ell][\texttt{if}\ le\ \texttt{then}\ s_1\ \texttt{else}\ s_2][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, TC[\ell][s'][m] \uplus \mathcal{T})} \ \textsc{If}$$

$$\frac{m' = \text{if } (m = \top \wedge P = Q) \text{ then RM else } m}{(g, TC[\ell][\texttt{call}\ P][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, (\ell, \mathcal{P}.P, m'){\cdot}TC[\ell][\texttt{skip}][m] \uplus \mathcal{T})} \ \textsc{Call}$$

$$\frac{m' = \text{if } (m_2 = \top) \text{ then } \top \text{ else } m_1}{(g, (\ell_1, \texttt{skip}, m_1){\cdot}TC[\ell_2][s][m_2] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, TC[\ell_1 \rhd \ell_2][s][m'] \uplus \mathcal{T})} \ \textsc{Return}$$

$$\frac{m' = \text{if } (m \neq \top) \text{ then LM else } \top \quad m'' = \text{if } (m \neq \top \wedge P \in \Sigma) \text{ then LM else } \top}{(g, TC[\ell][\texttt{async}\ P][m] \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, TC[\ell][\texttt{skip}][m'] \uplus (\ell, \texttt{call}\ P, m'') \uplus \mathcal{T})} \ \textsc{Async}$$

$$(g, (\ell, \texttt{skip}, m) \uplus \mathcal{T}) \xrightarrow{M,Q,\Sigma} (g, \mathcal{T}) \ \textsc{End}$$

$$\xrightarrow{\Sigma}$$

$$(g, TC[\ell][\texttt{skip}; s], \Omega) \xrightarrow{\Sigma} (g, TC[\ell][s], \Omega) \ \textsc{Seq}$$

$$\frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \in \rho \quad (g{\cdot}\ell, g'{\cdot}\ell', \Omega) \in \alpha}{(g, TC[\ell][\texttt{call}\ A], \Omega') \xrightarrow{\Sigma} (g', TC[\ell'][\texttt{skip}], \Omega \uplus \Omega')} \ \textsc{ActionStep} \qquad \frac{\mathcal{P}.A = (\rho, \alpha) \quad g{\cdot}\ell \notin \rho}{(g, TC[\ell][\texttt{call}\ A], \Omega) \xrightarrow{\Sigma} \lightning} \ \textsc{ActionFail}$$

$$\frac{s' = \text{if } (\ell \in le) \text{ then } s_1 \text{ else } s_2}{(g, TC[\ell][\texttt{if}\ le\ \texttt{then}\ s_1\ \texttt{else}\ s_2], \Omega) \xrightarrow{\Sigma} (g, TC[\ell][s'], \Omega)} \ \textsc{If}$$

$$(g, TC[\ell][\texttt{call}\ P], \Omega) \xrightarrow{\Sigma} (g, (\ell, \mathcal{P}.P){\cdot}TC[\ell][\texttt{skip}], \Omega) \ \textsc{Call}$$

$$(g, (\ell_1, \texttt{skip}){\cdot}TC[\ell_2][s], \Omega) \xrightarrow{\Sigma} (g, TC[\ell_1 \rhd \ell_2][s], \Omega) \ \textsc{Return}$$

$$(g, TC[\ell][\texttt{async}\ P], \Omega) \xrightarrow{\Sigma} \begin{cases} (g, TC[\ell][\texttt{skip}], (\ell, P) \uplus \Omega) & \text{if } P \notin \Sigma \\ (g, (\ell, \texttt{call}\ P)^\sharp{\cdot}TC[\ell][\texttt{skip}], \Omega) & \text{if } P \in \Sigma \end{cases} \ \textsc{Async}$$

$$(g, (\ell, \texttt{skip})^\sharp{\cdot}\vec{f}, \Omega) \xrightarrow{\Sigma} (g, \vec{f}, \Omega) \ \textsc{AsyncReturn}$$

Figure 5 Concurrent tracking semantics $\xrightarrow{M,Q,\Sigma}$ and sequential synchronized semantics $\xrightarrow{\Sigma}$.

With the concurrent tracking semantics and the sequential synchronized semantics we can now formally define *Sync* and *Refinement*.

Sync. $\mathit{Sync}(\mathcal{P}, M, Q, \Sigma)$ comprises the following two conditions:

S1 $(g, (\ell, \texttt{call}\ \mathit{Main}, \top)) \xrightarrow{M,Q,\Sigma}{}^* (g', TC[\ell'][s][m] \uplus \mathcal{T})$ implies $m \neq \bot$;

S2 $(g, (\ell, \texttt{call}\ \mathit{Main}, \top)) \xrightarrow{M,Q,\Sigma}{}^* (g', TC[\ell'][\texttt{call}\ P][\text{LM}] \uplus \mathcal{T})$ implies $(g', (\ell', \texttt{call}\ P), \varnothing) \xrightarrow{\Sigma}{}^* (g'', (\ell'', \texttt{skip}), \Omega'')$.

S1 states that executions respect the required mover sequences, i.e., no violation is reachable in the tracking semantics. S2 (the cooperation condition) states that every procedure call in the left-mover phase can be completed. The repeated application of S1 allows us to complete partial executions of Q. Note that S2 also captures asynchronous calls to procedures P with $P \in \Sigma$, since the operational semantics rewrites `async` P into `call` P.

Refinement. $\mathit{Refinement}(\mathcal{P}, Q, \Sigma, \rho, \alpha)$ comprises the following two conditions:

R1 $\rho \cap \{g{\cdot}\ell \mid (g, (\ell, \mathcal{P}.Q), \varnothing) \xrightarrow{\Sigma}{}^{*} \lightning\} = \varnothing$;

R2 $\rho \circ \{(g{\cdot}\ell, g'{\cdot}\ell', \Omega) \mid (g, (\ell, \mathcal{P}.Q), \varnothing) \xrightarrow{\Sigma}{}^{*} (g', (\ell', \texttt{skip}), \Omega)\} \subseteq \alpha$.

R1 states that the gate of A is strong enough to filter out all failures of Q, and R2 states that the transition relation of A captures all non-failing executions of Q.

5 Verifying Synchronization

In this section we show how the premises of the SYNCHRONIZE rule can be efficiently checked in practice. The *MoverValid* and *Refinement* premises both lead to standard verification conditions. In particular, the constraints of *MoverValid* state the commutativity of individual atomic actions, and the constraints of *Refinement* state that a sequential procedure is summarized by a transition relation, which can be readily handed off to logical reasoning engines. Thus we focus on *Sync* which we decompose as follows:

$$\frac{\mathit{StaticSync}(\mathcal{P}, M, Q, \Sigma, \mathit{Pre}) \quad \mathit{Safe}(\mathcal{P}, \mathit{Pre}) \quad \mathit{Terminates}(\mathcal{P}, \Sigma, \mathit{Pre}, \mathit{Red})}{\mathit{Sync}(\mathcal{P}, M, Q, \Sigma)}$$

We establish *Sync* in three steps. First, *StaticSync* is a static control-flow analysis that over-approximates the tracking semantics. It uses the domain of a *precondition mapping* *Pre*, a partial mapping from procedure names to sets of stores. If *StaticSync* succeeds, it guarantees S1 (i.e., that $\bot$ cannot be reached) and that all procedures P called with mover phase LM in S2 are in $\mathit{dom}(\mathit{Pre})$. Second, we over-approximate the possible stores $g'{\cdot}\ell'$ at these calls. For that, *Safe* requires that *Pre* denotes valid preconditions, i.e., if `call` P is reachable with store $g'{\cdot}\ell'$, then $g'{\cdot}\ell' \in \mathit{Pre}(P)$ for all $P \in \mathit{dom}(\mathit{Pre})$. Then finally, to establish S2, it remains to show that there is some terminating sequential execution from $(g', (\ell', \texttt{call}\ P), \varnothing)$ for every $P \in \mathit{dom}(\mathit{Pre})$ and $g'{\cdot}\ell' \in \mathit{Pre}(P)$. *Terminates* reduces these cooperation checks to standard termination checks on a restricted program. In particular, the *restriction function* *Red* limits the nondeterministic behavior of some atomic actions. Then showing that *all* executions in the restricted program terminate implies that there is *some* terminating execution in the original program (given that *Red* is not allowed to make atomic actions blocking).

StaticSync. Let $\mathcal{E}$ be the function that maps a mover type to the corresponding set of edges in $\mathcal{A}$, e.g., $\mathcal{E}(\mathrm{R}) = \{\mathrm{RM} \to \mathrm{RM}, \mathrm{RM} \to \mathrm{LM}\}$. We define an interprocedural control flow analysis that lifts $\mathcal{E}$ to a mapping $\widehat{\mathcal{E}}$ on statements, corresponding to the paths a statement may take in the tracking semantics. We write $\mathit{StaticSync}(\mathcal{P}, M, Q, \Sigma, \mathit{Pre})$ if there is a solution $\widehat{\mathcal{E}}(\mathcal{P}.Q) \neq \varnothing$ to the following equations w.r.t. M, Σ and Pre:

$$\widehat{\mathcal{E}}(\texttt{skip}) = \mathcal{E}(\mathrm{B})$$
$$\widehat{\mathcal{E}}(\texttt{call}\ A) = \mathcal{E}(M(A))$$
$$\widehat{\mathcal{E}}(s_1; s_2) = \widehat{\mathcal{E}}(s_1) \circ \widehat{\mathcal{E}}(s_2)$$
$$\widehat{\mathcal{E}}(\texttt{if}\ \mathit{le}\ \texttt{then}\ s_1\ \texttt{else}\ s_2) = \widehat{\mathcal{E}}(s_1) \cap \widehat{\mathcal{E}}(s_2)$$

$$\widehat{\mathcal{E}}(\texttt{call}\ P) = \begin{cases} \widehat{\mathcal{E}}(\mathcal{P}.P) & \text{if } P \in \mathit{dom}(\mathit{Pre}) \\ \widehat{\mathcal{E}}(\mathcal{P}.P) \cap \{\mathrm{RM}\}^2 & \text{if } P \notin \mathit{dom}(\mathit{Pre}) \end{cases}$$

$$\widehat{\mathcal{E}}(\texttt{async}\ P) = \begin{cases} \{\mathrm{LM}\}^2 & \text{if } P \notin \Sigma \\ \{\mathrm{LM}\}^2 \cap \widehat{\mathcal{E}}(\mathcal{P}.P) & \text{if } P \in \Sigma \cap \mathit{dom}(\mathit{Pre}) \\ \varnothing & \text{if } P \in \Sigma \setminus \mathit{dom}(\mathit{Pre}) \end{cases}$$

The equations on the left capture regular control-flow propagation. The equation for `call` P has two cases. If $P \in \mathit{dom}(\mathit{Pre})$, we do not restrict the call since P is cooperative. However, if $P \notin \mathit{dom}(\mathit{Pre})$ we must restrict the call to stay in the right-mover phase, because we cannot rely on the cooperation condition to complete partial executions of Q. The equation for `async` P has three cases. If $P \notin \Sigma$, we do not synchronize P and thus only require the

caller to be followed by only left movers. If $P \in \Sigma \cap dom(Pre)$, we additionally require the invoked procedure P to be only left movers. For synchronized procedures we always have to establish cooperation, thus the case $P \in \Sigma \setminus dom(Pre)$ is not allowed.

If $StaticSync(\mathcal{P}, M, Q, \Sigma, Pre)$, then S1 holds and for every `call` P reachable with LM in S2 we have $P \in dom(Pre)$. Hence, we must check cooperation for all procedures in $dom(Pre)$.

Safe. Now that we know the procedures that need to be checked for cooperation, we want to know the stores from which to check cooperation. For that, Pre must denote valid preconditions. We write $Safe(\mathcal{P}, Pre)$, if $(g, (\ell, \texttt{call } Main)) \Rightarrow^* (g', TC[\ell'][\texttt{call } P] \uplus \mathcal{T})$ implies $g' \cdot \ell' \in Pre(P)$ for all $P \in dom(Pre)$.

Terminates. Finally, we establish S2 by showing that all procedures P in $dom(Pre)$ cooperate from states in $Pre(P)$. Suppose that cooperation holds, but termination (which is stronger) does not. Such a difference between termination and cooperation must be due to nondeterminism. Thus, if we suitably restrict the nondeterminism to eliminate nonterminating behaviors, proving termination for the restricted program implies cooperation for the original program. Formally, a *restriction function* Red is a partial mapping from atomic action names to atomic actions, such that for all $A \in dom(Red)$ with $\mathcal{P}.A = (\rho, \alpha)$ it holds that $Red(A) = (\rho, \alpha')$ with $\alpha' \subseteq \alpha$ and $Red(A)$ is nonblocking. Let $\mathcal{P}^{Red}$ be the program equal to $\mathcal{P}$, except that $\mathcal{P}^{Red}.A = Red(A)$ for $A \in dom(Red)$.

We write $Terminates(\mathcal{P}, \Sigma, Pre, Red)$, if for all $P \in dom(Pre)$ and $g \cdot \ell \in Pre(P)$, there is no infinite sequential synchronized $\mathcal{P}^{Red}$-execution $(g, (\ell, \texttt{call } P), \varnothing) \xrightarrow{\Sigma} \cdots$. Notice that these termination checks can now be solved by a standard termination checker for sequential programs. While it is possible for the programmer to explicitly provide restricted atomic actions, in practice we did not found this necessary for any of our examples. Instead, a fixed policy to resolve nondeterministic branching (e.g., always take the *then* branch) was enough. For example, recall the program in Figure 1 (c). Always taking the *then* branch (i.e., resolving the nondeterministic choice to *true*) allows us to prove termination and thus implies cooperation of the original program.

▶ **Theorem 2.** *If we have* $StaticSync(\mathcal{P}, M, Q, \Sigma, Pre)$, $Safe(\mathcal{P}, Pre)$, *and* $Terminates(\mathcal{P}, \Sigma, Pre, Red)$, *then* $Sync(\mathcal{P}, M, Q, \Sigma)$ *holds.*

6 Eliminating Pending Asynchrony

In the previous two sections we showed how the Synchronize rule allows to summarize procedures to atomic actions, by either directly synchronizing asynchronous calls or keeping them as pending asyncs. In this section we present the complementary PendingAsyncElim rule to eliminate pending asyncs from atomic actions.

Let A be an atomic action with pending asyncs to a procedure P. Eliminating those pending asyncs requires that (1) P is summarized to an atomic action, say B, and (2) B must be a left mover, since we will directly compose its effect with A. Now we show the construction of the new gate and update for A. The new gate is obtained by filtering out all states from the gate of A that can cause B to fail. In other words, we strengthen A's gate such that it cannot make a transition to a state where B fails:

$$Gt(\rho_A, \alpha_A, \rho_B, P) = \left\{ \sigma \in \rho_A \mid \forall \begin{array}{c} g', \ell', \\ \ell_P, \Omega \end{array} : \begin{array}{c} (\sigma, g' \cdot \ell', (\ell_P, P) \uplus \Omega) \in \alpha_A \\ \implies g' \cdot \ell_P \in \rho_B \end{array} \right\}$$

The new update consists of two parts. First, we take all transitions without pending asyncs to P:

$$Upd_1(\alpha_A, P) = \{(\sigma, \sigma', \Omega) \in \alpha_A \mid \neg\exists \ell_P : (\ell_P, P) \in \Omega\}$$

Second, we compose all transitions with a pending async to P with the transitions of B:

$$Upd_2(\alpha_A, \alpha_B, P) = \left\{ (\sigma, g''\cdot\ell', \Omega \uplus \Omega') \mid \exists \begin{array}{c} g', \ell_P, \\ \Omega, \ell'' \end{array} : \wedge \begin{array}{l} (\sigma, g'\cdot\ell', (\ell_P, P) \uplus \Omega) \in \alpha_A \\ (g'\cdot\ell_P, g''\cdot\ell'', \Omega') \in \alpha_B \end{array} \right\}$$

Notice that the transitions of B can have pending asyncs that are absorbed into the resulting transition. Combining all pieces, we obtain the following rule for eliminating pending asyncs:

$$\boxed{\textsc{PendingAsyncElim}}$$
$$\frac{\begin{array}{ccc} \mathcal{P}.P = \texttt{call}\ B & \mathcal{P}.B = (\rho_B, \alpha_B) & M(B) \in \{\mathrm{L}, \mathrm{B}\} \\ \rho'_A = Gt(\rho_A, \alpha_A, \rho_B, P) & \alpha'_A = Upd_1(\alpha_A, P) \cup Upd_2(\alpha_A, \alpha_B, P) & \end{array}}{\mathcal{P} \uplus [A \mapsto (\rho_A, \alpha_A)] \rightsquigarrow \mathcal{P} \uplus [A \mapsto (\rho'_A, \alpha'_A)]}$$

▶ **Example 3.** Recall our motivating lock service example from Section 2.2. Eliminating the pending async in `ACQUIRE` is a formal application of PendingAsyncElim with $P =$ `Callback`, $A =$ `ACQUIRE`, and $B =$ `CALLBACK`. The resulting action (the new A) is `ACQUIRE'`.

▶ **Theorem 4.** *If* $\mathcal{P}_1 \rightsquigarrow \mathcal{P}_2$ *using the* PendingAsyncElim *rule, then* $\mathcal{P}_1 \preccurlyeq \mathcal{P}_2$.

PendingAsyncElim eliminates a single pending async to P in A. Iterative application of the rule allows us to eliminate finitely many pending asyncs. In theory, PendingAsyncElim can be generalized with an induction schema to eliminate unboundedly many pending asyncs, but we did not find this necessary in practice.

7 Evaluation

We implemented our verification method in CIVL [9], a verification system for concurrent programs based on automated and modular refinement reasoning. In CIVL, a program is specified and verified across multiple layers of refinement. At each layer, procedures can be declared to refine atomic actions and henceforth appear atomic to higher layers. This means that an input program with layer annotations implicitly describes the program at multiple levels of abstraction, and CIVL automatically checks refinement between programs on adjacent layers.

We implemented and verified a collection of nine benchmarks, of which five expand on our motivating example from Section 2.1, one is a ping-pong agreement protocol that exercises the notion of cooperation, and the remaining three examples are discussed in the remainder of this section to illustrate (1) the interaction with CIVL and modular verification via pending asyncs, (2) the applicability to challenging concurrency, and (3) one-shot synchronization of nested asynchronous calls. Overall, our benchmarks capture realistic patterns of asynchronous computation. All benchmarks are verified by our tool in less than three seconds. The implementation and benchmarks are available at `https://github.com/boogie-org/boogie`.

The proof rules introduced in this paper are crucial to preserving the layered verification approach in CIVL and exploiting it to construct compact and highly-automated proofs with simple invariants [12]. Without our new rules, CIVL proofs of our benchmarks would amount to single-layer proofs with monolithic invariants in a style similar to classical proofs of distributed systems in modeling frameworks such as TLA+ [13].

```
action {:atomic}{:layer 1,1}
CAS_l (oldval:Tid, newval:Tid) returns (b:bool) {
  if   (l == oldval) { l := newval; b := true;  }
  else               {              b := false; }
}

procedure {:layer 1}{:refines ACQUIRE}
Acquire (tid:Tid) {
  var b:bool;
  b := false;
  while (!b) call b := CAS_l(nil, tid);
  async call Callback(tid);
}
```

```
action {:atomic}{:layer 2,3} ACQUIRE (tid:Tid) {
  assert tid != nil;
  assume l == nil;
  l := tid;
  async call Callback(tid);
}

procedure {:layer 2}{:refines CALLBACK}
Callback (tid:Tid) { /* not shown */ }

action {:left}{:layer 3} CALLBACK (tid:Tid) {
  assert tid != nil && l == tid;
  x := x + 1;
  l := nil;
}
```

Figure 6 Lock service in CIVL (excerpt).

7.1 Lock Service

In this section we illustrate how synchronization and pending async elimination are offered to a programmer in CIVL by revisiting the lock service example from Section 2.2.

Figure 6 shows a fragment of our CIVL implementation. First, let us understand the layer annotations in more detail. A procedure has a single layer number x that denotes the layer at which the procedure is shown to refine an atomic action. At all layers up to x calls to the procedure behave according to its implementation, and at layers higher than x calls to the procedure behave according to its refined atomic action. Atomic actions have an associated layer range $[x, y]$, which denotes at which layers the action is "available". For each layer, the set of available atomic actions is subject to pairwise commutativity checks. In Figure 6, the procedure `Acquire` is declared to refine the atomic action `ACQUIRE` at layer 1, which causes CIVL to apply synchronization. The implementation makes two calls, a synchronous call to a compare-and-swap operation which is already atomic at layer 1, and an asynchronous call to `Callback`. Since `Callback` is refined at the higher layer 2, the asynchronous call results in a pending async in the atomic action `ACQUIRE`. Thus, at layer 2, `ACQUIRE` is exactly the client-independent specification of `Acquire` we presented in Figure 2 (b).

Now `Callback` (whose implementation is not shown) is declared to refine `CALLBACK` at layer 2. This causes CIVL to apply pending async elimination in `ACQUIRE` at layer 3; the pending async to `Callback` is replaced with the effect of `CALLBACK`. Thus, at layer 3, `ACQUIRE` corresponds to `ACQUIRE'` in Figure 2 (e).

This example illustrates two important aspects of our proof method and its integration into CIVL. First, on the conceptual side, our method enables independent and modular reasoning about the lock service implementation and its client. The atomic action `ACQUIRE` can be (1) proved for a different implementation of the lock without the need to re-verify the client, and (2) used to reason about a different client by letting CIVL apply pending async elimination for a different client (i.e., `Callback` implementation). Second, on the practical side, the application of synchronization and pending async elimination in CIVL is driven by layer annotations. The programmer does not have to explicitly write the program under consideration at every layer of abstraction and specify the transformation that connects them. Instead, CIVL automatically constructs per-layer versions of procedures and atomic actions.

7.2 Two-phase Commit

In this section we show that our method applies to realistic programs with intricate concurrency by verifying full functional correctness of the two-phase commit (2PC) protocol. The protocol employs a *coordinator* process to consistently replicate transactions among a set of

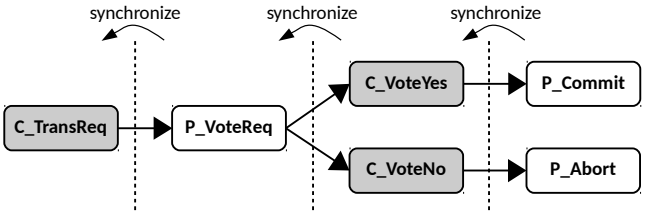


Figure 7 2PC call hierarchy (from left to right) and proof outline (right to left).

participant processes. In the first phase, the coordinator broadcasts incoming request to all participants, which respond either with a "yes" vote to commit, or a "no" vote to abort. In the second phase, the coordinator processes incoming votes as follows: (1) If *all* participants voted "yes" it broadcasts a "commit" message, or (2) as soon as *a single* participant votes "no" it broadcasts an "abort" message. Due to asynchrony and message reordering, the protocol implementation must be robust against unexpected situations. For example, a participant can receive an abort message before it receives the corresponding vote request.

Figure 7 shows the message handlers of the protocol we implemented in CIVL, together with the asynchronous communication structure. For example, `P_VoteReq` is a participants handler for vote requests, which asynchronously invokes either the coordinators `C_VoteYes` or `C_VoteNo` handler. To reason about the protocol, we use a variable `state` such that for every transaction `xid` and process `pid`, `state[xid][pid]` is one of `INIT`, `COMMIT`, or `ABORT`. We prove a top-level atomic action specification for `C_TransReq` that states that for a fresh `xid`, `state[xid]` is consistently updated, i.e., there are no two processes such that one is `COMMIT` and the other one `ABORT`. Figure 7 also shows the proof outline, making repeated use of synchronization. Here we focus on the first synchronization of `P_Commit` and `P_Abort`, which requires them to be left movers. A priori these operations do not commute, because they write the conflicting values `COMMIT` and `ABORT` to `state[xid][pid]`, respectively. However, by making it explicit that the coordinator has to decide on a transaction first, the following abstractions are commutative:

```
action P_Commit (pid,xid):
  assert state[xid][C] == COMMIT
  state[xid][pid] := COMMIT
```

```
action P_Abort (pid,xid):
  assert state[xid][C] == ABORT
  state[xid][pid] := ABORT
```

Our proof of 2PC confirms that the benefit of reduced invariant complexity in structured multi-layer refinement proofs [9] carries over to the asynchronous setting. In particular, we could state the central correctness invariant in terms of the protocol mechanism (i.e., voting and phases) after hiding low-level implementation details (i.e., counting).

7.3 Task Distribution Service

Finally, we verified a task distribution service inspired by a set of benchmarks from [1]. This example captures a whole class of similar benchmarks, where a set of *independent* tasks is processed by passing through a sequence of stages. The result of every stage is asynchronously communicated to the next stage, and different tasks can run through different stages. However, concurrent tasks do not interfere with each other. With this key difference to examples like 2PC, we can avoid the overhead of stepwise synchronization over several layers. Instead, synchronization can be applied to eliminate long (and even unbounded) chains of asynchronous calls in a single layer.

To summarize, synchronization is applicable to tightly interfering programs using program layers, and less interference leads to even simpler proofs.

8 Related Work

The idea of taming concurrency through synchrony is also at the heart of other works. Brisk [1] computes canonical sequentializations of message-passing programs by matching sends with corresponding receives. Our work differs in the programming model (dynamic thread creation vs. parametric processes with blocking receives) and the verification goal (deductive functional correctness vs. automatic deadlock-freedom). The work in [2] proposes the notion of robustness against concurrency as correctness condition for a class of event-driven programs. That is, the sequential behavior of a program is the underlying specification, and asynchronous executions are checked to conform to sequential executions. In contrast, we use synchronization to simplify the verification of safety properties.

There are several recent papers on mechanized verification of distributed systems. IronFleet [8] embeds TLA-style state-machine modeling [13] into the Dafny verifier [14] to refine high-level distributed systems specifications into low-level executable implementations. They use a fixed 3-layer design and one-shot reductions to atomic actions, while our program layers are more flexible. Ivy [18] organizes the search for an inductive invariant as a collaborative process between automatic verification attempts and user guided generalizations of counterexamples to induction in a graphical model. They use a restricted modeling and specification language that makes their verification conditions decidable. We rely on small partitioned verification conditions that can be discharged by an SMT solver [3]. PSync [4] uses a synchronous round-based model of communication for the purpose of program design and verification, shifting the complexity of efficient asynchronous execution to a runtime system. We allow explicit control over low-level details at the potential cost of increased verification effort. Verdi [21] lets the programmer provide a specification, implementation, and proof of a distributed system under an idealized network model. Then the application is automatically transformed into one that handles faults via verified system transformers. The rely-guarantee rule of [7] and the ALS types of [11] target a weaker form of asynchrony, where a single task queue atomically executes one task at a time.

Concurrent separation logic (CSL) [16] was devised for modular reasoning about multi-threaded shared-memory programs, focusing on the verification of fine-grained concurrent data structures. CSL adequately addresses the problem of reasoning about low-level concurrency related to dynamic memory allocation, but still suffers from the complications of a monolithic approach to invariant discovery for protocol-level concurrency. Recently, CSL has been applied to message-passing programs. The approach in [17] uses CSL to link implementation steps to atomic actions, and then relies on a model checker to explore the interleavings of those atomic actions. The work in [19] addresses the composition of verified protocols using ideas from separation logic. The actor services of [20] focus on compositional verification of response properties of message-passing programs.

9 Conclusion

The contribution of this paper are proof rules to simplify the reasoning about asynchronous concurrent programs. The impact of our work must be understood in the context of our two-pronged strategy for aiding interactive and automated verification of asynchronous programs. First, our proof rules enable asynchronous computation to be summarized analogous to the summarization of synchronous computation by pre- and post-conditions. This capability enables the construction of syntax-driven and structured proofs of asynchronous programs. Second, the program simplification enabled by our proof rules attacks the nemesis of complex invariants induced by a large number of interleaved executions. Instead of writing a large

and complex invariant justifying the overall correctness of the program, the programmer may now write a sequence of simpler invariants, each justifying a program simplification.

Our proof method decomposes the task of proving the correctness of a large asynchronous program into formulating and automatically discharging smaller independent proof obligations. These proof obligations show that an atomic action commutes with other atomic actions; that an atomic action summarizes the effect of a statement in a given context; and that an assertion is an inductive invariant for a simpler program, where asynchronous procedure calls are replaced by synchronous (immediate) atomic actions. Using our method, the automatable part of a concurrent verification problem – i.e., the safety proof given an inductive invariant – remains automatable, and the creative part – i.e., the discovery of an appropriate invariant – is greatly simplified by structuring it into smaller proof obligations, each of which can still be discharged automatically.

References

1 Alexander Bakst, Klaus von Gleissenthall, Rami Gökhan Kici, and Ranjit Jhala. Verifying distributed programs via canonical sequentialization. In *OOPSLA*, 2017. doi:10.1145/3133934.

2 Ahmed Bouajjani, Michael Emmi, Constantin Enea, Burcu Kulahcioglu Ozkan, and Serdar Tasiran. Verifying robustness of event-driven asynchronous programs against concurrency. In *ESOP*, 2017. doi:10.1007/978-3-662-54434-1_7.

3 Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, 2008. doi:10.1007/978-3-540-78800-3_24.

4 Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. PSync: a partially synchronous language for fault-tolerant distributed algorithms. In *POPL*, 2016. doi:10.1145/2837614.2837650.

5 Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. A calculus of atomic actions. In *POPL*, 2009. doi:10.1145/1480881.1480885.

6 Cormac Flanagan and Shaz Qadeer. A type and effect system for atomicity. In *PLDI*, 2003. doi:10.1145/781131.781169.

7 Ivan Gavran, Filip Niksic, Aditya Kanade, Rupak Majumdar, and Viktor Vafeiadis. Rely/guarantee reasoning for asynchronous programs. In *CONCUR*, 2015. doi:10.4230/LIPIcs.CONCUR.2015.483.

8 Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. IronFleet: proving practical distributed systems correct. In *SOSP*, 2015. doi:10.1145/2815400.2815428.

9 Chris Hawblitzel, Erez Petrank, Shaz Qadeer, and Serdar Tasiran. Automated and modular refinement reasoning for concurrent programs. In *CAV*, 2015. doi:10.1007/978-3-319-21668-3_26.

10 Chris Hawblitzel, Erez Petrank, Shaz Qadeer, and Serdar Tasiran. Automated and modular refinement reasoning for concurrent programs. Technical Report MSR-TR-2015-8, Microsoft Research, February 2015. URL: https://www.microsoft.com/en-us/research/publication/automated-and-modular-refinement-reasoning-for-concurrent-programs/.

11 Johannes Kloos, Rupak Majumdar, and Viktor Vafeiadis. Asynchronous liquid separation types. In *ECOOP*, 2015. doi:10.4230/LIPIcs.ECOOP.2015.396.

12 Bernhard Kragl and Shaz Qadeer. Layered concurrent programs. In *CAV*, 2018. doi:10.1007/978-3-319-96145-3_5.

13 Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

14 K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR*, 2010. doi:10.1007/978-3-642-17511-4_20.

15 Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975. doi:10.1145/361227.361234.

16 Peter W. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007. doi:10.1016/j.tcs.2006.12.035.

17 Wytse Oortwijn, Stefan Blom, and Marieke Huisman. Future-based static analysis of message passing programs. In *PLACES*, 2016. doi:10.4204/EPTCS.211.7.

18 Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *PLDI*, 2016. doi:10.1145/2908080.2908118.

19 Ilya Sergey, James R. Wilcox, and Zachary Tatlock. Programming and proving with distributed protocols. In *POPL*, 2018. doi:10.1145/3158116.

20 Alexander J. Summers and Peter Müller. Actor services - modular verification of message passing programs. In *ESOP*, 2016. doi:10.1007/978-3-662-49498-1_27.

21 James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. Verdi: a framework for implementing and formally verifying distributed systems. In *PLDI*, 2015. doi:10.1145/2737924.2737958.

A Semantics for Hybrid Iteration

Sergey Goncharov[1]
Lehrstuhl für Theoretische Informatik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
sergey.goncharov@fau.de

Julian Jakob
Lehrstuhl für Theoretische Informatik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
julian.jakob@fau.de

Renato Neves[2]
INESC TEC (HASLab) & University of Minho, Portugal
nevrenato@di.uminho.pt

Abstract

The recently introduced notions of *guarded traced (monoidal) category* and *guarded (pre-)iterative monad* aim at unifying different instances of partial iteration whilst keeping in touch with the established theory of total iteration and preserving its merits. In this paper we use these notions and the corresponding stock of results to examine different types of iteration for hybrid computations. As a starting point we use an available notion of *hybrid monad* restricted to the category of sets, and modify it in order to obtain a suitable notion of guarded iteration with guardedness interpreted as *progressiveness* in time – we motivate this modification by our intention to capture *Zeno behaviour* in an arguably general and feasible way. We illustrate our results with a simple programming language for hybrid computations and interpret it over the developed semantic foundations.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Elgot iteration, guarded iteration, hybrid monad, Zeno behaviour

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.22

Related Version A full version of the paper is available at [13], `https://arxiv.org/abs/1807.01053`.

1 Introduction

Iteration is a basic concept of computer science that takes different forms across numerous strands, from formal languages, to process algebras and denotational semantics. From a categorical point of view, using the definite perspective of Elgot [10], iteration is an operator

$$\frac{f : X \to Y + X}{f^\dagger : X \to Y} \tag{1}$$

[1] Research supported by Deutsche Forschungsgemeinschaft (DFG) under project GO 2161/1-2.
[2] Research supported by ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia within projects POCI-01-0145-FEDER-016692 and 02/SAICT/2017.

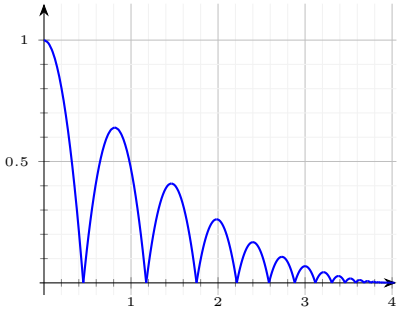


Figure 1 Bouncing ball's movement.

that runs the function f and terminates if the result is in Y, otherwise it proceeds with the result repetitively. One significant difficulty in the unification of various forms of iteration is that the latter need not be total, but can be defined only for a certain class of morphisms whose definition depends on the nature of the specific example at hand. In process algebra, for example, one typically considers recursive solutions of *guarded* process definitions, in complete metric spaces only fixpoints of *contractive* maps (which can then be found uniquely thanks to *Banach's fixpoint theorem*), and in domain theory only least fixpoints over *pointed predomains* (i.e. *domains*). These examples have recently been shown as instances of the unifying notion of *guarded traced category* [16, 15].

In this work we aim to extend the stock of examples of this notion by including iteration on *hybrid computations*, which are encoded in the recently introduced hybrid monad [26, 25]. We argue that in the hybrid context guardedness corresponds to *progressiveness* – the property of trajectories to progressively extend over time during the iteration process (possibly converging to a finite trajectory in the limit) – we illustrate and examine the corresponding iteration operator and use it to develop while-loops for hybrid denotational semantics.

Hybrid computations are inherent to systems that combine discrete and continuous, *physical behaviour* [31, 28, 1]. Traditionally qualified as *hybrid* and born in the context of control theory [32], they range from computational devices interacting with their physical, external environment to chemical/biological reactions and physical processes that are subjected to discrete changes, such as combustions and impacts. Typical examples include pacemakers, cellular division processes, cruise control systems, and electric/water grids. Let us consider, for example, the following *hybrid program*, written in an algebraic programming style, and with $(\dot{\mathtt{x}} = \mathtt{t}\ \&\ \mathtt{r})$ denoting "let variable $\mathtt{x}$ evolve according to t during r milliseconds".

$$(\dot{\mathtt{v}} = 1\ \&\ 1) +_{\mathtt{v} \leqslant 120} (\dot{\mathtt{v}} = -1\ \&\ 1)$$

It represents a (simplistic) cruise controller that either accelerates $(\dot{\mathtt{v}} = 1\ \&\ 1)$ or brakes $(\dot{\mathtt{v}} = -1\ \&\ 1)$ during one millisecond depending if the car's velocity $\mathtt{v}$ is lower or greater than 120km/h. This program naturally fits in a slightly more sophisticated scenario obtained by wrapping a non-terminating while-loop around it:

$$\mathsf{while\ true}\ \{(\dot{\mathtt{v}} = 1\ \&\ 1) +_{\mathtt{v} \leqslant 120} (\dot{\mathtt{v}} = -1\ \&\ 1)\} \tag{2}$$

Now the resulting program runs *ad infinitum*, measuring the car's velocity every millisecond and changing it as specified by the if-then-else condition. *How should we systematically interpret such while-loops?*

Iteration on hybrid computations is notoriously difficult to handle due to the so called *Zeno behaviour* [18, 2, 33], a phenomenon of unfolding an iteration loop infinitely often in finite time, akin to the scenarios famously described by the greek philosopher Zeno, further

analyzed by Aristotle [3, Physics, 231a–241b], and since then by many others. To illustrate this, consider a bouncing ball dropped at a positive height and with no initial velocity. Due to the gravitational acceleration g, it falls into the ground and bounces back up, losing a portion of its kinetic energy. In order to model this system, one can start by writing the program,

$$(\dot{\mathsf{p}} = \mathsf{v}, \dot{\mathsf{v}} = \mathsf{g} \,\&\, \mathsf{p} \leqslant 0 \wedge \mathsf{v} \leqslant 0); (\mathsf{v} := \mathsf{v} \times -0.5) \tag{3}$$

to specify the (continuous) change of height p, and also the (discrete) change of velocity v when the ball touches the ground; the expression $\mathsf{p} \leqslant 0 \wedge \mathsf{v} \leqslant 0$ provides the termination condition: the ball stops when both its height and velocity do not exceed zero. Then, abbreviating program (3) to b, one writes,

$$(\mathsf{p} := 1, \mathsf{v} := 0); \underbrace{\mathsf{b}; \ldots; \mathsf{b}}_{\text{n times}}$$

as the act of dropping the ball and letting it bounce exactly n times. One may also wish to drop the ball and let it bounce *until it stops* (see Fig. 1), using some form of infinite iteration on b and thus giving rise to Zeno behaviour. Only a few existing approaches aim to systematically work with Zeno behaviour, e.g. in [18, 19] this is done by relying on non-determinism, although the results seem to introduce undesirable behaviour in some occasions (see details in the following subsection). Here, we do regard Zeno behaviour as an important phenomenon to be covered and as such helping to design and classify notions of iteration for hybrid semantics in a systematic and compelling way.

1.1 Related Work, Contributions, Roadmap, and Notation

There exist two well-established program semantics for hybrid systems: Höfner's "*Algebraic calculi for hybrid systems*" [18] where programs are interpreted as sets of trajectories, and Platzer's Kleene algebra [28] interpreting programs as maps $X \to \mathcal{P}X$ for the powerset functor $\mathcal{P}$. Both approaches are inherently non-deterministic and the corresponding iteration operators crucially rely on non-determinism. In [28], the iteration operator is modelled by the Kleene star $(-)^*$, i.e. essentially by the non-deterministic choice between all possible *finite* iterates of a given program p; more formally, p^* is the *least fixpoint* of

$$x \mapsto \mathsf{p}; x + \mathsf{skip}$$

Semantics based on Kleene star deviates from the (arguably more natural) intuition given above for the non-terminating while-loop (2). It is also possible to extend the non-deterministic perspective summarized above to a more abstract setting via a monad that combines hybrid computations and nondeterminism [9], but in the present work we restrict ourselves to a *purely hybrid setting*, in order to study genuinely hybrid computations in isolation, without being interfered with other computational effects such as non-determinism.

One peculiarity of the Kleene star in [28] is that it is rather difficult to use for modelling programs with Zeno behaviour, the problem the authors are confronted with in [18, 19]. The authors of op. cit. extend the Kleene star setting with an *infinite iteration* operator $(-)^\omega$ that for a given program p it returns the *largest fixpoint* of the function on programs,

$$x \mapsto \mathsf{p}; x$$

As argued in [18, 19], this operator still does not adequately capture the semantics of hybrid iteration, as it yields "too much behaviour", e.g. if $\mathsf{p} = \mathsf{skip}$, p^ω is the program containing

all trajectories while we are expecting it to be skip. This is fixed by combining various techniques for obtaining a desirable set of behaviours, but unexpected behaviour could still appear at the smallest instant of time that is not reached by finite iterations [18, 19]. For the bouncing ball, this entails that at the instant in which it is supposed to stop, it can appear below ground or shoot up to the sky.

Other types of formalisms for hybrid systems were proposed in the last decades, including e.g. the definite case of hybrid automata [17], whose distinguishing feature is the ability of state variables to evolve continuously, and Hybrid CSP [8], an extension of CSP by expressions with time derivatives. More recently, an elegant specification language handling continuous behaviour of hybrid systems via *non-standard analysis* was introduced in [30].

Contributions. We propose semantic foundations for (Elgot) iteration in a hybrid setting: we identify two new monads for hybrid computations, one of which supports a partial guarded iteration operator, characterized as a least solution of the corresponding fixpoint equation, and another one extending the first and carrying a total iteration operator, although not generally being characterized in an analogous way. We show that both operators do satisfy the standard equational principles of iteration theories [5, 10] together with uniformity [29]. Moreover, we develop a language for hybrid computations with full-fledged while-loops as a prominent feature and interpret it using the underlying monad-based semantics. We discuss various use case scenarios and demonstrate various aspects of the iterative behaviour.

Plan of the paper. We proceed by defining a simple programming language for hybrid computations in Section 2, in order to present and discuss challenges related to defining a desirable semantics for it. In Section 3 we provide a summary of guarded (Elgot) iteration theory. In Sections 4 and 5 we present our main technical developments, including two new monads $\mathbf{H}_+$ and $\mathbf{H}$ for hybrid computations and the corresponding iteration operators. In Section 6 we provide a semantics for the while-loops of our programming language and then conclude in Section 7.

All omitted proofs can be found in an extended version of the paper [13] (available on `arXiv`).

Notation. We assume basic familiarity with the language of category theory [21], monads [21, 4], and topology [11]. Some conventions regarding notation are in order. By $|\mathbf{C}|$ we denote the class of objects of a category $\mathbf{C}$ and by $\mathsf{Hom}_{\mathbf{C}}(A, B)$ ($\mathsf{Hom}(A, B)$, if no confusion arises) the set of morphisms $f : A \to B$ from $A \in |\mathbf{C}|$ to $B \in |\mathbf{C}|$. We denote the set of *Kleisli endomorphisms* $\mathsf{Hom}_{\mathbf{C}}(X, TX)$ by $\mathsf{End}_{\mathbf{T}}(X)$. We agree to omit indices at natural transformations. We identify monads with the corresponding Kleisli triples, and use blackboard characters to refer to a monad and the corresponding roman letter to the monad's functorial part, e.g. $\mathbf{T} = (T, \eta, (\text{–})^\star)$ denotes a monad over a functor T with $\eta : \mathrm{Id} \to T$ being the *unit* and $(\text{–})^\star : \mathsf{Hom}(X, TY) \to \mathsf{Hom}(TX, TY)$ being the corresponding *Kleisli lifting*. Most of the time we work in the category **Set** of sets and functions. We write $\mathbb{R}_+$ and $\overline{\mathbb{R}}_+$ for the sets of non-negative reals, and non-negative reals extended with infinity ∞ respectively. Given $e : \mathbb{R}_+ \to X$ and $t \in \mathbb{R}_+$, we denote by e^t the application $e(t)$. Given $x \in X$, $\underline{x} : Y \to X$ is the function constantly equal to x. We use if-then-else constructs of the form $p \lhd b \rhd q$ returning p if b evaluates to true and q otherwise.

2 A Simple Hybrid Programming Language

Let us build a simple hybrid programming language to illustrate some of our challenges and results. Intuitively, this language adds differential equation constructs to the standard imperative features, namely assignments, sequencing, and conditional branching. It was first presented in [25, Chapter 3] and we will use this paper's results to extend it with a notion of iteration. We start by recalling the definition of the hybrid monad [26] here denoted by $\mathbf{H}_0$, as a candidate semantic domain for this language. In the following sections, we will extend $\mathbf{H}_0$ in order to obtain additional facilities for interpreting progressive and hybrid iteration.

▶ **Definition 1** ([26]). The monad $\mathbf{H}_0$ on **Set** is defined in the following manner.

- The set H_0X has as elements the pairs (d, e) with $d \in \overline{\mathbb{R}}_+$ and $e : \mathbb{R}_+ \to X$ a function satisfying the *flattening condition*: for every $x \geqslant d$, $e(x) = e(d)$. We call the elements of (d, e) *duration* and *evolution*, respectively, and use the subscripts d and e to access the corresponding fields, i.e. given $f = (d, e) \in H_0X$, we mean f_d and f_e to denote d and e respectively. This convention extends to Kleisli morphisms as follows: given $f : X \to H_0Y$, $f_\mathsf{d}(x) = (f(x))_\mathsf{d}$, $f_\mathsf{e}(x) = (f(x))_\mathsf{e}$.
- The unit is defined by $\eta(x) = (0, \underline{x})$, where $\underline{x}$ denotes the constant trajectory on x;
- For every Kleisli morphism $f : X \to H_0Y$ and every value $(d, e) \in H_0X$,

$$(f^\star(d,e))_\mathsf{d} = d + f_\mathsf{d}(e^d) \lhd d \in \mathbb{R}_+ \rhd \infty \qquad (f^\star(d,e))^t_\mathsf{e} = f^0_\mathsf{e}(e^t) \lhd t < d \rhd f^{t-d}_\mathsf{e}(e^d)$$

(recall that, according to our conventions, $(f_\mathsf{e})^0$ refers to $f_\mathsf{e}(0)$; here we additionally simplify $(f_\mathsf{e})^0$ to f^0_e for the sake of readability).

We now fix a finite set of real-valued variables $X = \{x_1, \ldots, x_n\}$ and denote by $\mathsf{At}(X)$ the set of atomic programs given by the grammar,

$$\varphi \ni (x_1 := t, \ldots, x_n := t) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, r) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, \psi),$$
$$t \ni r \mid r \cdot x \mid t + t, \qquad \psi \ni t \leqslant t \mid t \geqslant t \mid \psi \wedge \psi \mid \psi \vee \psi$$

where $x \in X$ and $r \in \mathbb{R}_+$. The next step is to construct an interpretation map,

$$[\![-]\!] : \mathsf{At}(X) \to \mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n) \tag{4}$$

that sends atomic programs a to endomorphisms $[\![\mathsf{a}]\!] : \mathbb{R}^n \to H_0(\mathbb{R}^n)$ in the Kleisli category of $\mathbf{H}_0$. This map extends to terms and predicates as $[\![t]\!](v_1, \ldots, v_n) \in \mathbb{R}^n$ and $[\![\psi]\!] \subseteq \mathbb{R}^n$ in the standard way by structural induction. We interpret each assignment $(x_1 := t, \ldots, x_n := t)$ as the map,

$$(v_1, \ldots, v_n) \mapsto \eta_{\mathbb{R}^n}([\![t_1]\!](v_1, \ldots, v_n), \ldots, [\![t_n]\!](v_1, \ldots, v_n))$$

Recall that linear systems of ordinary differential equations $\dot{x}_1 = t, \ldots, \dot{x}_n = t$ always have unique solutions $\phi : \mathbb{R}^n \to (\mathbb{R}^n)^{\mathbb{R}_+}$ [27]. We use this property to interpret each program $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, r)$ as the respective solution $\mathbb{R}^n \to (\mathbb{R}^n)^{\mathbb{R}_+}$ but restricted to $\mathbb{R}^n \to (\mathbb{R}^n)^{[0,r]}$. In order to interpret programs of the type $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, \psi)$ we can call on the following result.

▶ **Theorem 2** ([9]). *Consider a program $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, \psi)$, the solution $\phi : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ of the system $\dot{x}_1 = t, \ldots, \dot{x}_n = t$, and a valuation $(v_1, \ldots, v_n) \in \mathbb{R}^n$. If there exists a time instant $r \in \mathbb{R}_+$ such that $\phi(v_1, \ldots, v_n, r) \in [\![\psi]\!]$ then there exists a* smallest *time instant that also satisfies this condition.*

Using this theorem, we interpret each program $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, \psi)$ as the function defined by,

$$(v_1, \ldots, v_n) \mapsto (d, \phi(v_1, \ldots, v_n, -))$$

where d is the smallest time instant that intersects $[\![\psi]\!]$ if $(\mathsf{Img}(\phi(v_1, \ldots, v_n, -))) \cap [\![\psi]\!] \neq \emptyset$ and ∞ otherwise. This final step provides the desired interpretation map of atomic programs (4). We can now systematically build the hybrid programming language using standard algebraic results, as observed in [9, 25]. The set $\mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n)$ of endomorphisms $\mathbb{R}^n \to H_0(\mathbb{R}^n)$ together with Kleisli composition $\bullet$ and the unit $\eta : \mathrm{Id} \to H_0$ form a monoid $(\mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n), \bullet, \eta)$. Therefore, the free monoidal extension of $[\![-]\!] : \mathsf{At}(X) \to (\mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n), \bullet, \eta)$ is well-defined and induces a semantics for program terms,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At(X)} \mid \mathsf{skip} \mid \mathsf{p}\,;\mathsf{p}$$

▶ **Example 3.** Let us consider some programs written in this language.

1. We can have classic, discrete assignments, such as $\mathsf{x} := \mathsf{x} + 1$ or $\mathsf{x} := 2 \cdot \mathsf{x}$, and their sequential composition.
2. We can also write a $\mathsf{wait(r)}$ call, frequently used in the context of embedded systems for making the system halt its execution during r time units. This is achieved with the program $(\dot{x}_1 = 0, \ldots, \dot{x}_n = 0 \,\&\, r)$.
3. It is also possible to consider oscillators using histeresis [12], in particular via the sequential composition $(\dot{\mathsf{x}} = 1 \,\&\, 1)\,;(\dot{\mathsf{x}} = -1 \,\&\, 1)$.
4. The bouncing ball system that was examined in the introduction is another program of this language.

We next extend our language with if-then-else clauses. This can be achieved in the following manner. Denote by B the free Boolean algebra generated by the expressions $t = t$ and $t < t$. Each $b \in B$ induces an obvious predicate map $[\![b]\!] : \mathbb{R}^n \to 2$.

Any b induces a binary function $+_b : \mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n) \times \mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n) \to \mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n)$ defined as follows: $(f +_b g)(x) = f(x) \lhd b(x) \rhd g(x)$. This allows us to freely extend the interpretation map,

$$[\![-]\!] : \mathsf{At}(X) \to (\mathsf{End}_{\mathbf{H}_0}(\mathbb{R}^n), \bullet, \eta, (+)_{b \in B})$$

into a hybrid programming language with if-then-else clauses $\mathsf{p} +_{\mathsf{b} \in \mathsf{B}} \mathsf{p}$.

▶ **Example 4.** Let us consider some programs of this language with control decision features.

1. Aside from while-loops, our language carries the basic features of classic programs with discrete assignments, sequential composition, and if-then-else constructs.
2. The (simplistic) cruise controller, $(\dot{\mathsf{v}} = 1 \,\&\, 1) +_{\mathsf{v} \leqslant 120} (\dot{\mathsf{v}} = -1 \,\&\, 1)$ discussed in the introduction is also a program of this language.

To be able to address more complex behaviours we need some means for forming iterative computations, such as *while-loops*

$$\mathsf{while\ b\ \{p\}} \tag{5}$$

This poses the main challenge of our present work, which is to give a semantics of such constructs w.r.t. to a suitably designed *hybrid monad*. As a starting point, we refer to [26, 25] where $\mathbf{H}_0$ and an iteration operator $(-)^\# : \mathsf{Hom}(X, H_0X) \to \mathsf{Hom}(X, H_0X)$, which we call *basic iteration*, were introduced. One limitation of this approach can already be read from the type profile: $(-)^\#$ can only interpret non-terminating loops of the form $\mathsf{while\ true\ \{p\}}$. The semantics of $(-)^\#$ in $\mathbf{H}_0$ is given by virtue of metric spaces and Cauchy sequences, making

$$\textbf{(trv)}\quad \frac{f : X \to TY}{(T\,\mathsf{in}_1)\,f : X \to_{\mathsf{in}_2} T(Y+Z)} \qquad \textbf{(sum)}\quad \frac{f : X \to_\sigma TZ \qquad g : Y \to_\sigma TZ}{[f,g] : X+Y \to_\sigma TZ}$$

$$\textbf{(cmp)}\quad \frac{f : X \to_{\mathsf{in}_2} T(Y+Z) \qquad g : Y \to_\sigma TV \qquad h : Z \to TV}{[g,h]^\star\, f : X \to_\sigma TV}$$

Figure 2 Axioms of abstract guardedness.

difficult to identify the corresponding domain of definiteness. Here we take a different avenue of introducing an Elgot iteration (1), for which, as we shall see, the monad $\mathbf{H}_0$ must be modified. We then show (in Section 5) that basic iteration can be recovered, albeit with a semantics subtly different from the one via $\mathbf{H}_0$.

3 Guarded Monads and Elgot Iteration

We proceed to give the necessary definitions related to guardedness for monads [16]. A monad $\mathbf{T}$ (on **Set**) is *(abstractly) guarded* if it is equipped with a notion of guardedness, which is a relation between Kleisli morphisms $f : X \to TY$ and injections $\sigma : Y' \hookrightarrow Y$ closed under the rules in Fig. 2 where $f : X \to_\sigma Y$ denotes the fact that f and σ are in the relation in question. In the sequel, we also write $f : X \to_i TY$ for $f : X \to_{\mathsf{in}_i} TY$. More generally, we use the notation $f : X \to_{p,q,\dots} TY$ to indicate guardedness in the union of injections $\mathsf{in}_p, \mathsf{in}_q, \dots$ where $p, q, \dots$ are sequences over $\{1, 2\}$ identifying the corresponding coproduct summand in Y. For example, we write $f : X \to_{12,2} T((Y+Z)+Z)$ to mean that f is $[\mathsf{in}_1\ \mathsf{in}_2, \mathsf{in}_2]$-guarded.

▶ **Definition 5** (Guarded Elgot monads)**.** A monad $\mathbf{T}$ is a *guarded Elgot monad* if it is equipped with a *guarded iteration operator*,

$$(f : X \to_2 T(Y+X)) \mapsto (f^\dagger : X \to TY)$$

satisfying the following laws:

- *fixpoint law*: $f^\dagger = [\eta, f^\dagger]^\star\, f$;
- *naturality*: $g^\star f^\dagger = ([(T\,\mathsf{inl})\,g, \eta\,\mathsf{inr}]^\star\, f)^\dagger$ for $f : X \to_2 T(Y+X)$, $g : Y \to TZ$;
- *codiagonal*: $(T[\mathsf{id}, \mathsf{inr}]\,f)^\dagger = f^{\dagger\dagger}$ for $f : X \to_{12,2} T((Y+X)+X)$;
- *uniformity*: $f\,h = T(\mathsf{id}+h)\,g$ implies $f^\dagger\,h = g^\dagger$ for $f : X \to_2 T(Y+X)$, $g : Z \to_2 T(Y+Z)$ and $h : Z \to X$.

We drop the adjective "guarded" for guarded Elgot monads for which guardedness is total, i.e. $f : X \to_\sigma TY$ for any $f : X \to TY$ and σ.

The notion of guarded monad is a common generalization of various cases occurring in practice. Every monad can be equipped with a least notion of guardedness, called *vacuous guardedness* and defined as follows: $f : X \to_2 T(Y+Z)$ iff f factors through $T\,\mathsf{inl} : TY \to T(Y+Z)$. Every vacuously guarded monad is guarded iterative, for every fixpoint $f^\dagger$ unfolds precisely once. On the other hand, the greatest notion of guardedness is *total guardedness* and is defined as follows: $f : X \to_2 T(Y+Z)$ for every $f : X \to T(Y+Z)$. This addresses *total iteration* operators on $\mathbf{T}$ (e.g. for $\mathbf{T}$ being Elgot), whose existence depends on special properties of $\mathbf{T}$, such as being enriched over complete partial orders. Motivating examples, however, are those properly between these two extreme situations, e.g. *completely iterative monads* [22] for which the notion of guardedness is defined via monad modules and the iteration operator is partial, but uniquely satisfies the fixpoint law.

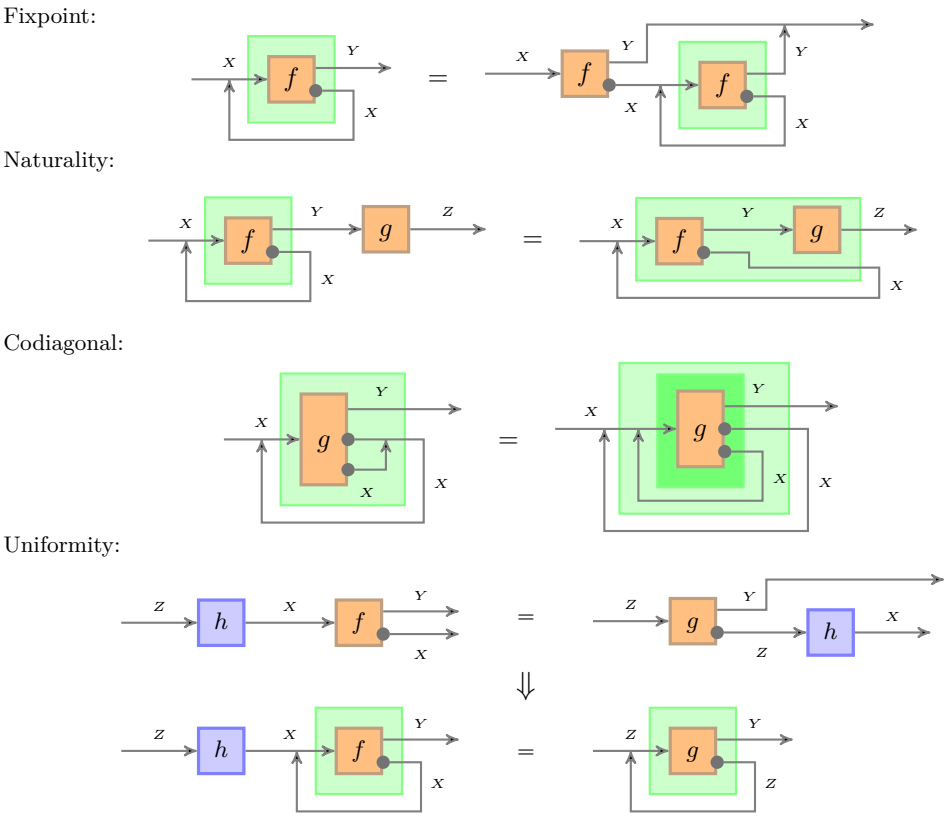


Figure 3 Axioms of guarded iteration.

▶ **Example 6.** We illustrate the above concepts with the following simplistic examples.

1. The powerset monad $\mathcal{P}$ is Elgot, with the iteration operator sending $f : X \to \mathcal{P}(Y + X)$ to $f^\dagger : X \to \mathcal{P}Y$ calculated as the least solution of the fixpoint law $f^\dagger = [\eta, f^\dagger]^\star f$.
2. An example of partial guarded iteration can be obtained from the previous clause by replacing $\mathcal{P}$ with the *non-empty powerset monad* $\mathcal{P}^+$. The total iteration operator from the previous clause does not restrict to a total iteration operator on this monad, because empty sets can arise from solving systems not involving empty sets, e.g. $\eta\ \mathsf{inr} : 1 \to \mathcal{P}^+(1 + 1)$ would not have a solution in this sense. However, it is easy to see that the iteration operator from the previous clause restricts to a guarded one for $\mathcal{P}$ with the notion of guardedness defined as follows: $f : X \to_2 \mathcal{P}^+(Y + X)$ iff for every element $x \in X$, $f(x)$ contains at least one element from Y.

The axioms of guarded Elgot monads are given in Fig. 3 in an intuitive pictorial form. The shaded boxes indicate the scopes of the corresponding iteration loops and bullets attached to output wires express the corresponding guardedness predicates. As shown in [16], other standard principles such as *dinaturality* and the *Bekić law* follow from this axiomatization.

4 A Fistful of Hybrid Monads

According to Moggi [23], Kleisli morphisms can be viewed as generalized functions carrying a computational effect, e.g. non-determinism, process algebra actions, or their combination. In this context, hybrid computations can be seen as computations extended in time.

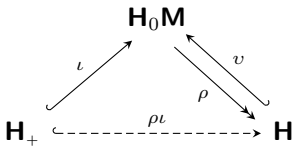


Figure 4 Connecting $\mathbf{H}_0\mathbf{M}$, $\mathbf{H}_+$ and $\mathbf{H}$.

By definition, the pairs $(d, e) \in H_0X$ fall into two classes: *closed trajectories* with $d \neq \infty$ and *open trajectories* with $d = \infty$. Due to the flattening condition (see Definition 1), closed trajectories are completely characterized by their restrictions to $[0, d]$. We proceed by extending $\mathbf{H}_0$ to a larger monad that brings open trajectories over arbitrary intervals $[0, d)$ with $d > 0$ into play, and call the resulting monad $\mathbf{H}_+$. It is instrumental in our study to cope with open trajectories, as in the presence of Zeno behaviour iteration might produce open trajectories $[0, d) \to X$ which we cannot sensibly extend into $[0, d] \to X$ without assuming some structure on X [18, 19, 24]. Furthermore, we introduce a variant of $\mathbf{H}_+$, which we call $\mathbf{H}$ and that extends the facilities of $\mathbf{H}_+$ even further by including the *empty trajectory* $[0, 0) \to X$ which will be used to accommodate divergent computations (see Remark 11). The notation for $\mathbf{H}$ and $\mathbf{H}_+$ is selected to be suggestive, and is a reminiscent of $\mathcal{P}$ and $\mathcal{P}_+$ for the powerset and the non-empty powerset monads as in Example 6. Indeed, the analogy goes further, as in the next section we show that $\mathbf{H}_+$ supports guarded (progressive) iteration, $\mathbf{H}$ supports total iteration, and the former is a restriction of the latter.

In order to develop $\mathbf{H}_+$, we first introduce a partial version of $\mathbf{H}_0$ that will greatly facilitate obtaining some of our results. Essentially, this partial version amounts to the combination of $\mathbf{H}_0$ with the *maybe monad* $\mathbf{M}$. Recall that $MX = X + 1$, that the unit of $\mathbf{M}$ is given by the left coproduct injection $\mathsf{inl} : X \to X + 1$, and that the Kleisli lifting sends $f : X \to Y + 1$ to $[f, \mathsf{inr}] : X + 1 \to Y + 1$. We conventionally see Kleisli morphisms $X \to MY$ as partial functions from X to Y and thus write $f(x)\downarrow$ to indicate that $f(x)$ is defined on x, i.e. $f(x) \neq \mathsf{inr}\star$. Let $\mathsf{dom}(f) = \{x \in X \mid f(x)\downarrow\} \subseteq X$ and denote by $\bot$ both $\mathsf{inr}\star \in X + 1$ and the totally undefined function $x \mapsto \bot$. Finally, write $f(x)\uparrow$ as a shorthand notation to $f(x) = \bot$. We will also need the following result.

▶ **Proposition 1.** *Every monad* $\mathbf{T} = (T, \eta, (\text{–})^\star)$ *induces a monad* $\mathbf{TM}$ *whose functor is defined by* $X \mapsto TMX$*, the unit by* $\eta\,\mathsf{inl} : X \to TMY$*, and the Kleisli lifting by* $[f, \eta\,\mathsf{inr}]^\star : TMX \to TMY$ *for every* $f : X \to TMY$*.*

Proof. This is a consequence of the standard fact that every monad distributes over the maybe monad [20]. ◀

▶ **Definition 7.** Let $\mathbf{H}_0\mathbf{M}$ be the monad identified in Proposition 1 with $\mathbf{T} = \mathbf{H}_0$. Then let H_+ be the subfunctor of H_0M that is defined by,

$$(d, e) \in H_+X \quad \text{iff} \quad e \neq \bot \quad \text{and} \quad e^t\downarrow \quad \text{for all} \quad t \in [0, d). \tag{6}$$

This yields a monad $\mathbf{H}_+$, by restricting the monad structure of $\mathbf{H}_0\mathbf{M}$. Explicitly, $\eta(x) = (0, \underline{x})$ and for every $f : X \to H_+Y$ and every $(d, e) \in H_+X$,

$$(f^\star(d, e))_{\mathsf{d}} = d \qquad (f^\star(d, e))^t_{\mathsf{e}} = f^0_{\mathsf{e}}(e^t) \lhd t < d \rhd \bot \qquad (\text{if } e^d\uparrow)$$
$$(f^\star(d, e))_{\mathsf{d}} = d + f_{\mathsf{d}}(e^d) \qquad (f^\star(d, e))^t_{\mathsf{e}} = f^0_{\mathsf{e}}(e^t) \lhd t < d \rhd f^{t-d}_{\mathsf{e}}(e^d) \qquad (\text{if } e^d\downarrow)$$

Note that the set H_+X consists precisely of elements (d, e) for which either $\mathsf{dom}(e) = [0, d]$ or $\mathsf{dom}(e) = [0, d)$ and $d > 0$. Of course, we need to verify that Definition 7 correctly introduces a monad.

Proof. It is easy to see that the monad structure of $\mathbf{H}_0\mathbf{M}$ restricts as above. So we only need to show that for every $f : X \to H_+Y$, $(d, e) \in H_+X$ implies that $f^\star(d, e) \in H_+Y$. Let $t \in [0, (f^\star(d, e))_\mathsf{d})$ and proceed by case distinction:

- $e^d \uparrow$. Then $(f^\star(d, e))_\mathsf{d} = d$, hence $t < d$, and $(f^\star(d, e))^t_\mathsf{e} \downarrow$ iff $f^0_\mathsf{e}(e^t) \downarrow$, which is true unless $f_\mathsf{d}(e^t) = 0$ and $f_\mathsf{e}(e^t)$ is the totally undefined function, however, the latter would contradict Definition 7.
- $e^d \downarrow$. Then $(f^\star(d, e))^t_\mathsf{e} \downarrow$ iff either $t \leqslant d$ and $f^0_\mathsf{e}(e^t) \downarrow$ or $t > d$ and $f^{t-d}_\mathsf{e}(e^d) \downarrow$. In the former case we are done in the same way as in the previous clause. In the latter case, note that $t - d < (f^\star(d, e))_\mathsf{d} - d = f_\mathsf{d}(e^d)$, which by assumption implies that $f^{t-d}_\mathsf{e}(e^d) \downarrow$. ◀

The condition $e \neq \bot$ in (6) is essential for the construction above, for otherwise we cannot ensure that computations with totally undefined trajectories are compatible with Kleisli liftings, as detailed in Remark 9 below. Such computations can be thought of as representing unproductive divergence, and are required for the semantics of programs like

$$\mathsf{while\ true}\ \{\mathtt{x} := \mathtt{x} + 1\}$$

We therefore need to extend $\mathbf{H}_+$ to a larger monad $\mathbf{H}$ in which such divergent computations exist. Technically, this will amount to quotienting the monad $\mathbf{H}_0\mathbf{M}$ in a suitable manner.

▶ **Definition 8.** Let $\mathbf{H}_0\mathbf{M}$ be the monad identified in Proposition 1 with $\mathbf{T} = \mathbf{H}_0$ and let H be the subfunctor of H_0M formed as follows:

$$(d, e) \in HX \quad \text{iff} \qquad e^t \downarrow \quad \text{for all } t \in [0, d). \tag{7}$$

Let υ be the inclusion of H into H_0M and let $\rho : H_0M \to H$ be the natural transformation whose components are defined by,

$$(\rho_X(d, e))_\mathsf{d} = d_\star, \qquad (\rho_X(d, e))^t_\mathsf{e} = e^t \lhd t < d_\star \rhd (e^d \lhd d_\star = d \rhd \bot)$$

where $d_\star = \sup\{t < d \mid [0, t) \subseteq \mathsf{dom}(e)\}$.

We extend $\mathbf{H}$ to a monad by defining $x \mapsto \rho(\eta(x))$ to be the unit and the Kleisli lifting the one that sends $f : X \to HY$ to $\rho(\upsilon f)^\star \upsilon$. Explicitly, the monad structure of $\mathbf{H}$ is as follows: $\eta(x) = (0, \underline{x})$ and for every $f : X \to HY$, and every $(d, e) \in HX$,

$$\begin{aligned} &(f^\star(d, e))_\mathsf{d} = \sup\{t < d \mid f^0_\mathsf{e}(e^t) \downarrow\} \\ &(f^\star(d, e))^t_\mathsf{e} = f^0_\mathsf{e}(e^t) \lhd t < \sup\{t < d \mid f^0_\mathsf{e}(e^t) \downarrow\} \rhd \bot \end{aligned} \qquad (\text{if } \exists t < d.\, f^0_\mathsf{e}(e^t) \uparrow \text{ or } e^d \uparrow)$$

$$\begin{aligned} &(f^\star(d, e))_\mathsf{d} = d + f_\mathsf{d}(e^d) \\ &(f^\star(d, e))^t_\mathsf{e} = f^0_\mathsf{e}(e^t) \lhd t < d \rhd f^{t-d}_\mathsf{e}(e^d) \end{aligned} \qquad (\text{if } \forall t < d.\, f^0_\mathsf{e}(e^t) \downarrow \text{ and } e^d \downarrow)$$

Like in the case of $\mathbf{H}_+$, we need to verify that $\mathbf{H}$ is a monad; this is shown in the paper's extended version [13].

▶ Remark 9. As indicated above, $\mathbf{H}$ is a quotient of $\mathbf{H}_0\mathbf{M}$ and not a submonad, specifically υ is not a monad morphism. Indeed, given $(0, \bot) : \mathbb{R}_+ \to H\mathbb{R}_+$ and $(1, \mathsf{id}) \in H\mathbb{R}_+$, by definition, $(0, \bot)^\star(1, \mathsf{id}) = (0, \bot)$, but in $\mathbf{H}_0\mathbf{M}$, $(0, \bot)^\star(1, \mathsf{id}) = (1, \bot)$.

In summary, the monads $\mathbf{H}_0\mathbf{M}$, $\mathbf{H}_+$, $\mathbf{H}$ are connected as depicted in Fig. 4. Here, ι and ρ are monad morphisms, and the induced composite morphism $\rho\iota : \mathbf{H}_+ \to \mathbf{H}$ is pointwise injective, which is a straightforward consequence of the fact that condition (6) entails condition (7).

5 Progressive Iteration and Hybrid Iteration

We start off by equipping the monad $\mathbf{H}_+$ from the previous section with a suitable notion of guardedness.

▶ **Definition 10** (Progressiveness). A Kleisli morphism $(d, e) : X \to H_+(Y + Z)$ is *progressive* in Z (in Y) if $e^0 : X \to Y + Z$ factors through inl (respectively, inr).

Given $(d, e) : X \to H_+(Y + X)$, progressiveness in X means precisely that $e^0 = \mathsf{inl}\, u : X \to Y + X$ for a suitable $u : X \to Y$, which is intuitively the candidate for $(d, e)^\dagger_{\mathsf{e}}$ at 0. In other words, progressiveness rules out the situations in which the iteration operator needs to handle compositions of zero-length trajectories.

▶ Remark 11. A simple example of a morphism $(d, e) : X \to H_+(Y + X)$ not progressive in X is obtained by taking $X = \{0, 1\}$, $Y = \emptyset$, $d = \underline{0}$ and $e^0 = \mathsf{inr}\,\mathsf{swap}$ where swap interchanges the elements of $\{0, 1\}$. In attempts of defining $(d, e)^\dagger$ we would witness oscillation between 0 and 1 happening at time 0, i.e. not progressing over time, which is precisely the reason why there is no candidate semantic for $(d, e)^\dagger$ in this case.

▶ **Lemma 12.** $\mathbf{H}_+$ *is a guarded monad with* $f : X \to_2 H_+(Y + Z)$ *iff* f *is progressive in* Z*.*

Instead of directly equipping $\mathbf{H}_+$ with progressive iteration, we take the following route: we enrich the monad $\mathbf{H}_0\mathbf{M}$ over complete partial orders and devise a total iteration operator for it using the standard least-fixpoint argument. Then we restrict iteration from $\mathbf{H}_0\mathbf{M}$ to $\mathbf{H}_+$ via ι and to $\mathbf{H}$ via υ (see Fig. 4). The latter part is tricky, because υ is not a monad morphism (Remark 9), and thus we will call on the machinery of *iteration-congruent retractions*, developed in [16], to derive a (total) Elgot iteration on $\mathbf{H}$.

Consider the following order on H_0MX. Given $(d, e), (d_\star, e_\star) \in H_0MX$, let $(d, e) \sqsubseteq (d_\star, e_\star)$ if $d \leqslant d_\star$ and e is smaller or equal $e_\star$ as partial maps, i.e. $\mathsf{dom}(e) \subseteq \mathsf{dom}(e_\star)$ and $e^t = e^t_\star$ for all $t \in \mathsf{dom}(e)$. This order extends to the hom-sets $\mathsf{Hom}(X, H_0MY)$ pointwise.

▶ **Theorem 13.** *The following properties hold.*

1. *Every set* H_0MX *is an* ω*-complete partial order under* $\sqsubseteq$ *with* $(0, \bot)$ *as the bottom element;*
2. *Kleisli composition is monotone and continuous w.r.t.* $\sqsubseteq$ *on both sides;*
3. *Kleisli composition is right-strict, i.e. for every* $f : X \to H_0MY$*,* $f^\star(0, \bot) = (0, \bot)$*.*

Note that Kleisli composition is not left strict (c.f. Remark 9). Using the previous result and [14, Theorem 5.8], we immediately obtain

▶ **Corollary 14.** $\mathbf{H}_0\mathbf{M}$ *possesses a total iteration operator* $(\text{–})^\ddagger$ *obtained as a least solution of equation* $f^\ddagger = [\eta, f^\ddagger]^\star f$*. This makes* H_0MX *into an Elgot monad.*

We readily obtain a *progressive iteration* on $\mathbf{H}_+$ by restriction via ι (see Fig. 4).

▶ **Corollary 15.** $\mathbf{H}_+$ *possesses a guarded iteration operator* $(\text{–})^\upharpoonright$ *obtained by restriction from* $\mathbf{H}_0\mathbf{M}$ *with guardedness being progressiveness and* $f^\upharpoonright$ *being the least solution of equation* $f^\upharpoonright = [\eta, f^\upharpoonright]^\star f$*.*

Explicitly, $f^\upharpoonright$ is calculated via the *Kleene fixpoint theorem* as follows. Consider $f : X \to_2 H_+(Y + X)$, let $f^{\langle 0 \rangle} = (0, \bot)$ and $f^{\langle i+1 \rangle} = [\eta, f^{\langle i \rangle}]^\star f$. This yields an ω-chain

$$(0, \bot) \sqsubseteq f^{\langle 1 \rangle} \sqsubseteq f^{\langle 2 \rangle} \sqsubseteq \cdots$$

and $f^\upharpoonright = \bigsqcup_i f^{\langle i \rangle}$. We proceed to obtain an iteration operator for $\mathbf{H}$. Remarkably, we cannot use the same technique of restricting the iteration operator from $\mathbf{H}_0\mathbf{M}$ to $\mathbf{H}$, even though H embeds into H_0M: the following example illustrates the problem.

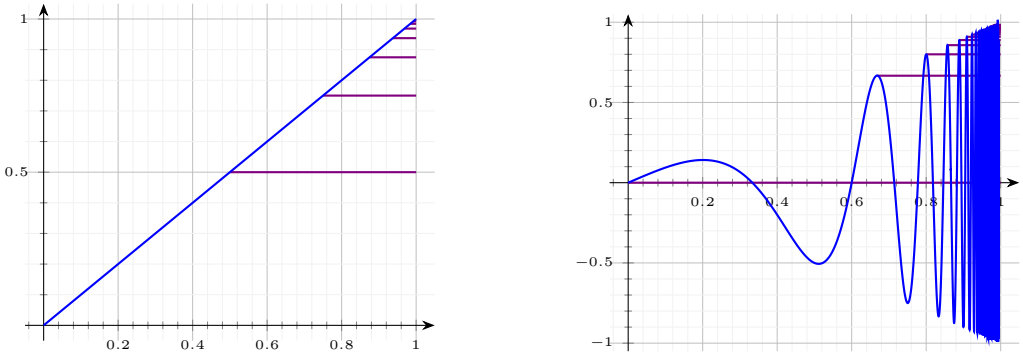


Figure 5 Examples of (un-)definiteness of basic iteration.

▶ **Example 16.** Let $f = (\underline{1}, e) : \mathbb{R}_+ \to H_0M(\mathbb{R}_+ + \mathbb{R}_+)$ with $e(x) = \underline{\mathsf{inr}\,1}$ for every $x \in \mathbb{R}_+$. The result of calculating $f^\dagger$ is the trajectory $(\infty, \bot)$, which is not present in $\mathbf{H}$.

Note that every HX inherits the ω-complete partial order from H_0MX.

▶ **Theorem 17.** *Let $\rho : \mathbf{H_0M} \to \mathbf{H}$ and $\upsilon : \mathbf{H} \to \mathbf{H_0M}$ be the pair of natural transformations from Definition 8. Then for every $f : X \to H_0M(Y + X)$, $\rho f^\ddagger = \rho(\upsilon\rho f)^\ddagger$.*

In the terminology of [16], Theorem 17 states that the pair (ρ, υ) is an *iteration-congruent retraction*. Therefore, per [16, Theorem 21], $\mathbf{H}$ inherits a total Elgot iteration from $\mathbf{H_0M}$.

▶ **Corollary 18.** $\mathbf{H}$ *is an Elgot monad with the iteration operator $(\text{–})^\dagger$ defined as follows: for every $f : X \to H(Y + X)$, $f^\dagger = \rho(\upsilon f)^\ddagger$ assuming that $(\text{–})^\ddagger$ is the iteration operator on $\mathbf{H_0M}$.*

▶ **Corollary 19.** *The progressive iteration operator $(\text{–})^\flat$ of $\mathbf{H}_+$ is the restriction of the total iteration operator $(\text{–})^\dagger$ of $\mathbf{H}$ along $\rho\iota : H_+ \to H$ (as in Fig. 4), i.e. for every $f : X \to_2 H_+(Y + X)$, $\rho\,\iota f^\flat = (\rho\,\iota f)^\dagger$.*

Proof. Let $(\text{–})^\ddagger$ be the iteration operator of $\mathbf{H_0M}$. Then, by definition,

$$(\rho\,\iota f)^\dagger = \rho(\upsilon\rho\,\iota f)^\ddagger = \rho(\iota f)^\ddagger = \rho\,\iota f^\flat.$$ ◀

Using the fact that the iteration operator for $\mathbf{H}$ satisfies the codiagonal law (see Definition 5), we factor the former through progressive iteration as follows.

▶ **Theorem 20** (Decomposition Theorem)**.** *Given $f : X \to H(Y + X)$, let $\hat{f} : X \to_{12} H((Y + X) + X)$ be defined as follows:*

$$\hat{f}_\mathsf{d}(x) = f_\mathsf{d}(x) \qquad \hat{f}^0_\mathsf{e}(x) = (\mathsf{id} + \mathsf{inl})(f^0_\mathsf{e}(x)) \quad \hat{f}^t_\mathsf{e}(x) = (\mathsf{id} + \mathsf{inr})(f^t_\mathsf{e}(x)) \qquad (x \in X, t > 0)$$

Then $f^\dagger = (\hat{f}^\dagger)^\flat$.

Proof. Note that $f = H[\mathsf{id}, \mathsf{inr}]\hat{f}$. Hence, by the codiagonal law: $f^\dagger = (H[\mathsf{id}, \mathsf{inr}]\hat{f})^\dagger = (\hat{f}^\dagger)^\dagger$, and the latter is $(f^\dagger)^\flat$ per Corollary 19, as $f^\dagger$ happens to be progressive in the second argument. ◀

Theorem 20 presents the iteration of $\mathbf{H}$ as a nested combination of progressive iteration and what can be called *singular iteration*, as it is precisely the restriction of $(\text{–})^\dagger$ responsible for iterating computations of zero duration. Finally, we recover basic iteration, discussed in Section 2, on $\mathbf{H}_+$ (and hence on $\mathbf{H}$) by turning a morphism $X \to H_+X$ into a progressive one $X \to H_+(X + X)$.

▶ **Definition 21** (Basic Iteration). We define *basic iteration* $(d, e)^\dagger : X \to H_+X$ to be

$$\big((d, \lambda x.\, \lambda t.\, \mathsf{inl}\, e^0(x) \lhd t = 0 \rhd \mathsf{inr}\, e^t(x)) : X \to_2 H_+(X + X)\big)^\dagger : X \to H_+X.$$

▶ **Example 22.** We illustrate our design decisions behind $\mathbf{H}_+$ (and $\mathbf{H}$) with the following two examples of Zeno behaviour, computing $f^\dagger = (d^\dagger, e^\dagger)$ for specific morphisms $f : X \to H_+X$.

1. Let $f = (d, e) : [0,1] \to H_+[0,1]$ be defined as follows: for every $x \in [0,1]$, $d(x) = (1-x)/2$ and $e^t(x) = x + t$ for $t \in [0, (1-x)/2]$. It is easy to see that $d^\dagger(0) = 1/2 + 1/4 + \ldots = 1$, however, by definition, $(e^\dagger(0))^1 \uparrow$. This is indeed a prototypical example of Zeno behaviour (specifically, this is precisely Zeno's *"Dichotomy" paradox* analyzed by Aristotle [3, Physics, 231a–241b]): Given a distance of total length 1 to be covered, suppose some portion $x < 1$ of it has been covered already. Then the remaining distance has the length $1 - x$. As originally argued by Zeno, in order to cover this distance, one has to pass the middle, i.e. walk the initial interval of length $(1-x)/2$ and our function f precisely captures the dynamics of this motion. The resulting evolution $e^\dagger(0)$ together with the corresponding approximations are depicted on the left of Fig. 5. In this formalization, the traveler can not reach the end of the track, but only because we designed $(\text{–})^\dagger$ to be so. We could also justifiably define $(e^\dagger(0))^1$ to be 1, for this is what $(e^\dagger(0))^t$ tends to as t tends to 1. This is indeed the case of the approach from [26, 25] developed for the original monad $\mathbf{H}_0$.
2. It is easy to obtain an example of an open trajectory produced by Zeno iteration that cannot be continuously extended to a closed one by adapting a standard example of *essentially discontinuous* function from analysis: let e.g. $u^t : [0,1) \to [0,1)$ be as follows:

$$u^t(x) = (t + x)\cos\left(\frac{\pi t}{(1-x)(1-x-t)}\right) \qquad (t \in [0, 1-x))$$
$$u^t(x) = 1 \qquad (t \in [1-x, 1))$$

The graph of $u(0)$ is depicted on the right of Fig. 5 where one can clearly see the discontinuity at $t = 1$. It is easy to verify that $(1, u) \in H[0,1]$ is obtained by applying basic iteration to $f = (d, e) : [0,1) \to H_+[0,1)$ given as follows:

$$d(x) = \frac{2(1-x)^2}{3-2x} \qquad e^t(x) = (t + x)\cos\left(\frac{\pi t}{(1-x)(1-x-t)}\right) \qquad (t \in [0, d(x)))$$
$$e^t(x) = d(x) + x \qquad (t \in [d(x), 1))$$

Even though we carried our developments in the category of sets, we designed $\mathbf{H}_+$ and $\mathbf{H}$ keeping in touch with a topological intuition. The following instructive example shows that the iteration operators developed in the previous section cannot be readily transferred to the category of topological spaces and continuous maps, for reasons of *instability*: small changes in the definition of a given system may cause drastic changes in its behaviour. In particular, even if a morphism $(d, e) : X \to H_0X$ is continuous (for the topology described in [26]) the duration component $d^\dagger : X \to [0, \infty]$ of $(d^\dagger, e^\dagger) = (d, e)^\dagger$ need not be continuous.

▶ **Example 23** (Hilbert Cube). Let $X = [0,1]^\omega$ be the *Hilbert cube*, i.e. the topological product of ω copies of $[0,1]$ and let $\mathsf{hd} : X \to [0,1]$ and $\mathsf{tl} : X \to X$ be the obvious projections realizing the isomorphism $[0,1]^\omega \cong [0,1] \times [0,1]^\omega$. Let $f = (\mathsf{hd}, e) : X \to H_+X$ with $e : X \to X^{\mathbb{R}_+}$ be defined as follows:

- $e^t(x) = x$ if $\mathsf{hd}(x) = 0$, $t \in \mathbb{R}_+$;
- $e^t(x) = \big((\mathsf{hd}(x) - t) \cdot x + t \cdot \mathsf{tl}(x)\big)/\mathsf{hd}(x)$ if $0 < \mathsf{hd}(x)$ and $t < \mathsf{hd}(x)$;
- $e^t(x) = \mathsf{tl}(x)$ if $\mathsf{hd}(x) > 0$ and $t \geqslant \mathsf{hd}(x)$.

In the second clause we use a *convex combination* of x and $\mathsf{tl}(x)$ as vectors of X seen as a vector space (indeed, even a Hilbert space) over the reals. It can now be checked that the cumulative duration $d^{\#}$ in $(d^{\#}, e^{\#}) = (d, e)^{\#}$ is not continuous. To see why, note that $d^{\#}(x)$ is the (possibly infinite) sum of the components of x from left to right up to the first zero element, and therefore each $U = (d^{\#})^{-1}([0, a))$ contains all such vectors $x \in [0, 1]^{\omega}$ for which this sum is properly smaller than a. Then recall that a basic open set of $[0, 1]^{\omega}$ must be a *finite* intersection of sets of the form $\pi_i^{-1}(V)$, $V \subseteq [0, 1]$ open, $i \in \mathbb{N}$. Therefore, if U was open the definition of the product topology on $[0, 1]$ would imply that for every vector x in U there exists a position such that by altering the components of x arbitrarily after this position, the result would still belong to U. This is obviously not true for U, because by replacing the elements of any infinite vector from $[0, 1]^{\omega}$ after any position with 1, would give a vector summing to infinity.

6 Bringing While-loops Into The Scene

In Section 2, we started building a simple hybrid programming language. We sketched a monad-based semantics for the expected programs constructs, except the while-loops. Here we extend it by taking $\mathbf{H}$, which is a supermonad of $\mathbf{H}_0$, as the underlying monad and interpret while-loops (5) via the iteration operator of $\mathbf{H}$.

Recall that b is an element of the free Boolean algebra generated by the expressions $t = t$ and $t < t$, and that there exists a predicate map $\mathsf{b} : \mathbb{R}^n \to 2$. Now for each $\mathsf{b} : \mathbb{R}^n \to 2$ and $f : \mathbb{R}^n \to H(\mathbb{R}^n)$ denote the function,

$$\left(\mathbb{R}^n \xrightarrow{\mathsf{dist}\,\langle \mathsf{id},\mathsf{b}\rangle} \mathbb{R}^n + \mathbb{R}^n \xrightarrow{[\eta\,\mathsf{inl},\,(H\,\mathsf{inr})\,f]} H(\mathbb{R}^n + \mathbb{R}^n) \xrightarrow{m} H(\mathbb{R}^n + \mathbb{R}^n)\right)^{\dagger}$$

by $w(\mathsf{b}, f)$ where $\mathsf{dist} : X \times 2 \to X + X$ is the obvious distributivity transformation, and $m(d, e) = (d, e')$ with $e'(t) = \mathsf{inl}(x) \lhd (\mathsf{inr}(x) = e(t)$ and $t < d) \rhd e(t)$. Intuitively, the function m makes the last point of the trajectory be the only one that is evaluated by the test condition of the while-loop. Then, we define $[\![\mathsf{while\ b\ \{p\}}]\!] = w(\mathsf{b}, [\![\mathsf{p}]\!])$ and this gives a hybrid programming language,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At(X)} \mid \mathsf{skip} \mid \mathsf{p}\,;\mathsf{p} \mid \mathsf{p} +_{\mathsf{b}} \mathsf{p} \mid \mathsf{while\ b\ \{p\}}$$

with while-loops.

▶ **Example 24.** Let us consider some programs written in this language.

1. We start again with a classic program, in this case $\mathsf{while\ true\ \{x := x + 1\}}$. It yields the empty trajectory $\bot$.
2. Another example of a classic program is,

   $$\mathsf{while\ x \leqslant 10\ \{x := x + 1\,; wait(1)\}}$$

   If for example the initial value is 0 the program takes eleven time units to terminate.
3. Let us consider now the program $\mathsf{while\ x \geqslant 1\ \{\ (\dot{x} = -1\ \&\ 1)\ \}}$. If the initial value is 0 the program outputs the trajectory with duration 0 and constant on 0, since it never enters in the loop. If we start e.g. with 3 as initial value then the program inside the while-loop will be executed precisely three times, continuously decreasing x over time.
4. In contrast to classic programming languages, here infinite while-loops need not be undefined. The cruise controller discussed in the introduction,

   $$\mathsf{while\ true\ \{(\dot{v} = 1\ \&\ 1) +_{v \leqslant 120}\ (\dot{v} = -1\ \&\ 1)\}}$$

   is a prime example of this.

5. Finally, the bouncing ball, $(\mathsf{p} := 1, \mathsf{v} := 0); (\mathsf{while\ true}\ \{\mathsf{b}\})$ which has Zeno behaviour, outputs a trajectory describing the ball's movement over the time interval $[0, d)$ where d is the instant of time at which the ball stops.

7 Conclusions and Further Work

We developed a semantics for hybrid iteration by bringing together two abstraction devices introduced recently: guarded Elgot iteration [16] and the hybrid monad [26, 25]. Our analysis reveals that, on the one hand, the abstract notion of guardedness can be interpreted as a suitable form of progressiveness of hybrid trajectories, and on the other hand, the original hybrid monad from [26, 25] needs to be completed for the sake of a smooth treatment of iteration, specifically, iteration producing Zeno behaviour. In our study we rely on Zeno behaviour examples as important test cases helping to design the requisite feasible abstractions. As another kind of guidance, we rely on Elgot's notion of iteration [10] and the corresponding laws of iteration theories [5]. In addition to the new hybrid monad $\mathbf{H}_+$ equipped with (partial) progressive iteration, we introduced a larger monad $\mathbf{H}$ with total hybrid iteration extending the progressive one. In showing the iteration laws we heavily relied on the previously developed machinery for unifying guarded and unguarded iteration [14, 16]. We illustrated the developed semantic foundations by introducing a simple language for hybrid iteration with while-loops interpreted over the Kleisli category of $\mathbf{H}$.

We regard our present work as a stepping stone for further developments in various directions. After formalizing hybrid computations via (guarded) Elgot monads, one obtains access to further results involving (guarded) Elgot monads, e.g. it might be interesting to explore the results of applying the *generalized coalgebraic resumption monad transformer* [14] to $\mathbf{H}$ and thus obtain in a principled way a semantic domain for hybrid processes in the style of CCS. As shown by Theorem 20, the iteration of $\mathbf{H}$ is a combination of progressive iteration and "singular iteration". An interesting question for further work is if this combination can be framed as a universal construction. We also would like to place $\mathbf{H}$ in a category more suitable than **Set**, but as Example 23 suggests, this is expected to be a very difficult problem.

Every monad on **Set** determines a corresponding *Lawvere theory*, whose presentation in terms of operations and equations is important for reasoning about the corresponding – in our case hybrid – programs. We set as a goal for further research the task of identifying the underlying Lawvere theories of hybrid monads and integrating them into generic diagrammatic reasoning in the style of Fig. 3. This should prospectively connect our work to the line of research by Bonchi, Sobociński, and Zanasi (see e.g. [6, 7]), who studied various axiomatizations of PROPs (i.e. monoidal generalizations of Lawvere theories) and their diagrammatic languages. For a proper treatment of guarded iteration (i.e. a specific instance of guarded monoidal trace in the sense of [15]), one would presumably need to develop the corresponding notions of *guarded Lawvere theory* and *guarded PROP*.

References

1 Rajeev Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.

2 Aaron Ames, Alessandro Abate, and Shankar Sastry. Sufficient conditions for the existence of zeno behavior. In *CDC-ECC'05: Decision and Control and European Control Conference, 44th IEEE Conference, Seville, Spain, December, 2005*, pages 696–701. IEEE, 2005.

3 Aristotle. *Physics*. Oxford University Press, 2008.

4 Steve Awodey. *Category Theory*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 2010.

5 Stephen Bloom and Zoltán Ésik. *Iteration theories: the equational logic of iterative processes*. Springer, 1993.

6 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 – Concurrency Theory*, pages 435–450, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

7 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. The calculus of signal flow diagrams I: linear relations on streams. *Inf. Comput.*, 252:2–29, 2017.

8 Zhou Chaochen, Wang Ji, and Anders P. Ravn. A formal description of hybrid systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 511–530. Springer Berlin Heidelberg, 1996.

9 Fredrik Dahlqvist and Renato Neves. Compositional semantics for new paradigms: probabilistic, hybrid and beyond. *arXiv preprint arXiv:1804.04145*, 2018.

10 Calvin Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium 1973*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 175–230. Elsevier, 1975.

11 Ryszard Engelking. *General topology*, volume 6 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1989. Translated from the Polish by the author.

12 R. Goebel, J.P. Hespanha, A.R. Teel, C. Cai, and R. G. Sanfelice. Hybrid systems: generalized solutions and robust stability. In *Proc. 6th IFAC Symposium in Nonlinear Control Systems*, page 1–12, 2004.

13 Sergey Goncharov, Julian Jakob, and Renato Neves. A semantics for hybrid iteration. *arXiv*, 2018. `arXiv:1807.01053`.

14 Sergey Goncharov, Christoph Rauch, and Lutz Schröder. Unguarded recursion on coinductive resumptions. In *Mathematical Foundations of Programming Semantics, MFPS 2015*, volume 319 of *ENTCS*, pages 183–198. Elsevier, 2015.

15 Sergey Goncharov and Lutz Schröder. Guarded traced categories. In Christel Baier and Ugo Dal Lago, editors, *Proc. 21th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2018)*, LNCS. Springer, 2018.

16 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. Unifying guarded and unguarded iteration. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FoSSaCS 2017*, volume 10203 of *LNCS*, pages 517–533. Springer, 2017.

17 Thomas A. Henzinger. The theory of hybrid automata. In *LICS96': Logic in Computer Science, 11th Annual Symposium, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE, 1996.

18 Peter Höfner. *Algebraic calculi for hybrid systems*. PhD thesis, University of Augsburg, 2009. URL: `http://opus.bibliothek.uni-augsburg.de/volltexte/2010/1481/`.

19 Peter Höfner and Bernhard Möller. Fixing Zeno gaps. *Theoretical Computer Science*, 412(28):3303–3322, 2011. Festschrift in Honour of Jan Bergstra.

20 Christoph Lüth and Neil Ghani. Composing monads using coproducts. In M. Wand and S. L. Peyton Jones, editors, *ICFP'02: Functional Programming, 7th ACM SIGPLAN International Conference*, pages 133–144. ACM, 2002.

21 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

22 Stefan Milius. Completely iterative algebras and completely iterative monads. *Inf. Comput.*, 196(1):1–41, 2005.

23 Eugenio Moggi. A modular approach to denotational semantics. In *Category Theory and Computer Science, CTCS 1991*, volume 530 of *LNCS*, pages 138–139. Springer, 1991.

24 Katsunori Nakamura and Akira Fusaoka. On transfinite hybrid automata. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 495–510, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

25 Renato Neves. *Hybrid programs*. PhD thesis, Minho University, 2018.

26 Renato Neves, Luis S. Barbosa, Dirk Hofmann, and Manuel A. Martins. Continuity as a computational effect. *Journal of Logical and Algebraic Methods in Programming*, 85(5, Part 2):1057–1085, 2016. Articles dedicated to Prof. J. N. Oliveira on the occasion of his 60th birthday.

27 Lawrence Perko. *Differential equations and dynamical systems*, volume 7. Springer Science & Business Media, 2013.

28 André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

29 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Logic in Computer Science, LICS 2000*, pages 30–41, 2000.

30 Kohei Suenaga and Ichiro Hasuo. Programming with infinitesimals: A while-language for hybrid system modeling. In *International Colloquium on Automata, Languages, and Programming*, pages 392–403. Springer, 2011.

31 Paulo Tabuada. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.

32 Hans Witsenhausen. A class of hybrid-state continuous-time dynamic systems. *IEEE Transactions on Automatic Control*, 11(2):161–167, 1966.

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

GPU Schedulers: How Fair Is Fair Enough?

Tyler Sorensen
Imperial College London, UK
t.sorensen15@imperial.ac.uk

Hugues Evrard
Imperial College London, UK
h.evrard@imperial.ac.uk

Alastair F. Donaldson
Imperial College London, UK
alastair.donaldson@imperial.ac.uk

Abstract

Blocking synchronisation idioms, e.g. mutexes and barriers, play an important role in concurrent programming. However, systems with semi-fair schedulers, e.g. graphics processing units (GPUs), are becoming increasingly common. Such schedulers provide varying degrees of fairness, guaranteeing enough to allow some, but not all, blocking idioms. While a number of applications that use blocking idioms do run on today's GPUs, reasoning about liveness properties of such applications is difficult as documentation is scarce and scattered.

In this work, we aim to clarify fairness properties of semi-fair schedulers. To do this, we define a general temporal logic formula, based on weak fairness, parameterised by a predicate that enables fairness per-thread at certain points of an execution. We then define fairness properties for three GPU schedulers: HSA, OpenCL, and occupancy-bound execution. We examine existing GPU applications and show that none of the above schedulers are strong enough to provide the fairness properties required by these applications. It hence appears that existing GPU scheduler descriptions do not entirely capture the fairness properties that are provided on current GPUs. Thus, we present two new schedulers that aim to support existing GPU applications. We analyse the behaviour of common blocking idioms under each scheduler and show that one of our new schedulers allows a more natural implementation of a GPU protocol.

2012 ACM Subject Classification Software and its engineering → Semantics, Software and its engineering → Scheduling, Computing methodologies → Graphics processors

Keywords and phrases GPU scheduling, Blocking synchronisation, GPU semantics

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.23

Acknowledgements We are grateful to Jeroen Ketema, John Wickerson, and Qiyi Tang for feedback and insightful discussions around this work. We are grateful to Joseph Greathouse for pointing out additional applications that require HSA progress guarantees. We thank the CONCUR reviewers for their thorough evaluations and feedback. This work was supported by the EPSRC, through an Early Career Fellowship (EP/N026314/1), the IRIS Programme Grant (EP/R006865/1) and a gift from Intel Corporation.

1 Introduction

The *scheduler* of a concurrent system is responsible for the placement of virtual threads onto hardware resources. There are often insufficient resources for all threads to execute in parallel, and it is the job of the scheduler to dictate resource sharing, potentially influencing the temporal semantics of concurrent programs. For example, consider a two threaded

program where thread 0 waits for thread 1 to set a flag. If the scheduler never allows thread 1 to execute then the program will hang due to starvation. Thus, to reason about liveness properties, developers must understand the *fairness* guarantees provided by the scheduler.

GPUs are highly parallel co-processors found in many devices, from mobile phones to super computers. While these devices initially accelerated graphics computations, the last ten years have seen a strong push towards supporting general purpose computation on GPUs. Today, through programming models such as OpenCL [11], CUDA [15], and HSA [8] we have mature frameworks to execute C-like programs on GPUs.

Current GPU programming models offer a compelling case study for schedulers for three reasons: (1) some blocking idioms, e.g. barriers, are known to hang due to starvation on current GPUs [20]; (2) other blocking idioms, e.g. mutexes, run without starvation on current GPUs; yet (3) documentation for some GPU programming models explicitly states that no guarantees are provided, while others state only minimal guarantees that are insufficient to ensure starvation-freedom even for mutexes. Because GPU schedulers are embedded in closed proprietary frameworks, we do not look at concrete scheduling logic, but instead aim to derive formal fairness guarantees from prose documentation and observed behaviours.

GPUs have a hierarchical programming model: in OpenCL threads are partitioned into *workgroups*. Threads in the same workgroup can synchronise via intrinsic instructions (e.g. the OpenCL `barrier` instruction [10, p. 99]). Yet, despite practical use cases (see Section 4), there are no such intrinsics for inter-workgroup synchronisation. Instead, inter-workgroup synchronisation is achieved by building constructs, e.g. mutexes, using finer-grained primitives, e.g. atomic read-modify-write instructions (RMWs). However, reasoning about such constructs is difficult as inter-workgroup thread interactions are relatively unstudied, especially in relation to fairness. Given this, we focus only on inter-workgroup interactions and threads will be assumed to be in disjoint workgroups. Under this constraint, it is cumbersome to use the word *workgroup*. Because we can think of a workgroup as being a "composite thread", we henceforth use the word *thread* to mean *workgroup*.

The unfair OpenCL scheduler and non-blocking programs. OpenCL is a programming model for parallel systems with wide support for GPUs. Due to scheduling concerns, OpenCL disallows all blocking synchronisation, stating [11, p. 29]: "A conforming implementation may choose to serialize the [threads] so a correct algorithm cannot assume that [threads] will execute in parallel. There is no safe and portable way to synchronize across the independent execution of [threads] . . . ." Such weak guarantees are acceptable for many GPU programs, e.g. matrix multiplication, as they are *non-blocking*. That is, these programs will terminate under an *unfair* scheduler, i.e. a scheduler that provides no fairness guarantees.

Blocking idioms and fair schedulers. On the other hand, there are many useful *blocking* synchronisation idioms, which require fairness properties from the scheduler to ensure starvation-freedom. Three common examples of blocking idioms considered throughout this work, *barrier*, *mutex* and *producer-consumer (PC)*, are described in Table 1. Intuitively, a *fair* scheduler provides the guarantee that any thread that is able to execute will eventually execute. Fair schedulers are able to guarantee starvation-freedom for the idioms of Table 1.

1.1 Semi-fair schedulers: HSA and occupancy-bound execution

We have described two schedulers: fair and unfair, under which starvation-freedom for blocking idioms is either always or never guaranteed. However, some GPU programming models have *semi-fair* schedulers, under which starvation-freedom is guaranteed for only

Table 1 Blocking synchronisation constructs considered in this work.

Barrier	Mutex	Producer-consumer (PC)
Aligns the execution of all participating threads: a thread waits at the barrier until all threads have reached the barrier. Blocking, as a thread waiting at the barrier relies on the other threads to make enough progress to also reach the barrier.	Provides mutual exclusion for a critical section. A thread *acquires* the mutex before executing the critical section, ensuring exclusive access. Upon leaving, the mutex is *released*. Blocking, as a thread waiting to acquire relies on the the thread in the critical section to eventually release the mutex.	Provides a handshake between threads. A *producer* thread prepares some data and then sets a flag. A *consumer* thread waits until the flag value is observed and then reads the data. Blocking, as the consumer thread relies on the the producer thread to eventually set the flag.

Table 2 Blocking synchronisation idioms guaranteed starvation-freedom under various schedulers.

	fair	HSA	OBE	unfair (e.g. OpenCL)
barrier	yes	no	occupancy-limited	no
mutex	yes	no	yes	no
PC	yes	one-way	occupancy-limited	no

some blocking idioms. We describe two such schedulers and informally analyse the idioms of Table 1 under these schedulers (summarised in Table 2). If starvation-freedom is guaranteed for all threads executing idiom i under scheduler s then we say that i is *allowed* under s.

Similar to OpenCL, Heterogeneous System Architecture (HSA) is a parallel programming model designed to efficiently target GPUs [8]. Unlike OpenCL however, the HSA scheduler conditionally allows blocking between threads based on *thread ids*, a unique contiguous natural number assigned to each thread. Namely, thread B can block thread A, if: "[thread] A comes after B in [thread] flattened id order" [8, p. 46]. Under this scheduler: a barrier is *not* allowed, as all threads wait on all other threads regardless of id; a mutex is *not* allowed, as the ids of threads are not considered when acquiring or releasing the mutex; PC is *conditionally* allowed *if* the producer has a lower id than the consumer.

Occupancy-bound execution (OBE) is a pragmatic GPU execution model that aims to capture the guarantees that current GPUs have been shown experimentally to provide [20]. While OBE is not officially supported, many GPU programs (discussed in Section 4) depend on its guarantees. OBE guarantees fairness among the threads that are currently *occupant* (i.e., are actively executing) on the hardware resources. The fairness properties are described as [20, p. 5]: "A [thread that has executed at least one instruction] is guaranteed to eventually be scheduled for further execution on the GPU." Under this scheduler: a barrier is *not* allowed, as all threads wait on all other threads regardless of whether they have been scheduled previously; a mutex *is* allowed, as a thread that has previously acquired a mutex will be fairly scheduled such that it eventually releases the mutex; PC is *not* allowed, as there is no guarantee that the producer will be scheduled relative to the consumer.

While general barrier and PC idioms are not allowed under OBE, constrained variants have been shown to be allowed by using an occupancy discovery protocol [20] (described in Section 5.1). The protocol works by identifying a subset of threads that have been observed

to take an execution step, i.e. it *discovers* a set of co-occupant threads[1]. Barrier and PC idioms are then able to synchronise threads that have been discovered; thus we say OBE allows *occupancy-limited* variants of these idioms.

It is worth noticing that the two variants of PC shown in Table 2 (occupancy-limited and one-way) are incomparable. That is, one-way is not occupancy-limited, as the OBE scheduler makes no guarantees about threads with lower ids being scheduled before threads with higher ids. Similarly, occupancy-limited is not one-way, as the OBE scheduler allows bi-directional PC synchronisation if both threads have been observed to be co-occupant.

▶ Remark (CUDA). Like OpenCL, CUDA gives no scheduling guarantees, stating [15, p. 11]: "[Threads] are required to execute independently: It must be possible to execute them in any order, in parallel or in series." Still, some CUDA programs rely on OBE or HSA guarantees (see Section 4). The recent version 9 of CUDA introduces *cooperative groups* [15, app. C], which provide primitive barriers between programmer specified threads. Because only barriers are provided, we do not consider cooperative groups. Indeed, we aim to reason about fine-grained fairness guarantees, as required by general blocking synchronisation.

1.2 Contributions and outline

The results of Table 2 raise the following points, which we aim to address:

1. The temporal correctness of common blocking idioms varies under different GPU schedulers; however, we are unaware of any formal scheduler descriptions that are able to validate these observations.
2. Two GPU models, HSA and OBE, have schedulers that are incomparable. However, for each scheduler, there are real programs that rely on its scheduling guarantees. Thus, neither of these schedulers captures all of the guarantees observed on today's GPUs.

To address (1), we develop a formalisation, based on weak fairness, for describing the fairness guarantees of semi-fair schedulers. This formula is parameterised by a *thread fairness criterion* (TFC), a predicate over a thread and the program state, that can be tuned to provide a desired degree of fairness. We illustrate our ideas by defining thread fairness criteria for HSA and OBE (Section 3).

To address (2), we first substantiate the claim by examining blocking GPU programs that run on current GPUs. We show that there are real programs that rely on HSA guarantees, as well as programs that rely on OBE guarantees (Section 4). That is, neither the HSA nor the OBE schedulers entirely capture guarantees on which existing GPUs applications rely.

Thus, we define fairness properties for two new schedulers: *HSA+OBE*, a combination of HSA and OBE, and *LOBE* (linear OBE), an intuitive strengthening of OBE based on contiguous thread ids. Both provide the guarantees required by current programs (Section 5), however we argue that LOBE corresponds to a more intuitive scheduler. We then present an optimisation to the occupancy discovery protocol [20] that exploits exclusive LOBE guarantees (Section 5.1). The schedulers we discuss are summarised in Figure 1.

To summarise, our contributions are as follows:

- We formalise the notion of semi-fair schedulers using a temporal logic formula and use this definition to describe the HSA and OBE GPU schedulers (Section 3).

[1] Some prior work uses *co-occupant* to describe the over-subscription of threads (workgroups) on a physical GPU core. In this work, we use *co-occupant* to mean threads (workgroups) that are logically executed in parallel on the GPU, potentially spanning many physical cores.

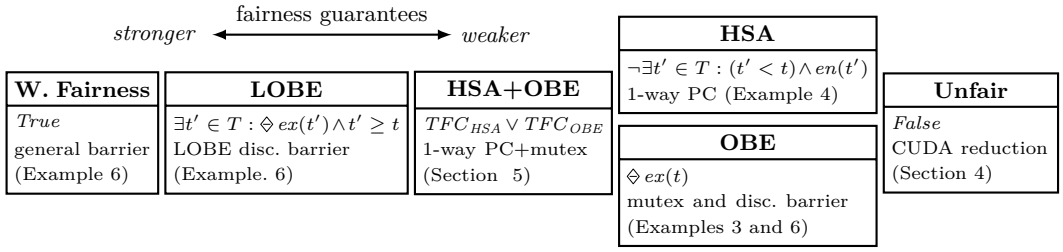


Figure 1 The semi-fair schedulers we define in this work from strongest to weakest. The first line in the box shows the scheduler's TFC (over a thread t), followed by the idiom(s) allowed under the scheduler and where the idiom is analysed. For any scheduler: any idiom to the right of a scheduler is allowed by the scheduler and any idiom to the left is disallowed. HSA and OBE are vertically aligned as they are not comparable.

- We examine blocking GPU applications and show that no existing GPU scheduler definition is strong enough to describe the guarantees required by all such programs (Section 4).
- We present two new semi-fair schedulers that meet the requirements of current blocking GPU programs: HSA+OBE and LOBE (Section 5). LOBE is shown to provide a more natural implementation of a GPU protocol (Section 5.1).

We have discussed related work on programming models and GPU schedules above, while applications that depend on specific schedulers are surveyed in Section 4.

2 Background

2.1 GPU programming

GPU programming models provide a hierarchical thread organisation. A GPU program, or a *kernel*, is executed by a set of workgroups, which we refer to as threads for convenience. Threads have access to a thread id, which can be used to partition inputs of data-parallel programs. We assume these constraints, which are common to GPU applications:

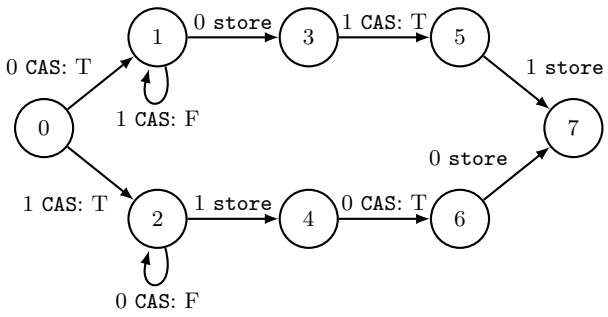
1. *Termination*: Programs are expected to terminate under a fair scheduler. GPU programs generally terminate, and in fact, they get killed by the OS if they execute for too long [19].
2. *Static thread count*: while dynamic thread creation has recently become available, e.g. nested parallelism [11, p. 30], we believe static parallelism should be studied first.
3. *Deterministic threads*: we assume that the scheduler is the only source of nondeterminism; the computation performed by a thread depends only on the program input and the order in which threads interleave. This is the case for all GPU programs examined.
4. *Enabled threads*: we assume all threads are *enabled*, i.e. able to be executed, at the beginning of the program and do not cease to be enabled until they terminate. While some systems contain scheduler-aware intrinsics, e.g. condition variables [3], GPU programming models do not. As a result, the idioms of Table 1 are implemented using atomic operations and busy-waiting, which do not change whether a thread is enabled or not.
5. *Sequential consistency*: while GPUs have relaxed memory models (e.g. see [18]), we believe scheduling under the interleaving model should be understood first.

2.2 Formal program reasoning

A *sequential* program is a sequence of instructions and its behaviour can be reasoned about by step-wise execution of instructions. We do not provide instruction-level semantics, but examples can be found in the literature (e.g. for GPUs, see [4]). A *concurrent* program is the

```
1  void thread0(mutex m) {
2    // acquire
3    while(!CAS(m,0,1));
4    // release
5    store(m,0);
6  }
7  void thread1(mutex m) {
8    // acquire
9    while(!CAS(m,0,1));
10   // release
11   store(m,0);
12 }
```

(a)



(b)

Figure 2 Two threaded mutex idiom (a) program code and (b) corresponding LTS.

parallel composition of n sequential programs, for some $n > 1$. The set $T = \{0, 1, \ldots, n-1\}$ provides a unique id for each thread, often called the *tid*. The behaviour of a concurrent program is defined by all possible *interleavings* of atomic (i.e. indivisible) instructions executed by the threads. Let A be the set of available atomic instructions.

For example, Figure 2a shows two sequential programs, `thread0` (with tid of 0) and `thread1` (with tid of 1), which both have access to a shared mutex object (initially 0). The set A of atomic instructions is {`CAS(m,old,new)`, `store(m,v)`}, whose semantics are as follows:

- `CAS(m,old,new)` – atomically checks whether the value of `m` is equal to `old`. If so, updates the value to `new` and returns true (T). Otherwise returns false (F).
- `store(m,v)` – atomically stores the value of `v` to `m`.

Using these two instructions, Figure 2a implements a simple mutex idiom, in which each thread loops trying to acquire a mutex (via the `CAS` instruction), and then immediately releases the mutex (via the `store` instruction). While other mutex implementations exist, e.g. see [7, ch. 7.2], the blocking behaviour shown in Figure 2a is idiomatic to mutexes.

Labelled transition systems. To reason about concurrent programs, we can use use a *labelled transition system* (LTS). Formally, an LTS L is a 4-tuple $(S, I, L, \rightarrow)$ where

- S is a finite set of states, with $I \subseteq S$ the set of initial states. A state contains values for all program variables and a program counter for each thread.
- $L \subseteq T \times A$ is a set of labels. A label is a pair (t, a) consisting of a thread id $t \in T$ and an atomic instruction $a \in A$.
- $\rightarrow \subseteq S \times L \times S$ is a transition relation. For convenience, given $(p, (t, a), q) \in \rightarrow$, we write $p \xrightarrow{t} q$. This is not ambiguous as we consider only per-thread deterministic programs. We use dot notation to refer to members of the tuple; e.g., given $\alpha \in \rightarrow$, we write $\alpha.t$ to refer to the thread id component of α.

Given a concurrent program, the LTS can be constructed iteratively. A start state s is created with program initial values (0 unless stated otherwise). For each thread $t \in T$, the next instruction $a \in A$ is executed to create state s' to explore. L is updated to include (t, a) and $(s, (t, a), s')$ is added to $\rightarrow$. This process iterates until there are no more states to explore. We show the LTS for the program of Figure 2a in Figure 2b. For ease of presentation, we omit state program values; labels show the thread id followed by the atomic action. If the action has a return value (e.g. `CAS`), it is shown following the action.

For a thread id $t \in T$ and state $p \in S$, we say that t is *enabled* in p, and write $en(p,t)$, if there exists a state $q \in S$ such that $p \xrightarrow{t} q$. We call p a *terminal* state, and write $terminal(p)$, if no thread is enabled in p; that is, $\neg en(p,t)$ holds for all $t \in T$. Intuitively, $en(p,t)$ states that it is possible for a thread t to take a step at state p and $terminal(p)$ states that all threads have completed execution at state p. The program constraints of Section 2.1 ensure that all threads are enabled until their termination.

Program executions and temporal logic. The executions E of a concurrent program are all possible *paths* through its LTS. Formally, a path $z \in E$ is a (possibly infinite) sequence of transitions: $\alpha_0\alpha_1\ldots$, with each $\alpha_i \in \rightarrow$, such that: the path starts in an initial state, i.e. $\alpha_0.p \in I$; adjacent transitions are connected, i.e. $\alpha_i.q = \alpha_{i+1}.p$; and if the path is finite, with n transitions, then it leads to a terminal state, i.e. $terminal(\alpha_{n-1}.q)$.

▶ Remark (Infinite paths). Because we assume programs terminate under fair scheduling (Section 2.1), infinite paths are discarded by the fair scheduler, but not necessarily by semi-fair schedulers. Indeed, these infinite paths allow us to distinguish semi-fair schedulers.

Given a path z and a transition α_i in z, $pre(\alpha_i, z)$ is used to denote the transitions up to, and including, i of z, that is, $\alpha_0\alpha_1\ldots\alpha_i$. Similarly, $post(\alpha_i, z)$ is used to denote the (potentially infinite) transitions of z from α_i, that is, $\alpha_i\alpha_{i+1}\ldots$. For convenience, we use $en(\alpha,t)$ to denote $en(\alpha.p,t)$ and $terminal(\alpha)$ to denote $terminal(\alpha.q)$. Finally, we define a new predicate $ex(\alpha,t)$ which holds if and only if $t = \alpha.t$. Intuitively, $ex(\alpha,t)$ indicates that thread t executes the transition α.

The notion of executions E over an LTS allows reasoning about *liveness* properties of programs. However, the full LTS may yield paths that realistic schedulers would exclude, illustrated in Example 1. Thus, fairness properties, provided by the scheduler, are modelled as a *filter* over the paths in E.

▶ **Example 1** (Mutex without fairness)**.** The two-threaded mutex LTS given in Figure 2b shows that it is possible for a thread to loop indefinitely waiting to acquire the mutex if the other thread is in the critical section, as seen in states 1 and 2. Developers with experience writing concurrent programs for traditional CPUs know that on most systems, these non-terminating paths do not occur in practice!

Fairness filters and liveness properties can be expressed using *temporal logic*. For a path z and a transition α in z, temporal logic allows reasoning over $post(\alpha, z)$ and $pre(\alpha, z)$, i.e. reasoning about future and past behaviours. Following the classic definitions of fairness, linear time temporal logic (LTL), is used in this work (see, e.g. [2, ch. 5] for an in-depth treatment of LTL). For ease of presentation, a less common operator, $\diamondsuit$, from past-time temporal logic (which has the same expressiveness as LTL [12]) is used. Temporal operators are evaluated with respect to z (a path) and α (a transition) in z. They take a formula ϕ, which is either another temporal formula or a transition predicate, ranging over $\alpha.p$, $\alpha.t$, or $\alpha.q$ (e.g. *terminal*). The three temporal operators used in this work are:

- The global operator $\square$, which states that ϕ must hold for all $\alpha' \in post(\alpha, z)$.
- The future operator $\diamondsuit$, which states that ϕ must hold for at least one $\alpha' \in post(\alpha, z)$.
- The past operator $\diamondsuit$, which states that ϕ must hold for at least at one $\alpha' \in pre(\alpha, z)$.

To show that a liveness property f holds for a program with executions E, it is sufficient to show that f holds for all pairs (z, α) such that $z \in E$ and α is the first transition in z. For example, one important liveness property is eventual termination: $\diamondsuit terminal$. Applying this

formula to the LTS of Figure 2b, a counter-example (i.e. a path that does not terminate) is easily found: $0 \xrightarrow{1} 2, (2 \xrightarrow{0} 2)^\omega$. In this path, thread 0 loops indefinitely trying to acquire the mutex. Infinite paths are expressed using ω-regular expressions [2, ch. 4.3].

However, many systems reliably execute mutex programs similar to Figure 2a. Such systems have *fair* schedulers, which do not allow the infinite paths described above. A fairness guarantee is expressed as a temporal predicate on paths and is used to filter out problematic paths before a liveness property, e.g. eventual termination, is considered.

In this work, *weak fairness* [2, p. 258] is considered, which is typically expressed as:

$$\forall t \in T : \Diamond\Box en(t) \implies \Box\Diamond ex(t) \tag{1}$$

Recall that en and ex are both evaluated over a transition α and a tid t. In this case, both predicates are partially evaluated with respected to a t. Intuitively, weak fairness states that if a thread *is able to* execute, then it will *eventually* execute.

▶ **Example 2** (Mutex with weak fairness)**.** We now return to the task of proving termination for the LTS of Figure 2b. If the scheduler provides weak fairness, then we can discard all paths that do not satisfy the weak fairness definition. Namely, the two problematic paths are: $0 \xrightarrow{1} 2, (2 \xrightarrow{0} 2)^\omega$ and $0 \xrightarrow{0} 1, (1 \xrightarrow{1} 1)^\omega$. Neither path satisfies weak fairness: in both cases the thread that can break the cycle is always enabled, yet it is not eventually executed once the infinite cycle begins. Thus, if executed on system which provides weak fairness, the program of Figure 2a is guaranteed to eventually terminate.

3 Formalising semi-fairness

We now detail our formalism for reasoning about fairness properties for semi-fair schedulers. Semi-fairness is parameterised by a thread predicate called the *thread fairness criterion*, or TFC. Intuitively, the TFC states a condition which, if satisfied by a thread t, guarantees fair execution for t.

Formally an execution is semi-fair with respect to a TFC if the following holds:

$$\forall t \in T : \Diamond\Box(en(t) \wedge TFC(t)) \implies \Box\Diamond ex(t) \tag{2}$$

The formula is similar to weak fairness (Eq. 1), but in order for a thread t to be guaranteed eventual execution, not only must t be enabled, but the TFC for t must also hold. Semi-fairness for different schedulers, e.g. HSA and OBE, can be instantiated by using different TFCs, which in turn will yield different liveness properties for programs under these schedulers, e.g. as shown in Table 2.

The weaker the TFC is, the stronger the fairness condition is. Semi-fairness with the the weakest TFC, i.e. true, yields classic weak fairness. Conversely, semi-fairness with the strongest TFC, i.e. false, yields no fairness.

Formalising a specific notion of semi-fairness now simply requires a TFC. We illustrate this by defining TFCs to describe the semi-fair guarantees provided by the OBE and HSA GPU schedulers, introduced informally in Section 1.

Formalising OBE semi-fairness. The prose definition for the OBE scheduler fits this formalism nicely, as it describes the per-thread condition for fair scheduling: once a thread has been scheduled (i.e. executed an instruction), it will continue to be fairly scheduled. This is

straightforward to encode in a TFC using the $\diamondsuit$ temporal logic operator (see Section 2.2), which holds for a given predicate if that predicate has held at least once in the past. Thus the TFC for the OBE scheduler can be stated formally as follows:

$$TFC_{OBE}(t) = \diamondsuit\, ex(t) \tag{3}$$

Formalising HSA semi-fairness. A TFC for the HSA scheduler is less straightforward because the prose documentation is given in terms of relative allowed blocking behaviours, rather than in terms of thread-level fairness. Recall the definition from Section 1: thread B can block thread A if: "[thread] A comes after B in [thread] flattened id order" [8, p. 46]. Searching the documentation further, another snippet phrased closer to a TFC is found, stating [8, p. 28]: "[Thread] $i + j$ might start after [thread] i finishes, so it is not valid for a [thread] to wait on an instruction performed by a later [thread]." We assume here that j refers to any positive integer. Because these prose documentation snippets do not discuss fairness explicitly, it is difficult to directly extract a TFC. We make a best-effort attempt following this reasoning: (1) if thread i is fairly scheduled, no thread with id greater than i is guaranteed to be fairly scheduled; and (2) threads that are not enabled (i.e. they have terminated) have no need to be fairly scheduled. Using these two points, we can derive a TFC for HSA: a thread is guaranteed to be fairly scheduled if there does not exist another thread that has a lower id and is enabled. Formally:

$$TFC_{HSA}(t) = \neg\exists t' \in T : (t' < t) \wedge en(t') \tag{4}$$

Although this TFC is somewhat removed from the prose snippets in the HSA documentation, this formal definition has value in enabling precise discussions about fairness. For example, we can increase confidence in our definition by showing that the idioms informally analysed in Section 1 behave as expected; see Examples 3, 4 and 6. Our formalism for HSA provides few progress guarantees, and as we discuss in Section 4, current GPUs appear to offer stronger guarantees than HSA. The HSA programming model may offer such weak guarantees to allow for a variety of devices to be targeted by this programming model and also to allow flexibility in future framework implementations.

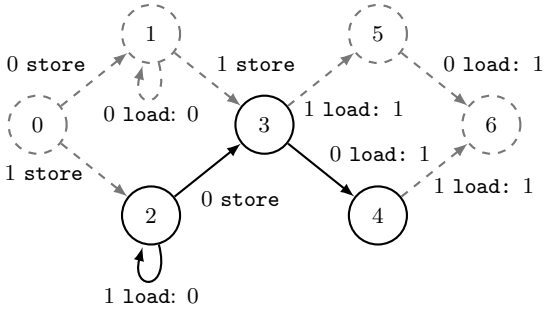
▶ **Example 3** (Mutex with semi-fairness)**.** Here we analyse the mutex LTS of Figure 2b under OBE and HSA semi-fairness guarantees. Recall the two problematic paths (causing starvation) are: $0 \xrightarrow{0} 1, (1 \xrightarrow{1} 1)^{\omega}$. and $0 \xrightarrow{1} 2, (2 \xrightarrow{0} 2)^{\omega}$

- **OBE**: In both problematic paths, one thread t acquires the mutex, and the other thread t' spins indefinitely. However, thread t has executed an instruction (acquiring the mutex) and is thus guaranteed eventual execution under OBE; the problematic paths violate this guarantee as thread t never executes after it acquires. Therefore both paths are discarded, guaranteeing starvation-freedom for mutexes under OBE.
- **HSA**: The second problematic path: $0 \xrightarrow{1} 2, (2 \xrightarrow{0} 2)^{\omega}$, cannot be discarded as thread 0 waits for thread 1 to release. Thread 1 does not have the lowest id of the enabled threads, thus there is no guarantee of eventual execution. Therefore starvation-freedom for mutexes cannot be guaranteed under HSA.

▶ **Example 4** (Producer-consumer with semi-fairness)**.** Figure 3 illustrates a two-threaded producer-consumer program. We use a new atomic instruction, `load`, which simply reads a value from memory (the return value is given on the LTS edges). Thread 0 produces a value via `x0` and then spins, waiting to consume a value via `x1`. Thread 1 is similar, but with

```
1  void thread0(int x0, int x1) {
2    // produce to tid 1
3    store(x0,1);
4    // consume from tid 1
5    while(load(x1) != 1);
6  }
7  void thread1(int x0, int x1) {
8    // produce to tid 0
9    store(x1,1);
10   // consume from tid 0
11   while(load(x0) != 1);
12 }
```

(a)



(b)

Figure 3 Two threaded PC idiom (a) code and (b) LTS. Omitting in (a) lines in gray and in (b) states and transitions in gray and dashed lines yields the one-way variant of this idiom.

the variables swapped. A subset of this program, omitting lines 4, 5, 8, and 9, shows the *one-way* producer-consumer idiom, where threads only consume from threads with lower ids, i.e., only thread 1 consumes from thread 0. The LTS for the one-way variant omits states 0, 1, 5, and 6 and the start state changes to state 2.

There are two problematic paths for the general test, in which one of the threads spins indefinitely waiting for the other thread to produce a value: $0 \xrightarrow{0} 1, (1 \xrightarrow{0} 1)^\omega$, and $0 \xrightarrow{1} 2, (2 \xrightarrow{1} 2)^\omega$. For the one-way variant, there is one problematic path: $(2 \xrightarrow{1} 2)^\omega$. We now analyse this program under OBE and HSA semi-fairness.

- **OBE**: Consider the problematic path $0 \xrightarrow{1} 2, (2 \xrightarrow{1} 2)^\omega$. Because thread 0 has not executed an instruction, OBE does not guarantee eventual execution for thread 0 and thus this path cannot be discarded. Similar reasoning shows that the problematic path for the one-way variant cannot be discarded either. Thus, neither general nor one-way producer consumer idioms are allowed under OBE.
- **HSA**: Consider the problematic path $0 \xrightarrow{0} 1, (1 \xrightarrow{0} 1)^\omega$. Because thread 1 does not have the lowest id of the enabled threads, HSA does not guarantee eventual execution for thread 1 and this path cannot be discarded. On the other hand, consider the problematic path for the one-way variant: $(2 \xrightarrow{1} 2)^\omega$. Because thread 0 has the lowest id of the enabled threads, HSA guarantees thread 0 will eventually execute, thus causing this path to be invalid. Therefore, general producer-consumer is not allowed under HSA, but one-way producer-consumer, following increasing order of thread ids, is allowed.

4 Inter-workgroup synchronisation in the wild

We now examine existing GPU applications to determine what scheduling guarantees they assume. This provides a basis for understanding (1) what scheduling guarantees are actually provided by existing GPUs, as these applications run without issues on current devices, and (2) the utility of schedulers, i.e. whether their fairness guarantees are exploited in current applications. We limit our exploration to GPU applications that use inter-workgroup synchronisation, and we perform a best-effort search through popular works in this domain. We manually examined the programs, searching for the idioms in Table 1, and relate them to the corresponding scheduler under which they are guaranteed to not starve.

OBE programs. We begin by looking at applications that assume guarantees from the OBE scheduler. The most prevalent example seems to be the occupancy-limited barrier (see Section 1). That is, developers use a priori knowledge about how many threads can be

simultaneously occupant on a given GPU, and only run the program with at most that many threads. The first work on such barriers is a 2010 paper by Xiao and Feng [22]. This work has many citations, many of which describe applications that use the barrier. Additionally, a barrier implementation following this work appears in the popular CUDA library CUB [14]. Thus, the OBE scheduler guarantees appear to be well-tested and useful on current GPUs.

In 2012, Kupta et al. present the *persistent thread model* [6], which more clearly characterises the scheduling guarantees required by the Xiao and Feng Barrier and proposes work-stealing as a potential use case under this model. This work again, has many citations describing use cases. One such work-stealing application that requires OBE scheduling guarantees was published in a 2011 GPU programming cookbook *GPU Computing Gems* [9, ch. 35]. Recent interest in barriers appears in graph analytic applications (e.g. BFS, SSSP), where the 2016 IRGL application benchmark suite is reported to have competitive performance in this domain and uses both barriers and mutexes [17].

HSA programs. We found only four applications that use the one-way PC idiom: two scan implementations, a sparse triangular solve (SpTRSV) application, and a sparse matrix vector multiplication (SpMV) application. While there are few applications in this category, we argue that they are important, as they appear in vendor-endorsed libraries.

The two scan applications, one found in the popular Nvidia CUB GPU library [14] and the second presented in [23], use a straightforward one-way PC idiom. Both scans work by computing workgroup-local scans on independent chunks of a large array. Threads compute chunks according to their thread id, e.g. thread 0 processes the first chunk. A thread t then passes its local sum to its immediate neighbour, thread $t+1$, who spins while waiting for this value. The neighbour factors in this sum and then passes an updated value to its neighbour, and so forth.

The SpMV application, presented in [5], has several workgroups cooperate to calculate the result for a single row. Before any cooperation, the result must first be initialised, which is performed by the workgroup with the lowest id out of the cooperating workgroups. The other workgroups spin, waiting for the initialisation. This algorithm is implemented in the clSPARSE library [1], a joint project between AMD and Vratis.

The SpTRSV application, presented in [13], allows multiple producers to accumulate data to send to a consumer. However, in the triangular solver system, all producers will have lower ids than the relative consumers. Thus the PC idiom remains one-way.

OpenCL programs. We also searched for applications that contain non-trivial inter-workgroup synchronisation and are non-blocking, and thus guaranteed starvation-freedom under any scheduler. By non-trivial synchronisation, we mean inter-workgroup interactions that cannot be achieved by a single atomic read-modify-write (RMW) instruction. While we found examples of non-blocking data-structures (e.g. in the work-stealing example of [9, ch. 35]), the top level loop was blocking as threads without work waited on other threads to complete work. Interestingly, we found only one application that appeared to be globally non-blocking: a reduction application in the CUDA SDK [16], called `threadFenceReduction`, in which the final workgroup to finish local computations also does a final reduction over all other local computations.

5 Unified GPU semi-fairness.

The exploration of applications in Section 4 shows that there are current applications that rely on either HSA or OBE guarantees, and that these applications run without starvation on current GPUs. Hence, it appears that current GPUs provide stronger fairness guarantees than either HSA or OBE describe. In this section, we propose new semi-fairness guarantees that unify HSA and OBE guarantees, and as such, potentially provide a more accurate description of current GPUs scheduling guarantees.

HSA+OBE semi-fairness. A straightforward approach to create a unified fairness property from two existing semi-fair properties is to create a new TFC defined as the disjunction of the two existing TFCs. Thus, threads guaranteed fairness under either existing scheduler are guaranteed fairness under the unified scheduler. We can do this with the HSA and OBE semi-fair schedulers to create a new unified semi-fairness condition, called HSA+OBE, i.e.,

$$TFC_{HSA+OBE}(t) = TFC_{HSA}(t) \vee TFC_{OBE}(t) \tag{5}$$

Thinking about the set of programs for which a scheduler guarantees starvation-freedom, let P_{HSA} be the set of programs allowed under HSA, with P_{OBE} and $P_{HSA+OBE}$ defined similarly. We note that $P_{HSA} \cup P_{OBE} \subset P_{HSA+OBE}$; that is, there are programs in $P_{HSA+OBE}$ that are neither in P_{HSA} nor P_{OBE}. For example, consider a program that uses one-way producer-consumer synchronisation and also a mutex. This program is not allowed under the OBE or HSA scheduler in isolation, but is allowed under the semi-fair scheduler defined as their disjunction. However, this idiom combination seems contrived as the applications discussed in Section 4 that exploit the one-way PC idiom do not require mutexes.

LOBE semi-fairness. The HSA+OBE fairness guarantees are useful for reasoning about existing applications, but these guarantees do not seem like they would naturally be provided by a system scheduler implementation. Namely, HSA+OBE guarantees fairness to (1) the thread with the lowest id that has not terminated (thanks to HSA) and (2) threads that have taken an execution step (thanks to OBE). For example, it might allow relative fair scheduling only between threads 0, 23, 29, and 42, if they were scheduled at least once in the past. Thus, HSA+OBE allows for "gaps", where threads with relative fairness do not have contiguous ids. We believe a more intuitive scheduler would guarantee that threads with relative fairness have contiguous ids.

Given these intuitions, we describe a new semi-fair guarantee, which we call *LOBE* (linear occupancy-bound execution). Similar to OBE, LOBE guarantees fair scheduling to any thread that has taken a step. Additionally, LOBE guarantees fair scheduling to any thread t if another thread t' (1) has taken a step, and (2) has an id greater than or equal to t (hence the word *linear*). Formally, the LOBE TFC can be written:

$$TFC_{LOBE}(t) = \exists t' \in T : \diamondsuit\, ex(t') \wedge t' \geq t \tag{6}$$

We will now show that LOBE is a unified scheduler, i.e. any program allowed under HSA or OBE is allowed under LOBE. It is sufficient to show that $TFC_{OBE} \implies TFC_{LOBE}$ and $TFC_{HSA} \implies TFC_{LOBE}$. First, we consider $TFC_{OBE} \implies TFC_{LOBE}$: this is trivial as the comparison check in TFC_{LOBE} includes equality, thus any thread that has taken a step is guaranteed to be fairly scheduled.

Considering now $TFC_{HSA} \implies TFC_{LOBE}$: we first recall a property of executions from Section 2.2, namely that an execution either ends in a state where all threads have terminated, or it is infinite. Thus, at an arbitrary non-terminal point in an execution, some

thread t must take a step. If t has the lowest id of the enabled threads, then both LOBE and HSA guarantee that t will be fairly executed. If t does not have the lowest id of the enabled threads, then LOBE guarantees that *all* threads with lower ids than t will be fairly executed, including the thread with the lowest id of the enabled threads, thus satisfying the fairness constraint of HSA.

5.1 LOBE discovery protocol

Because the $TFC_{HSA+OBE}$ is defined as the disjunction of TFC_{HSA} and TFC_{OBE}, the reasoning in Section 5 is sufficient to show that LOBE fairness guarantees are at least as strong as HSA+OBE. A practical GPU program is now discussed for which correctness relies on the stronger guarantees provided by LOBE compared to HSA+OBE. This example shows that (1) LOBE guarantees are strictly stronger than HSA+OBE, and (2) fairness guarantees exclusive to LOBE can be useful in GPU applications.

Our example is a modified version of the discovery protocol from [20], which dynamically discovers threads that are guaranteed to be co-occupant, and are thus guaranteed relative fairness by OBE. The protocol works using a virtual *poll*, in which threads have a short time window to indicate, using shared memory, that they are co-occupant. The protocol acts as a filter: discovered co-occupant threads execute a program, and undiscovered threads exit without performing any meaningful computation. Because only co-occupant threads execute the program, OBE guarantees that blocking idioms such as barriers can be used reliably.

GPU programs are often data-parallel, and threads use their ids to efficiently partition arrays; thus having contiguous ids is vital. Because OBE fairness does not consider thread ids, in order to provide contiguous ids, the discovery protocol dynamically assigns *new* ids to discovered threads. While functionally this approach is sound, there are two immediate drawbacks: (1) programs must be written using new thread ids, which can require intrusive changes, and (2) the native thread id assignment on GPUs may be optimised by the driver for memory accesses in data-parallel programs; using new ids would forgo these optimisations. Exploiting the scheduling guarantees of LOBE, we modify the discovery protocol to preserve native thread ids and also ensuring contiguous ids.

▶ **Example 5** (thread ids and data locality)**.** It is possible that the protocol discovers four threads (with tids 2-5) and creates the following mapping for their new dynamic ids: $\{(5 \to 0), (2 \to 1), (3 \to 3), (4 \to 4)\}$. The GPU runtime might have natively assigned threads 2 and 3 to one processor (often called a compute unit on GPUs) and threads 4 and 5 to another. Because these compute units often have caches, data-locality between threads on the same compute unit could offer performance benefits [21]. In data-parallel programs, there is often data-locality between threads with consecutive ids. Thus, in our example mapping, the (native) threads, 2 and 5 could not exploit data locality, as their new ids are consecutive, but their native ids are not.

While it may seem straightforward to remap the ids of discovered threads to facilitate data locality, as is done in [21], we note that this depends on the ability to query the physical core id of a thread. Nvidia provides this functionality in CUDA, which is exploited in [21], but OpenCL offers no support for such a feature. Thus, the relation between thread ids and data locality is hidden by the OpenCL framework. We assume the natively assigned ids take data locality into account and that dynamically assigned ids might not be as efficient.

We show the algorithm for the discovery protocol in Algorithm 1. The changes we make to exploit LOBE guarantees are indicated by dashed boxes for removed code and solid boxes for added code. We first describe the original protocol. The algorithm has two phases, both protected by the mutex m. The first phase is the *polling phase* (lines 2-10), where threads are able to indicate that they are currently occupant (i.e. executing). The

Algorithm 1 Occupancy discovery protocol. Applying LOBE optimisation removes the code in dashed boxes and adds the code in solid boxes.

```
1:  function DISCOVERY_PROTOCOL(open, count, id_map, m)
2:      Lock(m)
3:      if open ∨(tid < count) then
4:          id_map[tid] ← count
5:          count ← count + 1
6:          count ← max(count, tid + 1)
7:          Unlock(m)
8:      else
9:          Unlock(m)
10:         return False
11:     Lock(m)
12:     if open then
13:         open ← False
14:     Unlock(m)
15:     return True
```

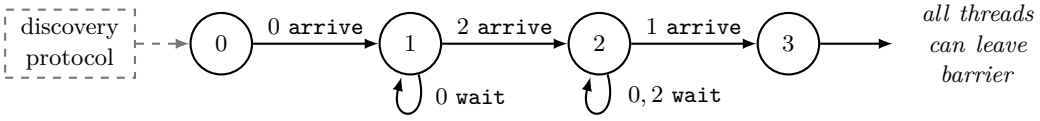


Figure 4 Sub-LTS of a barrier, with an optional discovery protocol preamble.

open shared variable is initialised to true to indicate that the poll is open. A thread first checks whether the poll is open (line 3). If so, then the thread marks itself as discovered; this involves obtaining a new id (line 4) and incrementing the number of discovered threads, via the shared variable *count* (line 5). The thread can then continue to the closing phase (starting line 11). If the poll was not open, the thread indicates that it was not discovered by returning false (lines 8-10). In the closing phase, a thread checks to see if the poll is open; if so, the thread closes the poll and no other threads can be discovered at this point (lines 12-13). All threads who enter the closing phase have been discovered to be co-occupant, thus they return true (line 15). The number of discovered threads will be stored in *count*.

We can optimise this protocol by exploiting fairness guarantees of LOBE. In particular, because LOBE guarantees that threads are fairly scheduled in contiguous id order, the protocol can allow a thread with a higher id to *discover* all threads with lower ids. As a result, threads are able to keep their native ids, although the number of discovered threads is still dynamic. The optimisation to the discovery protocol is simple: first the *id_map*, which originally mapped threads to their new dynamic ids is not needed (lines 1 and 4). Next, the number of discovered threads is no longer based on how many threads were observed to poll, but rather on the highest id of the discovered threads (line 6). Finally, even if the poll is closed, a thread entering the poll may have been discovered by a thread with a higher id; this is now reflected by each thread comparing its id with *count* (line 3). In Example 6, we show that a barrier prefaced by the LOBE optimised protocol is not allowed under HSA+OBE guarantees, and thus illustrate that LOBE fairness guarantees are strictly stronger than HSA+OBE.

▶ **Example 6** (Barriers under semi-fairness)**.** We now analyse the behaviour of barriers, with optional discovery protocols, under our semi-fair schedulers. Figure 4 shows a subset of an LTS for a barrier idiom that synchronises three threads with tids 0, 1, and 2. For the sake of clarity, instead of using atomic actions that correspond to concrete GPU atomic instructions, we use abstract instructions `arrive` and `wait`, which correspond to a thread marking its arrival at the barrier and a thread waiting at the barrier, respectively.

The sub-LTS shows one possible interleaving of threads arriving at the barrier, in the order 0, 1, 2. The final thread to arrive (thread 1) allows all threads to leave. The sub-LTS shows the various spin-waiting scenarios that can occur in a barrier at states 1 and 2. A discovery protocol can optionally be used before the barrier synchronisation.

We analyse the sub-LTS using the LOBE optimised discovery protocol (Section 5.1) here. A similar analysis for the general barrier and original discovery protocol is done in Appendix A. Recall that the LOBE discovery protocol discovers a thread if it has seen a step from a thread with an equal or greater id. In our example with three threads, the fewest behaviours the protocol is guaranteed to have seen is a step by thread 2, denoted: $DP \xrightarrow{2} 0$.

- **HSA+OBE**: consider the starvation path: $DP \xrightarrow{2} 0,0 \xrightarrow{0} 1,1 \xrightarrow{0} 2,(2 \xrightarrow{0} 2,2 \xrightarrow{2} 2)^{\omega}$. This path cannot be disallowed by HSA+OBE as at state 2, HSA+OBE guarantees fair scheduling for the thread with the lowest id (thread 0) and any threads that have taken a step (threads 0 and 2). This path requires fair execution from thread 1 to break the starvation loop. Thus, barrier synchronisation using the LOBE discovery protocol is not allowed under HSA+OBE.
- **LOBE**: The above starvation path is disallowed by LOBE, as LOBE guarantees fair execution for any thread t that has executed *and* any thread with a lower id than t. Thus, at state 0, the LOBE discovery protocol has observed a step from thread 2, we are guaranteed fair scheduling for threads 2, 1, and 0. Thus barriers with LOBE discovery protocol are allowed under LOBE.

6 Conclusion

While general purpose usage of GPUs is on the rise, current GPU programming models provide loose scheduling fairness guarantees in English prose. In practice, GPUs feature *semi-fair* schedulers. Our goal is to clarify the fairness guarantees that GPU programmers can rely on, or at least the ones they assume. To this aim, we have introduced a formalism that combines the classic weak fairness with a thread fairness criterion (TFC), enabling fairness to be specified at a per-thread level. We have illustrated this formalism by defining the TFC for HSA (from its specification) and OBE (from its description based on empirical evidence) and by reasoning with such TFCs on three classic concurrent programming idioms: barrier, mutex and producer-consumer.

We notice that while some popular existing GPU programs rely on either HSA or OBE guarantees, these two models are not comparable, hence current GPUs must support stronger guarantees that neither HSA nor OBE entirely capture. Our formalism lets easily combine the TFCs of HSA and OBE to define the HSA+OBE scheduler; and we additionally craft the LOBE scheduler which offers slightly stronger fairness guarantees than HSA+OBE. We illustrate that LOBE guarantees can be useful by showing a GPU protocol optimisation for which other GPU semi-fair schedulers do not guarantee starvation-freedom, but LOBE does.

References

1 clSPARSE. Retrieved June 2018 from `https://github.com/clMathLibraries/clSPARSE`.

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

3 Blaise Barney. POSIX threads programming: Condition variables. (visited January 2018). URL: `https://computing.llnl.gov/tutorials/pthreads/#ConditionVariables`.

4 Adam Betts, Nathan Chong, Alastair F. Donaldson, Jeroen Ketema, Shaz Qadeer, Paul Thomson, and John Wickerson. The design and implementation of a verification technique for gpu kernels. *TOPLAS*, 37(3):10:1–10:49, 2015.

5 M. Daga and J. L. Greathouse. Structural agnostic SpMV: Adapting CSR-adaptive for irregular matrices. In *HiPC*, pages 64–74. IEEE, 2015.

6 Kshitij Gupta, Jeff Stuart, and John D. Owens. A study of persistent threads style GPU programming for GPGPU workloads. In *InPar*, pages 1–14, 2012.

7 Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., 2008.

8 HSA Foundation. HSA programmer's reference manual: HSAIL virtual ISA and programming model, compiler writer, and object format (BRIG). (rev 1.1.1), March 2017. URL: `http://www.hsafoundation.com/standards/`.

9 Wen-mei W. Hwu. *GPU Computing Gems Jade Edition*. Morgan Kaufmann, 2011.

10 Khronos Group. The OpenCL C specification version 2.0 (rev. 33), May 2017. URL: `https://www.khronos.org/registry/OpenCL/specs/opencl-2.0-openclc.pdf`.

11 Khronos Group. The OpenCL specification version: 2.2 (rev. 2.2-7), May 2018. URL: `https://www.khronos.org/registry/OpenCL/specs/opencl-2.2.pdf`.

12 Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In *Logics of Programs*, pages 196–218. Springer Berlin Heidelberg, 1985.

13 Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, and Brian Vinter. A synchronization-free algorithm for parallel sparse triangular solves. In *Euro-Par*, pages 617–630. Springer, 2016.

14 Nvidia. CUB. (visited January 2018). URL: `http://nvlabs.github.io/cub/`.

15 Nvidia. CUDA C programming guide, version 9.1, January 2018. URL: `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`.

16 Nvidia. CUDA Code Samples, 2018. URL: `https://developer.nvidia.com/cuda-code-samples`.

17 Sreepathi Pai and Keshav Pingali. A compiler for throughput optimization of graph algorithms on GPUs. In *OOPSLA*, pages 1–19, 2016.

18 Tyler Sorensen and Alastair F. Donaldson. Exposing errors related to weak memory in GPU applications. In *PLDI*, pages 100–113. ACM, 2016.

19 Tyler Sorensen and Alastair F. Donaldson. The hitchhiker's guide to cross-platform OpenCL application development. In *IWOCL*, pages 2:1–2:12, 2016.

20 Tyler Sorensen, Alastair F. Donaldson, Mark Batty, Ganesh Gopalakrishnan, and Zvonimir Rakamaric. Portable inter-workgroup barrier synchronisation for GPUs. In *OOPSLA*, pages 39–58, 2016.

21 Bo Wu, Guoyang Chen, Dong Li, Xipeng Shen, and Jeffrey Vetter. Enabling and exploiting flexible task assignment on GPU through SM-centric program transformations. In *ICS*, pages 119–130. ACM, 2015.

22 Shucai Xiao and Wu-chun Feng. Inter-block GPU communication via fast barrier synchronization. In *IPDPS*, pages 1–12, 2010.

23 Shengen Yan, Guoping Long, and Yunquan Zhang. Streamscan: Fast scan algorithms for GPUs without global barrier synchronization. In *PPoPP*, pages 229–238. ACM, 2013.

A Barrier example cont.

We continue the analysis of the barrier sub-LTS of Figure 4 that was started in Example 6. That is, we analyse the general barrier (i.e. with no discovery protocol) and the the barrier using the original discovery protocol (as described in Section 5.1).

- **general barrier**:
  - *LOBE* - The starvation path $0 \xrightarrow{0} 1, (1 \xrightarrow{0} 1)^{\omega}$ is not disallowed by LOBE, as LOBE cannot guarantee fair execution for any thread other than thread 0 at state 1 where the infinite starvation path begins. Thus, general barriers are not allowed under LOBE. Because LOBE is stronger than HSA+OBE, HSA and OBE, we know that the general barrier is not allowed under these schedulers either.
- **original discovery protocol:** This discovery protocol ensures that all three threads, i.e. threads 0, 1, and 2, have taken a step before state 0. We denote this transition as $DP \xrightarrow{0,1,2} 0$.
  - *HSA* - The starvation path $DP \xrightarrow{0,1,2} 0, 0 \xrightarrow{0} 1, (1 \xrightarrow{0} 1)^{\omega}$ is not disallowed by HSA, as HSA only guarantees fair execution to the lowest enabled thread (i.e. thread 0). To break this starvation loop in the sub LTS, thread 2 would need fairness guarantees. Thus barriers using the original discovery protocol are not allowed under HSA.
  - *OBE* - Because the original discovery protocol guarantees all threads have taken a step before the barrier execution (i.e. state 0), OBE guarantees all three threads fair scheduling. Thus all starvation loops in the sub LTS are guaranteed to be broken, and the barrier using the original discovery protocol is allowed under OBE. Because HSA+OBE and LOBE are stronger than OBE, this synchronisation idiom is also allowed under those schedulers.

Linear Equations with Ordered Data

Piotr Hofman[1]
University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
piotrek.hofman@gmail.com
https://orcid.org/0000-0001-9866-3723

Sławomir Lasota[2]
University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
sl@mimuw.edu.pl
https://orcid.org/0000-0001-8674-4470

Abstract

Following a recently considered generalization of linear equations to *unordered data* vectors, we perform a further generalization to *ordered data* vectors. These generalized equations naturally appear in the analysis of vector addition systems (or Petri nets) extended with ordered data. We show that nonnegative-integer solvability of linear equations is computationally equivalent (up to an exponential blowup) to the reachability problem for (plain) vector addition systems. This high complexity is surprising, and contrasts with NP-completeness for unordered data vectors. This also contrasts with our second result, namely polynomial time complexity of the solvability problem when the nonnegative-integer restriction on solutions is relaxed.

2012 ACM Subject Classification Theory of computation → Parallel computing models, Theory of computation → Timed and hybrid models, Theory of computation → Automata over infinite objects

Keywords and phrases Linear equations, Petri nets, Petri nets with data, vector addition systems, sets with atoms, orbit-finite sets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.24

Related Version All the missing proofs are to be found in the full version of this paper, at [10], `https://arxiv.org/abs/1802.06660`.

Acknowledgements We thank anonymous reviewers for their detailed comments.

1 Introduction

Systems of linear equations are useful for approximate analysis of vector addition systems (VAS), or Petri nets. For instance, the relaxation of semantics of Petri nets, where the configurations along a run are not required to be nonnegative, yields the so called *state equation*, a system of linear equations with nonnegative-integer restriction on solutions. This is equivalent to *integer linear programming*, a well-known NP-complete problem [13]. When the nonnegative-integer restriction is further relaxed to nonnegative-rational one (or nonnegative-real one), we get a weaker but more tractable approximation, equivalent to *linear*

[1] Partially supported by Polish NCN grant 2016/21/D/ST6/01368.
[2] Partially supported by Polish NCN grant 2016/21/B/ST6/01505.

programming and solvable in PTime. We refer to [24] for an exhaustive overview of linear-algebraic and integer-linear-programming techniques in analysis of Petri nets; usefulness of these techniques is confirmed by multiple applications including, for instance, recently proposed efficient tools for the coverability problem of Petri nets [9, 1].

Motivations. Our starting point is an extension of Petri nets, or VAS, with data [17, 11]. The extension significantly enhances expressibility of the model but also increases the complexity of analysis. In case of *unordered* data (a countable set of data values that can be tested for equality only), the coverability problem is decidable (in non-elementary complexity) [17] but the decidability status of the reachability problem remains open. In case of *ordered* data (a countable dense total order), the coverability problem is still decidable [17] while reachability is not [23]. (Petri nets with ordered data are equivalent to a timed extension of Petri nets, as shown in [5].) One can also consider other data domains, and the coverability problem remains decidable as long as the data domain is homogeneous [16] (not to be confused with *homogeneous* systems of linear equations), but always in non-elementary complexity. In view of these high complexities, a natural need arises for efficient over-approximations.

A configuration of a Petri net with data domain $\mathbb{D}$ is a nonnegative integer *data vector*, i.e., a function $\mathbb{D} \to \mathbb{N}^d$ that maps only finitely many data values to a non-zero vector in $\mathbb{N}^d$. In a search for efficient over-approximations of Petri nets with data, a natural question appears: May linear algebra techniques be generalised so that the role of vectors is played by data vectors? In case of unordered data, this question was addressed in [12], where first promising results have been shown: the nonnegative-integer solvability of linear equations over unordered data domain is NP-complete. Thus, for unordered data, the problem remains within the same complexity class as its plain (data-less) counterpart. The same question for the second most natural data domain, i.e. ordered data, seems to be even more important; ordered data enables modelling features like fresh names creation [23] or time dependencies [5].

Contributions. In this paper we do a further step and investigate linear equations with ordered data, for which we fully characterise the complexity of the solvability problem. Firstly, we show that nonnegative-integer solvability of linear equations is computationally equivalent (up to an exponential blowup in one direction) with the reachability problem for plain Petri nets (or VAS). In consequence, decidability and ExpSpace-hardness of our problem follows. This high complexity is surprising, and contrasts NP-completeness for unordered data vectors.

Secondly, we prove that the complexity of the solvability problem drops to PTime, when the nonnegative-integer restriction on solutions is relaxed to rational, nonnegative-rational, or integer. The two latter problems may be thus used as two tractable but incomparable over-approximations of the reachability relation for VAS-es with ordered data. Thirdly, as a conceptual contribution we notice that systems of linear equations with (un)ordered data are a special case of systems of linear equations that are infinite but finite up to an automorphism of data domain. This can be formalized in the framework of *sets with atoms* [3, 4, 14], where finiteness is relaxed to *orbit-finiteness*.

Outline. In Section 2 we introduce the setting we work in, and formulate our results. Then the rest of the paper is devoted to proofs. First, in Section 3 we provide a lower bound for the nonnegative-integer solvability problem, by a reduction from the VAS reachability problem. Then, in Section 4 we suitably reformulate our problem in terms *multihistograms*, which are matrices satisfying certain combinatorial property. This reformulation is used in

the next Section 5 to provide a reduction from the nonnegative-integer solvability problem to the VAS reachability problem, thus proving decidability of our problem. In Section 6 we investigate the relaxations of the nonnegative-integer restriction on solutions and work out a PTIME decision procedure in each case. In the concluding Section 7 we sketch upon a generalised setting of orbit-finite systems of linear equations.

2 Vector addition systems and linear equations

In this section we introduce the setting of linear equations with data, and formulate our results. For a gentle introduction of the setting, we start by recalling classical linear equations.

Let $\mathbb{Q}$ denote the set of rationals, and $\mathbb{Q}_+, \mathbb{Z}$, and $\mathbb{N}$ denote the subsets of nonnegative rationals, integers, and nonnegative integers. Classical linear equations are of the form

$$a_1 x_1 + \ldots + a_m x_m = a,$$

where $x_1 \ldots x_m$ are variables (unknowns), and $a_1 \ldots a_m \in \mathbb{Z}$ are integer coefficients (equivalently, rational coefficients could be allowed). For a finite system $\mathcal{U}$ of such equations over the same variables $x_1, \ldots, x_m$, a solution of $\mathcal{U}$ is a vector $(n_1, \ldots, n_m) \in \mathbb{Q}^m$ such that the valuation $x_1 \mapsto n_1, \ldots, x_m \mapsto n_m$ satisfies all equations in $\mathcal{U}$. In the sequel we are most often interested in nonnegative integer solutions $(n_1, \ldots, n_m) \in \mathbb{N}^m$, but one may consider also other solution domains than $\mathbb{N}$. It is well known that the *nonnegative-integer solvability problem* ($\mathbb{N}$-solvability problem) of linear equations, i.e. the question whether $\mathcal{U}$ has a nonnegative-integer solution, is NP-complete (for hardness see [13]; NP-membership is a consequence of [21]). The complexity remains the same for other natural variants of this problem, for instance for inequalities instead of equations (a.k.a. integer linear programming). On the other hand, for any $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+\}$, the $\mathbb{X}$-*solvability problem*, i.e., the question whether $\mathcal{U}$ has a solution $(n_1, \ldots, n_m) \in \mathbb{X}^m$, is decidable in PTIME.

The $\mathbb{X}$-solvability problem is equivalently formulated as follows: for a given finite set of coefficient vectors $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\} \subseteq_{\text{fin}} \mathbb{Z}^d$ and a target vector $\mathbf{a} \in \mathbb{Z}^d$ (we use bold fonts to distinguish vectors from other elements), check whether $\mathbf{a}$ is an $\mathbb{X}$-*sum* of A, i.e., a sum of the following form $\mathbf{a} = n_1 \cdot \mathbf{a}_1 + \ldots + n_m \cdot \mathbf{a}_m$ for some $n_1, \ldots, n_m \in \mathbb{X}$. The dimension d corresponds to the number of equations in $\mathcal{U}$.

Linear equations may serve as an over-approximation of the reachability set of a Petri net, or equivalently, of a *vector addition system* – we prefer to work with the latter model. *A vector addition system (VAS)* $\mathcal{A} = (A, \mathbf{i}, \mathbf{f})$ is defined by a finite set of integer vectors $A \subseteq_{\text{fin}} \mathbb{Z}^d$ together with two nonnegative integer vectors $\mathbf{i}, \mathbf{f} \in \mathbb{N}^d$, the initial one and the final one. The set A determines a transition relation $\longrightarrow$ between configurations, that are nonnegative integer vectors $\mathbf{c} \in \mathbb{N}^d$: there is a transition $\mathbf{c} \longrightarrow \mathbf{c}'$ if $\mathbf{c}' = \mathbf{c} + \mathbf{a}$ for some $\mathbf{a} \in A$. The VAS reachability problems asks whether the final configuration is reachable from the initial one by a sequence of transitions (called a *run*), i.e., $\mathbf{i} \longrightarrow^* \mathbf{f}$. We stress that intermediate configurations are required to be nonnegative. The problem is EXPSPACE-hard [19] and decidable [20, 15], but nothing is known about its complexity except for the cubic Ackermann upper bound of [18]. For a given VAS, a necessary condition for $\mathbf{i} \longrightarrow^* \mathbf{f}$ is $\mathbb{N}$-solvability of the system of linear equations defined by the set A and the target vector $\mathbf{a} = \mathbf{f} - \mathbf{i}$, called (in case of Petri nets) the *state equation*. For further details we refer the reader to an exhaustive overview of linear-algebraic approximations for Petri nets [24], where both $\mathbb{N}$- and $\mathbb{Q}_+$-solvability problems are considered.

2.1 Vector addition systems and linear equations, with ordered data

The model of VAS, and linear equations, can be naturally extended with data. In this paper we assume that the data domain $\mathbb{D}$ is a countable set, ordered by a dense total order $\leq$ with no minimal nor maximal element. Thus, up to isomorphism, $(\mathbb{D}, \leq)$ is the set of rational numbers with the natural ordering. We call elements of $\mathbb{D}$ *data values*. In the following we use order preserving permutations (called *data permutations* in short) of $\mathbb{D}$, i.e. bijections $\rho : \mathbb{D} \to \mathbb{D}$ such that $x \leq y$ implies $\rho(x) \leq \rho(y)$.

A *data vector* is a function $\mathbf{v} : \mathbb{D} \to \mathbb{Q}^d$ such that the *support*, i.e. the set $supp(\mathbf{v}) \stackrel{\text{def}}{=} \{\alpha \in \mathbb{D} \mid \mathbf{v}(\alpha) \neq \mathbf{0}\}$, is finite (again, we use bold fonts to distinguish data vectors from other elements). The vector addition is lifted to data vectors pointwise: $(\mathbf{v}+\mathbf{w})(\alpha) \stackrel{\text{def}}{=} \mathbf{v}(\alpha)+\mathbf{w}(\alpha)$. A data vector $\mathbf{v}$ is *nonnegative* if $\mathbf{v} : \mathbb{D} \to (\mathbb{Q}_+)^d$, and $\mathbf{v}$ is *integer* if $\mathbf{v} : \mathbb{D} \to \mathbb{Z}^d$. Writing $\circ$ for function composition, we see that $\mathbf{v} \circ \rho$ is a data vector for any data vector $\mathbf{v}$ and any order preserving data permutation $\rho : \mathbb{D} \to \mathbb{D}$. For a set V of data vectors we define

$$\textsc{Orbit}(V) = \{\mathbf{v} \circ \rho \mid \mathbf{v} \in V, \rho \text{ a data permutation}\}.$$

A data vector $\mathbf{x}$ is said to be a *permutation sum* of a finite set of data vectors V if, for some $\mathbf{v_1}, \ldots, \mathbf{v_m} \in \textsc{Orbit}(V)$, not necessarily pairwise different, $\mathbf{x} = \sum_{i=1}^{m} \mathbf{v}_i$. We investigate the following decision problem:

Permutation sum problem.
Input: a finite set V of integer data vectors and an integer data vector $\mathbf{x}$.
Output: is $\mathbf{x}$ a permutation sum of V?

In the special case when the supports of $\mathbf{x}$ and all vectors in V are all singletons, the Permutation sum problem is just $\mathbb{N}$-solvability of linear equations and thus it is trivially NP-hard.

▶ **Proviso 1.** *For complexity estimations we assume binary encoding of numbers appearing in the input to all decision problems discussed in this paper.*

As the first main result, we prove the following inter-reducibility:

▶ **Theorem 1.** *There is a polynomial-time reduction from the VAS reachability problem to the* Permutation sum problem*, and an exponential-time reduction in the opposite direction.*

As a direct consequence, the Permutation sum problem is decidable and ExpSpace-hard. Our setting generalises the setting of *unordered* data, where the data domain $\mathbb{D}$ is *not* ordered, and hence data permutations are all bijections $\mathbb{D} \to \mathbb{D}$. In the case of unordered data the Permutation sum problem is NP-complete, as shown in [12]. The increase of complexity caused by the order in data is thus remarkable.

Similarly as the state equation in the data-less setting, Permutation sum problem may be used as an overapproximation of the reachability in vector addition systems with ordered data, which are defined exactly as ordinary VAS but in terms of data vectors instead of ordinary vectors. A *VAS with ordered data* $\mathcal{V} = (V, \mathbf{i}, \mathbf{f})$ consists of $V \subseteq_{\text{fin}} \mathbb{D} \to \mathbb{Z}^d$ a finite set of integer data vectors, and the initial and final nonnegative integer data vectors $\mathbf{i}, \mathbf{f} \in \mathbb{D} \to \mathbb{N}^d$. The configurations are nonnegative integer data vectors, and the set V induces a transition relation between configurations as follows: $\mathbf{c} \longrightarrow \mathbf{c}'$ if $\mathbf{c}' = \mathbf{c} + \mathbf{v}$ for some $\mathbf{v} \in \textsc{Orbit}(V)$. Similarly as for plain VAS, the reachability problem asks whether the final configuration is reachable from the initial one by a sequence of transitions (called a *run*), i.e., $\mathbf{i} \longrightarrow^* \mathbf{f}$; but contrarily to plain VAS, the problem is undecidable for VAS with

ordered data [17]. (The decidability status of the problem for VAS with *unordered* data is unknown.) As long as reachability is concerned, VAS with (un)ordered data are equivalent to Petri nets with (un)ordered data [11].

The Permutation sum problem is easily generalised to other domains $\mathbb{X} \subseteq \mathbb{Q}$ of solutions. To this end we introduce scalar multiplication: for $c \in \mathbb{Q}$ and a data vector $\mathbf{v}$ we put $(c \cdot \mathbf{v})(\alpha) \stackrel{\text{def}}{=} c\mathbf{v}(\alpha)$. A data vector $\mathbf{x}$ is said to be a $\mathbb{X}$-*permutation sum* of a finite set of data vectors V if for some $\mathbf{v_1}, \ldots, \mathbf{v_m} \in \text{Orbit}(V)$ and coefficients $n_1, \ldots, n_m \in \mathbb{X}$,

$$\mathbf{x} = n_1 \cdot \mathbf{v_1} + \ldots + n_m \cdot \mathbf{v_m}.$$

This leads to the following version of the problem, parametrized by the choice of solution domain $\mathbb{X}$ (the Permutation sum problem is a particular case, for $\mathbb{X} = \mathbb{N}$):

$\mathbb{X}$-Permutation sum problem.
Input: a finite set V of integer data vectors and an integer data vector $\mathbf{x}$.
Output: is $\mathbf{x}$ an $\mathbb{X}$-permutation sum of V?

Our second main result is the following:

▶ **Theorem 2.** *For any* $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+\}$, *the* $\mathbb{X}$-Permutation sum problem *is in* PTime.

For $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}\}$, the above theorem is a direct consequence of a more general fact, where $\mathbb{Q}$ or $\mathbb{Z}$ is replaced by any commutative ring $\mathbb{R}$, under a proviso that data vectors are defined in a more general way, as finitely supported functions $\mathbb{D} \to \mathbb{R}^d$. With this more general notion, we prove that the $\mathbb{R}$-Permutation sum problem reduces polynomially to the $\mathbb{R}$-solvability of linear equations with coefficients from $\mathbb{R}$ (cf. Theorem 17 in Section 6.2).

The case $\mathbb{X} = \mathbb{Q}_+$ in Theorem 2 is more involved but of particular interest, as it recalls continuous Petri nets [22, 8] where fractional firings of transitions are allowed. Moreover, faced with the high complexity of Theorem 1, it is expected that Theorem 2 may become a cornerstone of linear-algebraic techniques for VAS with ordered data.

3 Lower bound for the Permutation Sum Problem

In this section we assume all data vectors to be integer data vectors. We are going to show a polynomial-time reduction from the VAS reachability problem to the Permutation sum problem. Fix a VAS $\mathcal{A} = (A, \mathbf{i}, \mathbf{f})$. We are going to define a set of data vectors V and a target data vector $\mathbf{x}$ such that the following conditions are equivalent:

C1: $\mathbf{f}$ is reachable from $\mathbf{i}$ in $\mathcal{A}$;

C2: $\mathbf{x}$ is a permutation sum of V.

The set V, to be defined below, will contain only data vectors $\mathbf{v}$ satisfying the following conditions (such data vectors we call *increasing*):

- $\mathbf{v}$ is supported by two data values: $\mathit{supp}(\mathbf{v}) = \{\alpha, \beta\}$ for some data values $\alpha < \beta$;
- $\mathbf{v}(\alpha) \in (-\mathbb{N})^d$ is nonpositive;
- $\mathbf{v}(\beta) \in \mathbb{N}^d$ is nonnegative.

The choice of data values α, β is irrelevant, as we only need to define V up to data permutation. Up to data permutation, the vectors $\mathbf{a} = \mathbf{v}(\alpha)$ and $\mathbf{b} = \mathbf{v}(\beta)$ determine the increasing vector as above uniquely. We thus write $[\mathbf{a}, \mathbf{b}]$ to denote the increasing data vector determined by $\mathbf{a}$ and $\mathbf{b}$ (and some arbitrary but fixed data values $\alpha < \beta$).

Every integer vector $\mathbf{a} \in \mathbb{Z}^d$ is uniquely presented as a sum $\mathbf{a} = \mathbf{a}^- + \mathbf{a}^+$ of a nonnegative vector $\mathbf{a}^+ \in \mathbb{N}^d$ and a nonpositive one $\mathbf{a}^- \in (-\mathbb{N})^d$, defined as follows:

$$\mathbf{a}^+(i) = \begin{cases} \mathbf{a}(i), & \text{if } \mathbf{a}(i) \geq 0 \\ 0, & \text{if } \mathbf{a}(i) < 0 \end{cases} \qquad \mathbf{a}^-(i) = \begin{cases} \mathbf{a}(i), & \text{if } \mathbf{a}(i) \leq 0 \\ 0, & \text{if } \mathbf{a}|(i) > 0. \end{cases}$$

The idea of the reduction is to simulate every vector $\mathbf{a} = \mathbf{a}^- + \mathbf{a}^+ \in A$ by the increasing data vector $[\mathbf{a}^-, \mathbf{a}^+]$, which we call *data realization* of $\mathbf{a}$. In addition, we will need the increasing data vectors of the form $[-\mathbf{1}_i, \mathbf{1}_i]$, where $\mathbf{1}_i \in \mathbb{N}^d$ has 1 on coordinate i and 0 on all other coordinates. We call data vectors $[-\mathbf{1}_i, \mathbf{1}_i]$ *unit increases*. We thus define V as:

$$V \;=\; \{[\mathbf{a}^-, \mathbf{a}^+] \mid \mathbf{a} \in A\} \quad \cup \quad \{[-\mathbf{1}_i, \mathbf{1}_i] \mid i = 1, \ldots, d\}.$$

As the target data vector we take $\mathbf{x} = [-\mathbf{i}, \mathbf{f}]$. It remains to show the equivalence of conditions C1 and C2.

For the proof it will be useful to consider a *VAS with ordered data* $\mathcal{V} = (V, \overline{\mathbf{i}}, \overline{\mathbf{f}})$ (recall the definition in Section 2.1) with the same set of data vectors V, the initial configuration $\overline{\mathbf{i}}$ a data vector supported by one data value, which maps this data value to $\mathbf{i}$, and similarly the final configuration $\overline{\mathbf{f}}$, with the proviso that the singleton support of $\overline{\mathbf{f}}$ is greater than the singleton support of $\overline{\mathbf{i}}$. Clearly, the permutation sum problem overapproximates reachability in $\mathcal{V}$: existence of a run $\overline{\mathbf{i}} \longrightarrow^* \overline{\mathbf{f}}$ in $\mathcal{V}$ implies that $\mathbf{x} = [-\mathbf{i}, \mathbf{f}] = \overline{\mathbf{f}} - \overline{\mathbf{i}}$ is a permutation sum of V. Furthermore, $\mathcal{A}$ also overapproximates $\mathcal{V}$: every run in $\mathcal{V}$ can be transformed into a run in $\mathcal{A}$, by simply getting rid of data in data realizations and dropping all the unit increases.

Condition C1 implies condition C2. Indeed, every run $\mathbf{i} \longrightarrow^* \mathbf{f}$ in $\mathcal{A}$ can be transformed into a run $\overline{\mathbf{i}} \longrightarrow^* \overline{\mathbf{f}}$ in $\mathcal{V}$: replace every vector $\mathbf{a} \in A$ appearing in the former run with its data realization $[\mathbf{a}^-, \mathbf{a}^+] \circ \theta$ (for a suitably chosen data permutation θ), preceded, if necessary, by a number of unit increases of the form $[-\mathbf{1}_i, \mathbf{1}_i] \circ \theta$ (again, for suitably chosen θ), in order to gather, intuitively speaking, the whole vector $\mathbf{a}^-$ at the same data value. Then C2 follows by the overapproximation of reachability in $\mathcal{V}$ by the permutation sum problem.

For the converse implication suppose C2 holds, i.e., $\mathbf{x} = \sum_{i=1}^{n} \mathbf{w}_i$, where $\mathbf{w}_i = \mathbf{v}_i \circ \theta_i$ and $\mathbf{v}_i \in V$. By construction of V, for every $i \leq n$ the data vector $\mathbf{v}_i$ is either a data realization of some $\mathbf{a} \in A$, or a unit increase. Let $\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_l}$ denote the subsequence of $\mathbf{v}_1, \ldots, \mathbf{v}_n$ containing the former ones. We claim that the corresponding vectors $\mathbf{a}_1 \ldots \mathbf{a}_l$, of which $\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_l}$ are data realizations, can be arranged into a sequence being a correct run of the VAS $\mathcal{A}$ from $\mathbf{i}$ to $\mathbf{f}$. For this purpose we define a binary relation of *succession* on data vectors $\mathbf{w}_i$: we say that $\mathbf{w}_j$ succeeds $\mathbf{w}_i$ if $\max(supp(\mathbf{w}_i)) \leq min(supp(\mathbf{w}_j))$. We observe that the succession relation is a partial order – indeed, antisymmetry follows due to the fact that all data vectors $\mathbf{w}_i$ are increasing. Let $\prec$ denote an arbitrary extension of the partial order to a total order, and assume w.l.o.g. that $\mathbf{w}_1 \prec \mathbf{w}_2 \prec \ldots \prec \mathbf{w}_n$. We argue that the corresponding sequence $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l$ of vectors from A is a correct run of the VAS $\mathcal{A}$ from $\mathbf{i}$ to $\mathbf{f}$. As $\mathcal{A}$ overapproximates $\mathcal{V}$, it is enough to demonstrate that the sequence $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$ is a correct run in $\mathcal{V}$ from $\overline{\mathbf{i}}$ to $\overline{\mathbf{f}}$. We thus need to prove that the data vector $\mathbf{u}_i = \overline{\mathbf{i}} + \sum_{i=1}^{j} \mathbf{w}_i$ is nonnegative for every $j \in \{0, \ldots, n\}$. To this aim fix $\alpha \in \mathbb{D}$ and $l \in \{1, \ldots, d\}$, and consider the sequence of numbers

$$\mathbf{u}_0(\alpha, l), \quad \mathbf{u}_1(\alpha, l), \quad \ldots \quad \mathbf{u}_n(\alpha, l) \tag{1}$$

appearing as the value of the consecutive data vectors $\mathbf{u}_0$, $\mathbf{u}_1$, …, $\mathbf{u}_n$ at data value α and coordinate l. We know that the first element of the sequence $\mathbf{u}_0(\alpha, l) = \overline{\mathbf{i}}(\alpha, l) \geq 0$ and the last element of the sequence $\mathbf{u}_n(\alpha, l) = \overline{\mathbf{f}}(\alpha, l) \geq 0$. Furthermore, by the definition of the ordering $\preceq$ we know that the sequence (1) is first non-decreasing, and then non-increasing. These conditions imply nonnegativeness of all numbers in the sequence.

4 Histograms

The purpose of this section is to transform the Permutation sum problem to a more manageable form. As the first step, we eliminate data by rephrasing the problem in terms of matrices (in Lemma 4). Then, we distinguish matrices with certain combinatorial property, called *histograms*. Finally, in Lemma 12 we provide a final characterisation of the problem, using *multihistograms*. The characterisation will be crucial for effectively solving the Permutation sum problem in Section 5.

▶ **Proviso 2.** *In this section, all matrices are integer ones, and all data vectors are integer ones.*

Eliminating data. Matrices with r rows and c columns we call $r\times c$-matrices, and r (resp. c) we call row (resp. column) dimension of an $r\times c$-matrix. We are going to represent a data vector $\mathbf{v} \in \mathbb{D} \to \mathbb{Z}^d$ as a $d\times|supp(\mathbf{v})|$-matrix $M_{\mathbf{v}}$ as follows: if $supp(\mathbf{v}) = \{\alpha_1 < \alpha_2 < \ldots < \alpha_n\}$, we put $M_{\mathbf{v}}(i,j) \stackrel{\text{def}}{=} \mathbf{v}(i)(\alpha_j)$. A *0-extension* of an $r\times c$-matrix M is any $r\times c'$-matrix M', $c' \geq c$, obtained from M by inserting into M arbitrarily $c'-c$ additional zero columns $\mathbf{0} \in \mathbb{Z}^r$. Thus row dimension is preserved by 0-extension, and column dimension may grow arbitrarily. We denote by 0-ext(M) the (infinite) set of all 0-extensions of a matrix M. In particular, $M \in$ 0-ext(M). For a set $\mathcal{M}$ of matrices we denote by 0-ext$(\mathcal{M})$ the set of all 0-extensions of all matrices in $\mathcal{M}$.

▶ **Example 3.** For a data vector $\mathbf{v}$ with $supp(\mathbf{v}) = \{\alpha_1 < \alpha_2\}$, $\mathbf{v}(\alpha_1) = (1,3,0) \in \mathbb{Z}^3$ and $\mathbf{v}(\alpha_2) = (2,0,2) \in \mathbb{Z}^3$, here is the corresponding matrix and two of possible 0-extension:

$$M_{\mathbf{v}} = \begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 2 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \in \text{0-ext}(M_{\mathbf{v}}).$$

Below, whenever we add matrices we silently assume that they have the same row and column dimensions. For a finite set $\mathcal{M}$ of matrices, we say that a matrix N is a sum of 0-extensions of $\mathcal{M}$ if

$$N = M_1 + \ldots + M_m \tag{2}$$

for some matrices $M_1, \ldots, M_m \in$ 0-ext$(\mathcal{M})$, necessarily all of the same row and column dimension. We claim that the Permutation sum problem is equivalent to the question whether some 0-extension of a given matrix X is a sum of 0-extensions of $\mathcal{M}$.

Up to 0-extension sum problem.
Input: a finite set $\mathcal{M}$ of matrices, and a matrix X, all of the same row dimension d.
Output: is some 0-extension of X a sum of 0-extensions of $\mathcal{M}$?

▶ **Lemma 4.** *The* Permutation sum problem *is polynomially time equivalent to the* Up to 0-extension sum problem.

Histograms. From now on we concentrate on solving the Up to 0-extension sum problem. For a matrix H, we write $\sum H(i, 1\ldots j)$ instead of $\sum_{1\leq l\leq j} H(i,l)$. In particular, $\sum H(i, 1\ldots 0) = 0$ by convention. We call an integer matrix nonnegative if it only contains nonnegative integers. Histograms, to be defined now, are an adaptation (a strengthening) of histograms of [12] to ordered data.

▶ **Definition 5.** A nonnegative integer $r \times c$-matrix H we call a *histogram* if the following conditions are satisfied:

- for some $s \geq 0$, called the *degree* of H, for every $1 \leq i \leq r$ we have $\sum H(i, 1 \dots c) = s$;
- for every $1 \leq i < r$ and $0 \leq j < c$, we have $\sum H(i, 1 \dots j) \geq \sum H(i+1, 1 \dots j+1)$.

Note that the zero matrix is a histogram, for $s = 0$. If $s > 0$, the definition enforces $r \leq c$. Histograms of degree 1 are called *simple*. The following combinatorial property of histograms will be crucial in the sequel:

▶ **Lemma 6.** *H is a histogram of degree $s > 0$ if and only if H is a sum of s simple histograms.*

▶ **Example 7.** A histogram of degree 2 may be decomposed as a sum of two simple histograms:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Below, whenever we multiply matrices we assume that the column dimension of the first one is the same as the row dimension of the second one. Simple histograms are useful for characterising 0-extensions:

▶ **Lemma 8.** *For matrices N and M, $N \in 0\text{-}ext(M)$ if and only if $N = M \cdot S$, for a simple histogram S.*

▶ **Example 9.** Recall the matrix $M = M_{\mathbf{v}}$ from Example 3. One of the matrices from $0\text{-ext}(M)$ is presented as the multiplication of M and a simple histogram as follows:

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

We use Lemmas 6 and 8 to characterise the Up to 0-extension sum problem:

▶ **Lemma 10.** *For a matrix N and a finite set of matrices $\mathcal{M}$, the following conditions are equivalent:*

1. *N is a sum of 0-extensions of $\mathcal{M}$;*
2. *$N = \sum_{M \in \mathcal{M}} M \cdot H_M$, for some histograms $\{H_M \mid M \in \mathcal{M}\}$.*

Multihistograms. Using Lemma 10 we are now going to work out our final characterisation of the Up to 0-extension sum problem, as formulated in Lemma 12 below. The characterisation will use the notion of *multihistogram*, which is an indexed family $\mathcal{H} = \{H_1, \dots, H_k\}$ of histograms satisfying Definition 11 below.

We write $H(i, \_)$ and $H(\_, j)$ for the i-th row and the j-th column of a matrix H, respectively. For an indexed family $\{H_1, \dots, H_k\}$ of matrices, its j-th column is defined as the indexed family of j-th columns of respective matrices $\{H_1(\_, j), \dots, H_k(\_, j)\}$.

Fix an input of the Up to 0-extension sum problem: a matrix X and a finite set $\mathcal{M} = \{M_1, \dots, M_k\}$ of matrices, all of the same row dimension d. Let c_l stand for the column dimension of M_l. Relying on Lemma 10, suppose that some $N \in 0\text{-ext}(X)$ and some indexed family $\mathcal{H} = \{H_1, \dots, H_k\}$ of histograms satisfies

$$N = M_1 \cdot H_1 + \dots + M_k \cdot H_k.$$

(The row dimension of every H_l is necessarily c_l.) Boiling down the equation to entries of a single column $N(\_, j) \in \mathbb{Z}^d$ of N we get the system of d linear equations:

$$N(\_, j) \;=\; M_1 \cdot H_1(\_, j) \;+\; \ldots \;+\; M_k \cdot H_k(\_, j) \;=\; \Big[M_1 \mid \ldots \mid M_k\Big] \cdot \begin{bmatrix} H_1(\_, j) \\ \ldots \\ H_k(\_, j) \end{bmatrix}.$$

Therefore, the j-th column of $\mathcal{H}$, treated as a single column vector of length $s = c_1 + \ldots + c_k$, is a nonnegative-integer solution of a system of d linear equations $\mathcal{U}_{\mathcal{M}, N(\_,j)}$ with s unknowns $x_1 \ldots x_s$ of the form:

$$N(\_, j) \quad = \quad \Big[M_1 \mid \ldots \mid M_k\Big] \cdot \begin{bmatrix} x_1 \\ \ldots \\ x_s \end{bmatrix}.$$

Observe that the system $\mathcal{U}_{\mathcal{M}, N(\_,j)}$ depends on $\mathcal{M}$ and $N(\_, j)$ but not on j. For succinctness, for $\mathbf{a} \in \mathbb{Z}^d$ we put $\mathcal{C}_{\mathbf{a}} := \mathbb{N}\text{-sol}(\mathcal{U}_{\mathcal{M},\mathbf{a}})$ to denote the set of all nonnegative integer solutions of $\mathcal{U}_{\mathcal{M},\mathbf{a}}$. Thus every j-th column of $\mathcal{H}$ belongs to $\mathcal{C}_{N(\_,j)}$.

Now recall that $N \in 0\text{-ext}(X)$. Treating $\mathcal{H}$ as a sequence of its column vectors in $\mathbb{N}^s$ we arrive at the following condition:

▶ **Definition 11.** Let *the word* of an indexed family $\mathcal{H} = \{H_1, \ldots, H_k\}$ of histograms be the sequence of its consecutive column vectors. We say that $\mathcal{H}$ is an $(X, \mathcal{M})$-*multihistogram* if its word belongs to the following language (where n is the column dimension of X):

$$(\mathcal{C}_{\mathbf{0}})^* \; \mathcal{C}_{X(\_,1)} \; (\mathcal{C}_{\mathbf{0}})^* \; \mathcal{C}_{X(\_,2)} \;\; \ldots \;\; (\mathcal{C}_{\mathbf{0}})^* \; \mathcal{C}_{X(\_,n)} \; (\mathcal{C}_{\mathbf{0}})^*. \tag{3}$$

We have just shown existence of an $(X, \mathcal{M})$-*multihistogram* whenever some 0-extension N of X is a sum of 0-extensions of $\mathcal{M}$. As the reasoning above is reversible, we obtain:

▶ **Lemma 12.** *The* Up to 0-extension sum problem *is equivalent to the following one:*

Multihistogram problem.
Input: *a finite set $\mathcal{M}$ of matrices and a matrix X, all of the same row dimension d.*
Output: *does there exist an $(X, \mathcal{M})$-multihistogram?*

5 Upper bound for the Permutation Sum Problem

We reduce in this section the Multihistogram problem (and hence also the Permutation sum problem, due to Lemmas 4 and 12) to the VAS reachability problem (with single exponential blowup), thus obtaining decidability. Fix in this section an input to the Multihistogram problem: an integer matrix X (of column dimension n) and a finite set $\mathcal{M} = \{M_1, \ldots, M_k\}$ of integer matrices, all of the same row dimension d. We perform the reduction in two steps: we start by proving an effective exponential bound on vectors appearing as columns of $(X, \mathcal{M})$-multihistograms; then we construct a VAS whose runs correspond to the words of exponentially bounded $(X, \mathcal{M})$-multihistograms.

Exponentially bounded multihistograms. First, we need to recall a characterisation of nonnegative-integer solution sets of systems of linear equations as exponentially bounded hybrid-linear sets, i.e., of the form $B + P^{\oplus}$, for $B, P \subseteq \mathbb{N}^k$, where k is the number of variables and $P^{\oplus}$ stands for the set of all finite sums of vectors from P (see e.g. [6, 7, 21]). We denote

system of linear equations determined by a matrix M and a column vector $\mathbf{a}$ by $\mathcal{U}_{M,\mathbf{a}}$ and the corresponding *homogeneous* systems of linear equations by $\mathcal{U}_{M,\mathbf{0}}$. Again, for the size $|\mathcal{U}_{M,\mathbf{a}}|$ of $\mathcal{U}_{M,\mathbf{a}}$ we assume that numbers in M and $\mathbf{a}$ are encoded in binary.

▶ **Lemma 13** ([6] Prop. 2). *$\mathbb{N}$-sol$(\mathcal{U}_{M,\mathbf{a}}) = B + P^{\oplus}$, where $B, P \subseteq \mathbb{N}^k$ such that all vectors in $B \cup P$ are bounded exponentially w.r.t. $|\mathcal{U}_{M,\mathbf{a}}|$ and $P \subseteq \mathbb{N}$-sol$(\mathcal{U}_{M,\mathbf{0}})$.*

We will use Lemma 13 together with the following operation on multihistograms. A *j-smear* of a histogram H is any nonnegative matrix H' obtained by replacing j-th column $H(\_, j)$ of H by two columns that sum up to $H(\_, j)$. Here is an example ($j = 5$):

$$\begin{bmatrix} 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 2 \end{bmatrix} \to \begin{bmatrix} 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 \end{bmatrix}.$$

Formally, a j-smear of H is any nonnegative matrix H' satisfying:

$$\begin{aligned} H'(\_, l) &= H(\_, l) \quad \text{for } l < j & \qquad H'(\_, j) + H'(\_, j+1) &= H(\_, j) \\ H'(\_, l+1) &= H(\_, l) \quad \text{for } l > j. \end{aligned}$$

One easily verifies that a smear preserves the defining condition of the histogram:

▶ **Claim 5.1.** *A smear of a histogram is a histogram.*

Finally, a j-smear of a family of matrices $\{H_1, \ldots, H_k\}$ is any indexed family of matrices $\{H'_1, \ldots, H'_k\}$ obtained by applying a j-smear simultaneously to all matrices H_l. We omit the index j when it is irrelevant.

So prepared, we claim that every $(X, \mathcal{M})$-multihistogram $\mathcal{H} = \{H_1, \ldots, H_k\}$ can be transformed by a number of smears into an $(X, \mathcal{M})$-multihistogram containing only numbers exponentially bounded with respect to the sizes of X, $\mathcal{M}$. Indeed, recall (3) and suppose that $N = M_1 \cdot H_1 + \ldots + M_k \cdot H_k \in 0\text{-ext}(X)$. Take an arbitrary (say j-th) column $\mathbf{w} \in \mathcal{C}_{\mathbf{a}} = \mathbb{N}\text{-sol}(\mathcal{U}_{\mathcal{M},\mathbf{a}})$ of $\mathcal{H}$, where $\mathbf{a} = N(\_, j)$, treated as a single column vector $\mathbf{w} \in \mathbb{N}^s$ (for s the sum of row dimensions of $H_1, \ldots, H_k$), and present it (using Lemma 13) as a sum $\mathbf{w} = \mathbf{b} + \mathbf{p}_1 + \ldots + \mathbf{p}_m$, for some exponentially bounded $\mathbf{b} \in \mathcal{C}_{\mathbf{a}}$ and $\mathbf{p}_1, \ldots, \mathbf{p}_m \in \mathcal{C}_{\mathbf{0}}$. Apply smear m times, replacing the j-th column by $m+1$ columns $\mathbf{b}, \mathbf{p}_1, \ldots, \mathbf{p}_m$. As $\mathbf{b}$ is a solution of the system $\mathcal{U}_{\mathcal{M},\mathbf{a}}$ and every $\mathbf{p}_l$ is a solution of the homogeneous system $\mathcal{U}_{\mathcal{M},\mathbf{0}}$,

$$\Big[M_1 \mid \ldots \mid M_k\Big] \cdot \mathbf{b} = \Big[M_1 \mid \ldots \mid M_k\Big] \cdot \mathbf{w} \qquad \Big[M_1 \mid \ldots \mid M_k\Big] \cdot \mathbf{p}_l = \mathbf{0},$$

the family $\mathcal{H}' = \{H'_1, \ldots, H'_k\}$ obtained in the same way still satisfies the condition $M_1 \cdot H'_1 + \ldots + M_k \cdot H'_k \in 0\text{-ext}(X)$. Using Claim 5.1 we deduce that $\mathcal{H}'$ is an $(X, \mathcal{M})$-multihistogram. Repeating the same operation for every column of $\mathcal{H}$ yields the required exponential bound.

Construction of a VAS. Given X and $\mathcal{M}$ we now construct a VAS whose runs correspond to the words of exponentially bounded $(X, \mathcal{M})$-multihistograms. Think of the VAS as reading (or nondeterministically guessing) consecutive column vectors (i.e., the word) of a potential $(X, \mathcal{M})$-multihistogram $\mathcal{H} = \{H_1, \ldots, H_k\}$. The VAS has to check two conditions:

(A) the word of $\mathcal{H}$ belongs to the language (3);

(B) the matrices $H_1, \ldots, H_k$ satisfy the histogram condition (cf. Definition 5).

The first condition, under the exponential bound proved above, amounts to the membership in a regular language and can be imposed by a VAS in a standard way. The second

condition is a conjunction of k histogram conditions, and again the conjunction can be realised in a standard way. We thus focus, from now on, only on showing that a VAS can check that its input is a histogram.

To this aim it will be profitable to have the following characterisation of histograms. For an arbitrary $r\times c$-matrix H, define the $(r-1)\times c$-matrix Δ_H:

$$\Delta_H(i,j+1) \quad \stackrel{\text{def}}{=} \quad \sum H(i,1\ldots j) - \sum H(i+1,1\ldots j+1).$$

Intuitively, Δ_H represents the excess in the second condition in Definition 5. Moreover, consider the $(r-1)\times c$-matrix $(\overline{H}+\Delta_H)$, where $\overline{H}$ is H with the last row truncated.

▶ **Lemma 14.** *A nonnegative $r\times c$-matrix H is a histogram if and only if Δ_H is nonnegative and $(\overline{H}+\Delta_H)(\_,c)=\mathbf{0}$.*

Proof. Indeed, nonnegativeness of Δ_H is equivalent to saying that

$$\sum H(i,1\ldots j) \;\geq\; \sum H(i+1,1\ldots j+1)$$

for every $1\leq i<r$ and $0\leq j<c$; moreover, $(\overline{H}+\Delta_H)(\_,c)=\mathbf{0}$ is equivalent to saying that $\sum H(i,1\ldots c)$ is the same for every $i=1,\ldots,r$. ◀

For the construction of a VAS it is important to note that every two consecutive entries $(\overline{H}+\Delta_H)(i,j-1)$ and $(\overline{H}+\Delta_H)(i,j)$ are related by the following formula:

$$(\overline{H}+\Delta_H)(i,j) \;=\; (\overline{H}+\Delta_H)(i,j-1) - H(i+1,j) + H(i,j). \tag{4}$$

Let r be the row dimension of the input matrix, and let $\mathcal{C}\subseteq\mathbb{N}^r$ denote the exponential set of all column vectors that can appear in a histogram, as derived above. We define a VAS of dimension $2(r-1)$ that reads consecutive columns of an exponentially bounded matrix H and accepts if and only if the matrix is a histogram. The VAS transitions will obey the following invariant: after j steps,

$$\text{counter}_i + \text{counter}_{r-1+i} \;=\; (\overline{H}+\Delta_H)(i,j), \qquad \text{for } i=1,\ldots,r-1. \tag{5}$$

The counters are initially set to 0. Informally, in its j-th step, the VAS will subtract $H(i+1,j)$ from the counter $(r-1)+i$ and simultaneously add $H(i,j)$ to the counter i, for $i=1\ldots r-1$, in accordance with (4); due to the duplication of counters, by sole nonnegativeness of every counter $(r-1)+i$ the VAS will thereby check that $\Delta_H(i,j+1)\geq 0$. Formally, for every vector $\mathbf{w}=(w_1,\ldots,w_r)\in\mathcal{C}$, the VAS has a 'reading' transition that adds $(w_1,\ldots,w_{r-1})\in\mathbb{N}^{r-1}$ to its counters $1,\ldots,r-1$, and subtracts $(w_2,\ldots,w_r)\in\mathbb{N}^{r-1}$ from its counters $r,\ldots,2(r-1)$ (think of $\mathbf{w}(i)=H(i,j)$ in the equation (4)). Furthermore, for every $i=1,\ldots,r-1$ the VAS has a 'moving' transition that subtracts 1 from counter i and adds 1 to counter $r-1+i$. Observe that these transitions preserve the invariant (5).

Relying on Lemma 14 we claim that the VAS defined in this way reaches nontrivially (i.e., along a nonempty run) the zero configuration (all counters equal 0) iff its input H is a histogram with all entries belonging to $\mathcal{C}$. In one direction, the nonnegativeness of counters $r\ldots 2(r-1)$ (as discussed above) assures that Δ_H is nonnegative; and the invariant (5) together with the final zero configuration assures that $(\overline{H}+\Delta_H)(\_,c)=\mathbf{0}$. In the opposite direction, if the VAS inputs a histogram, it has a run ending in the zero configuration. The VAS is computable in exponential time (as the set $\mathcal{C}$ above can be computed in exponential time).

Thus, given X and $\mathcal{M}$ one can effectively (in exponential time) build a VAS that admits reachability if and only if there exists an $(X,\mathcal{M})$-multihistogram.

6 PTime decision procedures

In this section we prove Theorem 2, namely we provide polynomial-time decision procedures for the $\mathbb{X}$-Permutation sum problem, where $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+\}$. The most interesting case $\mathbb{X} = \mathbb{Q}_+$ is treated in Section 6.1. The remaining ones are in fact special cases of a more general result, shown in Section 6.2, that applies to an arbitrary commutative ring.

6.1 $\mathbb{X} = \mathbb{Q}_+$

We start by noticing that the whole development of (multi-)histograms in Section 4 is not at all specific for $\mathbb{X} = \mathbb{N}$ and works equally well for $\mathbb{X} = \mathbb{Q}_+$. First, one adapts the Up to 0-extension sum problem and considers a sum of 0-extensions of $\mathcal{M}$ *multiplied by nonnegative rationals*. Accordingly, one relaxes the definition of histogram: instead of a nonnegative integer matrix, let histogram be now a *nonnegative rational* matrix satisfying exactly the same conditions as in Definition 5 in Section 4. In particular, the degree of a histogram is now a nonnegative rational, and simple histograms are these with exactly one nonzero entry in every row. The same relaxation as for histograms we apply to multihistograms, and in the definition of the latter (cf. the language (3) at the end of Section 4) we consider nonnegative-rational solutions of linear equations instead of nonnegative-integer ones. With these adaptations, the $\mathbb{Q}_+$-Permutation sum problem is equivalent to the following decision problem (whenever a risk of confusion arises, we specify explicitly which matrices are integer ones, and which rational ones):

$\mathbb{Q}_+$-Multihistogram problem.
Input: a finite set $\mathcal{M}$ of integer matrices, and an integer matrix X, all of the same row dimension d.
Output: does there exist a rational $(X, \mathcal{M})$-multihistogram?

From now on we concentrate on the polynomial-time decision procedure for this problem. We proceed in two steps. First, we define *homogeneous linear Petri nets*, a variant of Petri nets generalising continuous PN [22], and show how to solve its reachability problem using $\mathbb{Q}_+$-solvability of a slight generalisation of linear equations (linear equations with implications), following the approach of [8]. Next, using a similar construction as in Section 5, combined with the above characterisation of reachability, we encode the $\mathbb{Q}_+$-Multihistogram problem by systems of linear equations with implications.

Homogeneous linear Petri nets. A *homogeneous linear Petri net* (homogeneous linear PN) of dimension d is a finite set of homogeneous[3] systems of linear equations $\mathcal{V} = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$, called *transition rules*, all over the same $2d$ variables $x_1, \ldots, x_{2d}$. The transition rules determine a transition relation $\longrightarrow$ between configurations, which are nonnegative rational vectors $\mathbf{c} \in (\mathbb{Q}_+)^d$, as follows: there is a transition $\mathbf{c} \longrightarrow \mathbf{c}'$ if, for some $i \in \{1, \ldots, m\}$ and $\mathbf{v} \in \mathbb{Q}_+\text{-sol}(\mathcal{U}_i)$, the vector $\mathbf{c} - \pi_{1\ldots d}(\mathbf{v})$ is still nonnegative, and

$$\mathbf{c}' = \mathbf{c} - \pi_{1\ldots d}(\mathbf{v}) + \pi_{d+1\ldots 2d}(\mathbf{v}).$$

(The vectors $\pi_{1\ldots d}(\mathbf{v})$ and $\pi_{d+1\ldots 2d}(\mathbf{v})$ are projections of $\mathbf{v}$ on respective coordinates.) The binary reachability relation $\mathbf{c} \longrightarrow^* \mathbf{c}'$ holds, if there is a sequence of transitions from $\mathbf{c}$ to $\mathbf{c}'$.

[3] If non-homogeneous systems were allowed, the model would subsume (ordinary) Petri nets.

A class of *continuous PN* [22] can be seen as a subclass of homogeneous linear PN, where every system $\mathcal{U}_i$ has 1-dimensional solution set of the form $\{c\mathbf{v} \mid c \in \mathbb{Q}_+\}$, for some fixed $\mathbf{v} \in \mathbb{N}^{2d}$.

Linear equations with implications. A $\Rightarrow$*-system* is a finite set of linear equations, all over the same variables, plus a finite set of implications of the form $x > 0 \implies y > 0$, where x, y are variables appearing in the linear equations. The solutions of a $\Rightarrow$-system are defined as usually, but additionally they must satisfy all implications. The $\mathbb{Q}_+$-solvability problem asks if there is a nonnegative-rational solution. In [8] (Algorithm 2) and also in [2] (where a PTIME fragment of existential FO($\mathbb{Q}$, + ,<) has been identified that captures $\Rightarrow$-system), it has been shown (within a different notation) how to solve the problem in PTIME:

▶ **Lemma 15** ([8, 2])**.** *The $\mathbb{Q}_+$-solvability problem for $\Rightarrow$-systems is decidable in* PTIME.

Due to [8], the reachability problem for continuous PNs reduces to the $\mathbb{Q}_+$-solvability of $\Rightarrow$-systems. We generalise this result and prove the reachability relation of a homogeneous linear PN to be effectively described by a $\Rightarrow$-system:

▶ **Lemma 16.** *Given a homogeneous linear PN $\mathcal{V}$ of dimension d one can compute in* PTIME *a $\Rightarrow$-system whose $\mathbb{Q}_+$-solution set, projected onto a subset of $2d$ variables, describes the binary reachability relation of $\mathcal{V}$.*

Polynomial-time decision procedure. Now, we are ready to sketch out a decision procedure for the $\mathbb{Q}_+$-MULTIHISTOGRAM PROBLEM, by a polynomial-time reduction to the $\mathbb{Q}_+$-solvability problem of $\Rightarrow$-systems.

Fix an input to the $\mathbb{Q}_+$-MULTIHISTOGRAM PROBLEM, i.e., X and $\mathcal{M} = \{M_1, \ldots, M_k\}$. As in Section 4, for $\mathbf{a} \in \mathbb{Z}^d$ we denote the solution set of a system $\mathcal{U}_{\mathcal{M},\mathbf{a}}$ of linear equations determined by the matrix $\left[M_1 \mid \ldots \mid M_k\right]$ and the column vector $\mathbf{a}$ by $\mathcal{C}_\mathbf{a}$; but this time we care about *nonnegative-rational* solutions. We thus put $\mathcal{C}_\mathbf{a} := \mathbb{Q}_+\text{-sol}(\mathcal{U}_{\mathcal{M},\mathbf{a}}) \subseteq (\mathbb{Q}_+)^r$. Recall the language (3). Our aim is to check existence of a rational $(X, \mathcal{M})$-multihistogram, i.e., of a family $\mathcal{H} = \{H_1, \ldots, H_k\}$ of nonnegative rational matrices, such that the following conditions are satisfied:

(A) the word of $\mathcal{H}$ belongs to the language (3) (interpreted in nonnegative rationals);
(B) the matrices $H_1, \ldots, H_k$ satisfy the histogram condition.

We construct in polynomial time a $\Rightarrow$-system $\mathcal{S}$ that is solvable if and only if conditions (A) and (B) are met. The solvability of $\mathcal{S}$ itself is decidable in PTIME according to Lemma 15. The idea is to characterise conditions (A)–(B) by a sequence of runs in a homogeneous linear PN interleaved by single steps described by non-homogeneous systems of linear equations (where n is the column dimension of X):

$$\mathbf{0} \xrightarrow{\mathcal{C}_\mathbf{0}^*} \mathbf{c}_1 \xrightarrow{\mathcal{C}_{X(\_,1)}} \mathbf{c}_2 \xrightarrow{\mathcal{C}_\mathbf{0}^*} \mathbf{c}_3 \xrightarrow{\mathcal{C}_{X(\_,2)}} \ldots \xrightarrow{\mathcal{C}_\mathbf{0}^*} \mathbf{c}_{2n-2} \xrightarrow{\mathcal{C}_\mathbf{0}^*} \mathbf{c}_{2n-1} \xrightarrow{\mathcal{C}_{X(\_,n)}} \mathbf{c}_{2n} \xrightarrow{\mathcal{C}_\mathbf{0}^*} \mathbf{0}.$$

Conceptually, the construction follows the construction of a VAS in Section 5. We define a homogeneous linear PN $\mathcal{V}_\mathbf{0}$, recognizing the language $(\mathcal{C}_\mathbf{0})^*$ and, using Lemma 16, we compute in PTIME a $\Rightarrow$-system $\mathcal{S}_\mathbf{0}$ such that the projection $P_\mathbf{0}$ of $\mathbb{Q}_+$-sol($\mathcal{S}_\mathbf{0}$) to some of its variables describes the reachability relation of $\mathcal{V}_\mathbf{0}$. Ignoring some technical details, the final $\Rightarrow$-system $\mathcal{S}$ imposes the following constraints (for all j):

1. there is a run from $\mathbf{c}_{2j}$ to $\mathbf{c}_{2j+1}$ in $\mathcal{V}_\mathbf{0}$, i.e., $(\mathbf{c}_{2j}, \mathbf{c}_{2j+1}) \in P_\mathbf{0}$;
2. $\mathbf{c}_{2j} - \mathbf{c}_{2j-1} \in \mathcal{C}_{X(\_,j)} = \mathbb{Q}_+\text{-sol}(\mathcal{U}_{\mathcal{M},X(\_,j)})$.

Now, $\mathcal{S}$ is solvable iff some rational $(X, \mathcal{M})$-multihistogram exists.

6.2 $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}\}$

In this, and only in this section we generalise slightly our setting and consider a fixed commutative ring $\mathbb{R}$, instead of just the ring of integers $\mathbb{Z}$ or rationals $\mathbb{Q}$. Accordingly, by a data vector we mean in this section a function $\mathbb{D} \to \mathbb{R}^d$ from data values to d-tuples of elements of $\mathbb{R}$ that maps almost all data values (i.e. all except for a finite number of data values) to the zero vector $\mathbf{0} \in \mathbb{R}^d$. With this more general notion of data vectors, we define $\mathbb{R}$-permutation sums and the $\mathbb{R}$-Permutation sum problem analogously as in Section 2.1. Furthermore, we define analogously $\mathbb{R}$-sums and consider linear equations with coefficients from $\mathbb{R}$ and their $\mathbb{R}$-solvability problem.

▶ **Theorem 17.** *For any commutative ring $\mathbb{R}$, the $\mathbb{R}$-*Permutation sum problem *reduces polynomially to the $\mathbb{R}$-solvability problem of linear equations.*

Clearly, Theorem 17 implies the remaining cases of Theorem 2, namely $\mathbb{X} \in \{\mathbb{Z}, \mathbb{Q}\}$, as in these cases the $\mathbb{X}$-solvability of linear equations is in PTime. Theorem 17 follows immediately by Lemma 18 stated below. For a data vector $\mathbf{v}$, we define the vector $\textsc{sum}(\mathbf{v}) \in \mathbb{R}^d$ and a finite set of vectors $\textsc{vectors}(\mathbf{v}) \subseteq_{\text{fin}} \mathbb{R}^d$:

$$\textsc{sum}(\mathbf{v}) \stackrel{\text{def}}{=} \sum_{\alpha \in \mathit{supp}(\mathbf{v})} \mathbf{v}(\alpha) \qquad \textsc{vectors}(\mathbf{v}) \stackrel{\text{def}}{=} \{\mathbf{v}(\alpha) \mid \alpha \in \mathit{supp}(\mathbf{v})\}.$$

Clearly both operations commute with data permutations: $\textsc{sum}(\mathbf{v}) = \textsc{sum}(\mathbf{v} \circ \theta)$ and $\textsc{vectors}(\mathbf{v}) = \textsc{vectors}(\mathbf{v} \circ \theta)$, and can be lifted naturally to finite sets of data vectors:

$$\textsc{sum}(V) \stackrel{\text{def}}{=} \{\textsc{sum}(\mathbf{v}) \mid \mathbf{v} \in V\} \qquad \textsc{vectors}(V) \stackrel{\text{def}}{=} \bigcup_{\mathbf{v} \in V} \textsc{vectors}(\mathbf{v}).$$

▶ **Lemma 18.** *Let $\mathbf{x}$ be a data vector and V be a finite set of data vectors V. Then $\mathbf{x}$ is an $\mathbb{R}$-permutation sum of V if and only if*

1. $\textsc{sum}(\mathbf{x})$ *is an $\mathbb{R}$-sum of* $\textsc{sum}(V)$*, and*
2. *every* $\mathbf{a} \in \textsc{vectors}(\mathbf{x})$ *is an $\mathbb{R}$-sum of* $\textsc{vectors}(V)$.

Proof. The proof is inspired by Theorem 15 in [12]. The only if direction is immediate: if $\mathbf{x} = z_1 \cdot \mathbf{w}_1 + \ldots + z_n \cdot \mathbf{w}_n$ for $z_1, \ldots, z_n \in \mathbb{R}$ and $\mathbf{w}_1, \ldots, \mathbf{w}_n \in \textsc{Orbit}(V)$, then clearly $\textsc{sum}(\mathbf{x}) = z_1 \cdot \textsc{sum}(\mathbf{w}_1) + \ldots + z_n \cdot \textsc{sum}(\mathbf{w}_n)$ and hence $\textsc{sum}(\mathbf{x})$ is a $\mathbb{R}$-sum of $\textsc{sum}(V)$ (using the fact that $\textsc{sum}(\_)$ commutes with data permutations). Also $\mathbf{x}(\alpha)$ is necessarily an $\mathbb{R}$-sum of $\textsc{vectors}(V)$ for every $\alpha \in \mathit{supp}(\mathbf{x})$.

Now we focus on the if direction. For a vector $\mathbf{a} \in \mathbb{R}^d$, we define an $\mathbf{a}$-*move* as an arbitrary data vector that maps some data value to $\mathbf{a}$, some other data value to $-\mathbf{a}$, and all other data values to $\mathbf{0}$.

▶ **Claim 6.1.** *Every $\mathbf{a}$-move, for $\mathbf{a} \in \textsc{vectors}(\mathbf{v})$, is an $\mathbb{R}$-permutation sum of $\{\mathbf{v}\}$.*

Indeed, for $\mathbf{a} = \mathbf{v}(\alpha)$, consider a data permutation θ that preserves all elements of $\mathit{supp}(\mathbf{v})$ except that it maps α to a data value α' related in the same way as α by the order $\leq$ to all other data values in $\mathit{supp}(\mathbf{v})$. Then $\mathbf{a}$-moves are exactly data vectors $(\mathbf{v} - \mathbf{v} \circ \theta) \circ \rho = \mathbf{v} \circ \rho - \mathbf{v} \circ (\theta \circ \rho)$.

For the if direction, suppose point 1. holds: $\textsc{sum}(\mathbf{x})$ is an $\mathbb{R}$-sum of $\textsc{sum}(V)$. Treat the vector $\textsc{sum}(\mathbf{x})$ and the vectors in $\textsc{sum}(V)$ as data vectors with the same singleton support. Observe that $\textsc{sum}(\mathbf{v})$ for any $\mathbf{v} \in V$ is an $\mathbb{R}$-permutation sum of $\{\mathbf{v}\}$; indeed, by Claim 6.1 we can use $\mathbf{a}$-moves to transfer all nonzero vectors for data in $\mathit{supp}(\mathbf{v})$ into one datum. With this view in mind we have:

- $\textsc{sum}(\mathbf{x})$ is an $\mathbb{R}$-permutation sum of V.

Furthermore, suppose point 2. holds: every $\mathbf{a} \in \text{VECTORS}(\mathbf{x})$ is an $\mathbb{R}$-sum of $\text{VECTORS}(V)$. Thus every $\mathbf{a}$-move, for $\mathbf{a} \in \text{VECTORS}(\mathbf{x})$, is an $\mathbb{R}$-sum of $\{\mathbf{b}\text{-move} \mid \mathbf{b} \in \text{VECTORS}(V)\}$. By Claim 6.1 we know that every element of the latter set is an $\mathbb{R}$-permutation sum of V. Thus we entail:

- every $\mathbf{a}$-move, for $\mathbf{a} \in \text{VECTORS}(\mathbf{x})$, is an $\mathbb{R}$-permutation sum of V.

We have shown that $\text{SUM}(\mathbf{x})$, as well as all $\mathbf{a}$-moves (for all $\mathbf{a} \in \text{VECTORS}(\mathbf{x})$), are $\mathbb{R}$-permutation sums of V. We use the $\mathbf{a}$-moves to transform $\text{SUM}(\mathbf{x})$ into $\mathbf{x}$. This proves that $\mathbf{x}$ is an $\mathbb{R}$-permutation sum of V as required. ◀

7 Concluding remarks

The main result of this paper is determining the computational complexity of solving linear equations with integer (or rational) coefficients, in the setting of ordered data. We observed the huge gap: while the $\mathbb{N}$-solvability problem is equivalent (up to an exponential blowup) to the VAS reachability problem, the $\mathbb{Z}$-, $\mathbb{Q}$-, and $\mathbb{Q}_+$-solvability problems are all in PTIME. This has a consequence for possible linear-algebraic overapproximations of the reachability in VAS with ordered data: instead of $\mathbb{N}$-solvability, one should apply $\mathbb{Z}$- or $\mathbb{Q}_+$-solvability, or even the combination of both.

Except for the last Section 6.2, the coefficients and solutions are assumed to belong to the ring $\mathbb{Q}$ of rationals, but clearly one can consider other commutative rings as well. There is another possible axis of generalisation, namely orbit-finite systems of linear equations over an orbit-finite set of variables, which can be introduced as follows. Fix an arbitrary commutative ring $\mathbb{R}$ and an arbitrary data domain $\mathbb{D}$. Consider *orbit-finite* sets (see, e.g., [3, 4]), i.e., sets that are finite up to the natural action of data automorphisms of $\mathbb{D}$. For instance, in case of the ordered data domain $\mathbb{D}$, the natural action of a monotonic bijection $\theta : \mathbb{D} \to \mathbb{D}$ maps a pair $(d, i) \in \mathbb{D} \times \{1, \ldots, d\}$ to $(\theta(d), i)$; and maps a data vector $\mathbf{v}$ to $\mathbf{v} \circ \theta^{-1}$. Therefore $\mathbb{D} \times \{1, \ldots, d\}$ is orbit-finite (the number of orbits is d) and $\text{ORBIT}(V)$ is orbit-finite whenever V is finite (the number of orbits is at most the cardinality of V). For an orbit-finite set $\mathcal{Y}$, by an *$\mathcal{Y}$-vector* we mean (think of $\mathcal{Y} = \mathbb{D} \times \{1, \ldots, d\}$) any function $\mathcal{Y} \to \mathbb{R}$ that maps almost all elements of $\mathcal{Y}$ to $0 \in \mathbb{R}$; let $\mathbb{R}^{\mathcal{Y}}$ be the set of all $\mathcal{Y}$-vectors. A $\mathcal{Y}$-matrix is an orbit-finite family of (column) $\mathcal{Y}$-vectors, $\mathcal{M} \subseteq_{\text{orbit-finite}} \mathbb{R}^{\mathcal{Y}}$. Such a $\mathcal{Y}$-matrix $\mathcal{M}$, together with a (column) $\mathcal{Y}$-vector $\mathbf{a}$, determines a system of linear equations $\mathcal{U}_{\mathcal{M},\mathbf{a}}$, whose solutions are those $\mathcal{M}$-vectors that, treated as coefficients of a linear combination of vectors $m \in \mathcal{M}$, yield $\mathbf{a} \in \mathcal{Y}$:

$$\text{sol}(\mathcal{U}_{M,\mathbf{a}}) \quad = \quad \{\mathbf{v} \in \mathbb{R}^{\mathcal{M}} \mid \sum_{m \in \mathcal{M}} \mathbf{v}(m) \cdot m = \mathbf{a}\}.$$

Note that the sum is well defined as $\mathbf{v}(m) \neq 0$ for only finitely many elements $m \in \mathcal{M}$. The setting of this paper is nothing but a special case, where $\mathbb{R} = \mathbb{Q}$ and $\mathcal{Y} = \mathbb{D} \times \{1, \ldots, d\}$ and $\mathcal{M} = \text{ORBIT}(V)$ for a finite set V of data vectors. Similarly, another special case has been investigated in [12], where finiteness up to the natural action of automorphisms of the data domain $(\mathbb{D}, =)$ played a similar role. As another example, in [14] the solvability problem has been investigated (in the framework of CSP) for the same data domain $(\mathbb{D}, =)$, in the case where $\mathbb{R}$ is a finite field.

It is an exciting research challenge to fully understand the complexity landscape of orbit-finite systems of linear equations, as a function of the choice of data domain. The results of this paper are a step towards this goal, and indicate that development of the uniform theory will be hard: the case of ordered data, compared to the case of unordered data investigated in [12], requires significantly new techniques and the complexity of the

nonnegative integer solvability differs significantly too. Even more broadly, investigation of orbit-finite dimensional linear algebra, together with its possible applications in the analysis of data-enriched systems, seems to be a tempting continuation of this work.

References

1 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *Proc. TACAS 2016*, pages 480–496, 2016. doi:10.1007/978-3-662-49674-9_28.

2 Michael Blondin and Christoph Haase. Logics for continuous reachability in Petri nets and vector addition systems with states. In *Proc. LICS 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005068.

3 Mikołaj Bojańczyk. Slightly infinite sets. A draft of a book available at https://www.mimuw.edu.pl/~bojan/paper/atom-book. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

4 Mikołaj Bojańczyk, Bartek Klin, Slawomir Lasota, and Szymon Toruńczyk. Turing machines with atoms. In *Proc. LICS 2013*, pages 183–192, 2013. doi:10.1109/LICS.2013.24.

5 Rémi Bonnet, Alain Finkel, Serge Haddad, and Fernando Rosa-Velardo. Comparing Petri data nets and timed Petri nets. Technical Report LSV-10-23, Laboratoire Spécification et Vérification, ENS Cachan, France, 2010.

6 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *Proc. ICALP'16*, pages 128:1–128:13, 2016.

7 Eric Domenjoud. Solving systems of linear Diophantine equations: An algebraic approach. In *Proc. MFCS 1991*, pages 141–150, 1991.

8 Estíbaliz Fraca and Serge Haddad. Complexity analysis of continuous Petri nets. *Fundam. Inform.*, 137(1):1–28, 2015. doi:10.3233/FI-2015-1168.

9 Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. Occam's razor applied to the Petri net coverability problem. In *Proc. Reachability Problems 2016*, pages 77–89, 2016. doi:10.1007/978-3-319-45994-3_6.

10 Piotr Hofman and Sławomir Lasota. Linear equations with ordered data. *CoRR*, arXiv:1802.06660, 2018. URL: http://arxiv.org/abs/1802.06660.

11 Piotr Hofman, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability Trees for Petri Nets with Unordered Data. In *Proc. FoSSaCS*, Eindhoven, Netherlands, 2016. URL: https://hal.inria.fr/hal-01252674.

12 Piotr Hofman, Jérôme Leroux, and Patrick Totzke. Linear combinations of unordered data vectors. In *Proc. LICS 2017*, pages 1–11, 2017.

13 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. Complexity of Computer Computations*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf.

14 Bartek Klin, Eryk Kopczyński, Joanna Ochremiak, and Szymon Toruńczyk. Locally finite constraint satisfaction problems. In *Proc. LICS 2015*, pages 475–486, 2015. doi:10.1109/LICS.2015.51.

15 S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC'82*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

16 Slawomir Lasota. Decidability border for Petri nets with data: WQO dichotomy conjecture. In *Proc. PETRI NETS 2016*, pages 20–36, 2016. doi:10.1007/978-3-319-39086-4_3.

17 Ranko Lazic, Thomas Christopher Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. Nets with tokens which carry data. *Fundam. Inform.*, 88(3):251–274, 2008.

18 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proc. LICS 2015*, pages 56–67, 2015. doi:10.1109/LICS.2015.16.

19 Richard Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.

20 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. STOC 1981*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

21 Loic Pottier. Minimal solutions of linear Diophantine systems: Bounds and algorithms. In *Proc. RTA 1991*, pages 162–173, 1991. URL: http://dl.acm.org/citation.cfm?id=647192.720494.

22 Laura Recalde, Serge Haddad, and Manuel Silva Suárez. Continuous Petri nets: Expressive power and decidability issues. *Int. J. Found. Comput. Sci.*, 21(2):235–256, 2010. doi:10.1142/S0129054110007222.

23 Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theor. Comput. Sci.*, 412(34):4439–4451, 2011. doi:10.1016/j.tcs.2011.05.007.

24 Manuel Silva Suárez, Enrique Teruel, and José Manuel Colom. Linear algebraic and linear programming techniques for the analysis of place or transition net systems. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, pages 309–373, 1996. doi:10.1007/3-540-65306-6_19.

A Coalgebraic Take on Regular and ω-Regular Behaviour for Systems with Internal Moves

Tomasz Brengos
Faculty of Mathematics and Information Science, Warsaw University of Technology,
ul. Koszykowa 75, 00-662 Warszawa, Poland
t.brengos@mini.pw.edu.pl

Abstract

We present a general coalgebraic setting in which we define finite and infinite behaviour with Büchi acceptance condition for systems with internal moves. Since systems with internal moves are defined here as coalgebras for a monad, in the first part of the paper we present a construction of a monad suitable for modelling (in)finite behaviour. The second part of the paper focuses on presenting the concepts of a (coalgebraic) automaton and its (ω-) behaviour. We end the paper with coalgebraic Kleene-type theorems for (ω-) regular input. We discuss the setting in the context of non-deterministic (tree) automata and Segala automata.

2012 ACM Subject Classification Theory of computation Models of computation

Keywords and phrases coalgebras, regular languages, omega regular languages, automata, Büchi automata, silent moves, internal moves, monads, saturation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.25

Acknowledgements I want to thank Marco Peressotti for his continuous support and feedback. I am grateful to the anonymous referees for valuable comments and remarks.

1 Introduction

Automata theory is one of the core branches of theoretical computer science and formal language theory. One of the most fundamental state-based structures considered in the literature is a non-deterministic automaton and its relation with languages. Non-deterministic automata with a finite state-space are known to accept *regular languages*, characterized as subsets of words over a fixed finite alphabet that can be obtained from the languages consisting of words of length less than or equal to one via a finite number of applications of three types of operations: union, concatenation and the Kleene star operation [22]. This result is known under the name of *Kleene theorem for regular languages* and readily generalizes to other types of finite input (see e.g. [31]).

$$R ::= \varnothing \mid a, a \in \Sigma_\varepsilon \mid R + R \mid R \cdot R \mid R^*$$

Figure 1 Reg. exp. grammar.

On the other hand, non-deterministic automata have a natural infinite semantics which is given in terms of infinite input satisfying the so-called Büchi acceptance condition (or *BAC* in short). The condition takes into account the terminal states of the automaton and requires them to be visited infinitely often. It is a common practise to use the term *Büchi automata* in order to refer to automata whenever their infinite semantics is taken into consideration. Although the standard type of infinite input of a Büchi automaton is the set of infinite words over a given alphabet, other types (e.g. trees) are also commonly studied [31]. The class of languages of infinite words accepted by Büchi automata can also be characterized akin to the characterization

input type	*Kleene theorem*	*where*
ω-words	$\bigcup_{i=1}^{n} R_i \cdot L_i^\omega$	$R_i, L_i =$ regular lang.
ω-trees	$T_0 \cdot [T_1 \dots T_n]^\omega$	$T_i =$ regular tree lang.

Figure 2 Kleene thm. for ω-regular input.

of regular languages. This result is known under the name of *Kleene theorem for ω-regular languages* and its variants hold for many input types (see e.g. [17,31]). Roughly speaking, any language recognized by a Büchi automaton can be represented in terms of regular languages and the infinite iteration operator $(-)^\omega$. This begs the question of a unifying framework these systems can be put in and reasoned about on a more abstract level so that the analogues of Kleene theorems for (ω-)regular input are derived. The recent developments in the theory of coalgebra [11, 32, 35, 36] show that the coalgebraic framework may turn out to be suitable to achieve this goal.

A coalgebra $X \to FX$ is an abstract (categorical) representation of a single step of computation of a given process [18, 32]. The coalgebraic setting has already proved itself useful in modelling finite behaviour via least fixpoints (e.g. [8, 21, 35]) and infinite behaviour via greatest fixpoints of suitable mappings [12, 24]. The infinite behaviour with BAC can be modelled by a combination of the two [30, 36].

Our paper plans to revisit the coalgebraic framework of (in)finite behaviour from the perspective of *systems with internal moves*. A unifying theory of systems with internal steps has been part of the focus of the coalgebraic community in recent years [6–10, 35] and was mainly motivated by the research in finite behaviour of such systems. Intuitively, these systems have a special computation branch that is silent. This special branch, usually denoted by the letter τ or ε, is allowed to take several steps and in some sense remain neutral to the structure of a process. These systems arise in a natural manner in many branches of theoretical computer science, among which are process calculi [29] (labelled transition systems with τ-moves and their weak bisimulation) or automata theory (automata with ε-moves), to name only two. The approach from [8, 9] suggests that these systems should be defined as coalgebras whose type is a monad. This treatment allows for an elegant modelling of weak behavioural equivalences [9, 10] among which we find Milner's weak bisimulation [29]. Each coalgebra $\alpha : X \to TX$ becomes an endomorphism $\alpha : X \to X$ in the Kleisli category for the monad T and Milner's weak bisimulation on a labelled transition system α is defined to be a strong bisimulation on its *saturation* α^* which is the smallest LTS over the same state space satisfying $\alpha \leq \alpha^*$, $\mathsf{id} \leq \alpha^*$ and $\alpha^* \cdot \alpha^* \leq \alpha^*$ (where the composition and the order are given in the Kleisli category for the LTS monad) [8]. Hence, intuitively, α^* is the reflexive and transitive closure of α and is formally defined as the least fixpoint $\alpha^* = \mu x.(\mathsf{id} \vee x \cdot \alpha)$. Since a reflexive and transitive closure is understood as an accumulation of a *finite* number of compositions of the structure with itself, the concept of coalgebraic saturation is intrinsically related to *finite* behaviour of systems. A similar treatment of infinite behaviour (and/or their combination) in the context of systems with internal moves has not been considered so far.

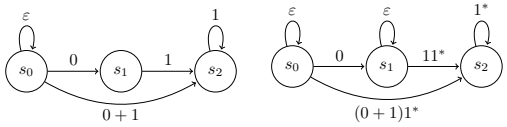


Figure 3 LTS with ε-moves and its saturation.

The aim of the paper. We plan to:

1. revisit non-deterministic (Büchi) automata and their behaviour in the coalgebraic context of systems with internal moves,
2. provide a type monad suitable for modelling (in)finite behaviour of general systems,
3. present a setting for defining (in)finite behaviour for abstract automata with silent moves,
4. state coalgebraic Kleene theorems for (ω-)regular behaviour.

The first point in the list is achieved by describing non-deterministic (Büchi) automata and their finite and infinite behaviour in terms of different coalgebraic (categorical) fixpoint

constructions calculated in the Kleisli category for a suitable monad. Section 3 serves as a motivation for the framework presented later in Section 4 and Section 5.

Originally [20,35], coalgebras with internal moves were considered as systems $X \to TF_\varepsilon X$ for a monad T and an endofunctor F, where $F_\varepsilon \triangleq F + \mathcal{I}d$. The functor TF_ε could be embedded into the monad TF^*, where F^* is the free monad over F [8]. The monad TF^* is enough to model systems with internal moves and their finite behaviour [6,8,9]. However, it will prove itself useless in the context of infinite behaviour. Hence, by revisiting and tweaking the construction of TF^* from [8], Section 4 gives a general description of the monad TF^∞, the type functor TF_ε embeds into, which is used in the remaining part of the paper to model the combination of finite and infinite behaviour. Point (3) in the above list is achieved by using two fixpoint operators: the saturation operator $(-)^*$ and a new operator $(-)^\omega$ calculated in (a full subcategory of) the Kleisli category for a monad which admits infinite behaviour. The combination of $(-)^*$ and $(-)^\omega$ allows us to define infinite behaviour with BAC. Since we are mainly interested in finite state systems, all our results are presented in the context of the full subcategory of the Kleisli category whose objects are sets $\{1,\dots,n\}$ for $n = 0, 1, \dots$, *a.k.a.* the Lawvere theory associated with the given monad. Kleene-type theorems of (4) are a direct consequence of the definition of finite and infinite behaviour with BAC using $(-)^*$ and $(-)^\omega$.

2 Basic notions

In this paper we assume the reader is familiar with basic category theory concepts like functor, (sub)monad, adjunction, Kleisli category, lifting of a functor to Kleisli category via distributive law, (initial) F-algebra, (final) F-coalgebra. For a thorough introduction to category theory the reader is referred to [28]. See also e.g. [7–9] for an extensive list of notions needed here.

Non-deterministic (Büchi) automata and their behaviour. Classically, a *nondeterministic automaton*, or simply *automaton*, is a tuple $\mathcal{Q} = (Q, \Sigma, \delta, q_0, \mathfrak{F})$, where Q is a finite set of *states*, Σ finite set called *alphabet*, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ a *transition function* and $\mathfrak{F} \subseteq Q$ set of *accepting states*. We write $q_1 \xrightarrow{a} q_2$ if $q_2 \in \delta(q_1, a)$. There are two standard types of semantics of automata: finite and infinite. The finite semantics, also known as the *language of finite words* of $\mathcal{Q}$, is defined as the set of all finite words $a_1 \dots a_n \in \Sigma^*$ for which there is a sequence of transitions $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots q_{n-1} \xrightarrow{a_n} q_n$ which ends in an accepting state $q_n \in \mathfrak{F}$ [22]. The infinite semantics, also known as the ω-*language* of $\mathcal{Q}$, is the set of infinite words $a_1 a_2 \dots \in \Sigma^\omega$ for which there is a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \dots$ for which the set of indices $\{i \mid q_i \in \mathfrak{F}\}$ is infinite, or in other words, the run r visits the set of final states $\mathfrak{F}$ infinitely often. Often in the literature, in order to emphasize that the infinite semantics is taken into consideration the automata are referred to as *Büchi automata* [31]. In our work we will consider (Büchi) automata without the initial state specified and define the (ω-)language in an automaton for any given state.

There are several other variants of input for non-deterministic Büchi automata known in the literature [17,31]. Here, we mention non-deterministic (Büchi) tree automaton, i.e. a tuple $(Q, \Sigma, \delta, \mathfrak{F})$, where $\delta : Q \times \Sigma \to \mathcal{P}(Q \times Q)$ and the rest is as before. The infinite semantics of this machine is the set of infinite binary trees with labels in Σ for which there is a run whose every branch visits $\mathfrak{F}$ infinitely often [17,31].

Coalgebras with internal moves and their type monads. As mentioned before coalgebras with internal moves were first introduced in the context of coalgebraic trace semantics as coalgebras of the type TF_ε for a monad T and an endofunctor F on C [20, 35]. If we take $F = \Sigma \times \mathcal{I}d$ then we have $TF_\varepsilon = T(\Sigma \times \mathcal{I}d + \mathcal{I}d) \cong T(\Sigma_\varepsilon \times \mathcal{I}d)$, where $\Sigma_\varepsilon \triangleq \Sigma + \{\varepsilon\}$. In [8] we showed that given some mild assumptions on T and F we may embed the functor TF_ε into the monad TF^*, where F^* is the free monad over F. In particular, if we apply this construction to $T = \mathcal{P}$ and $F = \Sigma \times \mathcal{I}d$ we obtain the monad $\mathcal{P}(\Sigma^* \times \mathcal{I}d)$ from Example 2.1 below. This construction is also revisited in this paper in Section 4. The trick of modelling the invisible steps via a monadic structure allows us *not* to specify the internal moves explicitly. Instead of considering TF_ε-coalgebras we consider T'-coalgebras for a monad T' on an arbitrary category.

The strategy of finding a suitable monad (for modelling the behaviour taken into consideration) will also be applied in this paper. Unfortunately, from the point of view of the infinite behaviour of coalgebras, considering systems of the type TF^* is not sufficient (see Section 3 for a discussion). Hence, in Section 4 we show how to obtain monads suitable for modelling infinite behaviour. Below, we list basic examples of monads considered in this paper. This list will be extended in sections to come.

▶ **Example 2.1.** The powerset endofunctor $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ carries a monadic structure for which the category $\mathcal{K}l(\mathcal{P})$ consists of sets as objects and maps $f : X \to \mathcal{P}Y$ and $g : Y \to \mathcal{P}Z$ with the composition $g \cdot f : X \to \mathcal{P}Z$ defined as follows $g \cdot f(x) = \{z \in Z \mid z \in \bigcup g(f(x))\}$. The identity morphisms $\mathsf{id} : X \to \mathcal{P}X$ are given for any $x \in X$ by $\mathsf{id}(x) = \{x\}$. Now, for a set Σ the functor $\mathcal{P}(\Sigma^* \times \mathcal{I}d)$ carries a monadic structure whose composition in the Kleisli category is given as follows [8]. For $f : X \to \mathcal{P}(\Sigma^* \times Y)$ and $g : Y \to \mathcal{P}(\Sigma^* \times Z)$ we have $g \cdot f(x) = \{(\sigma_1\sigma_2, z) \mid x \xrightarrow{\sigma_1}_f y \xrightarrow{\sigma_2}_g z \text{ for some } y \in Y\}$. The identity morphisms in this category are $\mathsf{id} : X \to \mathcal{P}(\Sigma^* \times X)$ given by $\mathsf{id}(x) = \{(\varepsilon, x)\}$. Finally, let Σ^ω be the set of all infinite sequences of elements from Σ. The functor $\mathcal{P}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$ carries a monadic structure whose Kleisli composition is the following. For $f : X \to \mathcal{P}(\Sigma^* \times Y + \Sigma^\omega)$ and $g : Y \to \mathcal{P}(\Sigma^* \times Z + \Sigma^\omega)$ the map $g \cdot f : X \to \mathcal{P}(\Sigma^* \times Z + \Sigma^\omega)$ is:

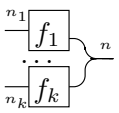
$$x \xrightarrow{\sigma}_{g\cdot f} z \iff \exists y \text{ s.t. } x \xrightarrow{\sigma_1}_f y \text{ and } y \xrightarrow{\sigma_2}_g z, \text{ where } \sigma = \sigma_1\sigma_2 \in \Sigma^*,$$
$$x \downarrow_{g\cdot f} v \iff x \downarrow_f v \text{ or } x \xrightarrow{\sigma}_f y,\ y \downarrow_g v' \text{ and } v = \sigma v' \in \Sigma^\omega.$$

In the above we write $x \xrightarrow{\sigma}_f y$ whenever $(\sigma, y) \in f(x)$ and $x \downarrow_f v$ if $v \in f(x)$ for $\sigma \in \Sigma^*$, $v \in \Sigma^\omega$. The identity morphisms in this category are the same as in the Kleisli category for the monad $\mathcal{P}(\Sigma^* \times \mathcal{I}d)$. The monadic structure of $\mathcal{P}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$ arises as a consequence of a general construction of monads modelling (in)finite behaviour described in detail in Section 4.

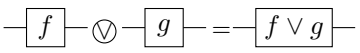
▶ **Example 2.2.** The subconvex distributions functor $\mathcal{CM}$ used to model Segala systems [33, 34] is defined as follows [16]. For any set X define $\mathcal{M}X$ to be the carrier of the free module for the semiring $[0, \infty)$ over X and put $\mathcal{CM}X = \{U \subseteq \mathcal{M}X \mid U = \overline{U} \text{ and } U \neq \varnothing\}$, where for $U \subseteq \mathcal{M}X$ we have $\overline{U} \triangleq \{\sum_{i=1}^n r_i \cdot u_i \mid u_i \in U, r_i \in [0, \infty) \ \&\ \sum_i r_i \leq 1\}$. For any map $f : X \to Y$ put $\mathcal{CM}(f) : \mathcal{CM}X \to \mathcal{CM}Y; U \mapsto \overline{\mathcal{M}f(U)}$. See also [8, 25] for a slightly different definition of $\mathcal{CM}$ and a more thorough discussion of this treatment. The functor can be equipped with a monadic structure which results in the Kleisli composition defined by: $g \cdot f(x) = \bigcup_{\phi \in f(x)} \sum_{y \in \mathsf{supp}\phi} \{\phi(y) \cdot \psi \mid \psi \in g(y)\} \in \mathcal{CM}Z$ for $x \in X$, $f : X \to \mathcal{CM}Y$ and $g : Y \to \mathcal{CM}Z$ [16].

Lawvere theories and categorical order enrichment. The primary interest of the theory of automata and formal languages focuses on automata over a *finite* state space. Hence, since we are interested in systems with internal moves (i.e. maps $X \to TX$ for a monad T), without any loss of generality we may focus our attention on coalgebras of the form $[n] \to T[n]$, where $[n] \triangleq \{1, \ldots, n\}$ with $n = 0, 1, \ldots$ for a Set-monad T. These morphisms are endomorphisms in a full subcategory of the Kleisli category for T known under the name of *Lawvere theory*. That is why we choose the setting of this paper to be Lawvere theories. Because we are interested in the coalgebraic essence of a Lawvere theory, we adopt the definition which is dual to the classical notion [27].

Formally, a *Lawvere theory*, or simply *theory*, is a category whose objects are natural numbers $n \geq 0$ such that each n is an n-fold coproduct of 1. For any element $i \in [n]$ let $i_n : 1 \to n$ denote the i-th coproduct injection and $[f_1, \ldots, f_k] : n_1 + \ldots + n_k \to n$ the cotuple of the family $\{f_l : n_l \to n\}_l$ depicted in the diagram on the right. The coprojection $n_i \to n_1 + \ldots + n_k$ into the i-th component of the coproduct will be denoted by $\mathsf{in}^{n_i}_{n_1+\ldots+n_k}$. Any morphism $k \to n$ of the form $[i^1_n, \ldots, i^k_n] : k \to n$ for $i^j \in [n]$ is called *base morphism* or *base map*. Finally, let $! : n \to 1$ be defined by $! \triangleq [1_1, 1_1, \ldots, 1_1]$. We say that a theory $\mathbb{T}'$ is a *subtheory* of $\mathbb{T}$ if there is a faithful functor $\mathbb{T}' \to \mathbb{T}$ which maps any object n onto itself. Any monad T on Set induces a theory $\mathbb{T}$ associated with it by restricting the Kleisli category $\mathcal{K}l(T)$ to objects $[n]$ for any $n \geq 0$. Conversely, for any theory $\mathbb{T}$ there is a Set based monad the theory is associated with (see e.g. [23] for details).



In order to establish the definition of the fixpoint operators $(-)^*$ and $(-)^\omega$ we require the Lawvere theory under consideration to be suitably order enriched. A category is said to be *order enriched*, or simply *ordered*, if each hom-set is a poset with the order preserved by the composition. It is $\vee$*-ordered* if all hom-posets admit arbitrary finite suprema. Note that, given such suprema exist, the composition in C does not have to distribute over them in general. We call such category *left distributive* (or *LD* in short) if $h \cdot (f \vee g) = h \cdot f \vee h \cdot g$. In this paper we will come across many left distributive categories that do not necessarily satisfy right distributivity. Still, however, all the examples taken into consideration satisfy a weaker form of right distributivity. To be precise, we say that a theory is *right distributive w.r.t. base morphisms* (or *bRD* in short) provided that $(f \vee g) \cdot j = f \cdot j \vee g \cdot j$ for any f, g and any base morphism j. We say that an order enriched category is ω-Cpo-enriched if any ascending ω-chain $f_1 \leq f_2 \leq \ldots$ of morphisms admits a supremum $\bigvee_i f_i$ which is preserved by the morphism composition. Finally, in an ordered category with finite coproducts we say that *cotupling preserves order* if $[f_1, f_2] \leq [g_1, g_2] \iff f_1 \leq g_1$ and $f_2 \leq g_2$ for any f_i, g_i with suitable domains and codomains.



▶ **Example 2.3.** The primary interest of the next section of this paper lies in the theories LTS and LTS^ω which are defined to be the theories that arise from the Kleisli categories of the monads $\mathcal{P}(\Sigma^* \times \mathcal{I}d)$ and $\mathcal{P}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$ respectively. Both theories are order-enriched with the hom-set ordering given by $f \leq g \iff f(i) \subseteq g(i)$ for any $i \in [n]$. It is easy to see that the hom-posets of LTS and LTS^ω are complete lattices, both theories are ω-Cpo-enriched and satisfy LD and bRD. Moreover, cotupling $[-, -]$ in LTS and LTS^ω preserves order.

3 Non-deterministic (Büchi) automata, coalgebraically

The purpose of this section is to give motivations for the development of the abstract theory done in the remainder of the paper. Here, we will focus on finite non-deterministic (Büchi) automata and their (in)finite behaviour from the perspective of the theories LTS and LTS^ω.

Without any loss of generality we may only consider automata over the state space $[n]$ for some natural number n. Any non-deterministic automaton with ε-moves $([n], \Sigma_\varepsilon, \delta, \mathfrak{F})$ may be modelled as a $\mathcal{P}(\Sigma_\varepsilon \times \mathcal{I}d + 1)$−coalgebra $[n] \to \mathcal{P}(\Sigma_\varepsilon \times [n] + 1)$ [32]. However, as it has been already noted in [36], from the point of view of infinite behaviour with BAC it is more useful to extract the information about the final states of the automaton and do not encode it into the transition map as above. Instead, we consider the given automaton as a pair $(\alpha, \mathfrak{F})$ where $\alpha : [n] \to \mathcal{P}(\Sigma_\varepsilon \times [n])$ is defined by $\alpha(i) = \{(a, j) \mid j \in \delta(a, i)\}$ and consider the map:

$$\mathfrak{f}_{\mathfrak{F}} : [n] \to \mathcal{P}(\Sigma_\varepsilon \times [n]); i \mapsto \begin{cases} \{(\varepsilon, i)\} & \text{if } i \in \mathfrak{F}, \\ \varnothing & \text{otherwise.} \end{cases}$$

The purpose of $\mathfrak{f}_{\mathfrak{F}}$ is to encode the set of accepting states with an endomorphism in the same Kleisli category in which the transition α is an endomorphism. Now, we have all the necessary ingredients to revisit finite and infinite behaviour (with BAC) of non-deterministic automata from the perspective of the theory LTS^ω.

Finite behaviour. Consider $\alpha^* : n \to n$ to be an endomorphism in LTS (or LTS^ω) given by $\alpha^* = \mu x.(\mathsf{id} \vee x \cdot \alpha) = \bigvee_{n \in \omega} \alpha^n$, where the order is as in Example 2.3. We have [8]:

$$\alpha^*(i) = \{(\sigma, j) \mid i \overset{\sigma}{\Longrightarrow} j\},$$

where $\overset{\sigma}{\Longrightarrow} \triangleq (\overset{\varepsilon}{\to})^* \circ \overset{a_1}{\to} \circ (\overset{\varepsilon}{\to})^* \circ \ldots (\overset{\varepsilon}{\to})^* \circ \overset{a_n}{\to} (\overset{\varepsilon}{\to})^*$ for $\sigma = a_1 \ldots a_n$, $a_i \in \Sigma$ and $\overset{\varepsilon}{\Longrightarrow} \triangleq (\overset{\varepsilon}{\to})^*$. Now, let us recall the definition of ! in any theory $\mathbb{T}$. In particular, when $\mathbb{T} = \mathsf{LTS}, \mathsf{LTS}^\omega$ the map $! : [n] \to \mathcal{P}(\Sigma_\varepsilon \times [1])$ satisfies $!(i) = \{(\varepsilon, 1)\}$. Finally, consider the morphism $! \cdot \mathfrak{f}_{\mathfrak{F}} \cdot \alpha^* : n \to 1$ in LTS (or LTS^ω)) which is explicitly given by:

$$! \cdot \mathfrak{f}_{\mathfrak{F}} \cdot \alpha^*(i) = \{(\sigma, 1) \mid \sigma \in \Sigma^* \text{ such that } i \overset{\sigma}{\Longrightarrow} j \text{ and } j \in \mathfrak{F}\}.$$

Since $\mathcal{P}(\Sigma^* \times [1]) \cong \mathcal{P}(\Sigma^*)$, the set $! \cdot \mathfrak{f}_{\mathfrak{F}} \cdot \alpha^*(i)$ represents the set of all finite words accepted by the state i in the automaton $([n], \Sigma_\varepsilon, \delta, \mathfrak{F})$.

Infinite behaviour with BAC. Note that both theories LTS and LTS^ω are complete and, hence (by Tarski-Knaster theorem), come equipped with an operator which assigns to any endomorphism $\beta : n \to n$ the morphism $\beta^\omega : n \to 0$ defined as the greatest fixpoint of $\lambda x.x \cdot \beta$. For α the map $\alpha^\omega : [n] \to \mathcal{P}(\Sigma^* \times \varnothing) = \{\varnothing\}$ is unique in LTS with $\alpha^\omega(i) = \varnothing$. However, if we compute α^ω in LTS^ω the result will be different. Indeed, we have the following.

▶ **Theorem 3.1.** *Let $\beta : [n] \to \mathcal{P}(\Sigma^* \times [n])$ be a transition map with no silent moves. Then $\beta^\omega : [n] \to \mathcal{P}(\Sigma^* \times \varnothing + \Sigma^\omega) = \mathcal{P}(\Sigma^\omega)$ in LTS^ω is given by:*

$$\beta^\omega(i) = \{\sigma_1\sigma_2 \ldots \in \Sigma^\omega \mid i \overset{\sigma_1}{\to} i_1 \overset{\sigma_2}{\to} i_2 \ldots \text{ for some } i_k \in [n] \text{ and } \sigma_k \in \Sigma^* \setminus \{\varepsilon\}\}.$$

Hence, if we, for now, assume that $\alpha : [n] \to \mathcal{P}(\Sigma_\varepsilon \times [n])$ has no silent transitions then by the above theorem: $\alpha^\omega(i) = \{a_1a_2 \ldots \in \Sigma^\omega \mid i \overset{a_1}{\to} i_1 \overset{a_2}{\to} i_2 \ldots \text{ for some } i_k \in [n]\}$. We will use the operation $(-)^\omega$ in LTS^ω to extract the information about the ω-language of $(\alpha, \mathfrak{F})$. However, we need one last ingredient. Let us define $\alpha^+ \triangleq \alpha^* \cdot \alpha$ and note

$$\alpha^+(i) = \{(\sigma, j) \mid i \overset{a_1}{\to} i_1 \ldots \overset{a_k}{\to} i_k \text{ in } \alpha \text{ and } \sigma = a_1 \ldots a_k \text{ for } k \geq 1\}.$$

Finally, consider the morphism $(\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+)^\omega : n \to 0$ in LTS^ω. In order to see the explicit formula for $(\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+)^\omega$ let us first note that the endomorphism $\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+ : [n] \to \mathcal{P}(\Sigma^* \times [n])$ has

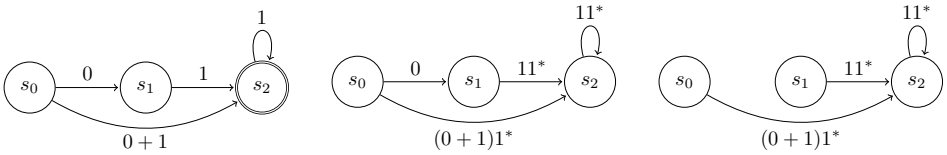


Figure 4 A non-deterministic automaton $(\alpha, \mathfrak{F})$ and the maps α^+ and $\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+$.

no silent moves and $\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+(i) = \{(\sigma, j) \mid i \xrightarrow{\sigma} j \text{ in } \alpha^+ \text{ and } j \in \mathfrak{F}\}$. Therefore, by Theorem 3.1, the map $(\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+)^\omega : [n] \to \mathcal{P}(\Sigma^\omega)$ satisfies:

$$(\mathfrak{f}_{\mathfrak{F}} \cdot \alpha^+)^\omega(i) = \text{ the } \omega\text{-language of } i \text{ in the Büchi automaton represented by } (\alpha, \mathfrak{F}).$$

This property suggests a general approach towards modelling (ω-)behaviours of abstract (coalgebraic) automata that we will develop in the sections to come.

▶ Remark. Note that throughout this paragraph we assumed the map α to have no ε-transitions. It may not be instantly clear why. It turns out that ε moves are problematic for the infinite behaviour operator $(-)^\omega$ defined as above. Indeed, in order to see this consider two finite languages $A, B \subseteq \{a, b\}^*$ defined by $A = \{\varepsilon, ab\}$ and $B = \{ab\}$. These languages can be viewed as endomorphisms $\alpha, \beta : 1 \to 1$ in LTS^ω given by $\alpha, \beta : [1] \to \mathcal{P}(\Sigma^* \times [1])$, where $\alpha(1) \triangleq \{(\varepsilon, 1), (ab, 1)\}$ and $\beta(1) \triangleq \{(ab, 1)\}$. Note that α has a silent loop, β has no silent transitions and both maps $\alpha^*, \beta^* : 1 \to 1$ satisfy $\alpha^*(1) = \{((ab)^n, 1) \mid n \geq 0\} = \beta^*(1)$. However, $\alpha^\omega \neq \beta^\omega$ in LTS^ω. Indeed, by Theorem 3.1, $\beta^\omega(1) = \{ababababab\ldots\}$ but $\alpha^\omega(1) = \mathcal{P}(\{a, b\}^\omega)$ is the set of *all* infinite words over $\{a, b\}$. The latter holds, since $\mathsf{id} \leq \alpha$ in LTS^ω and the greatest fixpoint of $\lambda x.x \cdot \alpha$ is the greatest morphism $\top : 1 \to 0$ in the given theory as $\top = \top \cdot \mathsf{id} \leq \top \cdot \alpha \leq \top$. The identity $\alpha^\omega = \mathcal{P}(\{a, b\}^\omega)$ seems to be unintuitive considering the fact that in many classical works on Büchi automata (e.g. [31]) $A^\omega = B^\omega = \{ababababab\ldots\}$. These papers use a slightly incompatible definition of the language operator $(-)^\omega : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^\omega)$ which explicitly removes ε from the argument set. Since it would be difficult to devise such an operator on a more abstract categorical level, we decide to keep with $\nu x.x \cdot \beta$ as the definition of β^ω and bear in mind this minor incompatibility with the classical work.

Why systems with internal moves? In the light of the above remark the reader may get the (wrong) impression that putting systems with internal moves into the context of infinite behaviour with BAC may seem rather *ad hoc*. To add to this, the need for categorical modelling of infinite behaviour for systems with silent steps is *not* sufficiently justified by the classical literature on the topic, where such systems rarely occur in practice (*conf.* [31]). However, as mentioned before, since putting systems with internal steps into the context is, in fact, extending the given setting to the setting of coalgebras $X \to TX$ whose type T is a monad, the main profit from this approach is the access to a simple and powerful language of the Kleisli category for the monad T. It allows us to abstract away from several "unnecessary" details and focus on the core properties. Hopefully, this paper demonstrates that the access to the language justifies the extension of the setting, as it makes it possible to formulate new results and provide their simple proofs which, in our opinion, would be tedious without such extension.

Büchi automata with non-standard input and beyond. As mentioned in Section 2, there are variants of non-deterministic (Büchi) automata that accept other types of input (e.g. binary trees). In general, given a functor $F : \mathsf{Set} \to \mathsf{Set}$ we define a non-deterministic (Büchi)

F-automaton as a pair $(\alpha, \mathfrak{F})$, where $\alpha : [n] \to \mathcal{P}F[n]$ (or $\alpha : [n] \to \mathcal{P}F_\varepsilon[n]$- to model systems with internal moves) and $\mathfrak{F} \subseteq [n]$. A natural question that arises is the following: are we able to build a setting in which we can reason about the (in)finite behaviour of systems for arbitrary non-deterministic Büchi F-automata (or even more generally, for systems of the type TF (or TF_ε) for a monad T)? If so, then is it possible to generalize the Kleene theorem for ($\omega-$)regular languages stated in the introduction to a coalgebraic level? We will answer these questions positively in the next sections.

4 Monads for (in)finite behaviour

Let C be a category which admits binary coproducts. We denote the coproduct operator by $+$ and the coprojection into the first and the second component of a coproduct by inl and inr respectively. Moreover, let $F : \mathsf{C} \to \mathsf{C}$ be a functor.

The purpose of this section is to present a monad the functor TF_ε embeds into that will prove itself sufficient to model the combination of finite and infinite behaviour (akin to the monad $\mathcal{P}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$ for the functor $\mathcal{P}(\Sigma_\varepsilon \times \mathcal{I}d)$). At first we list basic facts needed in the remainder of this section. In Subsection 4.2 we revisit the construction of the monad TF^* from [8]. Here, however, we show how it can be obtained by composing a different pair of adjunctions. Finally, we give a description of the definition of TF^∞ suitable for modelling (in)finite behaviour. In what follows, in this section we assume:

- (T, μ, η) is a monad on C and $F : \mathsf{C} \to \mathsf{C}$ lifts to $\mathcal{K}l(T)$ via a dist. law $\lambda : FT \Longrightarrow TF$,
- there is an initial $F(-) + X$-algebra for any object X and a terminal F-coalgebra $\zeta : F^\omega \to FF^\omega$.

4.1 Preliminaries

Existence of the initial $F(-) + X$-algebra $i_X : FF^*X + X \to F^*X$ (i.e. $i_X \circ \mathsf{inl}$ is the free F-algebra over X) for any object X yields an adjoint situation $\mathsf{C} \rightleftarrows \mathsf{Alg}(F)$, where the left adjoint is the free algebra functor which assigns to any object X the free algebra $i_X \circ \mathsf{inl} : FF^*X \to F^*X$ over it. The right adjoint is the forgetful functor which assigns to any F-algebra its carrier and is the identity on morphisms. This adjunction yields the monad $F^* : \mathsf{C} \to \mathsf{C}$ which assigns to any object X the carrier of the free F-algebra over X.

▶ **Example 4.1.** For any set Σ and X the initial $\Sigma \times \mathcal{I}d + X$-algebra is given by the morphism $i_X : \Sigma \times \Sigma^* \times X + X \to \Sigma^* \times X$, where $i_X(a, (\sigma, x)) = (a\sigma, x)$ and $i_X(x) = (\varepsilon, x)$.

Now we recall basic definitions and properties of Bloom F-algebras [1] which will be used to introduce monads for infinite behaviour in the next subsection. A pair $(a : FA \to A, (-)^\dagger)$ is called *Bloom F-algebra* provided that for any F-coalgebra $e : X \to FX$ the map $e^\dagger : X \to A$ satisfies:

$$\begin{array}{ccc} X & \xrightarrow{e^\dagger} & A \\ {\scriptstyle e}\downarrow & & \uparrow{\scriptstyle a} \\ FX & \xrightarrow[Fe^\dagger]{} & FA \end{array} \quad \text{and} \quad \begin{array}{ccc} X & \xrightarrow{h} & Y \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle f} \\ FX & \xrightarrow[Fh]{} & FY \end{array} \text{ implies } \begin{array}{ccc} X & \xrightarrow{h} & Y \\ & {\scriptstyle e^\dagger}\searrow \quad \swarrow{\scriptstyle f^\dagger} & \\ & A & \end{array}$$

By a *homomorphism* between Bloom algebras $(a : FA \to A, (-)^\dagger)$ and $(b : FB \to B, (-)^\ddagger)$ we mean a map $h : A \to B$ which is an F-algebra homomorphism from a to b and which additionally preserves the solution, i.e. $e^\dagger \circ h = e^\ddagger$. The category of Bloom algebras and homomorphisms between them is denoted by $\mathsf{Alg}_B(F)$. We assume that $\mathsf{Alg}(F)$ has binary coproducts which are denoted by $\oplus$. We have the following theorem.

▶ **Theorem 4.2.** *[1] The pair* $(\zeta^{-1} : FF^\omega \to F^\omega, [[-]])$*, where* $[[-]]$ *assigns to* $e : X \to FX$ *the unique coalgebra homomorphism* $[[e]] : X \to F^\omega$ *between* e *and* ζ*, is an initial object in* $\mathsf{Alg}_B(F)$*. Moreover,* $i_X \circ \mathsf{inl} \oplus \zeta^{-1}$ *is the free Bloom algebra over* X*.*

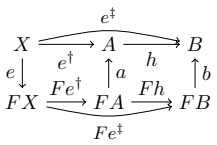
▶ Remark. Let $F^\infty : \mathsf{C} \to \mathsf{C}$ be defined as the composition of the left and right adjoints $\mathsf{C} \rightleftarrows \mathsf{Alg}_B(F)$ respectively, where the left adjoint is the free Bloom algebra functor and the right adjoint is the forgetful functor. The functor F^∞ carries a monadic structure which extends F^*. Indeed, by Th. 4.2, the monad F^* is a submonad of F^∞ (via the transformation induced by the coprojection into the first component of $i_X \circ \mathsf{inl} \oplus \zeta^{-1}$ in $\mathsf{Alg}(F)$). The construction of the free Bloom algebra from the above theorem indicates that F^∞ is a natural extension of F^* encompassing infinite behaviours of the final F-coalgebra. By abusing the notation slightly, we can write $F^\infty = F^* \oplus F^\omega$. The functor F_ε is a subfunctor of F^* [8, Lemma 4.12] and hence, by the above, also of F^∞. In the following sections this will let us turn any coalgebra $X \to TFX$ or $X \to TF_\varepsilon X$ into a system $X \to TF^\infty X$ and, by doing so, allow us to model their (in)finite behaviour.

▶ **Example 4.3.** The terminal $\Sigma \times \mathcal{I}d$-coalgebra is $\zeta : \Sigma^\omega \to \Sigma \times \Sigma^\omega; a_1a_2 \ldots \mapsto (a_1, a_2a_3 \ldots)$. The coproduct of $a : \Sigma \times A \to A$ and $b : \Sigma \times B \to B$ in $\mathsf{Alg}(F)$ is $a \oplus b : \Sigma \times (A + B) \to A + B; (\sigma, x) \mapsto$ **if** $x \in A$ **then** $a(\sigma, x)$ **else** $b(\sigma, x)$. Hence, the free Bloom algebra over X is: $\Sigma \times (\Sigma^* \times X + \Sigma^\omega) \to \Sigma^* \times X + \Sigma^\omega$, where $(a, (\sigma, x)) \mapsto (a\sigma, x)$ and $(a, a_1a_2 \ldots) \mapsto aa_1a_2 \ldots$.

Let $(a : FA \to A, (-)^\dagger)$ be a Bloom algebra, $b : FB \to B$ an F-algebra and $h : A \to B$ a homomorphism between F-algebras a and b. Then there is a unigue assignment $(-)^\ddagger$ which turns $(b : FB \to B, (-)^\ddagger)$ into a Bloom algebra and h into a Bloom algebra homomorphism and it is defined as follows [1]: for $e : X \to FX$ the map $e^\ddagger : X \to B$ is $e^\ddagger \triangleq h \circ e^\dagger$.

4.2 Lifting monads to algebras

Take an F-algebra $a : FA \to A$ and define $\bar{T}(a) \triangleq FTA \overset{\lambda_A}{\to} TFA \overset{Ta}{\to} TA$. If $h : A \to B$ is a homomorphism of algebras a and $b : FB \to B$ we put $\bar{T}(h) = T(h)$. $\bar{T} : \mathsf{Alg}(F) \to \mathsf{Alg}(F)$ is a functor for which the morphism $\eta_A : A \to TA$ is an F-algebra homomorphism from $a : FA \to A$ to $\bar{T}(a) : FTA \to TA$. Moreover, $\mu_A : T^2A \to TA$ is a homomorphism from $\bar{T}^2(a)$ to $\bar{T}(a)$ (see [4] for details). A direct consequence of this construction is the following.



▶ **Theorem 4.4.** *[4] The triple* $(\bar{T}, \bar{\mu}, \bar{\eta})$*, where for* $a : FA \to A$ *we put* $\bar{\mu}_a : \bar{T}^2(a) \to \bar{T}(a)$*;* $\bar{\mu}_a = \mu_A$ *and* $\bar{\eta}_a : a \to \bar{T}(a); \bar{\eta}_a = \eta_A$ *is a monad on* $\mathsf{Alg}(F)$*.*

The above theorem together with the assumption of existence of an arbitrary free F-algebra in $\mathsf{Alg}(F)$ leads to a pair of adjoint situations in Fig. 5. Since the composition of adjoint situations is an adjoint situation this yields a monadic structure on the functor $TF^* : \mathsf{C} \to \mathsf{C}$.

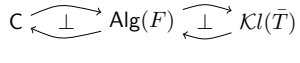


Figure 5

▶ **Example 4.5.** An example of this phenomenon is given by the monad $\mathcal{P}(\Sigma^* \times \mathcal{I}d)$ from Example 2.1 where in the above we set $T = \mathcal{P}$ and $F = \Sigma \times \mathcal{I}d$. This monad has already been described e.g. in [8], but it arose as a consequence of the composition of a different pair of adjunctions.

Monads on Bloom algebras. Above we gave a recipe for a general construction of a monadic structure on the functor TF^*. As witnessed in [6, 8], this monad is suitable to model coalgebras and their weak bisimulations and weak finite trace semantics (i.e. their

finite behaviour). Our primary interest is in modelling infinite behaviour and this monad will prove itself insufficient. The purpose of this subsection is to show how to tweak the middle category from Fig. 5 so that the monad obtained from the composition of two adjunctions is suitable to our needs.

Let $(a : FA \to A, (-)^\dagger)$ be a Bloom algebra and define $\bar{T}_B((a : FA \to A, (-)^\dagger)) \triangleq (\bar{T}(a) : FTA \to TA, (-)^\ddagger)$, where for any $e : X \to FX$ the map $e^\ddagger$ is given by $\eta_A \circ e^\dagger$. Since $\eta_A : A \to TA$ is a homomorphism between $a : FA \to A$ and $\bar{T}(a) : FTA \to TA$ the pair $(\bar{T}(a), (-)^\ddagger)$ is a Bloom algebra. For a pair of Bloom algebras $(a : FA \to A, (-)^\dagger)$ and $(b : FB \to B, (-)^\ddagger)$ and a Bloom algebra homomorphism $h : A \to B$ between them put $\bar{T}_B(h) = T(h)$. This defines a functor $\bar{T}_B : \mathsf{Alg}_B(F) \to \mathsf{Alg}_B(F)$. Analogously to the previous subsection we have the following direct consequence of the construction.

▶ **Theorem 4.6.** *The triple $(\bar{T}_B, \bar{\mu}^B, \bar{\eta}^B)$ is a monad on $\mathsf{Alg}_B(F)$, where for any Bloom algebra $(a : FA \to A, (-)^\dagger)$ the $(a, (-)^\dagger)$-components of the transformations $\bar{\mu}^B$ and $\bar{\eta}^B$ are $\bar{\mu}^B_{(a,(-)^\dagger)} : \bar{T}_B^2(a, (-)^\dagger) \to \bar{T}_B(a, (-)^\dagger)$; $\bar{\mu}^B_{(a,(-)^\dagger)} = \mu_A$ and $\bar{\eta}^B_{(a,(-)^\dagger)} : (a, (-)^\dagger) \to \bar{T}_B(a, (-)^\dagger)$ with $\bar{\eta}^B_{(a,(-)^\dagger)} = \eta_A$.*

Hence, we have the following two adjoint situations: $\mathsf{C} \rightleftarrows \mathsf{Alg}_B(F) \rightleftarrows \mathcal{K}l(\bar{T}_B)$. These adjunctions impose a monadic structure on $TF^\infty : \mathsf{C} \to \mathsf{C}$. The monad $\mathcal{P}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$ from Example 2.1 arises from the composition of the above adjoint situations (see also Example 4.3). It is important to note that since any Set-based monad T is strong, the functor $\Sigma \times \mathcal{I}d : \mathsf{Set} \to \mathsf{Set}$ always lifts to a functor on the Kleisli category for T. If we additionally assume T is a commutative monad then this is, in fact, true for any *polynomial functor* $F : \mathsf{Set} \to \mathsf{Set}$ [21], i.e. a functor defined by the grammar $F \triangleq \Sigma \in \mathsf{Set} \mid \mathcal{I}d \mid F \times F \mid \sum F$.

▶ **Example 4.7.** Let $F = \Sigma \times \mathcal{I}d^2$. Then $F^\infty = T_\Sigma(-)$ is a functor which assigns to any set X the set of complete binary trees (i.e. every node has either two children or no children) with inner nodes taking values in Σ and finitely many leaves, all taken from X [1]. This yields a monadic structure on $\mathcal{P}F^\infty = \mathcal{P}T_\Sigma$, where the Kleisli composition for $f : X \to \mathcal{P}T_\Sigma Y$ and $g : Y \to \mathcal{P}T_\Sigma Z$ is $g \cdot f : X \to \mathcal{P}T_\Sigma Z$ with $g \cdot f(x)$ being a set of trees obtained from trees in $f(x) \subseteq T_\Sigma Y$ by replacing any occurence of the leaf $y \in Y$ with a tree from $g(y) \subseteq T_\Sigma Z$. Let TTS^ω denote the theory associated with $\mathcal{P}T_\Sigma$. It is a simple exercise to prove that this category is order enriched with the order $f \leq g$ defined by $f(i) \subseteq g(i)$ for any $i \in [n]$ being complete, and that it is LD, ω-Cpo-enriched, and bRD.

▶ **Example 4.8.** For $T = \mathcal{CM}$ and $F = \Sigma \times \mathcal{I}d$ we get the monad $\mathcal{CM}(\Sigma^* \times \mathcal{I}d + \Sigma^\omega)$. The composition $\cdot$ for $f : X \to \mathcal{CM}(\Sigma^* \times Y + \Sigma^\omega)$ and $g : Y \to \mathcal{CM}(\Sigma^* \times Z + \Sigma^\omega)$ in its Kleisli category is as follows. If $\sum_{i=1}^n r_i \cdot (\sigma_i, y_i) + \sum_{i=n+1}^{n+k} r_i \cdot v_{i-n} \in f(x)$ and $\sum_{i=1}^{n_j} r_i^j \cdot (\sigma_i^j, z_i^j) + \sum_{i=n_j+1}^{n_j+k_j} r_i^j \cdot v_{i-n_j}^j \in g(y_j)$ for $j = 1, \ldots, n$, where $\sigma_i, \sigma_i^j \in \Sigma^*$ and $v_i, v_i^j \in \Sigma^\omega$, then the expression

$$\sum_{i=1}^{n} \left(\sum_{l=1}^{n_i} r_i \cdot r_l^i \cdot (\sigma_i \sigma_l^i, z_i^l) + \sum_{l=n_i+1}^{n_i+k_i} r_i \cdot r_l^i \cdot \sigma_i v_{l-n_i}^i \right) + \sum_{i=n+1}^{n+k} r_i \cdot v_{i-n}$$

is a member of the set $g \cdot f(x)$. The theory associated to this monad will be denoted by SGL^ω. It is order enriched with $f \leq g$ whenever $f(i) \subseteq g(i)$ for any i. For an arbitrary family of morphisms f_i their supremum $\bigvee_i f_i$ exists and is given by $\bigvee_i f_i(j) = \overline{\bigcup_i f_i(j)}$. Hence, the theory is complete with the infima $\bigwedge_i f_i(j) = \bigcap_i f_i(j)$. It is also LD, ω-Cpo-enriched, and bRD (the proof of this statement is analogous to the proof that the Kleisli categories for $\mathcal{CM}$ or $\mathcal{CM}(\Sigma^* \times \mathcal{I}d)$ have these properties [8, 9, 16] and, hence, is omitted).

5 Abstract (Büchi) automata and their behaviour

The purpose of this section is to generalize the concepts from Section 3 to an arbitrary theory with a suitable ordering. We start with the definition of an automaton for a theory $\mathbb{T}$.

▶ **Definition 5.1.** A $\mathbb{T}$-*automaton* or simply *automaton* is a pair $(\alpha, \mathfrak{F})$, where $\alpha : n \to n$ is an arbitrary endomorphism called *transition morphism* and $\mathfrak{F} \subseteq [n]$.

In order to define finite and infinite behaviour of $(\alpha, \mathfrak{F})$ we require the theory to satisfy more assumptions. An order enriched theory $\mathbb{T}$ is called *complete saturation theory* (or *CST* in short) provided that:

- **i** hom-posets are complete lattices,
- **ii** it is ω-Cpo-enriched, LD & bRD
- **iii** bottom maps 0 satisfy $f \cdot 0 = 0$ for any f,
- **iv** cotupling preserves the order.

From now on in this section we assume that $\mathbb{T}$ is a complete saturation theory. Note that the definition of a $\mathbb{T}$-automaton was stated in a more general framework. However, the finite and infinite behaviour of $(\alpha, \mathfrak{F})$ will be only considered for complete saturation theories.

▶ Remark. The assumption about completeness of the order, although a strong assumption, will guarantee existence of two types of fixpoints, namely $(-)^*$ and $(-)^\omega$. The former fixpoint operator was thoroughly studied in [7–10] in the context of coalgebraic weak bisimulation. Although it can be defined in an arbitrary completely ordered category, it requires left distributivity to be expressive enough [9] and ω-Cpo-enrichment to be calculated in terms of countable joins. Right distributivity w.r.t. the base morphisms is a technical assumption that is crucial in the proofs of theorems to come. This is a weak assumption as already discussed in [9, Lemma 3.25]. The bottom maps 0 provide us with a natural annihilator thanks to which given a set $\mathfrak{F} \subseteq [n]$ we can encode it as an endomorphism $\mathfrak{f}_\mathfrak{F} : n \to n$ defined as the cotuple of i_n's and 0^1_n's depending on whether the given coordinate is a member of $\mathfrak{F}$ or not. Finally, the last assumption guarantees that the order plays well with the coproduct.

For any endomorphism $\alpha : n \to n$ in $\mathbb{T}$ define $\alpha^*, \alpha^+ : n \to n$ and $\alpha^\omega : n \to 0$ by:

$$\alpha^* \triangleq \mu x.(\mathsf{id} \vee x \cdot \alpha), \quad \alpha^+ \triangleq \alpha^* \cdot \alpha \text{ and } \alpha^\omega \triangleq \nu x.x \cdot \alpha.$$

In a complete saturation theory we have $\alpha^* = \bigvee_{n<\omega}(\mathsf{id} \vee \alpha)^n$ [8] and $\alpha^\omega = \bigwedge_{\kappa \in \mathsf{Ord}}(\lambda x.x \cdot \alpha)^\kappa \top$, where $\top : n \to 0$ is the greatest element of $\mathbb{T}(n, 0)$ and $(\lambda x.x \cdot \alpha)^\kappa$ is defined by the transfinite induction by $(\lambda x.x \cdot \alpha)^{\kappa+1} = (\lambda x.x \cdot \alpha)(\lambda x.x \cdot \alpha)^\kappa$ for a successor ordinal $\kappa + 1$ and $(\lambda x.x \cdot \alpha)^\kappa = \bigwedge_{\lambda<\kappa}(\lambda x.x \cdot \alpha)^\lambda$ for a limit ordinal κ.

▶ **Theorem 5.2.** *For any $\alpha, \beta : n \to n$ we have:*

1. $\mathsf{id}^* = \mathsf{id}$, $\mathsf{id} \leq \alpha^*$ *and* $\alpha^* \cdot \alpha^* = \alpha^*$,
2. $(\alpha \cdot \beta)^\omega = (\beta \cdot \alpha)^\omega \cdot \beta$,
3. $(\alpha^n)^\omega = \alpha^\omega$ *for any* $n > 0$,
4. $\alpha^\omega = (\alpha^+)^\omega$.

▶ **Definition 5.3.** *Finite behaviour* (ω-*behaviour*) $||(\alpha, \mathfrak{F})|| : n \to 1$ (resp. $||(\alpha, \mathfrak{F})||_\omega : n \to 0$) of an automaton $(\alpha, \mathfrak{F})$ is defined by:

$$||(\alpha, \mathfrak{F})|| \triangleq !\cdot \mathfrak{f}_\mathfrak{F} \cdot \alpha^* \text{ and } ||(\alpha, \mathfrak{F})||_\omega \triangleq (\mathfrak{f}_\mathfrak{F} \cdot \alpha^+)^\omega.$$

Finite (ω-)behaviour of a state $i_n : 1 \to n$ of $(\alpha, \mathfrak{F})$ is $||(\alpha, \mathfrak{F})|| \cdot i_n$ (resp. $||(\alpha, \mathfrak{F})||_\omega \cdot i_n$).

▶ **Example 5.4.** The theories LTS^ω, TTS^ω and SGL^ω are complete saturation theories. As we have already seen in Section 3, the finite and ω-behaviour of LTS^ω-automata coincides with the classical notions. The same can be easily proven to be true for TTS^ω (see e.g. [31] for classical definitions in the theory of tree automata). According to our knowledge, *Segala automata* or, in other words, SGL^ω-automata, have not been considered in the computer science literature so far. See Subsection 5.1 for a discussion on these systems in the context of expressivity in language theory.

▶ Remark. So far in the coalgebraic literature, finite behaviour of systems was introduced in terms of the finite trace [6,26,35]. In the order enriched setting for systems with internal moves for which the type functor encodes accepting states, finite trace is given by $\alpha^\dagger = \mu x.x \cdot \alpha$ [7]. However, from the point of our setting, the terminal states are not part of the transition. In this case we can consider the *exception monad* $\mathcal{I}d + 1$ on any theory $\mathbb{T}$, denote its associated theory by $\widehat{\mathbb{T}}$, and encode any $\mathbb{T}$-automaton $(\alpha, \mathfrak{F})$ as a $\widehat{\mathbb{T}}$-endomorphism $\widehat{\alpha} : n \to n$ (or equivalently $\mathbb{T}$-morphism $\widehat{\alpha} : n \to n+1$) defined by $\widehat{\alpha} = \mathsf{in}^n_{n+1} \cdot \alpha \vee \widehat{\mathfrak{f}_{\mathfrak{F}}}$, where $\widehat{\mathfrak{f}_{\mathfrak{F}}} : n \to n+1$ is a morphism in $\mathbb{T}$ given by $\widehat{\mathfrak{f}_{\mathfrak{F}}}(i) =$ **if** $i \in \mathfrak{F}$ **then** $n+1$ **else** 0. It is a simple exercise to prove that, given the assumptions of this section, $\widehat{\alpha}^\dagger = ||(\alpha, \mathfrak{F})||$. Therefore, our definition of finite behaviour via $(-)^*$ coincides with the trace definition in an ordered category [6].

Kleene theorems. The prominent role in the theory of non-deterministic automata is played by regular languages. Using the nomenclature of Section 3 these languages are given by $! \cdot \mathfrak{f} \cdot \alpha^* \cdot i_n : 1 \to 1$ for an LTS^ω automaton $(\alpha, \mathfrak{F})$ in which we have $\alpha : [n] \to \mathcal{P}(\Sigma_\varepsilon \times [n])$. The set of regular languages, denoted by $\mathfrak{Reg}(1,1)$, is known to be closed under the language composition, finite union and Kleene star operation. These three operations are exactly the composition, finite joins and the saturation of morphisms $1 \to 1$ in the theory LTS^ω. Moreover, $\mathfrak{Reg}(1,1)$ is the smallest set of languages containing the empty language, single letter languages and being closed under the three operations. This classical result is known under the name of *Kleene theorem for regular languages* [22]. A similar theorem can be proven for automata that accept non-sequential data types, e.g. trees [17,31]. However for tree automata the result is slightly more involved as the set $\mathfrak{Reg}(1,1)$ of regular tree languages is closed under a more complex type of composition, namely the composition of regular tree languages with multiple variables. To be more precise, if $\mathfrak{Reg}(1,p)$ denotes the set of regular tree languages whose leaves may end in variables from $\{1,\dots,p\}$, then the morphism $[r_1,\dots,r_p] \cdot r$ is a member of $\mathfrak{Reg}(1,1)$ for any $r \in \mathfrak{Reg}(1,p)$ and $r_i \in \mathfrak{Reg}(1,1)$. These observations are generalized to the coalgebraic level below. As a direct consequence of this treatment we get a characterization of ω-regular behaviours.

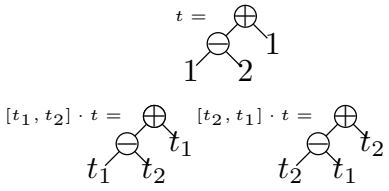


Figure 6 Tree composition with inner nodes in $\{+,-\}$ and variables in $\{1,2\}$.

Let $T = (T, \mu, \eta)$ be a monad on Set and F a Set-endofunctor satisfying the assumptions of Section 4. This allows us to consider the monad TF^∞ and the theory $\mathbb{T}_{TF^\infty}$ associated with it. We say that a map $\alpha : m \to n$ in $\mathbb{T}_{TF^\infty}$ is a (T, F_ε)-*map* if $\alpha : [m] \to TF_\varepsilon[n]$ in Set (it is a well defined notion as F_ε is a subfunctor of F^∞). Note that by the definition of F^∞ the family of (T, F_ε)-maps contains all base maps of $\mathbb{T}_{TF^\infty}$ and is closed under cotupling and the composition with base morphisms (it follows by the definition of the monadic structure of F^∞, TF^∞ and Remark in Subsection 4.1). $\mathbb{T}_{TF^\infty}$-automata whose transition maps are (T, F_ε)-maps will be referred to as (T, F_ε)-*automata*. In this paragraph we assume that $\mathbb{T}_{TF^\infty}$ is a CST and:

- (T, F_ε)-maps are closed under taking arbitrary suprema (hence, also contain 0's),
- $0 \cdot \alpha = 0$ for any (T, F_ε)-map α.

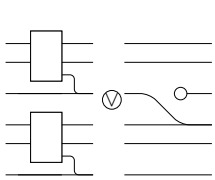
As a direct consequence of these assumptions and since id is a base morphism we get that $0 \cdot \alpha^* = \alpha^* \cdot 0 = 0$ for any (T, F_ε)-map α which is a $\mathbb{T}_{TF^\infty}$-endomorphism. We define the set of *regular morphisms* $m \to p$ by:

$$\mathfrak{Reg}(m,p) \triangleq \{j' \cdot \mathfrak{f}_{\mathfrak{F}} \cdot \alpha^* \cdot j \mid (\alpha : n \to n, \mathfrak{F}) \text{ is a } (T, F_\varepsilon)\text{-aut. and } j : m \to n, j' : n \to p \text{ are base maps}\}.$$

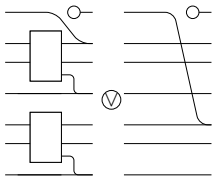
The set of regular morphisms $\mathfrak{Reg}(1, p)$ will be often referred to as the set of *regular trees with variables in* p. Note that $\mathfrak{Reg}(1, 1)$ is exactly the set of finite behaviours of states in (T, F_ε)-automata. A regular morphism $r \in \mathfrak{Reg}(m, p)$ is said to be in *normal form* (NF) if it is given by $r = [0_p^n, \mathsf{id}_p] \cdot [\alpha, \mathsf{in}_{n+p}^p]^* \cdot \mathsf{in}_{n+p}^m$ for a (T, F_ε)-map $\alpha : n \to n + p$ and $m \leq n$. The following lemma states that all regular morphisms can be given in their normal forms and that they can be obtained from regular trees via cotupling.

▶ **Lemma 5.5.** *The following equality is true:* $\mathfrak{Reg}(m, p) = \{[r_1, \ldots, r_m] \mid r_i \in \mathfrak{Reg}(1, p)\} = \{[0_p^n, \mathsf{id}_p] \cdot [\alpha, \mathsf{in}_{n+p}^p]^* \cdot \mathsf{in}_{n+p}^m \mid \alpha : n \to n + p$ *is a* (T, F_ε)*-map and* $m \leq n\}$.

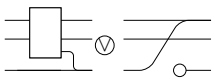
The next results (Lem. 5.6 and Th. 5.7) show, in particular, that regular morphisms with suitable domains and codomains are closed under composition, finite joins and saturation operation. The constructions used in the proofs of the results below are simple generalization of classical constructions of non-deterministic automata with ε-moves used in proving that concatenation/finite union/Kleene star of regular languages is regular (see e.g. [22]). Hence, in our opinion, it can be considered a computer science folklore which presents itself very aesthetically in terms of the string diagram calculus. Note that for classical regular languages it was enough to consider the case where the normal form $[0, \mathsf{id}_p] \cdot [\alpha, \mathsf{in}^p]^* \cdot \mathsf{in}^m$ of the expressions satisfied $m = p = 1$ (i.e. one initial and one final state). These constructions can be summarized by the following three diagrams.



$r_1 \cdot r_2$



$r_1 \vee r_2$



r^*

▶ **Lemma 5.6.** *The identity maps in* $\mathbb{T}_{TF^\infty}$ *are regular morphisms. Moreover, regular morphisms are closed under the composition from* $\mathbb{T}_{TF^\infty}$.

Let $\mathfrak{Reg}(T, F)$ be the category whose objects are the same as the objects of $\mathbb{T}_{TF^\infty}$ and whose hom-sets are $\mathfrak{Reg}(m, n)$ with the composition taken from $\mathbb{T}_{TF^\infty}$. By the above lemmas this definition is proper and, moreover, $\mathfrak{Reg}(T, F)$ is a theory. It is order enriched with the order from $\mathbb{T}_{TF^\infty}$. Moreover, the following statement holds.

▶ **Theorem 5.7** (Kleene thm. for regular behaviour)**.** $\mathfrak{Reg}(T, F)$ *is an ordered theory which:*

(a) *contains all* (T, F_ε)*-maps,*

(b) *admits finite suprema and each hom-set contains the bottom element,*

(c) *endomorphisms are closed under* $(-)^*$.

If $\mathfrak{Rat}(T, F)$ *is the smallest subtheory of* $\mathbb{T}_{TF^\infty}$ *satisfying (a)-(c) then* $\mathfrak{Rat}(T, F) = \mathfrak{Reg}(T, F)$.

Finally, put $\omega\mathfrak{Rat}(T,F) \triangleq \{[r_1,\ldots,r_m]^\omega \cdot r \mid r \in \mathfrak{Rat}(1,m), r_i \in \mathfrak{Rat}(1,m) \text{ for } m<\omega\}$ and

$$\omega\mathfrak{Reg}(T,F) \triangleq \{||(\alpha,\mathfrak{F})||_\omega \cdot i_m : 1 \to 0 \mid (\alpha,\mathfrak{F}) \text{ is } \mathbb{T}_{TF^\infty}\text{-aut. with } (T,F_\varepsilon)\text{-map } \alpha : m \to m\}.$$

▶ **Theorem 5.8** (Kleene thm. for ω-regular behaviour)**.** *We have* $\omega\mathfrak{Rat}(T,F) = \omega\mathfrak{Reg}(T,F)$.

5.1 Behaviours v. languages

The purpose of this subsection is to define *languages* for $\mathbb{T}_{TF^\infty}$-automata. Unlike behaviours, languages are simply subsets of $F^\infty 1$. As we will see below there is a natural way to introduce such languages for abstract automata. The theory presented in this subsection is motivated by (ω-)languages of probabilistic (ω-)automata [3] defined using a construction akin to the one presented below. However, since fully probabilistic automata are not considered in our paper (see the summary section for details), we will focus our attention on SGL^ω-automata and the languages they generate. We will show that the classes of (ω-)regular languages of these machines coincide with the class of (ω-)regular languages in the classical sense.

Let T be a functor on Set and consider the transformation $\tau : T \Longrightarrow \mathcal{P}$ whose X-component is defined by $\tau_X(t) = \bigcap\{Y \subseteq X \mid t \in TY\}$. If T preserves preimages and infinite intersections then the transformation is natural [19]. Here, we assume it is the case.

▶ **Example 5.9.** For $T = \mathcal{CM}$ the transformation τ is given by:

$$\tau_X(U) = \bigcup\{\{x_1,\ldots,x_n\} \mid r_1\cdot x_1 + \ldots + r_n \cdot x_n \in U \text{ for } r_i > 0\} \text{ for } U \in \mathcal{CM}X.$$

It is a simple exercise to prove that $\tau : \mathcal{CM} \Longrightarrow \mathcal{P}$ is a natural transformation.

Let T and F and $\mathcal{P}$ and F satisfy the assumptions of Section 4. Then the transformation $\tau : T \Longrightarrow \mathcal{P}$ imposes an assignment τ_{F^∞} between theories $\mathbb{T}_{TF^\infty}$ and $\mathbb{T}_{\mathcal{P}F^\infty}$ given by: $\tau_{F^\infty}(n) \triangleq n$ and $\tau_{F^\infty}(f : m \to TF^\infty m) \triangleq \tau_{F^\infty n} \circ f$. Assume that both $\mathbb{T}_{TF^\infty}$ and $\mathbb{T}_{\mathcal{P}F^\infty}$ are complete saturation theories. Given a $\mathbb{T}_{TF^\infty}$-automaton $(\alpha : n \to n, \mathfrak{F})$ and $i \in [n]$, we define its *language* (resp. *ω-language*) by $\mathcal{L}(\alpha,\mathfrak{F},i) \triangleq \tau_{F^\infty}(||(\alpha,\mathfrak{F})|| \cdot i_n)$ and $\mathcal{L}^\omega(\alpha,\mathfrak{F},i) \triangleq \tau_{F^\infty}(||(\alpha,\mathfrak{F})||_\omega \cdot i_n)$. Now, the sets of *regular* and *ω-regular languages* for $\mathbb{T}_{TF^\infty}$ are $\mathfrak{LReg}(T,F) \triangleq \{(\mathcal{L}(\alpha,\mathfrak{F},i) \mid \alpha \text{ is a } (T,F_\varepsilon)\text{-map}\}$ and

$$\omega\mathfrak{LReg}(T,F) \triangleq \{\mathcal{L}^\omega(\alpha,\mathfrak{F},i) \mid \alpha \text{ is a } (T,F_\varepsilon)\text{-map}\}.$$

▶ **Theorem 5.10.** *If $\tau_{F^\infty} : \mathbb{T}_{TF^\infty} \to \mathbb{T}_{\mathcal{P}F^\infty}$ is a functor which preserves cotupling, preserves 0's, finite suprema and suprema of ω-chains then $\mathfrak{LReg}(T,F) \subseteq \mathfrak{Reg}(\mathcal{P},F)(1,1)$. Moreover, if $\tau_{F^\infty}(\beta^\omega) = \tau_{F^\infty}(\beta)^\omega$ for any $\mathbb{T}_{TF^\infty}$-endomorphism β of the form $\mathfrak{f}_\mathfrak{F}\cdot\alpha^+$ for a (T,F_ε)-map α then $\omega\mathfrak{LReg}(T,F) \subseteq \omega\mathfrak{Reg}(\mathcal{P},F)$.*

▶ **Example 5.11.** The assignment $\mathsf{SGL}^\omega \to \mathsf{LTS}^\omega$ induced by the natural transformation from Example 5.9 satisfies the assumptions of the first part of Th. 5.10. Additionally, it preserves the assumptions of the second part of this statement, and, hence, from the point of view of regular and ω-regular languages Segala automata are equally expressive as the non-deterministic (Büchi) automata.

6 Summary, future and related work

The purpose of this paper was to develop a coalgebraic (categorical) framework to reason about abstract automata and their finite and infinite behaviours satisfying BAC. We achieved this goal by constructing a monad suitable to handle the types of behaviours we were

interested in and defining them in the right setting. A natural and direct consequence of this treatment was Theorem 5.7 and Theorem 5.8, i.e. the coalgebraic characterization of regular and ω-regular behaviour. These two results are the main reason why the primary interest of this paper is the Set-based finite structures. Note that several definitions and properties of Section 5 generalize to systems whose type monad is over a different category than Set (in this case an *automaton* should be simply defined as a pair of endomorphisms in the given Kleisli category).

Seemingly, the main restrictions of this framework are hidden behind the assumptions in the definition of a complete saturation theory. However, many of these axioms can be relaxed. For instance, in case of lack of left distributivity we may use a construction from our previous work [9] which embeds suitably ordered categories into left distributive ones. Secondly, the assumption about completeness of the order may be replaced with the assumption about existence of $(-)^*$ and $(-)^\omega$ satisfying the desired properties (note that the theory $\mathfrak{Reg}$ defined in Section 5 is not necessarily complete, yet finite joins, $(-)^*$ and $(-)^\omega$ are well defined).

Future work. We plan that the next step from here will be to put fully probabilistic automata into our framework, as this type of machines and their ω-languages play a significant role in infinite language theory [2]. Probabilistic systems have been successfully put into the saturation and weak bisimulation framework by embedding the category these systems are described in, into a category which admits left distributivity [9].

Given our natural characterization of coalgebraic ω-regular languages we ask if it is possible to characterize it in an *algebraic* way in terms of a preimage of a subset of a finite algebraic structure. Especially, considering the fact that by Th. 5.2 the pair of hom-sets $(\mathbb{T}(n,n), \mathbb{T}(n,0))$ equipped with suitable operations resembles a Wilke algebra used in the algebraic characterization of these languages (see e.g. [31] for details).

Related work. The first coalgebraic take on ω-languages was presented in [11], where authors put deterministic Muller automata with Muller acceptance condition into the framework. Our work is related to a more recent paper [36], where Urabe *et al.* give a coalgebraic framework for modelling behaviour with Büchi acceptance condition for (T, F)-systems. The main ingredient of their work is a solution to a system of equations which uses least and greatest fixpoints. This is done akin to Park's [30] classical characterization of ω-languages via a system of equations. In our paper we also use least and greatest fixpoints, however, the operators we consider are the two natural types of operators $(-)^* = \mu x.\mathsf{id} \vee x \cdot (-)$ and $(-)^\omega = \nu x.x \cdot (-)$ which generalize the language operators $(-)^*$ and $(-)^\omega$ known from the classical theory of regular and ω-regular languages. By calculating everything in the Kleisli category for the given monad and by using the aforementioned operators we simplify the language considerably. This allows us to state and prove Kleene-type theorems for (ω-)regular input which was not achieved in [36] and (in our opinion) would be difficult to obtain in that setting. To summarize, the major differences between our work and [36] are the following:

- we use the setting of systems with internal moves (i.e. coalgebras over a monad) to discuss infinite behaviour with BAC,
- the infinite behaviour with BAC is calculated in terms of a simple expression which uses $(-)^*$ and $(-)^\omega$ in the Kleisli category,
- we provide the definition of a finite behaviour of a system (using $(-)^*$) and build a bridge between regular and ω-regular behaviours on a coalgebraic level in terms of the Kleene theorem.

Abstract finite automata have already been considered in the computer science literature in the context of Lawvere iteration theories with analogues of Kleene theorems stated and proven (see e.g. [5, 13–15]). Some of these results seem to be presented in a more general setting than ours, using a slightly different language than ours (*conf.* Theorem 5.7 and e.g. [5, Theorem 1.4]). We decided to state Theorem 5.7 the way we did, in order to make a direct generalization of the classical Kleene theorem for regular input and to give a coalgebraic interpretation which is missing in [5, 13–15]. We should also mention that the infinite behaviour with BAC was defined in *loc. cit.* only for a very specific type of theories (i.e. the matricial theories over an algebra with an infinite iteration operator), which do not encompass e.g. non-deterministic Büchi tree automata and their infinite tree languages.

References

1 Jirí Adámek, Mahdieh Haddadi, and Stefan Milius. Corecursive algebras, corecursive monads and bloom monads. *Logical Methods in Computer Science*, 10(3), 2014. doi:10.2168/LMCS-10(3:19)2014.

2 Christel Baier and Marcus Grosser. Recognizing omega-regular languages with probabilistic automata. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, LICS '05, pages 137–146, Washington, DC, USA, 2005. IEEE Computer Society. doi:10.1109/LICS.2005.41.

3 Christel Baier, Marcus Grösser, and Nathalie Bertrand. Probabilistic omega-automata. *J. ACM*, 59(1):1:1–1:52, 2012. doi:10.1145/2108242.2108243.

4 Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg.

5 Stephen Bloom and Zoltán Ésik. *Iteration Theories. The Equational Logic of Iterative Processes.* Monographs in Theoretical Computer Science. Springer, 1993.

6 Filippo Bonchi, Stefan Milius, Alexandra Silva, and Fabio Zanasi. Killing epsilons with a dagger: A coalgebraic study of systems with algebraic label structure. *Theoretical Computer Science*, 604:102–126, 2015. doi:10.1016/j.tcs.2015.03.024.

7 Tomasz Brengos. On coalgebras with internal moves. In Marcello M. Bonsangue, editor, *Proc. CMCS*, Lecture Notes in Computer Science, pages 75–97. Springer, 2014. doi:10.1007/978-3-662-44124-4_5.

8 Tomasz Brengos. Weak bisimulation for coalgebras over order enriched monads. *Logical Methods in Computer Science*, 11(2):1–44, 2015. doi:10.2168/LMCS-11(2:14)2015.

9 Tomasz Brengos, Marino Miculan, and Marco Peressotti. Behavioural equivalences for coalgebras with unobservable moves. *Journal of Logical and Algebraic Methods in Programming*, 84(6):826–852, 2015. doi:10.1016/j.jlamp.2015.09.002.

10 Tomasz Brengos and Marco Peressotti. A Uniform Framework for Timed Automata. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2016.26.

11 Vincenzo Ciancia and Yde Venema. Stream automata are coalgebras. In *Proc. CMCS*, volume 7399 of *Lecture Notes in Computer Science*, pages 90–108, 2012. doi:10.1007/978-3-642-32784-1_6.

12 Corina Cîrstea. Generic infinite traces and path-based coalgebraic temporal logics. *Electr. Notes Theor. Comput. Sci.*, 264(2):83–103, 2010. doi:10.1016/j.entcs.2010.07.015.

13 Zoltán Ésik and Tamás Hajgató. Iteration grove theories with applications. In *Proc. Algebraic Informatics*, volume 5725 of *Lecture Notes in Computer Science*, pages 227–249. Springer, 2009. doi:10.1007/978-3-642-03564-7_15.

14 Zoltán Ésik and Werner Kuich. *A Unifying Kleene Theorem for Weighted Finite Automata*, pages 76–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-19391-0_6.

15 Zoltán Ésik and Werner Kuich. Modern automata theory, 2013. URL: http://www.dmg.tuwien.ac.at/kuich/.

16 Sergey Goncharov and Dirk Pattinson. Coalgebraic weak bisimulation from recursive equations over monads. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proc. ICALP*, volume 8573 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2014. doi:10.1007/978-3-662-43951-7_17.

17 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*, page 392. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

18 H. Peter Gumm. *Elements of the general theory of coalgebras*. LUATCS 99, Rand Afrikaans University, 1999.

19 H. Peter Gumm. From t-coalgebras to filter structures and transition systems. In José Luiz Fiadeiro, Neil Harman, Markus Roggenbach, and Jan Rutten, editors, *Algebra and Coalgebra in Computer Science*, pages 194–212, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:doi.org/10.1007/11548133_13.

20 Ichiro Hasuo. Generic forward and backward simulations. In *Prof. CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 406–420, 2006. doi:10.1007/11817949_27.

21 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:11)2007.

22 John E. Hopcroft, Rajeev Motwani, Rotwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

23 Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, 2007. doi:10.1016/j.entcs.2007.02.019.

24 Bart Jacobs. Trace semantics for coalgebras. *Electr. Notes Theor. Comput. Sci.*, 106:167–184, 2004. doi:/doi.org/10.1016/j.entcs.2004.02.031.

25 Bart Jacobs. Coalgebraic trace semantics for combined possibilistic and probabilistic systems. In *Proc. CMCS*, Electronic Notes in Theoretical Computer Science, pages 131–152, 2008. doi:10.1016/j.entcs.2008.05.023.

26 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. In *Proc. CMCS*, volume 7399 of *Lecture Notes in Computer Science*, pages 109–129, 2012. doi:10.1007/978-3-642-32784-1_7.

27 F. W. Lawvere. Functorial semantics of algebraic theories. *Proc. Nat. Acad. Sci. U.S.A.*, 50:869–872, 1963. doi:10.1073/pnas.50.5.869.

28 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlang New York, 1978. doi:10.1007/978-1-4757-4721-8.

29 Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

30 David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, pages 167–183, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

31 Jean-Eric Pin and Dominique Perrin. *Infinite Words: Automata, Semigroups, Logic and Games*, page 538. Elsevier, 2004.

32 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

33 Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

34 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In *Proc. CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496, 1994. doi:10.1007/978-3-540-48654-1_35.

35 Alexandra Silva and Bram Westerbaan. A coalgebraic view of ϵ-transitions. In Reiko Heckel and Stefan Milius, editors, *Proc. CALCO*, volume 8089 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2013. doi:10.1007/978-3-642-40206-7_20.

36 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic Trace Semantics for Buechi and Parity Automata. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2016.24.

Relating Syntactic and Semantic Perturbations of Hybrid Automata

Nima Roohi
Univeristy of Pennsylvania, USA
roohi2@cis.upenn.edu

Pavithra Prabhakar
Kansas State University, USA
http://people.cs.ksu.edu/~pprabhakar/
pprabhakar@ksu.edu

Mahesh Viswanathan
University of Illinois at Urbana-Champaign, USA
http://vmahesh.cs.illinois.edu/
vmahesh@illinois.edu

Abstract

We investigate how the semantics of a hybrid automaton deviates with respect to syntactic perturbations on the hybrid automaton. We consider syntactic perturbations of a hybrid automaton, wherein the syntactic representations of its elements, namely, initial sets, invariants, guards, and flows, in some logic are perturbed. Our main result establishes a continuity like property that states that small perturbations in the syntax lead to small perturbations in the semantics. More precisely, we show that for every real number $\epsilon > 0$ and natural number k, there is a real number $\delta > 0$ such that $\mathcal{H}^\delta$, the δ syntactic perturbation of a hybrid automaton $\mathcal{H}$, is ϵ-simulation equivalent to $\mathcal{H}$ up to k transition steps. As a byproduct, we obtain a proof that a bounded safety verification tool such as `dReach` will eventually prove the safety of a safe hybrid automaton design (when only non-strict inequalities are used in all constraints) if `dReach` iteratively reduces the syntactic parameter δ that is used in checking approximate satisfiability. This has an immediate application in counter-example validation in a CEGAR framework, namely, when a counter-example is spurious, then we have a complete procedure for deducing the same.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems, Theory of computation → Timed and hybrid models, Software and its engineering → Model checking

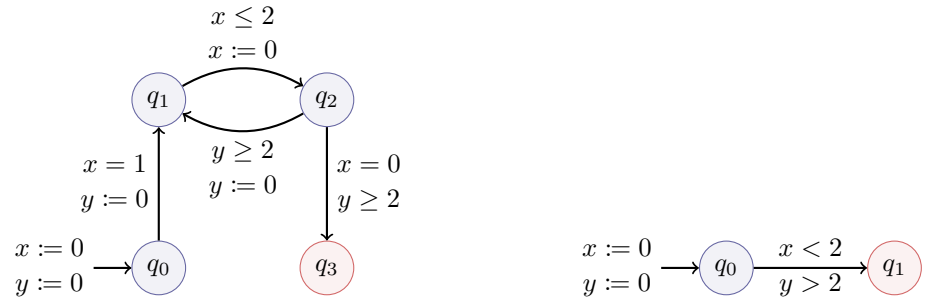
Keywords and phrases Model Checking, Hybrid Automata, Approximation, Perturbation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.26

Funding Mahesh Viswanathan was partially supported by NSF CSR 1422798, and Pavithra Prabhakar was partially supported by NSF CAREER Award No. 1552668 and ONR YIP Award No. N00014-17-1-257.

1 Introduction

Hybrid automata are a mathematical framework to model systems consisting of a digital controller interacting with a continuously evolving physical process. Such cyberphysical systems arise in a variety of applications ranging from every day smart home appliances to safety critical systems like avionics software and self driving cars. Hybrid automata have discrete modes corresponding to phases in the digital controller, where the physical



(a) Timed automaton T_A.

(b) Timed automaton T_B.

Figure 1 Example timed automata whose syntactic perturbations are not semantically close. Continuous variables x, y are clocks, i.e., $\dot{x} = \dot{y} = 1$ and the invariant in each state is $0 \leq x, y \leq 3$. The states q_3 in T_A and state q_1 in T_B are not reachable. But q_1 and q_3 are reachable in infinitesimal syntactic perturbations T_A and T_B, respectively.

environment evolves continuously according to physics laws based on the actuator inputs from the controller in that phase. Transitions between modes model discrete changes to actuator inputs from the controller based on sensor feedback.

Formal models of cyberphysical systems are typically "best effort" descriptions that may not be 100% faithful to the actual system. There are several sources of inaccuracies. Environment parameters, like network latency, are estimated based on extensive experimentation. Differential equations governing the behavior of the physical plant maybe imprecise, either because of limitations in our mathematical understanding of the physics, or because of a conscious effort to construct a tractable model by approximating. Sensor and actuator delays might either be unpredictable or have been ignored. Finally, inaccuracies in sensor input to the controller may not have been faithfully modeled.

For these reasons, a system modeled by hybrid automaton $\mathcal{H}$, may, in practice, behave like the automaton $\mathcal{H}^\delta$ which is obtained from $\mathcal{H}$ by syntactically perturbing constants, constraints on mode switches, and flow equations governing continuous evolution, by some $\delta > 0$. A natural question to ask is if the automaton $\mathcal{H}$ and its perturbation $\mathcal{H}^\delta$ are *semantically close* (in some well defined sense). Can a perturbed automaton $\mathcal{H}^\delta$ be arbitrarily close to $\mathcal{H}$? The challenge in answering this question lies in the presence of discrete mode changes – small changes to the behavior of a hybrid automaton could result in transitions becoming enabled that yield unexpected behavior in the perturbed automaton.

For example, consider the timed automaton T_A shown in Figure 1a. Variables x, y are clocks (i.e., $\dot{x} = \dot{y} = 1$), and the invariant in every location is $0 \leq x, y, \leq 3$. Observe that, when the automaton first visits q_1 from q_0, $x = 1$ and y is set to 0. Because of this, when the automaton switches between q_1 and q_2, it spends *at most* 1 time unit in q_1 and *at least* 1 time unit in q_2. Further since all transitions into q_1 set y to 0, whenever the automaton transitions to q_2, x is set to 0, and y is at most 1. Therefore, the transition to q_3 is never enabled, and q_3 is not reachable. However, perturbing T_A slightly by changing the guard from q_1 to q_2 to $x \leq 2 + \delta$ and the guard from q_2 to q_1 to $y \geq 2 - \delta$ (for some small δ) makes q_3 reachable – in the perturbed automaton, we can ensure that the automaton stays for $1 + n\delta$ units during its n^{th} visit to q_1, and so after visiting q_1 n_* times (for n_* that satisfies $1 + n_*\delta \geq 2$), when we reach q_2, the transition to q_3 will be enabled. Thus, even a small δ-perturbation of T_A, results in an automaton T_A^δ that is not semantically close, as q_3 is reachable in T_A^δ but not reachable in T_A. The difference between T_A and T_A^δ arises when

one considers executions with arbitrarily many transitions. Does the property of semantic closeness hold if one considers executions of bounded number of steps? The answer once again is no. Consider the example timed automaton T_B in Figure 1b. Once again, x, y are clocks and the invariant in every location is $0 \leq x, y \leq 3$. The transition from q_0 to q_1 is not enabled in T_B and so q_1 is not reachable. However, perturbing the guard to $x < 2 + \delta$ and $y > 2 - \delta$, results in an automaton where q_1 is reachable, and hence not semantically close to T_B.

The examples in Figure 1 illustrate that in general syntactic perturbations can result in models that are not semantically close. Any affirmative results, need to account for the subtle issues that arise in the examples of Figure 1. Our main result is that for a fairly general class of hybrid automata, that include hybrid automata with highly non-linear and non-deterministic dynamics, syntactic perturbations are closely related to semantic perturbations. We consider hybrid automata $\mathcal{H}$ all of whose components, like flows and invariants in modes, and guards and resets on transitions, are described using formulas in first-order logic over reals built from constraints of the form $f \geq 0$, where f is a continuous function, and using conjunction, disjunction, and first order quantification. For a formula φ in this logic, its perturbation by δ, φ^δ, is the formula obtained by replacing all atomic constraints $f \geq 0$ in φ by $f + \delta \geq 0$. Using the notion of perturbation of a constraint, we define $\mathcal{H}^\delta$ to be the hybrid automaton obtained from $\mathcal{H}$ by perturbing all constraints φ appearing in $\mathcal{H}$ by δ. We show that for any $\epsilon \in \mathbb{R}_+$ and $k \in \mathbb{N}$, there is a $\delta \in \mathbb{R}_+$ such that $\mathcal{H}^\delta$ is ϵ-simulated for k-steps by $\mathcal{H}$. In other words, every execution ρ of $\mathcal{H}^\delta$ (having at most k discrete transitions) is simulated by an execution ρ' of $\mathcal{H}$ such that the states of ρ and ρ' are within distance ϵ at all times. Our definition of perturbation ensures that $\mathcal{H}$ is always simulated (in the formal sense) by $\mathcal{H}^\delta$. Therefore, we show that $\mathcal{H}$ and $\mathcal{H}^\delta$ are approximately simulation equivalent for k steps. Thus, one way to informally interpret our results is as follows. Let us consider the function $[\![\mathcal{H}]\!]$ that maps an automaton to its transition system semantics. Our results can be seen as saying that $[\![\cdot]\!]$ is a "continuous" map with the metric on the hybrid automata induced by δ perturbations.

The crux of our result is a technical lemma that maybe of independent interest. Consider any formula φ in the logic defined in the previous paragraph. Let us assume that all free variables are constrained to take values from a bounded interval $\mathbb{I}$. If X is the set of free variables of φ, then we can see φ as defining a subset (denoted $[\![\varphi]\!]$) of $\mathbb{I}^X$ in the standard way. We show that, for any formula φ, and any $\epsilon > 0$, there is a $\delta > 0$ such that the set $[\![\varphi^\delta]\!]$ (φ^δ is the δ-perturbation of φ) is contained in the ϵ-ball around $[\![\varphi]\!]$.

Our results on relating syntactic and semantic perturbations have implications in satisfiability checking as well as verification, that served as our initial motivation for studying this problem. Our first application is in the realm of δ-complete decision procedures [10, 11], that take as input a formula in first order logic and a parameter δ, and return either that the formula is unsatisfiable or that a δ syntactic perturbation of it is satisfiable. Note that in the case that the formula is unsatisfiable, but a δ-perturbation of it is satisfiable, the procedure is allowed to return either of the answers. Our results guarantee that if the formula is unsatisfiable, then there is a δ for which the procedure will return `unsat`, and such a δ can be computed by simply starting from 1 and halving it iteratively. This has direct implications on the bounded safety verification using a tool such as `dReach` [12], that uses a δ-complete decision procedure to check the satisfiability of the formula encoding the bounded verification problem of hybrid automata. More precisely, `dReach` takes as input a hybrid automaton $\mathcal{H}$, a bound on the discrete steps k, an unsafe set U and a δ, and outputs either that $\mathcal{H}$ is safe with respect to U and k discrete steps, or that $\mathcal{H}^\delta$ is unsafe. Our results imply that if

$\mathcal{H}$ is safe, then there is a δ for which `dReach` will necessarily conclude safety, and such a δ again can be computed. Finally, these results are relevant in the context of counter-example guided abstraction refinement (CEGAR) based analysis for hybrid automata [26], wherein the above guarantees on bounded safety verification imply that the validation succeeds if the counter-example is spurious and hence the CEGAR loop makes progress.

Related Work

There has been previous work on relating syntactic perturbation of first order logic formulas over reals to their semantic perturbation [8, 24, 25]. These papers show that small syntactic perturbations result in small semantic perturbations. However, the notion of distance between semantic sets is different from ours. The distance between two subsets A, B of $\mathbb{R}^n$ is taken to be the volume of the symmetric difference between A and B. Thus, in [8, 24, 25], the distance between $[\![\varphi]\!]$ and $[\![\psi]\!]$ may be 0 even if $[\![\varphi]\!] \neq [\![\psi]\!]$. This is not true for us, and is one of the reasons our proofs rely significantly on the Bolzano-Weierstrass theorem. In addition, using the observations in [8, 24, 25], one cannot prove that syntactically perturbing a hybrid automaton results in a model that is ϵ-simulation equivalent.

The results in this paper are closely related to *robustness verification* [1, 3, 4, 7, 9, 14, 15, 23, 27, 29]. In robustness verification, the goal is to develop algorithms to determine if a system $\mathcal{H}$ and all its perturbations $\mathcal{H}^\delta$ in some small neighborhood, satisfy the correctness property φ. While establishing the semantic proximity of $\mathcal{H}$ and $\mathcal{H}^\delta$ may help answer the robustness verification question, verification per se, is not the focus of this paper. Our principal goal is to answer the meta-mathematical question of the relationship between syntactic and semantic perturbations of a hybrid automaton.

Another related line of work consists of methods for constructing simplified models of hybrid systems that are semantically close to the original system [13, 17, 18, 20–22] to reduce the complexity of verification. Semantic closeness is expressed using the notions of approximate simulations and bisimulations which establish that every execution of one system can be matched by a corresponding execution of the other system that stays "close" to it. Approximate bisimulation is weaker than bisimulation, however, it allows for the construction of simplified models that are semantically close for a large class of systems [20] under certain stability assumptions.

2 Preliminaries

2.1 Functions and Sets

The set of *natural*, *positive natural*, *real*, *non-negative real*, and *positive real* numbers are represented by $\mathbb{N}$, $\mathbb{N}_+$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_+$, respectively. For any two sets A and B, power set of A is denoted by 2^A, the Cartesian product of A and B is denoted by $A \times B$, and the set of functions from A to B is denoted by $A \to B$ or B^A. For any two functions $f, g \in A \to \mathbb{R}$, and number $r \in \mathbb{R}$, functions $f \pm g \in A \to \mathbb{R}$ given by $a \mapsto f(a) \pm g(a)$, maps a to addition/subtraction of $f(a)$ and $g(a)$, are pointwise addition/subtraction of f and g, function $rf \in A \to \mathbb{R}$ given by $a \mapsto rf(a)$ is the scalar product of r and f, and function $f + r \in A \to \mathbb{R}$ given by $a \mapsto f(a) + r$ is f shifted by r. For any set of variables X, we denote the set of *functions* and *continuous functions* from $\mathbb{R}^X$ to $\mathbb{R}$ by $\mathbb{F}_X$ and $\mathbb{C}_X$, respectively.

Let X and Y be two arbitrary disjoint sets of variables. For any two points $\nu_1 \in \mathbb{R}^X$ and $\nu_2 \in \mathbb{R}^Y$, concatenation of ν_1 with ν_2, denoted by $\nu_1 \frown \nu_2$, is defined to be $\nu \in \mathbb{R}^{X \cup Y}$ that maps x to $\nu_1(x)$ if $x \in X$ and to $\nu_2(x)$, otherwise (note that $\nu_1 \frown \nu_2 = \nu_2 \frown \nu_1$). Similarly,

for any two sets of points $\mathtt{V}_1 \subseteq \mathbb{R}^X$ and $\mathtt{V}_2 \subseteq \mathbb{R}^Y$, $\mathtt{V}_1 \frown \mathtt{V}_2 \coloneqq \{\nu_1 \frown \nu_2 \mid \nu_1 \in \mathtt{V}_1, \nu_2 \in \mathtt{V}_2\}$. We use X' to denote the *primed* version of X (*i.e.* for every $x \in X$, variable x' belongs to X').

For any two numbers $a, b \in \mathbb{R}$, we define $[a, b] \coloneqq \{x \in \mathbb{R} \mid a \leq x \leq b\}$ to be the *interval* of points between a and b. We use $\mathcal{I}$ to denote the set of intervals.

2.2 Extended Metric Space and Distance Functions

Let M be an arbitrary set and $d \in M \times M \to \mathbb{R} \cup \{\infty\}$ be an arbitrary function. Ordered pair (M, d) is called an *extended metric space* and d is called a *distance function* iff for any $x, y, z \in M$ the following conditions hold:

1. $d(x, y) \geq 0$,
2. $d(x, y) = 0 \Leftrightarrow x = y$,
3. $d(x, y) = d(y, x)$, and
4. $d(x, z) \leq d(x, y) + d(y, z)$.

Let X be a finite set of variables, and $M \subseteq \mathbb{R}^X$ be an arbitrary set. A well-known distance function on M, denoted by $\mathtt{d}_\infty(\nu_1, \nu_2)$, maps any two points $\nu_1, \nu_2 \in M$ to $\max\limits_{x \in X} |\nu_1(x) - \nu_2(x)|$.

Let $f \in M \times M \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be an arbitrary function. For any point $p \in M$ and set $A \subseteq M$, we define $f^{\to}(p, A)$ to be $\inf\limits_{a \in A} f(p, a)$; note that if $A = \emptyset$, this means that $f^{\to}(p, A) = \infty$. Intuitively, if f is a distance function then $f^{\to}(p, A)$ is the distance of p from A. Furthermore, for any set $B \subseteq M$, $f^{\to}(A, B)$ is defined to be $\sup\limits_{a \in A} f^{\to}(a, B)$. This means that when $A = \emptyset$, $f^{\to}(A, B) = 0$. Intuitively, if f is a distance function then $f^{\to}(A, B)$ is the *asymmetric distance* from A to B [1]. For any $\epsilon \in \mathbb{R}_{\geq 0}$, $\mathtt{B}^{\epsilon}_{\infty}(A) \coloneqq \{p \in M \mid \mathtt{d}^{\to}_{\infty}(p, A) \leq \epsilon\}$ is the ϵ-ball around A. Also, *closure* of A is denoted by $\mathtt{cl}(A)$ and is defined to be $\mathtt{B}^{0}_{\infty}(A)$. A set is *closed* iff it is equal to its closure. We say $A \subseteq M$ is *bounded* iff $\sup\limits_{a_1, a_2 \in A} \mathtt{d}_\infty(a_1, a_2) \in \mathbb{R}$. Finally, a set is *compact* iff it is closed and bounded.

2.3 Predicates and Perturbations

In this paper we will consider predicates described in first order logic, where the assignment to free variables is constrained to be within a bounded, closed interval. We fix $\mathbb{I}$ to be this interval bounding the domain of all variables, for the rest of this paper. Further for a finite set of variables X, $\mathbb{B}_X$ will denote $\mathbb{I}^X$, *i.e.* the box of dimension $|X|$ defined by interval $\mathbb{I}$.

Let us fix a finite set of variables X. An *atomic predicate with free variables in* X is a constraint of the form $f \geq 0$, where $f \in \mathbb{C}_X$ is a continuous function. We denote the set of atomic propositions by $\mathbb{P}$ and those with X as their set of free variables by $\mathbb{P}_X$. For any two functions $f, g \in \mathbb{F}_X$, we use $f \geq g$, $f \leq g$, and $f = g$, to denote $f - g \geq 0$, $g - f \geq 0$, and $\min(f - g, g - f) \geq 0$, respectively. A *predicate* is defined according to the following BNF rules, where $f \in \mathbb{C}$ is an arbitrary continuous function, x is an arbitrary variable, and $\mathtt{I} \subseteq \mathbb{I}$ is an arbitrary interval:

$$\varphi ::= f \geq 0 \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x \in \mathtt{I} \bullet \varphi \mid \forall x \in \mathtt{I} \bullet \varphi \mid \varphi(X'/X)$$

To simplify notation, instead of writing $\exists x_1, x_2, \ldots, x_n \in \mathtt{I} \bullet \varphi$ and $\forall x_1, x_2, \ldots, x_n \in \mathtt{I} \bullet \varphi$, we write $\exists Y \in \mathtt{I} \bullet \varphi$ and $\forall Y \in \mathtt{I} \bullet \varphi$, where $Y \coloneqq \{x_1, x_2, \ldots, x_n\}$. Formula $\varphi(X'/X)$ is the usual substitution of every free variable $x \in X$ by x', and can be found in a standard logic book like [5, pp. 45-51]. We implicitly assume that before carrying out the substitution, all

[1] $\max\{f^{\to}(A, B), f^{\to}(B, A)\}$ is the *Hausdorff distance* between A and B [19, Section 7.3].

bound variables in φ are renamed using fresh variables that are also distinct from X'. For any predicate φ, the set of *free variables* of φ, denoted by $\texttt{fvar}(\varphi)$, is inductively defined according to the following rules:

1. $\texttt{fvar}(f)$ is X, where f is a function from $\mathbb{R}^X$ to $\mathbb{R}$,
2. $\texttt{fvar}(\varphi \vee \psi)$ and $\texttt{fvar}(\varphi \wedge \psi)$ are defined to be $\texttt{fvar}(\varphi) \cup \texttt{fvar}(\psi)$,
3. $\texttt{fvar}(\exists y \in \texttt{I} \bullet \varphi)$ and $\texttt{fvar}(\forall y \in \texttt{I} \bullet \varphi)$ are defined to be $\texttt{fvar}(\varphi) \setminus \{y\}$, and
4. $\texttt{fvar}(\varphi(X'/X))$ is defined to be $(\texttt{fvar}(\varphi) \setminus X) \cup (\texttt{fvar}(\varphi) \cap X)'$.

For any predicate φ with $X \coloneqq \texttt{fvar}(\varphi)$, we use $[\![\varphi]\!]$ to refer to the subset of points in the box $\mathbb{B}_X$ that satisfy the predicate φ. We denote the set of predicates with Φ and those with X as their set of free variables by Φ_X. We may write $\varphi(X)$ to emphasize $\texttt{fvar}(\varphi) = X$. Also, for any point $\nu \in \mathbb{R}^X$, we use $\nu(X'/X)$ to denote the same point in $\mathbb{R}^{X'}$, *i.e.* a function that maps x' to $\nu(x)$.

For any predicate $\varphi \in \Phi$ and value $\delta \in \mathbb{R}$, *perturbation of φ by δ* is denoted by φ^δ and is a predicate constructed from φ by replacing all atomic predicates of the form $f \geq 0$ with $f + \delta \geq 0$. Note that $[\![\varphi]\!] = [\![\varphi^0]\!]$, and for any $\delta_1 \leq \delta_2 \in \mathbb{R}$, we have $[\![\varphi^{\delta_1}]\!] \subseteq [\![\varphi^{\delta_2}]\!]$. For any two predicates $\varphi_1 \in \Phi_X$ and $\varphi_2 \in \Phi_Y$, and $\delta \in \mathbb{R}$, it is easy to see that $[\![(\varphi_1 \wedge \varphi_2)^\delta]\!] = [\![\varphi_1^\delta \wedge \varphi_2^\delta]\!]$ and $[\![(\varphi_1 \vee \varphi_2)^\delta]\!] = [\![\varphi_1^\delta \vee \varphi_2^\delta]\!]$. Furthermore, if $X = Y$ then $[\![\varphi_1^\delta \wedge \varphi_2^\delta]\!] = [\![\varphi_1^\delta]\!] \cap [\![\varphi_2^\delta]\!]$ and $[\![\varphi_1^\delta \vee \varphi_2^\delta]\!] = [\![\varphi_1^\delta]\!] \cup [\![\varphi_2^\delta]\!]$.

▶ **Definition 1** (Arbitrary Over-Approximation)**.** For any predicate $\varphi \in \Phi$, we say *φ can be arbitrarily over-approximated* iff the following is true:

$$\forall \epsilon \in \mathbb{R}_+ \bullet \exists \delta \in \mathbb{R}_+ \bullet [\![\varphi^\delta]\!] \subseteq \texttt{B}^\epsilon_\infty([\![\varphi]\!])$$

We end this section with an important result that is used many times in our proofs.

▶ **Theorem 2** (Bolzano-Weierstrass [2])**.** *For any finite set of variables X, a bounded set $M \subset \mathbb{R}^X$, and τ an infinite sequence of points in M, τ has a convergent subsequence.*

2.4 Transition Systems and Hybrid Automata

▶ **Definition 3** (Transition Systems)**.** A transition system $\mathcal{T}$ is a tuple $(\texttt{S}, \Sigma, \rightarrow, \texttt{S}^{\mathsf{init}})$ in which

- $\texttt{S}$ is a (possibly infinite) set of *states*,
- Σ is a (possibly infinite) set of *labels*,
- $\rightarrow \subseteq \texttt{S} \times \Sigma \times \texttt{S}$ is a *transition relation*, and
- $\texttt{S}^{\mathsf{init}} \subseteq \texttt{S}$ is a set of *initial states*.

We denote different elements of $\mathcal{T}$ by adding a subscript to their names. For example, we use $\texttt{S}_\mathcal{T}$ to denote the set of states of $\mathcal{T}$. We may omit the subscript whenever it is clear from the context. We write $s \xrightarrow{\sigma} s'$ instead of $(s, \sigma, s') \in \rightarrow$, and $s \rightarrow s'$ to denote $\exists \sigma \in \Sigma \bullet s \xrightarrow{\sigma} s'$.

▶ **Definition 4** (Syntax of Hybrid Automata)**.** A hybrid automaton $\mathcal{H}$ is specified by a tuple $(\texttt{Q}, \texttt{X}, \texttt{E}, \texttt{T}, \texttt{I}, \texttt{F}, \texttt{S}, \texttt{D}, \texttt{R}, \texttt{Q}^{\mathsf{init}}, \texttt{X}^{\mathsf{init}})$ in which

- $\texttt{Q}$ is a non-empty finite set of *locations*.
- $\texttt{X}$ is a non-empty finite set of *variables* that does not contain variable t.
- $\texttt{E}$ is a finite set of *edges*.
- $\texttt{T} \in \mathbb{R}_{\geq 0}$ is a *continuous transition time-bound*. *Wlog.*, we assume $[0, \texttt{T}] \subseteq \mathbb{I}$.
- $\texttt{I} \in \texttt{Q} \rightarrow \Phi_\texttt{X}$ maps each location to a predicate as the *invariant* of that location.
- $\texttt{F} \in \texttt{Q} \rightarrow \Phi_{\texttt{X} \cup \texttt{X}' \cup \{t\}}$, maps each location to a predicate as the *flow* of that location.
- $\texttt{S}, \texttt{D} \in \texttt{E} \rightarrow \texttt{Q}$ map each edge to its *source* and *destination* locations, respectively.
- $\texttt{R} \in \texttt{E} \rightarrow \Phi_{\texttt{X} \cup \texttt{X}'}$ maps each edge to its *transition relation*.

- $\mathtt{Q}^{\mathsf{init}} \in 2^{\mathtt{Q}}$ is a set of *initial locations*.
- $\mathtt{X}^{\mathsf{init}} \in \mathtt{Q}^{\mathsf{init}} \to \Phi_{\mathtt{X}}$ maps each initial location to a predicate as the *initial valuations* of that location.

We denote different elements of $\mathcal{H}$ by adding a subscript to their names. For example, we use $\mathtt{X}_{\mathcal{H}}$ to denote the set of variables of $\mathcal{H}$. We may omit the subscript whenever it is clear from the context. Finally, we define $\mathtt{Inv}(\mathcal{H}) \coloneqq \bigvee_{q\in\mathtt{Q}} \mathtt{I}(q)$ to be the union of invariants in $\mathcal{H}$. Note that $\mathtt{Inv}(\mathcal{H}) \in \Phi_{\mathtt{X}}$.

▶ **Definition 5** (Semantics of Hybrid Automata)**.** Semantics of a hybrid automaton $\mathcal{H}$ is defined using the transition system $[\![\mathcal{H}]\!] = (\mathtt{S}, \Sigma, \to, \mathtt{S}^{\mathsf{init}})$ in which

- $\mathtt{S} \coloneqq \mathtt{Q} \times \mathbb{B}_{\mathtt{X}}$,
- $\Sigma \coloneqq \mathtt{E} \cup [0, \mathtt{T}]$ [2],
- $\mathtt{S}^{\mathsf{init}} \coloneqq \{(q,\nu) \mid q \in \mathtt{Q}^{\mathsf{init}} \wedge \nu \in [\![\mathtt{X}^{\mathsf{init}}(q) \wedge \mathtt{I}(q)]\!]\}$, and
- $\to \coloneqq \to_c \cup \to_d$ where
  - $\to_c$ is the set of *continuous* transitions and for any $r \in \mathbb{R}$ we have $(q,\nu) \xrightarrow{r}_c (q',\nu')$ iff
    1. $r \in [0, \mathtt{T}]$,
    2. $q = q'$,
    3. $\nu \in [\![\mathtt{I}(q)]\!]$,
    4. $\nu' \in [\![\mathtt{I}(q')]\!]$, and
    5. $\nu \frown \nu'(\mathtt{X}'/\mathtt{X}) \frown \{t \mapsto r\} \in [\![\mathtt{F}(q)]\!]$,
  - $\to_d$ is the set of *discrete* transitions and for any $e \in \mathtt{E}$ we have $(q,\nu) \xrightarrow{e}_d (q',\nu')$ iff
    1. $q = \mathtt{S}(e)$,
    2. $q' = \mathtt{D}(e)$,
    3. $\nu \in [\![\mathtt{I}(q)]\!]$,
    4. $\nu' \in [\![\mathtt{I}(q')]\!]$, and
    5. $\nu \frown \nu'(\mathtt{X}'/\mathtt{X}) \in [\![\mathtt{R}(e)]\!]$.

For any two states $s_1 \coloneqq (q_1,\nu_1), s_2 \coloneqq (q_2,\nu_2) \in \mathtt{S}$, we extend definition of $\mathtt{d}_\infty(s_1,s_2)$ to be ∞ if $q_1 \neq q_2$ and $\mathtt{d}_\infty(\nu_1,\nu_2)$ otherwise. For any state $s \in \mathtt{S}$, we define $\mathtt{CPost}_{\mathcal{H}}(s) \coloneqq \{s' \in \mathtt{S} \mid \exists t \in [0,\mathtt{T}] \bullet s \xrightarrow{t} s'\}$. Similarly, for any state $s \in \mathtt{S}$ and $e \in \mathtt{E}$, we define $\mathtt{DPost}^e_{\mathcal{H}}(s) \coloneqq \{s' \in \mathtt{S} \mid s \xrightarrow{e} s'\}$. Also, for any set of states $S \subseteq \mathtt{S}$, we define $\mathtt{CPost}_{\mathcal{H}}(S) \coloneqq \bigcup_{s\in S} \mathtt{CPost}_{\mathcal{H}}(s)$, and $\mathtt{DPost}^e_{\mathcal{H}}(S) \coloneqq \bigcup_{s\in S} \mathtt{DPost}^e_{\mathcal{H}}(s)$. To simplify notation, we may drop the subscript $\mathcal{H}$ if it is clear from the context. Finally, for any number $k \in \mathbb{N}$, we define $\mathtt{reach}_k(\mathcal{H})$ to be the set of states in $\mathtt{S}$ that can be reached from $\mathtt{S}^{\mathsf{init}}$ through *at most* k function applications of $\mathtt{CPost}$ or $\mathtt{DPost}$.

Next, we define a notion of distance between hybrid automata and a related notion of simulation equivalence.

▶ **Definition 6** (k-Step Asymmetric Distance)**.** For any two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ with the same set of locations, variables, and edges, and for any two states $s_1 \in \mathtt{S}$, and $s_2 \in \mathtt{S}$, we define $\mathtt{adist}_0(s_1,s_2)$ to be $\mathtt{d}_\infty(s_1,s_2)$. For any $k \in \mathbb{N}_+$, we inductively define $\mathtt{adist}_k(s_1,s_2)$ to be maximum of the following values:

$$\begin{aligned} &\mathtt{d}_\infty(s_1,s_2) \\ &\mathtt{adist}^{\to}_{k-1}(\mathtt{CPost}_{\mathcal{H}_1}(s_1), \mathtt{CPost}_{\mathcal{H}_2}(s_2)) \\ &\max_{e\in\mathtt{E}} \mathtt{adist}^{\to}_{k-1}\big(\mathtt{DPost}^e_{\mathcal{H}_1}(s_1), \mathtt{DPost}^e_{\mathcal{H}_2}(s_2)\big) \end{aligned}$$

[2] *Wlog.* we assume $\mathtt{E}$ and $\mathbb{R}$ are disjoint.

Whenever, $\mathcal{H}_1$ and/or $\mathcal{H}_2$ are not clear from the context, we use $\underset{\mathcal{H}_1,\mathcal{H}_2}{\mathtt{adist}}{}^{\rightarrow}_{k}$. Finally, we define $\mathtt{adist}^{\rightarrow}_{k}(\mathcal{H}_1,\mathcal{H}_2)$ to be $\underset{\mathcal{H}_1,\mathcal{H}_2}{\mathtt{adist}}{}^{\rightarrow}_{k}(\mathtt{S}^{\mathsf{init}}_{\mathcal{H}_1},\mathtt{S}^{\mathsf{init}}_{\mathcal{H}_2})$.

▶ **Definition 7** (k-Step ϵ-Simulation)**.** For any two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ with the same set of locations, variables, and edges, and value $\epsilon \in \mathbb{R}_{\geq 0}$, a relation $R \subseteq \mathtt{S} \times \mathtt{S}$ is called a 0-*step* ϵ-*simulation* iff $\forall s_1, s_2 \in \mathtt{S} \bullet s_1 R s_2 \Rightarrow \mathtt{d}_\infty(s_1, s_2) < \epsilon$. For any $k \in \mathbb{N}_+$, $R \subseteq \mathtt{S} \times \mathtt{S}$ is called a k-*step* ϵ-*simulation* iff there exists $R' \subseteq \mathtt{S} \times \mathtt{S}$, a $k-1$-step ϵ-simulation, such that for any $s_1, s_2 \in \mathtt{S}$, if $s_1 R s_2$ then $\mathtt{d}_\infty(s_1, s_2) < \epsilon$ and both the following conditions hold:

- $\forall s_1' \in \mathtt{S}, e \in \mathtt{E} \bullet s_1 \xrightarrow{e} s_1' \Rightarrow \exists s_2' \in \mathtt{S} \bullet s_2 \xrightarrow{e} s_2' \wedge s_1' R' s_2'$
- $\forall s_1' \in \mathtt{S}, t \in \mathbb{R} \bullet s_1 \xrightarrow{t} s_1' \Rightarrow \exists s_2' \in \mathtt{S}, t' \in \mathbb{R} \bullet s_2 \xrightarrow{t'} s_2' \wedge s_1' R' s_2'$

We say $\mathcal{H}_1$ is k-*step* ϵ-*similar* to $\mathcal{H}_2$, denoted by $\mathcal{H}_1 \preceq^{\epsilon}_{k} \mathcal{H}_2$ iff there is a k-step ϵ-simulation relation R such that $\forall s_1 \in \mathtt{S}^{\mathsf{init}}_{\mathcal{H}_1} \bullet \exists s_2 \in \mathtt{S}^{\mathsf{init}}_{\mathcal{H}_2} \bullet s_1 R s_2$.

▶ **Proposition 8** (Equivalence of k-Step Distance and ϵ-Simulation)**.** *For any two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$, numbers $k \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_+$, $\mathcal{H}_1 \preceq^{\epsilon}_{k} \mathcal{H}_2$ iff $\mathtt{adist}^{\rightarrow}_{k}(\mathcal{H}_1, \mathcal{H}_2) < \epsilon$.*

▶ **Definition 9** (Perturbation of hybrid automata)**.** For any hybrid automaton $\mathcal{H}$ and perturbation $\delta \in \mathbb{R}_{\geq 0}$, perturbation of $\mathcal{H}$ by δ, denoted by $\mathcal{H}^\delta$, is obtained from $\mathcal{H}$ by perturbing all of its predicates by δ. More precisely, $\mathtt{Q}_{\mathcal{H}^\delta} \coloneqq \mathtt{Q}_{\mathcal{H}}$, $\mathtt{X}_{\mathcal{H}^\delta} \coloneqq \mathtt{X}_{\mathcal{H}}$, $\mathtt{E}_{\mathcal{H}^\delta} \coloneqq \mathtt{E}_{\mathcal{H}}$, $\mathtt{T}_{\mathcal{H}^\delta} \coloneqq \mathtt{T}_{\mathcal{H}}$, $\mathtt{S}_{\mathcal{H}^\delta} \coloneqq \mathtt{S}_{\mathcal{H}}$, $\mathtt{D}_{\mathcal{H}^\delta} \coloneqq \mathtt{D}_{\mathcal{H}}$, and $\mathtt{Q}^{\mathsf{init}}_{\mathcal{H}^\delta} \coloneqq \mathtt{Q}^{\mathsf{init}}_{\mathcal{H}}$. Furthermore, for any $q \in \mathtt{Q}$ and $e \in \mathtt{E}$, we have $\mathtt{I}_{\mathcal{H}^\delta}(q) \coloneqq (\mathtt{I}_{\mathcal{H}}(q))^\delta$, $\mathtt{F}_{\mathcal{H}^\delta}(q) \coloneqq (\mathtt{F}_{\mathcal{H}}(q))^\delta$, $\mathtt{R}_{\mathcal{H}^\delta}(e) \coloneqq (\mathtt{R}_{\mathcal{H}}(e))^\delta$, and if $q \in \mathtt{Q}^{\mathsf{init}}$ then $\mathtt{X}^{\mathsf{init}}_{\mathcal{H}^\delta}(q) \coloneqq (\mathtt{X}^{\mathsf{init}}_{\mathcal{H}}(q))^\delta$.

It is easy to see, for any hybrid automaton $\mathcal{H}$, perturbation $\delta \in \mathbb{R}_{\geq 0}$, states $s, s' \in \mathtt{S}$, time $t \in \mathbb{R}$, and edge $e \in \mathtt{E}$, we have $s \xrightarrow{t}_{[\![\mathcal{H}]\!]} s'$ implies $s \xrightarrow{t}_{[\![\mathcal{H}^\delta]\!]} s'$, and $s \xrightarrow{e}_{[\![\mathcal{H}]\!]} s'$ implies $s \xrightarrow{e}_{[\![\mathcal{H}^\delta]\!]} s'$. Also, if $\delta = 0$ then $[\![\mathcal{H}^\delta]\!] = [\![\mathcal{H}]\!]$. Finally, for any $k \in \mathbb{N}$, it is easy to see $\mathtt{adist}^{\rightarrow}_{k}(\mathcal{H}, \mathcal{H}^\delta) = 0$. However, $\mathtt{adist}^{\rightarrow}_{k}(\mathcal{H}^\delta, \mathcal{H})$ is not necessarily 0.

2.5 Encoding States, CPost, and DPost as Predicates

Since in Section 2.3, we only allow variables to be quantified over intervals, *wlog.*, for the rest of this paper and for any hybrid automaton $\mathcal{H}$, we assume

1. $\mathtt{Q}_{\mathcal{H}}$ is the set $\{0, 1, \ldots, |\mathtt{Q}| - 1\}$, and
2. $\mathtt{X}_{\mathcal{H}}$ contains variable $x_{\mathtt{Q}}$.

Variable $x_{\mathtt{Q}}$ is used to model the current location. For any location $q \in \mathtt{Q}$ and edge $e \in \mathtt{E}$, *wlog.*, we assume

- $[\![\mathtt{I}(q)]\!] = [\![x_{\mathtt{Q}} = q \wedge \mathtt{I}(q)]\!]$
- $[\![\mathtt{F}(q)]\!] = [\![x_{\mathtt{Q}} = q \wedge \mathtt{F}(q)]\!]$
- $[\![\mathtt{R}(e)]\!] = [\![x_{\mathtt{Q}} = \mathtt{S}(e) \wedge x'_{\mathtt{Q}} = \mathtt{D}(e) \wedge \mathtt{R}(e)]\!]$
- $q \in \mathtt{Q}^{\mathsf{init}} \Rightarrow [\![\mathtt{X}^{\mathsf{init}}(q)]\!] = [\![x_{\mathtt{Q}} = q \wedge \mathtt{X}^{\mathsf{init}}(q)]\!]$

Let $\varphi \in \Phi_{\mathtt{X}}$ be a predicate encoding a subset of states in $\mathtt{S}_{[\![\mathcal{H}]\!]}$. For any edge $e \in \mathtt{E}$, we define $\mathtt{DPost}^{e}_{\mathcal{H}}(\varphi)$ to be $(\exists \mathtt{X} \in \mathbb{I} \bullet \varphi \wedge \psi_1(\mathtt{X}) \wedge \psi_2(\mathtt{X}') \wedge \psi_3(\mathtt{X} \cup \mathtt{X}'))(\mathtt{X}/\mathtt{X}')$, where

1. $\psi_1 \coloneqq \mathtt{I}(\mathtt{S}(e))$ ensures source location is correct and its invariant is satisfied,
2. $\psi_2 \coloneqq \mathtt{I}(\mathtt{D}(e))(\mathtt{X}'/\mathtt{X})$ ensures destination location is correct and its invariant is satisfied, and
3. $\psi_3 \coloneqq \mathtt{R}(e)$ ensures transition relation is satisfied.

At the end of formula, substituting every free variable $x' \in \mathtt{X}'$ by $x \in \mathtt{X}$, ensures all free variables of $\mathtt{DPost}^e_{\mathcal{H}}(\varphi)$ belong to $\mathtt{X}$ [3]. Note that $\mathtt{DPost}^e_{\mathcal{H}}(\varphi)$ belongs to $\Phi_{\mathtt{X}}$.

For any $q \in \mathtt{Q}$, we define $\mathtt{CPost}^q_{\mathcal{H}}(\varphi)$ to be $(\exists \mathtt{X} \in \mathbb{I}, t \in [0, \mathtt{T}] \bullet \varphi \wedge \psi_1(\mathtt{X}) \wedge \psi_2(\mathtt{X}') \wedge \psi_3(\mathtt{X} \cup \mathtt{X}' \cup \{t\}))(\mathtt{X}/\mathtt{X}')$, where

1. $\psi_1 \coloneqq \mathtt{I}(q)$ ensures source location is correct and its invariant is satisfied,
2. $\psi_2 \coloneqq \mathtt{I}(q)(\mathtt{X}'/\mathtt{X})$ ensures location does not change and its invariant will remain satisfied, and
3. $\psi_3 \coloneqq \mathtt{F}(q)$ ensures flow relation is satisfied.

At the end of formula, substituting every free variable $x' \in \mathtt{X}'$ by $x \in \mathtt{X}$, ensures all free variables of $\mathtt{CPost}^q_{\mathcal{H}}(\varphi)$ belong to $\mathtt{X}$. We also define $\mathtt{CPost}_{\mathcal{H}}(\varphi)$ to be $\bigvee_{q \in \mathtt{Q}} \mathtt{CPost}^q_{\mathcal{H}}(\varphi)$. Note that $\mathtt{CPost}^q_{\mathcal{H}}(\varphi)$ and $\mathtt{CPost}_{\mathcal{H}}(\varphi)$ are both members of $\Phi_{\mathtt{X}}$. Finally, it is easy to see that for any $e \in \mathtt{E}$ and $\varphi \in \Phi_{\mathtt{X}}$ we have $[\![\mathtt{DPost}^e_{\mathcal{H}}(\varphi)]\!] = \mathtt{DPost}^e_{\mathcal{H}}([\![\varphi]\!])$ and $[\![\mathtt{CPost}_{\mathcal{H}}(\varphi)]\!] = \mathtt{CPost}_{\mathcal{H}}([\![\varphi]\!])$ [9, 12].

3 Arbitrary Over-Approximation of a Predicate

The crux of our result on the relation between syntactic and semantic perturbations of hybrid automata relies on the observation that we can arbitrarily over-approximate any predicate, which is formalized in Theorem 10.

▶ **Theorem 10.** *For any set of variables X and predicate $\varphi \in \Phi_X$, φ can be arbitrarily over-approximated.*

The proof is by induction on the structure of φ, and relies on an important topological property of the formulas $\varphi \in \Phi_X$, namely, that the set of satisfiable assignments represented by the formulas is closed. The proofs have been moved to the appendix in the interest of space.

▶ **Theorem 11.** *For any finite set of variables X and predicate $\varphi \in \Phi_X$, the set $[\![\varphi]\!]$ is closed.*

Note that requiring quantified variables to range over bounded intervals is necessary here. For example, let $P \coloneqq (xy = 1)$. Clearly $[\![P]\!]$ is a closed set. However, $[\![\exists y \in \mathbb{R} \bullet P]\!] = \{x \in \mathbb{I} \mid x \neq 0\}$ may not be closed anymore (if $0 \in \mathbb{I}$). Also, even though we do not allow strict inequalities in the syntax, there are typically ways in which a constraint like $f(X) > 0$ can be encoded. However, these different ways are effectively ruled out. Here are some examples to illustrate this point.

1. The formula $\exists \epsilon \in \mathbb{R}_+ \bullet f(X) - \epsilon \geq 0$ is ruled out because we only allow quantification over closed intervals.
2. Consider $\exists y \in [0, 1] \bullet 0 \leq y \leq 1 \wedge f(X) - g(y) \geq 0$, where $g(y) = 1$ if $y = 0$, and $g(y) = y$ if $y \neq 0$. This is also not allowed because the function g is discontinuous.
3. Finally, $\exists y \in \mathbb{R} \bullet y \geq 1 \wedge f(X) - \frac{1}{y} \geq 0$ is ruled out because we only allow quantification over bounded intervals.

[3] As we mentioned in Section 2.3, in order to prevent variable capture that may happen as a result of a substitution, before every substitution, all bound variables are renamed to some fresh variable distinct from those in $\mathtt{X}$ or $\mathtt{X}'$.

4 Arbitrary Over-Approximation of Hybrid Automata

In Section 2.4, we defined hybrid automata, its semantics, its perturbation, as well as bounded step distance function. In Theorem 10, we proved that for any set of variables X, every predicate in Φ_X can be arbitrarily over-approximated. Our main result of this section is that for any bounded number of steps, any hybrid automaton can be arbitrarily over-approximated. In other words, it is always possible to make sure a hybrid automaton and its perturbation behave similarly for at least k steps. Theorem 14 formally states this result. Before, we present this theorem and its proof, we will introduce two lemmas that will help us prove our main result.

▶ **Lemma 12** (CPost and DPost are Continuous). *For any hybrid automaton $\mathcal{H}$ and edge $e \in \mathtt{E}$, functions $\mathtt{CPost}_{\mathcal{H}}$ and $\mathtt{DPost}^e_{\mathcal{H}}$ are continuous with respect to $\mathtt{d}^{\rightarrow}_{\infty}$ and perturbation. More precisely,*

$$\forall \nu \in [\![\mathtt{Inv}(\mathcal{H})]\!], \epsilon \in \mathbb{R}_+ \bullet \exists \delta \in \mathbb{R}_+ \bullet \forall \nu' \in [\![\mathtt{Inv}(\mathcal{H}^\delta)]\!] \bullet \mathtt{d}_\infty(\nu', \nu) \leq \delta \Rightarrow$$
$$\mathtt{d}^{\rightarrow}_{\infty}(\mathtt{CPost}_{\mathcal{H}^\delta}(\nu'), \mathtt{CPost}_{\mathcal{H}}(\nu)) < \epsilon \wedge \mathtt{d}^{\rightarrow}_{\infty}(\mathtt{DPost}^e_{\mathcal{H}^\delta}(\nu'), \mathtt{DPost}^e_{\mathcal{H}}(\nu)) < \epsilon$$

Proof. We prove this lemma for CPost. Proof of $\mathtt{DPost}^e$ can be obtained by replacing every CPost with $\mathtt{DPost}^e$. For the purpose of contradiction suppose this result does not hold for some $\nu \in [\![\mathtt{Inv}(\mathcal{H})]\!]$, $\epsilon \in \mathbb{R}_+$. For any $n \in \mathbb{N}$, define $\delta_n := \frac{1}{n+1}$, and let $\nu'_n \in [\![\mathtt{Inv}(\mathcal{H}^{\delta_n})]\!]$ be a point for which $\mathtt{d}_\infty(\nu'_n, \nu) \leq \delta_n$ and $\mathtt{d}^{\rightarrow}_{\infty}(\mathtt{CPost}_{\mathcal{H}^{\delta_n}}(\nu'_n), \mathtt{CPost}_{\mathcal{H}}(\nu)) \geq \epsilon$ are both true.

Define $\varphi := \bigwedge_{x \in \mathtt{X}}(x = \nu(x))$. Since $\varphi \in \mathbb{P}_{\mathtt{X}}$, we know $\mathtt{CPost}_{\mathcal{H}}(\varphi) \in \Phi_{\mathtt{X}}$. Use Theorem 10 and fix $n \in \mathbb{N}$ such that $[\![(\mathtt{CPost}_{\mathcal{H}}(\varphi))^{\delta_n}]\!] \subseteq \mathtt{B}^{\epsilon/2}_\infty([\![\mathtt{CPost}_{\mathcal{H}}(\varphi)]\!])$. Definition of $\mathtt{CPost}_{\mathcal{H}}(\varphi)$ ensures the following is true, because the two sides of the equality are syntactically the same.

$$\forall \delta \in \mathbb{R}_{\geq 0} \bullet [\![\mathtt{CPost}_{\mathcal{H}^\delta}(\varphi^\delta)]\!] = [\![(\mathtt{CPost}_{\mathcal{H}}(\varphi))^\delta]\!]$$

Therefore, knowing $\forall n \in \mathbb{N} \bullet \nu'_n \in [\![\varphi^{\delta_n}]\!]$, we conclude

$$\mathtt{CPost}_{\mathcal{H}^{\delta_n}}(\nu'_n) \subseteq \mathtt{CPost}_{\mathcal{H}^{\delta_n}}([\![\varphi^{\delta_n}]\!]) = [\![\mathtt{CPost}_{\mathcal{H}^{\delta_n}}(\varphi^{\delta_n})]\!] \subseteq$$
$$\mathtt{B}^{\epsilon/2}_\infty([\![\mathtt{CPost}_{\mathcal{H}}(\varphi)]\!]) = \mathtt{B}^{\epsilon/2}_\infty(\mathtt{CPost}_{\mathcal{H}}(\nu))$$

This means $\mathtt{d}^{\rightarrow}_{\infty}(\mathtt{CPost}_{\mathcal{H}^{\delta_n}}(\nu'_n), \mathtt{CPost}_{\mathcal{H}}(\nu)) \leq \frac{\epsilon}{2}$, which is a contradiction. ◀

▶ **Lemma 13** (adist is Continuous). *For any hybrid automaton $\mathcal{H}$ and number $k \in \mathbb{N}$, distance function $\mathtt{adist}^{\rightarrow}_k$ is continuous with respect to $\mathtt{d}^{\rightarrow}_{\infty}$ and perturbation. More precisely,*

$$\forall \varphi \in \Phi_{\mathtt{X}}, \epsilon \in \mathbb{R}_+ \bullet \exists \delta \in \mathbb{R}_+ \bullet \forall \psi \in \Phi_{\mathtt{X}} \bullet$$
$$[\![\varphi]\!] \subseteq [\![\mathtt{Inv}(\mathcal{H})]\!] \wedge [\![\psi]\!] \subseteq [\![\mathtt{Inv}(\mathcal{H}^\delta)]\!] \wedge \mathtt{d}^{\rightarrow}_{\infty}([\![\psi]\!], [\![\varphi]\!]) \leq \delta \Rightarrow \underset{\mathcal{H}^\delta, \mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_k([\![\psi]\!], [\![\varphi]\!]) < \epsilon$$

Proof. Proof is by induction on k. Base of induction, where $k = 0$, is trivial because $\mathtt{adist}^{\rightarrow}_0$ and $\mathtt{d}^{\rightarrow}_{\infty}$ are identical. For the purpose of contradiction, suppose the inductive step does not hold for some $\epsilon \in \mathbb{R}_+$ and $\varphi \in \Phi_{\mathtt{X}}$, where $[\![\varphi]\!] \subseteq [\![\mathtt{Inv}(\mathcal{H})]\!]$. For any $n \in \mathbb{N}$, define $\delta_n := \frac{\epsilon}{n+3}$, and let $\psi_n \in \Phi_{\mathtt{X}}$ be a predicate for which $[\![\psi_n]\!] \subseteq [\![\mathtt{Inv}(\mathcal{H}^{\delta_n})]\!]$, $\mathtt{d}^{\rightarrow}_{\infty}([\![\psi_n]\!], [\![\varphi]\!]) \leq \delta_n \leq \frac{\epsilon}{3}$, and $\underset{\mathcal{H}^\delta, \mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_{k+1}([\![\psi_n]\!], [\![\varphi]\!]) \geq \epsilon$ are all true. We immediately know $[\![\varphi]\!] \neq \emptyset \wedge \forall n \in \mathbb{N} \bullet \wedge [\![\psi_n]\!] \neq \emptyset$. Using the properties of supremum and infimum, for any $n \in \mathbb{N}$, let $\nu'_n \in [\![\psi_n]\!]$ be such that

$$\forall \nu \in [\![\varphi]\!] \bullet \mathtt{adist}_{k+1}(\nu'_n, \nu) \geq \frac{\epsilon}{2}$$

Let $\nu_n \in [\![\varphi]\!] \cap \mathtt{B}^{\delta_n}_\infty(\nu'_n)$ be an arbitrary point (the set $[\![\varphi]\!] \cap \mathtt{B}^{\delta_n}_\infty(\nu'_n)$ is never empty). Using definition of $\mathtt{adist}_{k+1}$, we know at least one of the following conditions is true:

$$\mathtt{adist}^{\rightarrow}_k(\mathtt{CPost}_{\mathcal{H}^{\delta_n}}(\nu'_n), \mathtt{CPost}_{\mathcal{H}}(\nu_n)) \geq \frac{\epsilon}{2}$$
$$\mathtt{adist}^{\rightarrow}_k(\mathtt{DPost}^e_{\mathcal{H}^{\delta_n}}(\nu'_n), \mathtt{DPost}^e_{\mathcal{H}}(\nu_n)) \geq \frac{\epsilon}{2}, \text{ for some } e \in \mathtt{E}$$

We assume the first condition is true. Proof of the other case can be obtained by replacing every $\texttt{CPost}$ with $\texttt{DPost}^e$ in the rest of this proof. Define $\varphi'_n \coloneqq \bigwedge_{x\in\mathtt{X}}(x = \nu_n(x))$ and $\psi'_n \coloneqq \bigwedge_{x\in\mathtt{X}}(x = \nu'_n(x))$. We know $\varphi'_n, \psi'_n \in \mathbb{P}_{\mathtt{X}}$, and hence $\varphi''_n \coloneqq \texttt{CPost}_{\mathcal{H}}(\varphi'_n)$ and $\psi''_n \coloneqq \texttt{CPost}_{\mathcal{H}^{\delta_n}}(\psi'_n)$ are both members of $\Phi_{\mathtt{X}}$. For any $n \in \mathbb{N}$, we know $[\![\varphi''_n]\!] \subseteq [\![\texttt{Inv}(\mathcal{H})]\!]$ and $[\![\psi''_n]\!] \subseteq [\![\texttt{Inv}(\mathcal{H}^{\delta_n})]\!]$ are both true. Use induction hypothesis and find $N \in \mathbb{N}$ such that

$$\forall n \in \mathbb{N} \bullet \mathtt{d}^{\rightarrow}_{\infty}([\![\psi''_n]\!], [\![\varphi''_n]\!]) \leq \delta_N \Rightarrow \underset{\mathcal{H}^{\delta_n},\mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_{k}([\![\psi''_n]\!], [\![\varphi''_n]\!]) < \frac{\epsilon}{2}$$

We conclude the following which is in contradiction with Lemma 12.

$$\forall n \in \mathbb{N} \bullet \nu_n \in [\![\texttt{Inv}(\mathcal{H})]\!] \wedge \nu'_n \in [\![\texttt{Inv}(\mathcal{H}^{\delta_n})]\!] \wedge \mathtt{d}^{\rightarrow}_{\infty}(\nu'_n, \nu_n) \leq \delta_n \wedge$$
$$\mathtt{d}^{\rightarrow}_{\infty}(\texttt{CPost}_{\mathcal{H}^{\delta_n}}(\nu'_n), \texttt{CPost}_{\mathcal{H}}(\nu_n)) > \delta_N$$

◀

▶ **Theorem 14.** *For any hybrid automaton $\mathcal{H}$, bounded step $n \in \mathbb{N}$, and $\epsilon \in \mathbb{R}_+$, there is $\delta \in \mathbb{R}_+$ such that*

$$\forall k \in \{0, \ldots, n\} \bullet \underset{\mathcal{H}^{\delta},\mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_{k}\left(\mathtt{S}^{\mathsf{init}}_{[\![\mathcal{H}^{\delta}]\!]}, \mathtt{S}^{\mathsf{init}}_{[\![\mathcal{H}]\!]}\right) < \epsilon$$

Proof. Since n is given in advance and $\underset{\mathcal{H}^{\delta},\mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_{k}(\cdot,\cdot)$ is a non-decreasing function with respect to k, it is enough to prove this result for only $k \coloneqq n$. Let $\varphi \coloneqq \texttt{Inv}(\mathcal{H}) \wedge \bigvee_{q\in\mathbb{Q}^{\mathsf{init}}_{\mathcal{H}}} \mathtt{X}^{\mathsf{init}}(q)$ be a formula in $\Phi_{\mathtt{X}}$ that represents initial states of $\mathcal{H}$. We know $\mathtt{S}^{\mathsf{init}}_{[\![\mathcal{H}]\!]} = [\![\varphi]\!]$ and $\forall \delta \in \mathbb{R} \bullet \mathtt{S}^{\mathsf{init}}_{[\![\mathcal{H}^{\delta}]\!]} = [\![\varphi^{\delta}]\!]$. Using Lemma 13, find $\eta \in \mathbb{R}_+$ such that $\forall \delta \in \mathbb{R} \bullet \mathtt{d}^{\rightarrow}_{\infty}([\![\varphi^{\delta}]\!], [\![\varphi]\!]) < \eta \Rightarrow \underset{\mathcal{H}',\mathcal{H}}{\mathtt{adist}}{}^{\rightarrow}_{k}([\![\varphi^{\delta}]\!], [\![\varphi]\!]) < \epsilon$. Complete the proof by using Theorem 10 and finding $\delta_k \in \mathbb{R}_+$ such that $\mathtt{d}^{\rightarrow}_{\infty}([\![\varphi^{\delta_k}]\!], [\![\varphi]\!]) < \eta$. ◀

▶ **Corollary 15** (Arbitrary Over-Approximation of Reachable Sets)**.** *For any hybrid automaton $\mathcal{H}$, reachable set of $\mathcal{H}$ can be arbitrarily over-approximated. More precisely,*

$$\forall \epsilon \in \mathbb{R}_+, k \in \mathbb{N} \bullet \exists \delta \in \mathbb{R}_+ \bullet \mathtt{reach}_k\left(\mathcal{H}^{\delta}\right) \subseteq \mathtt{B}^{\epsilon}_{\infty}(\mathtt{reach}_k(\mathcal{H}))$$

Proof. Immediate from Theorem 14. ◀

▶ **Corollary 16** (Bounded ϵ-Simulation)**.** *For any hybrid automaton $\mathcal{H}$, $k \in \mathbb{N}$, and $\epsilon \in \mathbb{R}_+$, there is $\delta \in \mathbb{R}_+$ such that $\mathcal{H}$ and $\mathcal{H}^{\delta}$ ϵ-simulate each other for at least k steps.*

$$\forall \epsilon \in \mathbb{R}_+, k \in \mathbb{N} \bullet \exists \delta \in \mathbb{R}_+ \bullet \mathcal{H}^{\delta} \preceq^{\epsilon}_{k} \mathcal{H} \wedge \mathcal{H} \preceq^{\epsilon}_{k} \mathcal{H}^{\delta}$$

Proof. $\mathcal{H} \preceq^{\epsilon}_{k} \mathcal{H}^{\delta}$ is trivially true for any $\epsilon \in \mathbb{R}_{\geq 0}$ and $k \in \mathbb{N}$. ϵ-simulation of $\mathcal{H}^{\delta}$ by $\mathcal{H}$ for at least k steps is immediate from Theorem 14 and Proposition 8. ◀

5 Applications to Safety Model Checking

In Section 3 and Section 4 we proved some results about the ability to arbitrarily over-approximate sets and hybrid automata. In this section, we demonstrate three applications of those results.

5.1 Co-completeness of δ-Complete Decision Procedures

It is well-known that the first order theory of reals with addition, multiplication, and trigonometric functions is undecidable (one can easily encode integers in this logic) [5, Chapter 3]. Authors in [10, 11] came up with an interesting compromise in their decision procedure for *checking satisfiability* of these formulas. For any formula $\varphi \in \Phi$, their so called δ-complete decision procedure returns either $\texttt{unsat}$ or δ-$\texttt{sat}$. If the output is $\texttt{unsat}$, we know $[\![\varphi]\!] = \emptyset$. However, if the output is δ-$\texttt{sat}$, we only know $[\![\varphi^{\delta}]\!] \neq \emptyset$. Parameter $\delta \in \mathbb{R}_+$

is an input to their algorithm and can be made arbitrary small. The algorithm in [10, 11], imposes one additional constraint on formulas: Every function used in atomic propositions of a formula must be type 2 computable [6, 16, 28]. Intuitively, a function is type 2 computable iff an arbitrary approximation of its value can be computed from arbitrary approximations of its inputs. This condition is stronger than continuity requirement that we have in this paper. Nevertheless, it still includes arithmetic, trigonometric, logarithmic, exponential, absolute value, minimum, and maximum functions as well as solutions of many ordinary differential equations.

Observe that δ-complete decision procedures provide approximate answers to the satisfiability question. So even if a formula φ is unsatisfiable, there is no guarantee that the δ-decision procedure will answer `unsat`. The procedure is guaranteed to answer `unsat` only if it is called with a δ such that both φ *and* φ^δ are unsatisfiable. Does such a δ exist for all unsatisfiable φ? Can it be computed? Theorem 10 answers in the affirmative for both these questions – the value of δ can also be computed by repeatedly calling a δ-decision procedure for smaller and smaller δ. Theorem 17 formalizes this result.

▶ **Theorem 17.** *Let $\varphi \in \Phi$ be an arbitrary predicate with $[\![\varphi]\!] = \emptyset$ and every function is type 2 computable. One can compute $\delta \in \mathbb{R}_+$ for which $[\![\varphi^\delta]\!] = \emptyset$.*

We require that the predicates only use non-strict inequalities, whereas no such restriction is imposed on the inputs to the δ-decision procedures [10, 11]. However, predicates with strict inequalities can be easily handled using predicates with non-strict inequalities. Consider φ, a formula with some strict inequalities, and let φ' be the formula obtained by changing each strict inequality in φ to the corresponding non-strict one. Observe that if a δ-complete decision procedure says φ' is `unsat`, we know `unsat` is a valid output for φ as well. More interestingly, output δ-`sat` for φ' means 2δ-`sat` is a valid output for φ.

5.2 Completeness of dReach

δ-complete decision procedures have been used in [12] to develop a tool called `dReach` for bounded time, bounded step safety model checking of non-linear hybrid automaton $\mathcal{H}$. The tool first takes $\mathcal{H}$ and unsafe predicate $U \in \Phi_{\mathtt{X}}$ as input, and then encodes the safety problem into a satisfiability problem of a formula that is supported by the algorithm in [10, 11]. Finally, it checks its satisfiability and outputs the results. Possible outputs are either `unsat`, which means $\mathtt{reach}_k(\mathcal{H}) \cap [\![U]\!] = \emptyset$, and δ-`sat`, which means $\mathtt{reach}_k(\mathcal{H}^\delta) \cap [\![U^\delta]\!] \neq \emptyset$. However, in case the input hybrid automaton is k-step safe, it is not known if a value of δ exists, such that $\mathtt{reach}_k(\mathcal{H}^\delta) \cap [\![U^\delta]\!]$ remains an empty set. Theorem 18 is an immediate consequence of Corollary 15 and Theorem 17.

▶ **Theorem 18.** *Let $\mathcal{H}$ be a hybrid automaton and $U \in \Phi_{\mathtt{X}}$ be an unsafe predicate. If syntax of $\mathcal{H}$ and U is supported by* `dReach` *then for any $k \in \mathbb{N}$, if $\mathcal{H}$ is k-step safe then one can compute $\delta \in \mathbb{R}_+$ for which* `dReach` *answers* `unsat` *(*i.e. *safe).*

Note that the formulas considered here are more general that what is supported in `dReach`. As argued in the previous section, our restriction to non-strict inequalities does not constrain things when using δ-complete decision procedures. Flows specified using ordinary differential equations in `dReach`, define continuous functions and thus are handled by our results. We could also easily change the formula describing the continuous post predicate by requiring $\forall t \in [0, \mathtt{T}] \bullet \mathtt{I}(q)(f(t))$, to ensure that the continuous state satisfies the invariant at all times during a continuous transition.

δ-complete decision procedures can also be used for counter-example validation in a CEGAR-based model checking framework [26]. Theorem 18 can also be used to provide guarantees of progress for such tools – if a counter-example is spurious then such δ-decision procedures are guaranteed to discover it.

6 Conclusion

We investigated whether syntactic perturbations of hybrid automata whose elements are specified in a logic, are semantically close. Such a result does not hold in general as illustrated by the timed automata in Figures 1a and 1b. We identify a fairly general class of systems for which, for every $\epsilon > 0$ and k, there is a $\delta > 0$ such that δ-syntactic perturbations are ϵ-simulation equivalent upto k-discrete steps. These results are important in the context of providing certain theoretical guarantees on δ-decision procedures and bounded verification using the same procedures.

Our results about the semantic closeness of δ-syntactic perturbations only apply when one considers a bounded number of discrete jumps. It seems unlikely that such a result can be extended when there is no a priori bound on the number of discrete steps; this is illustrated by the timed automaton in Figure 1a. One interesting future direction consists of exploring the computability issues with respect to the continuity property of the semantics. More precisely, given an ϵ, can we effectively compute a δ such that δ syntactic perturbation on the hybrid automaton lead to at most ϵ deviations in the semantics.

References

1 E. Asarin and A. Bouajjani. Perturbed turing machines and hybrid systems. In *LICS*, pages 269–278, 2001.

2 R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis, 4th Edition*. John Wiley & Sons, Incorporated, 2011.

3 P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In *Proceedings of LATIN*, pages 238–249, 2006.

4 Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In *Proceedings of FORMATS*, pages 97–112, 2011.

5 Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer-Verlag, 2007.

6 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. *A Tutorial on Computable Analysis*, pages 425–491. Springer New York, New York, NY, 2008.

7 Martin de Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1):45–84, 2008.

8 Peter Franek, Stefan Ratschan, and Piotr Zgliczynski. Quasi-decidability of a fragment of the first-order theory of real numbers. *Journal of Automated Reasoning*, 57(2):157–185, Aug 2016.

9 Martin Fränzle. *Analysis of Hybrid Systems: An Ounce of Realism Can Save an Infinity of States*, pages 126–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

10 Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-completee decision procedures for satisfiability over the reals. In *IJCAR*, pages 286–300, Berlin, Heidelberg, 2012. Springer-Verlag.

11 Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. Delta-decidability over the reals. In *LICS*, pages 305–314. IEEE Computer Society, 2012.

12 Sicun Gao, Soonho Kong, Wei Chen, and Edmund M. Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *CoRR*, abs/1404.7171, 2014.

13 Antoine Girard. Approximately bisimilar abstractions of incrementally stable finite or infinite dimensional systems. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pages 824–829, 2014.

14 Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. *Robust timed automata*, pages 331–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

15 Thomas A. Henzinger and Jean-François Raskin. *Robust Undecidability of Timed and Hybrid Systems*, pages 145–159. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

16 K.I. Ko. *Complexity theory of real functions*. Progress in theoretical computer science. Birkhäuser, 1991.

17 Ruggero Lanotte and Simone Tini. Taylor approximation for hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 402–416, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

18 Ruggero Lanotte and Simone Tini. Taylor approximation for hybrid systems. *Information and Computation*, 205(11):1575–1607, 2007.

19 M. O'Searcoid. *Metric Spaces*. Springer Undergraduate Mathematics Series. Springer London, 2006.

20 Giordano Pola, Antoine Girard, and Paulo Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

21 Pavithra Prabhakar and Mahesh Viswanathan. A dynamic algorithm for approximate flow computations. In *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*, pages 133–142, 2011.

22 Pavithra Prabhakar, Vladimeros Vladimerou, Mahesh Viswanathan, and Geir E. Dullerud. Verifying tolerant systems using polynomial approximations. In *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 181–190, 2009.

23 Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

24 Stefan Ratschan. Quantified constraints under perturbation. *Journal of Symbolic Computation*, 33(4):493–505, 2002.

25 Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Logic*, 7(4):723–748, 2006.

26 Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. *HARE: A Hybrid Abstraction Refinement Engine for Verifying Non-linear Hybrid Automata*, pages 573–588. Springer, 2017.

27 Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Robust model checking of timed automata under clock drifts. In *HSCC*, pages 153–162. ACM, 2017.

28 K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2000.

29 M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robustness and implementability of timed automata. In *Proceedings of FORMATS*, pages 118–133, 2004.

A Proofs

In this section we present all the proofs that we skipped in the main section of the paper.

▶ **Theorem 11**. *For any finite set of variables X and predicate $\varphi \in \Phi_X$, the set $[\![\varphi]\!]$ is closed.*

Proof. Immediate from an induction on the structure of φ, using Lemma 19, Lemma 20, and Lemma 21 (closedness of finite disjunctions and closedness of finite conjunctions of closed sets are trivial facts). ◀

▶ **Theorem 10**. *For any set of variables X and predicate $\varphi \in \Phi_X$, φ can be arbitrarily over-approximated.*

Proof. Immediate from an induction on the structure of φ using Lemma 22, Lemma 23, Lemma 24, Lemma 25, and Lemma 26. ◀

▶ **Lemma 19.** *For any finite set of variables X and predicate $P \in \mathbb{P}_X$, $[\![P]\!]$ is closed.*

Proof. This is a trivial proof once we noticed all functions in P are continuous and all inequalities in P are non-strict. ◀

▶ **Lemma 20.** *For any finite sets of variables X, variable y, interval $\mathtt{I}$, and predicate $\varphi \in \Phi_X$, if $[\![\varphi]\!]$ is closed then $[\![\exists y \in \mathtt{I} \bullet \varphi]\!]$ is also closed.*

Proof. Let ψ be $\exists y \in \mathtt{I} \bullet \varphi$. If $\mathtt{I}$ is an empty set then $[\![\psi]\!] = \emptyset$. Otherwise, if $y \notin X$ then $[\![\psi]\!] = [\![\phi]\!]$. Otherwise, let $\nu_0, \nu_1, \nu_2, \ldots$ be a sequence of points in $[\![\exists y \in \mathtt{I} \bullet \varphi]\!]$ that converges to ν^*. For every $n \in \mathbb{N}$, there is $\nu'_n \in [\![\varphi]\!]$ such that $\nu'_n(y) \in \mathtt{I}$ and $\forall x \in X \backslash \{y\} \bullet \nu'_n(x) = \nu_n(x)$. We know $[\![\varphi]\!]$ is bounded. Therefore, from Theorem 2 and closeness of $\mathtt{I}$, there is a subsequence with indices $m_0 < m_1 < m_2 < \cdots$ such that $\nu'_{m_0}, \nu'_{m_1}, \nu'_{m_2}, \ldots$ converges to ν'^* with $\nu'^*(y) \in \mathtt{I}$. We know $\nu'^* \in [\![\varphi]\!]$, since it is closed, and ν'^* agrees with ν^* on every variable $x \in X \setminus \{y\}$. Therefore, $\nu^* \in [\![\exists y \in \mathtt{I} \bullet \varphi]\!]$. ◀

▶ **Lemma 21.** *For any finite sets of variables X, variable y, interval $\mathtt{I}$, and predicate $\varphi \in \Phi_X$, if $[\![\varphi]\!]$ is closed then $[\![\forall y \in \mathtt{I} \bullet \varphi]\!]$ is also closed.*

Proof. Let ψ be $\forall y \in \mathtt{I} \bullet \varphi$. If $\mathtt{I}$ is an empty set then $[\![\psi]\!] = \mathbb{B}_{X \setminus \{y\}}$. Otherwise, if $y \notin X$ then $[\![\psi]\!] = [\![\phi]\!]$. Otherwise, let $\nu' \in \mathtt{I}^{\{y\}}$ be an arbitrary point, and let $\nu_0, \nu_1, \nu_2, \ldots$ be a sequence of points in $[\![\forall y \in \mathtt{I} \bullet \varphi]\!]$ that converges to ν^*. For any $n \in \mathbb{N}$, we know $\nu_n \frown \nu' \in [\![\varphi]\!]$. We also know $\nu_0 \frown \nu', \nu_1 \frown \nu', \nu_2 \frown \nu', \ldots$ converges to $\nu^* \frown \nu'$ which, by closedness of $[\![\varphi]\!]$, is a point in $[\![\varphi]\!]$. Therefore, $\nu^* \in [\![\forall y \in \mathtt{I} \bullet \varphi]\!]$, since we put no restriction on ν'. ◀

▶ **Lemma 22.** *For any finite set of variables X, any predicate in $\mathbb{P}_X$ can be arbitrarily over-approximated.*

Proof. For the purpose of contradiction, let $P \in \mathbb{P}_X$ be a predicate with the following condition: $\exists \epsilon \in \mathbb{R}_+ \bullet \forall \delta \in \mathbb{R}_+ \bullet \exists \nu \in [\![P^\delta]\!] \bullet \nu \notin \mathtt{B}^\epsilon_\infty([\![P]\!])$. Fix ϵ, and for any $n \in \mathbb{N}$, let $\delta_n := \frac{1}{n+1}$. Let $\nu_n \in [\![P^{\delta_n}]\!] \setminus \mathtt{B}^\epsilon_\infty([\![P]\!])$ be an arbitrary element. Using Theorem 2, there is a sequence $m_0 < m_1 < \cdots$ such that $\nu_{m_0}, \nu_{m_1}, \ldots$ converges to ν^*.

It is enough to show $\nu^* \in [\![P]\!]$, because there are infinitely many indices n such that $\mathtt{d}_\infty(\nu_{m_n}, \nu^*) \leq \epsilon$ and any one of them contradicts the fact $\nu_{m_n} \notin \mathtt{B}^\epsilon_\infty([\![P]\!])$. Suppose $\nu^* \notin [\![P]\!]$. Let the atomic constraint corresponding to P be $f \geq 0$. Since, $\nu^* \notin [\![P]\!]$ we have $f(\nu^*) < 0$, and since, $\nu_n \in [\![P^{\delta_n}]\!]$, we have $\forall n \in \mathbb{N} \bullet f(\nu_{m_n}) + \delta_{m_n} \geq 0$. Since f is a continuous function, $\exists N_1 \in \mathbb{N} \bullet \forall n > N_1 \bullet |f(\nu_{m_n}) - f(\nu^*)| < -\frac{1}{2} f(\nu^*)$. Let $N_2 \in \mathbb{N}$ be such that $\delta_{N_2} < -\frac{1}{2} f(\nu^*)$ and let $N := \max\{N_1, N_2\}$. We know $\forall n > N \bullet f(\nu_{m_n}) + \delta_{m_n} < \frac{1}{2} f(\nu^*) - \frac{1}{2} f(\nu^*) = 0$ which is a contradiction. ◀

▶ **Lemma 23.** *For any finite sets of variables Y, Z and predicates $\varphi \in \Phi_Y$ and $\psi \in \Phi_Z$, if φ and ψ can be arbitrarily over-approximated then $\varphi \vee \psi$ can be arbitrarily approximated as well.*

Proof. For any $\epsilon \in \mathbb{R}_+$, let $\delta \in \mathbb{R}_+$ be such that both $[\![\varphi^\delta]\!] \subseteq \mathtt{B}^\epsilon_\infty([\![\varphi]\!])$ and $[\![\psi^\delta]\!] \subseteq \mathtt{B}^\epsilon_\infty([\![\psi]\!])$ hold. If $[\![\varphi^\delta \vee \psi^\delta]\!] = \emptyset$ then the proof is complete. Otherwise, let $\nu \in [\![\varphi^\delta \vee \psi^\delta]\!]$ be an arbitrary value. At least one of $\nu\restriction_Y \in [\![\varphi^\delta]\!]$ and $\nu\restriction_Z \in [\![\psi^\delta]\!]$ are true. *Wlog.* assume that the first holds. Let $\nu' \in \mathtt{B}^\epsilon_\infty([\![\varphi]\!])$ be any point for which $\mathtt{d}_\infty(\nu', \nu\restriction_Y) \leq \epsilon$, and let $\nu'' \in \mathbb{R}^{Y \cup Z}$ be the point that maps $y \in Y$ to $\nu'(y)$ and $z \in Z \setminus Y$ to $\nu(z)$. We know $\mathtt{d}_\infty(\nu, \nu'') :=$

$\max_{x\in Y\cup Z}|\nu(x)-\nu''(x)|$. The right hand side is equal to the maximum of $\max_{y\in Y}|\nu(y)-\nu''(y)|$ and $\max_{z\in Z\setminus Y}|\nu(z)-\nu''(z)|=0$, and hence equal to $\max_{y\in Y}|\nu(y)-\nu'(y)|=\mathtt{d}_\infty(\nu\restriction_Y,\nu')\leq\epsilon$. What remains is to show $\nu''\in \mathtt{B}^\epsilon_\infty([\![\varphi\vee\psi]\!])$. Let $u'\in[\![\varphi]\!]$ be any point with $\mathtt{d}_\infty(\nu',u')\leq\epsilon$, and let $u''\in\mathbb{R}^{Y\cup Z}$ be the point that maps $y\in Y$ to $u'(x)$ and $z\in Z\setminus Y$ to $\nu''(z)$. Clearly, $\mathtt{d}_\infty(u'',\nu'')=\mathtt{d}_\infty(u',\nu')\leq\epsilon$. Therefore, it is enough to show $u''\in[\![\varphi\vee\psi]\!]$, which can be concluded from the facts $u''\restriction_Y=u'$ and $u'\in[\![\varphi]\!]$. ◀

▶ **Lemma 24.** *For any finite sets of variables Y,Z and predicates $\varphi\in\Phi_Y$ and $\psi\in\Phi_Z$, if φ and ψ can be arbitrarily over-approximated then $\varphi\wedge\psi$ can be arbitrarily approximated as well.*

Proof. For the purpose of contradiction, suppose $\exists\epsilon\in\mathbb{R}_+\bullet\forall\delta\in\mathbb{R}_+\bullet[\![\varphi^\delta\wedge\psi^\delta]\!]\not\subseteq\mathtt{B}^\epsilon_\infty([\![\varphi\wedge\psi]\!])$. Fix such an ϵ and for any $n\in\mathbb{N}$, let $\epsilon_n:=\frac{\epsilon}{n+1}$, and let $\delta_n\in\mathbb{R}_+$ be a value for which $[\![\varphi^{\delta_n}]\!]\subseteq\mathtt{B}^{\epsilon_n}_\infty([\![\varphi]\!])$ and $[\![\psi^{\delta_n}]\!]\subseteq\mathtt{B}^{\epsilon_n}_\infty([\![\psi]\!])$ hold. Also, make sure for any $n\in\mathbb{N}$, $\delta_{n+1}\leq\delta_n$ and hence $[\![\varphi^{\delta_{n+1}}\wedge\psi^{\delta_{n+1}}]\!]\subseteq[\![\varphi^{\delta_n}\wedge\psi^{\delta_n}]\!]$. Finally, let ν_n be an arbitrary element of the non-empty set $[\![\varphi^{\delta_n}\wedge\psi^{\delta_n}]\!]\setminus\mathtt{B}^\epsilon_\infty([\![\varphi\wedge\psi]\!])$. We know $\nu_n\restriction_Y\in[\![\varphi^{\delta_n}]\!]$ and $\nu_n\restriction_Z\in[\![\psi^{\delta_n}]\!]$.

Using Theorem 2, there is a sequence $m_0<m_1<m_2<\cdots$ such that $\nu_{m_0},\nu_{m_1},\nu_{m_2},\ldots$ converges to ν^*. It is enough to show $\nu^*\in[\![\varphi\wedge\psi]\!]$, because, there are infinitely many indices n such that $\mathtt{d}_\infty(\nu_{m_n},\nu^*)\leq\epsilon$ and any one of them contradicts the fact $\nu_{m_n}\notin\mathtt{B}^\epsilon_\infty([\![\varphi\wedge\psi]\!])$. Note that $\nu_{m_0}\restriction_Y,\nu_{m_1}\restriction_Y,\nu_{m_2}\restriction_Y,\ldots$ converges to $\nu^*\restriction_Y$ and $\nu_{m_0}\restriction_Z,\nu_{m_1}\restriction_Z,\nu_{m_2}\restriction_Z,\ldots$ converges to $\nu^*\restriction_Z$.

By Theorem 11, we know $[\![\varphi]\!]$ is closed and hence $\nu^*\restriction_Y\in[\![\varphi]\!]$ (otherwise, $\nu^*\restriction_Y$ will have a positive distance with $[\![\varphi]\!]$, which is a contradiction). Similarly, we know $\nu^*\restriction_Z\in[\![\psi]\!]$. Therefore, $\nu^*\in[\![\varphi\wedge\psi]\!]$. ◀

▶ **Lemma 25.** *For any finite set of variables X, variable y, interval $\mathtt{I}$, and a predicate $\varphi\in\Phi_X$, if φ can be arbitrarily over-approximated then $\psi:=\exists y\in\mathtt{I}\bullet\varphi$ can be arbitrarily over-approximated as well.*

Proof. The proof is trivial once we noticed for any two points $\nu_1,\nu_2\in\mathbb{R}^X$ we have $\mathtt{d}_\infty(\nu_1\restriction_{X\setminus\{y\}},\nu_2\restriction_{X\setminus\{y\}})\leq\mathtt{d}_\infty(\nu_1,\nu_2)$. ◀

▶ **Lemma 26.** *For any finite set of variables X, variable y, interval $\mathtt{I}$, and a predicate $\varphi\in\Phi_X$, if φ can be arbitrarily over-approximated then $\psi:=\forall y\in\mathtt{I}\bullet\varphi$ can be arbitrarily over-approximated as well.*

Proof. If $\mathtt{I}$ is an empty set then $\forall\delta\in\mathbb{R}\bullet[\![\psi^\delta]\!]=[\![\psi]\!]$. Otherwise, if $y\notin X$ then $\forall\delta\in\mathbb{R}\bullet[\![\psi^\delta]\!]=[\![\varphi^\delta]\!]$. Otherwise, For the purpose of contradiction, suppose $\exists\epsilon\in\mathbb{R}_+\bullet\forall\delta\in\mathbb{R}_+\bullet\exists\nu\in[\![\forall y\in\mathtt{I}\bullet\varphi^\delta]\!]\bullet\nu\notin\mathtt{B}^\epsilon_\infty([\![\forall y\in\mathtt{I}\bullet\varphi]\!])$. Fix such ϵ and let $\nu'\in\mathbb{R}^{\{y\}}$ be an arbitrary point. Also, for every $n\in\mathbb{N}$, let $\nu_n\in[\![\forall y\in\mathtt{I}\bullet\varphi^{\delta_n}]\!]\setminus\mathtt{B}^\epsilon_\infty([\![\forall y\in\mathtt{I}\bullet\varphi]\!])$ be an arbitrary point, where $\delta_n:=\frac{1}{n+1}$.

Using Theorem 2, we know there is a sequence $m_0<m_1<m_2<\ldots$ such that $\nu_{m_0},\nu_{m_1},\nu_{m_2},\ldots$ converges to ν^*. It is enough to show $\nu^*\in[\![\forall y\in\mathtt{I}\bullet\varphi]\!]$, because, there are infinitely many indices n such that $\mathtt{d}_\infty(\nu_{m_n},\nu^*)\leq\epsilon$ and any one of them contradicts the fact $\nu_{m_n}\notin\mathtt{B}^\epsilon_\infty([\![\forall y\in\mathtt{I}\bullet\varphi]\!])$. For any $n\in\mathbb{N}$, we know $\nu_{m_n}\frown\nu'\in[\![\varphi^{\delta_{m_n}}]\!]$. We also know, $\nu_{m_0}\frown\nu',\nu_{m_1}\frown\nu',\nu_{m_2}\frown\nu',\ldots$ converges to $\nu^*\frown\nu'$ which, by Theorem 11 and hence closedness of $[\![\varphi]\!]$, is a point in $[\![\varphi]\!]$. Therefore, $\nu^*\in[\![\forall y\in\mathtt{I}\bullet\varphi]\!]$, since there is no constraint on ν'. ◀

Updating Probabilistic Knowledge on Condition/Event Nets using Bayesian Networks

Benjamin Cabrera
University of Duisburg-Essen, Germany
benjamin.cabrera@uni-due.de

Tobias Heindel
University of Hawaii, USA
heindel@hawaii.edu

Reiko Heckel
University of Leicester, UK
rh122@leicester.ac.uk

Barbara König
University of Duisburg-Essen, Germany
barbara_koenig@uni-due.de

Abstract

The paper extends Bayesian networks (BNs) by a mechanism for dynamic changes to the probability distributions represented by BNs. One application scenario is the process of knowledge acquisition of an observer interacting with a system. In particular, the paper considers condition/event nets where the observer's knowledge about the current marking is a probability distribution over markings. The observer can interact with the net to deduce information about the marking by requesting certain transitions to fire and observing their success or failure.

Aiming for an efficient implementation of dynamic changes to probability distributions of BNs, we consider a modular form of networks that form the arrows of a free PROP with a commutative comonoid structure, also known as term graphs. The algebraic structure of such PROPs supplies us with a compositional semantics that functorially maps BNs to their underlying probability distribution and, in particular, it provides a convenient means to describe structural updates of networks.

2012 ACM Subject Classification Mathematics of computing → Bayesian networks, Software and its engineering → Petri nets

Keywords and phrases Petri nets, Bayesian networks, Probabilistic databases, Condition/Event nets, Probabilistic knowledge, Dynamic probability distributions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.27

Related Version A full version of the paper is available at [3], `https://arxiv.org/abs/1807.02566`.

Funding Research partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grant No. GRK 2167, Research Training Group "User-Centred Social Media".

1 Introduction

Representing uncertain knowledge by probability distributions is the core idea of Bayesian learning. We model the potential of an agent – the *observer* – interacting with a concurrent system with hidden or uncertain state to gain knowledge through "experimenting" with

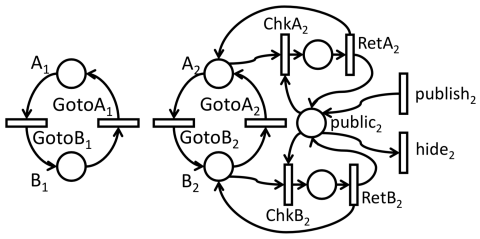


Figure 1 Example: Social network account with location privacy.

the system, focussing on the problem of keeping track of knowledge updates correctly and efficiently. Knowledge about states is represented by a probability distribution. Our system models are condition/even nets where states or possible worlds are markings and transitions describe which updates are allowed.

In order to clarify our intentions we consider an application scenario from social media: preventing inadvertent disclosure, the concern of location privacy [8]. Consider the example of a social network account, modelled as a condition/event net, allowing a user to update and share their location (see Figure 1). We consider two users. User 1 does not allow location updates to be posted to the social network, they are only recorded on their device. In the net this is represented by places A_1 and B_1 modelling the user at corresponding locations, and transitions GotoA_1 and GotoB_1 for moving between them. We assume that only User 1 can fire or observe these transitions. User 2 has a similar structure for locations and movements, but allows the network to track their location. The user can decide to make their location public or hide it by firing transition $\mathsf{publish}_2$ or hide_2. Any observer can attempt to fire $\mathsf{ChkA}_2; \mathsf{RetA}_2$ or $\mathsf{ChkB}_2; \mathsf{RetB}_2$ to query the current location of User 2. If public_2 is marked, this will allow the observer to infer the correct location. Otherwise the observer is left uncertain as to why the query fails, i.e. due to the wrong location being tested or the lack of permission, unless they test both locations. While our net captures the totality of possible behaviours, we identify different observers, the two users, the social network, and an unrelated observer. For each of these we define which transitions they can access. We then focus on one observer and only allow transitions they are authorised for. In our example, if we want to analyse the unrelated observer, we fix the users' locations and privacy choices before it is the observer's turn to query the system.

The observer may have prior knowledge about the dependencies between the locations of Users 1 and 2, for example due to photos with location information published by User 2, in which both users may be identifiable. The prior knowledge is represented in the initial probability distribution, updated according to the observations.

We also draw inspiration from probabilistic databases [28, 1] where the values of attributes or the presence of records are only known probabilistically. However, an update to the database might make it necessary to revise the probabilities. Think for instance of a database where the gender of a person (male or female) is unknown and we assume with probability 1/2 that they are male. Now a record is entered, stating that the person has married a male. Does it now become more probable that the person is female?

Despite its simplicity, our system model based on condition/event nets allows us to capture databases: the content of a database can be represented as a (hyper-)graph (where each record is a (hyper-)edge). If the nodes of the graph are fixed, updates can be represented by the transitions of a net, where each potential record is represented by a place.

Given a net, the observer does not know the initial marking, but has a prior belief, given by a probability distribution over markings. The observer can try to fire transitions and observe whether the firing is successful or fails. Then the probability distribution is updated accordingly. While the update mechanism is rather straightforward, the problem lies in the huge number of potential states: we have 2^n markings if n is the number of places.

To mitigate this state space explosion, we propose to represent the observer's knowledge using Bayesian networks (BNs) [22, 24], i.e., graphical models that record conditional dependencies of random variables in a compact form. However, we encounter a new problem as updating the observer's knowledge becomes non-trivial. To do this correctly and efficiently, we develop a compositional approach to BNs based on symmetric monoidal theories and PROPs [20]. In particular, we consider modular Bayesian networks as arrows of a freely generated PROP and (sub-)stochastic matrices as another PROP with a functor from the former to the latter. In this way, we make Bayesian networks compositional and we obtain a graphical formalism [27] that we use to modify Bayesian networks: in particular, we can replace entire subgraphs of Bayesian networks by equivalent ones, i.e., graphs that evaluate to the same matrix. The compositional approach allows us to specify complex updates in Bayesian networks by a sequence of simpler updates using a small number of primitives.

We furthermore describe an implementation and report promising runtime results.

The proofs of all results can be found in the full version of this paper [3].

2 Knowledge Update in Condition/Event Nets

We will formalise knowledge updates by means of an extension of Petri nets with probabilistic knowledge on markings. The starting point are condition/event nets [26].

▶ **Definition 1** (Condition/event net)**.** A condition/event net (CN) $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$ is a five-tuple consisting of a finite set of *places* S, a finite set of *transitions* T with *pre-conditions* ${}^\bullet() : T \to \mathcal{P}(S)$, *post-conditions* $()^\bullet : T \to \mathcal{P}(S)$, and $m_0 \subseteq S$ an *initial marking*. A *marking* is any subset of places $m \subseteq S$. We assume that for any $t \in T$, ${}^\bullet t \cap t^\bullet = \emptyset$.

A transition t can *fire* for a marking $m \subseteq S$, denoted $m \Rightarrow^t$, if ${}^\bullet t \subseteq m$ and $t^\bullet \cap m = \emptyset$. Then marking m is transformed into $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, written $m \Rightarrow^t m'$. We write $m \Rightarrow^t$ to indicate that there exists some m' with $m \Rightarrow^t m'$.

We will use two different notations to indicate that a transition cannot fire, the first referring to the fact that the pre-condition is not sufficiently marked, the second stating that there are tokens in the post-condition: $m \not\Rightarrow^t_{pre}$ whenever ${}^\bullet t \not\subseteq m$ and $m \not\Rightarrow^t_{post}$ whenever $t^\bullet \cap m \neq \emptyset$. We denote the *set of all markings* by $\mathcal{M} = \mathcal{P}(S)$.

For simplicity we assume that $S = \{1, \ldots, n\}$ for $n \in \mathbb{N}$. Then, a marking m can be characterized by a boolean vector $m : S \to \{0, 1\}$, i.e., $\mathcal{M} \cong \{0, 1\}^S$. Using the vector notation we write $m(A) = \{1\}$ for $A \subseteq S$ if all places in A are marked in m.

To capture the probabilistic observer we augment CNs by a probability distribution over markings modelling uncertainty about the hidden initial or current marking.

▶ **Definition 2** (Condition/Event net with Uncertainty)**.** A *Condition/Event Net with Uncertainty (CNU)* is a six-tuple $N = (S, T, {}^\bullet(), ()^\bullet, m_0, p)$ where $(S, T, {}^\bullet(), ()^\bullet, m_0)$ is a net as in Definition 1. Additionally, p is a function $p : \mathcal{M} \to [0, 1]$ with $\sum_{m \in \mathcal{M}} p(m) = 1$ that assigns a *probability mass* to each possible marking. This gives rise to a probability space $(\mathcal{M}, \mathcal{P}(\mathcal{M}), \mathbb{P})$ with $\mathbb{P} : \mathcal{P}(\mathcal{M}) \to [0, 1]$ defined by $\mathbb{P}\big(\{m_1, \ldots, m_k\}\big) = \sum_{i=1}^{k} p(m_i)$.

We assume that $p(m_0) > 0$, i.e. the initial marking is possible according to p.

We model the knowledge gained by observers when firing transitions and observing their outcomes. Firing $t \in T$ can either result in success (all places of ${}^\bullet t$ are marked and no place in $t^\bullet$ is marked) or in failure (at least one place of ${}^\bullet t$ is empty or one place in $t^\bullet$ is marked). Thus, there are two kinds of failure, the absence of tokens in the pre-condition or the presence of tokens in the post-condition. If a transition fails for both reasons, the observer will learn only one of them. To model the knowledge gained we define the following operations on distributions.

▶ **Definition 3** (Operations on CNUs). Given a CNU $N = (S, T, {}^\bullet(), ()^\bullet, m_0, p)$ the following operations update the mass function p and as a result the probability distribution $\mathbb{P}$.

- To assert that certain places $A \subseteq S$ *all contain* a token ($b = 1$) or that none contains a token ($b = 0$) we define the operation *assert*

$$\mathrm{ass}_{A,b}(p)(m) = \frac{p(m)}{\sum_{m' \in \mathcal{M}: m'(A) = \{b\}} p(m')}, \text{ if } m(A) = \{b\} \quad \text{and} \quad 0, \text{ otherwise.}$$

- To state that *at least one* place of a set $A \subseteq S$ *does* (resp. *does not*) *contain* a token we define operation *negative assert*

$$\mathrm{nas}_{A,b}(p)(m) = \frac{p(m)}{\sum_{m' \in \mathcal{M}: m'(A) \neq \{b\}} p(m')}, \text{ if } m(A) \neq \{b\} \quad \text{and} \quad 0, \text{ otherwise.}$$

- *Modifying* a set of places $A \subseteq S$ such that *all places contain a token ($b = 1$) or none contains a token ($b = 0$)* requires the following operation

$$\mathrm{set}_{A,b}(p)(m) = \sum_{m' : m'|_{S \setminus A} = m|_{S \setminus A}} p(m'), \text{ if } m(A) = \{b\} \quad \text{and} \quad 0, \text{ otherwise.} \tag{1}$$

- A successful firing of a transition t leads to an assert (ass) and set of the pre-conditions ${}^\bullet t$ and the post-conditions $t^\bullet$. A failed firing translates to a negative assert (nas) of the pre- or post-condition and nothing is set. Thus we define for a transition $t \in T$

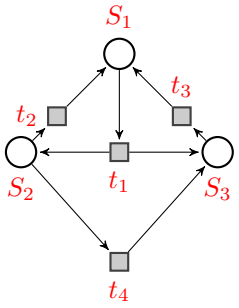
$$\mathrm{success}_t(p) = \mathrm{set}_{t^\bullet,1}(\mathrm{set}_{{}^\bullet t,0}(\mathrm{ass}_{t^\bullet,0}(\mathrm{ass}_{{}^\bullet t,1}(p)))), \qquad \mathrm{fail}_t^{pre}(p) = \mathrm{nas}_{{}^\bullet t,1}(p),$$
$$\mathrm{fail}_t^{post}(p) = \mathrm{nas}_{t^\bullet,0}(p).$$

Operations ass, nas are partial, defined whenever the sum in the denominator of their first clause is greater than 0. That means, the observer only fires transitions whose pre- and postconditions have a probability greater than zero, i.e., where according to their knowledge about the state it is possible that these transitions are enabled. By Definition 1 the initial marking is possible, and this property is maintained as markings and distributions are updated. If this assumption is not satisfied, the operations in Definition 3 are undefined.

The ass and nas operations result from conditioning the input distribution on (not) having tokens at A (compare Proposition 4). Also, set and ass for $A = \{s_1, \ldots, s_k\} \subseteq S$ can be performed iteratively, i.e., $\mathrm{set}_{A,b} = \mathrm{set}_{\{s_k\},b} \circ \cdots \circ \mathrm{set}_{\{s_1\},b}$ and $\mathrm{ass}_{A,b} = \mathrm{ass}_{\{s_k\},b} \circ \cdots \circ \mathrm{ass}_{\{s_1\},b}$. For a single place s we have $\mathrm{ass}_{\{s\},b} = \mathrm{nas}_{\{s\},1-b}$.

Figure 2 gives an example for a Petri net with uncertainty and explains how the observer can update their knowledge by interacting with the net. We can now show that our operations correctly update the probability assumptions according to the observations of the net.

▶ **Proposition 4.** *Let $N = (S, T, {}^\bullet(), ()^\bullet, m_0, p)$ be a CNU where $\mathbb{P}$ is the corresponding probability distribution. For given $t \in T$ and $m \in \mathcal{M}$ let $\mathcal{M}[\Rightarrow^t] = \{m' \in \mathcal{M} \mid m' \Rightarrow^t\}$, $\mathcal{M}[\Rightarrow^t m] = \{m' \in \mathcal{M} \mid m' \Rightarrow^t m\}$, $\mathcal{M}[\not\Rightarrow^t_{pre}] = \{m' \in \mathcal{M} \mid m' \not\Rightarrow^t_{pre}\}$ and $\mathcal{M}[\not\Rightarrow^t_{post}] =$*



– *places* –				*success*$_{t_4}$				*fail*$^{pre}_{t_1}$
S_1	S_2	S_3	init	$\text{as}_{\{S_2\},1}$	$\text{as}_{\{S_3\},0}$	$\text{set}_{\{S_2\},0}$	$\text{set}_{\{S_3\},1}$	$\text{nas}_{\{S_1\},1}$
1	1	1	1/12	1/6	0	0	0	0
1	1	0	1/6	1/3	1/2	0	0	0
1	0	1	1/8	0	0	0	1/2	0
1	0	0	1/8	0	0	1/2	0	0
0	1	1	1/12	1/6	0	0	0	0
0	1	0	1/6	1/3	1/2	0	0	0
0	0	1	1/8	0	0	0	1/2	1
0	0	0	1/8	0	0	1/2	0	0

Figure 2 Example of operations on a net with uncertainty. We set $m_0 = \{S_2\}$ and assume the observer first fires t_4 (and succeeds) and then tries to fire t_1 (and fails). Columns in the table represent updated distributions on the markings after each operation (ordered from left to right). For this example, in the end the observer knows that the final configuration is $\{S_3\}$ with probability 1.

$\{m' \in \mathcal{M} \mid m' \not\Rightarrow^t_{post}\}$. *Then, provided that* $\mathcal{M}[\Rightarrow^t]$, $\mathcal{M}[\not\Rightarrow^t_{pre}]$ *respectively* $\mathcal{M}[\not\Rightarrow^t_{post}]$ *are non-empty, it holds for* $m \in \mathcal{M}$ *that*

$$\text{success}_t(p)(m) = \mathbb{P}(\mathcal{M}[\Rightarrow^t m] \mid \mathcal{M}[\Rightarrow^t]), \qquad \text{fail}^{pre}_t(p)(m) = \mathbb{P}(\{m\} \mid \mathcal{M}[\not\Rightarrow^t_{pre}]),$$
$$\text{fail}^{post}_t(p)(m) = \mathbb{P}(\{m\} \mid \mathcal{M}[\not\Rightarrow^t_{post}]).$$

We shall refer to the the joint distribution (over all places) by $\mathbb{P}$. Note that it is unfeasible to explicitly store it if the number of places is large. To mitigate this problem we use a Bayesian network with a random variable for each place, recording dependencies between the presence of tokens in different places. If such dependencies are local, the BN is often moderate in size and thus provides a compact symbolic representation. However, updating the joint distribution of BNs is non-trivial. To address this problem, we propose a propagation procedure based on a term-based, modular representation of BNs.

3 Modular Bayesian Networks and Sub-Stochastic Matrices

Bayesian networks (BNs) are a graphical formalism to reason about probability distributions. They are visualized as directed, acyclic graphs with nodes random variables and edges dependencies between them. This is sufficient for static BNs whose most common operation is the inference of (marginalized or conditional) distributions of the underlying joint distribution.

For a rewriting calculus on dynamic BNs, we consider a modular representation of networks that do not only encode a single probability vector, but a matrix, with several input and output ports. The first aim is compositionality: larger nets can be composed from smaller ones via sequential and parallel composition, which correspond to matrix multiplication and Kronecker product of the encoded matrices. This means, we can implement the operations of Section 2 in a modular way.

PROPs with Commutative Comonoid Structure

We now describe the underlying compositional structure of (modular) BNs and (sub-) stochastic matrices, which facilitates a compositional computation of the underlying probability distribution of (modular) BNs. The mathematical structure are PROPs [20] (see also [13, Chapter 5.2]), i.e., strict symmetric monoidal categories $(C, \otimes, 0, \sigma)$ whose objects are (in

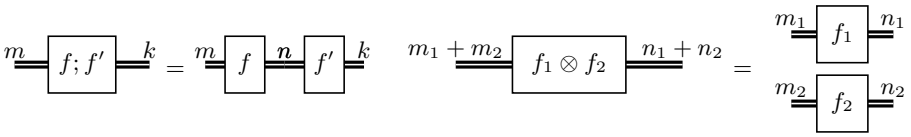


Figure 3 String diagrammatic composition (resp. tensor) of two arrows $f\colon m \to n$, $f'\colon n \to k$ (resp. $f_1\colon m_1 \to n_1$, $f_2\colon m_2 \to n_2$) of a PROP $(C, \otimes, 0, \sigma)$.

Table 1 Axioms for CC-structured PROPs and definition of operators of higher arity.

$$(t_1; t_3) \otimes (t_2; t_4) = (t_1 \otimes t_2); (t_3 \otimes t_4) \qquad (t_1; t_2); t_3 = t_1; (t_2; t_3)$$
$$\mathrm{id}_n; t = t = t; \mathrm{id}_m \qquad (t_1 \otimes t_2) \otimes t_3 = t_1 \otimes (t_2 \otimes t_3) \qquad \mathrm{id}_0 \otimes t = t = t \otimes \mathrm{id}_0$$
$$\sigma; \sigma = \mathrm{id}_2 \qquad (t \otimes \mathrm{id}_m); \sigma_{n,m} = \sigma_{m,n}; (\mathrm{id}_n \otimes t) \qquad \nabla; (\nabla \otimes \mathrm{id}_1) = \nabla; (\mathrm{id}_1 \otimes \nabla)$$
$$\nabla = \nabla; \sigma \qquad \nabla; (\mathrm{id}_1 \otimes \top) = \mathrm{id}_1$$

$$\mathrm{id}_1 = \mathrm{id} \qquad \mathrm{id}_{n+1} = \mathrm{id}_n \otimes \mathrm{id}_1$$
$$\sigma_{n,0} = \sigma_{0,n} = \mathrm{id}_n \qquad \sigma_{n+1,1} = (\mathrm{id} \otimes \sigma_{n,1}); (\sigma \otimes \mathrm{id}_n)$$
$$\sigma_{n,m+1} = (\sigma_{n,m} \otimes \mathrm{id}_1); (\mathrm{id}_m \otimes \sigma_{n,1})$$
$$\nabla_1 = \nabla \qquad \nabla_{n+1} = (\nabla_n \otimes \nabla); (\mathrm{id}_n \otimes \sigma_{n,1} \otimes \mathrm{id})$$
$$\top_1 = \top \qquad \top_{n+1} = \top_n \otimes \top$$

bijection with) the natural numbers, with monoidal product $\otimes$ as (essentially) addition, with unit 0. The compositional structure of PROPs can be intuitively represented using string diagrams with wires and boxes (see Figure 3). Symmetries σ serve for the reordering of wires.

A paradigmatic example is the PROP of 2^n-dimensional Euclidean spaces and linear maps, equipped with the tensor product: the tensor product of 2^n- and 2^m-dimensional spaces is 2^{n+m}-dimensional, composition of linear maps amounts to matrix multiplication, and the tensor product is also known as Kronecker product (as detailed below). We refer to the natural numbers of the domain and codomain of arrows in a PROP as their *type*; thus, a linear map from 2^n- to 2^m-dimensional Euclidean space has type $n \to m$.

We shall have the additional structure on symmetric monoidal categories that was dubbed *graph substitution* in work on term graphs [7], which amounts to a commutative comonoid structure on PROPs.

▶ **Definition 5** (PROPs with commutative comonoid structure)**.** A *CC-structured PROP* is a tuple $(C, \otimes, 0, \sigma, \nabla, \top)$ where $(C, \otimes, 0, \sigma)$ is a PROP and the last two components are arrows $\nabla\colon 1 \to 2$ and $\top\colon 1 \to 0$, which are subject to equations (2) (cf. Figure 4).

$$\nabla; (\nabla \otimes \mathrm{id}_1) = \nabla; (\mathrm{id}_1 \otimes \nabla), \qquad \nabla = \nabla; \sigma, \qquad \nabla; (\mathrm{id}_1 \otimes \top) = \mathrm{id}_1. \tag{2}$$

To give another, more direct definition, the arrows of a freely generated CC-structured PROP can be represented as terms over some set of generators $g \in G$ and constants $\mathrm{id}\colon 1 \to 1$, $\sigma\colon 2 \to 2$, $\nabla\colon 1 \to 2$, $\top\colon 1 \to 0$, combined with the operators sequential composition $(;)$ and tensor $(\otimes)$ and quotiented by the axioms in Table 1 (see [30]). This table also lists

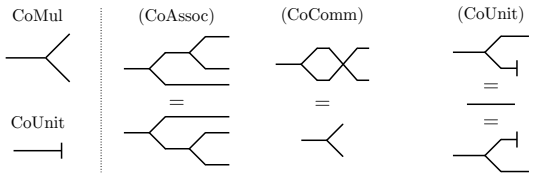


Figure 4 Comultiplication and counit arrows and the equations of commutative comonoids.

the definition of operators of higher arity. We often refer to the comultiplication Δ and its counit $\top$ as *duplicator* and *terminator*, resp. (cf. Figure 4). Intuitively, adding the commutative comonoid structure amounts to the possibility to have several or no connections to each one of the output port of gates and input ports. In other words, outputs can be shared.

(Sub-)Stochastic Matrices

We now consider (sub-)stochastic matrices as an instance of a CC-structured PROP. A *matrix* of type $n \to m$ is a matrix P of dimension $2^m \times 2^n$ with entries taken from the closed interval $[0,1] \subseteq \mathbb{R}$. We restrict attention to sub-stochastic matrices, i.e., column sums will be at most 1; if we require equality, we obtain stochastic matrices.

We index matrices over $\{0,1\}^m \times \{0,1\}^n$, i.e., for $\mathbf{x} \in \{0,1\}^m$, $\mathbf{y} \in \{0,1\}^n$ the corresponding entry is denoted by $P(\mathbf{x} \mid \mathbf{y})$. We use this notation to evoke the idea of conditional probability (the probability that the first index is equal to $\mathbf{x}$, whenever the second index is equal to $\mathbf{y}$.) When we write P as a matrix, the rows/columns are ordered according to a descending sequence of binary numbers ($1 \ldots 1$ first, $0 \ldots 0$ last).

$$\begin{array}{c} \\ 11 \\ 10 \\ 01 \\ 00 \end{array} \begin{array}{c} \begin{array}{cccc} 11 & 10 & 01 & 00 \end{array} \\ \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{array}$$

Sequential composition is *matrix multiplication*, i.e., given $P\colon n \to m$, $Q\colon m \to \ell$ we define $P;Q = Q \cdot P\colon n \to \ell$, which is a $2^\ell \times 2^n$-matrix. The tensor is given by the *Kronecker product*, i.e., given $P\colon n_1 \to m_1$, $Q\colon n_2 \to m_2$ we define $P \otimes Q\colon n_1 + n_2 \to m_1 + m_2$ as $(P \otimes Q)(\mathbf{x}_1\mathbf{x}_2 \mid \mathbf{y}_1\mathbf{y}_2) = P(\mathbf{x}_1 \mid \mathbf{y}_1) \cdot Q(\mathbf{x}_2 \mid \mathbf{y}_2)$ where $\mathbf{x}_i \in \{0,1\}^{n_i}$, $\mathbf{y}_i \in \{0,1\}^{m_i}$.

The constants are defined as follows:

$$\mathrm{id}_0 = (1) \quad \mathrm{id} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \nabla = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \quad \sigma = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \top = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

▶ **Proposition 6** ([12]). *(Sub-)stochastic matrices form a CC-structured PROP.*

Causality Graphs

We next introduce causality graphs, a variant of term graphs [7], to provide a modular representation of Bayesian networks. Nodes play the role of gates of string diagrams; the main difference to port graphs [13, Chapter 5] is the branching structure at output ports, which corresponds to (freely) added commutative comonoid structure. We fix a set of generators G (a.k.a. signature), elements of which can be thought of as blueprints of gates of a certain type; all generators $g \in G$ will be of type $n \to 1$, which means that each node can be identified with its single output port while it has a certain number of input ports.

▶ **Definition 7** (Causality Graph (CG)). A *causality graph (CG)* of type $n \to m$ is a tuple $B = (V, \ell, s, \text{out})$ where

- V is a set of *nodes*,
- $\ell\colon V \to G$ is a *labelling function* that assigns a generator $\ell(v) \in G$ to each node $v \in V$,
- $s\colon V \to W_B^*$ where $W_B = V \cup \{i_1, \dots, i_n\}$ is the *source function* that assigns a sequence of *wires* $s(v)$ to each node $v \in V$ such that $|s(v)| = n$ if $\ell(v)\colon n \to 1$,
- $\text{out}\colon \{o_1, \dots, o_m\} \to W_B$ is the *output function* that assigns each output port to a wire.

Moreover, the corresponding directed graph (defined by s) has to be acyclic.

By $\{i_1, \dots, i_n\}$ we denote the set of *input ports* and by $\{o_1, \dots, o_m\}$ the set of *output ports*. By pred and succ we denote the direct predecessors and successors of a node, i.e. $\text{pred}(v_0) = \{v \in V \mid v \in s(v_0)\}$ and $\text{succ}(v_0) = \{v \in V \mid v_0 \in s(v)\}$, respectively. By $\text{pred}^*(v_0)$ we denote the set of indirect predecessors, using transitive closure. Furthermore $\text{path}(v, w)$ denotes the set of all nodes which lie on paths from v to w.

A wire originates from a single input port or node and each node can feed into several successor nodes and/or output ports. Note that input and output are not symmetric in the context of causality graphs. This is a consequence of the absence of a monoid structure.

We equip CGs with operations of composition and tensor product, identities, and a commutative comonoid structure. We require that the node sets of Bayesian nets B_1, B_2 are disjoint.[1]

Composition. Whenever $m_1 = n_2$, we define $B_1; B_2 := B = (V, \ell, s, out)\colon n_1 \to m_2$ with $V = V_1 \uplus V_2$, $\ell = \ell_1 \uplus \ell_2$, $s = s_1 \uplus c \circ s_2$, $out = c \circ out_2$ where $c\colon W_{B_2} \to W_B$ is defined as follows and extended to sequences: $c(w) = w$ if $w \in V_2$ and $c(w) = out_1(o_j)$ if $w = i_j$.

Tensor. Disjoint union is parallel composition, i.e., $B_1 \otimes B_2 := B = (V, \ell, s, out)\colon n_1 + n_2 \to m_1 + m_2$ with $V = V_1 \uplus V_2$, $\ell = \ell_1 \uplus \ell_2$, $s = s_1 \uplus d \circ s_2$, where $d\colon W_{B_2} \to W_B$ and $out\colon \{o_1, \dots, o_{m_1+m_2}\} \to W_B$ are defined as follows: $d(w) = w$ if $w \in V_2$ and $d(w) = i_{n_1+j}$ if $w = i_j$. Furthermore $out(o_j) = out_1(o_j)$ if $1 \le j \le m_1$ and $out(o_j) = out_2(o_{j-m_1})$ if $m_1 < j \le m_1 + m_2$.

Operators. Finally the constants and generators are as follows:[2]

$$\text{id}_0 = (\emptyset, [\,], [\,], [\,])\colon 0 \to 0 \quad \text{id} = (\emptyset, [\,], [\,], [o_1 \mapsto i_1])\colon 1 \to 1 \quad \top = (\emptyset, [\,], [\,], [\,])\colon 1 \to 0$$
$$\sigma = (\emptyset, [\,], [\,], [o_1 \mapsto i_2, o_2 \mapsto i_1])\colon 2 \to 2 \quad \nabla = (\emptyset, [\,], [\,], [o_1 \mapsto i_1, o_2 \mapsto i_1])\colon 1 \to 2$$
$$B_g = (\{v\}, [v \mapsto g], [v \mapsto i_1 \dots i_n], [o_1 \mapsto v])\colon n \to 1, \text{ whenever } g \in G \text{ with type } g\colon n \to 1$$

Finally, all these operations lift to isomorphism classes of CGs.

▶ **Proposition 8** ([7]). *CGs quotiented by isomorphism form the freely generated CC-structured PROP over the set of generators G, where two causality graphs $B_i = (V_i, \ell_i, s_i, out_i)\colon n \to m$, $i \in \{1, 2\}$, are isomorphic if there is a bijective mapping $\varphi\colon V_1 \to V_2$ such that $\ell_1(v) = \ell_2(\varphi(v))$ and $\varphi(s_1(v)) = s_2(v)$ hold for all $v \in V_1$ and $\varphi(out_1(o_i)) = out_2(o_i)$ holds for all $i \in \{1, \dots, m\}$.*[3]

In the following, we often decompose a CG into a subgraph and its "context".

▶ **Lemma 9** (Decompositionality of CGs). *Let $B = (V, \ell, s, \text{out})\colon n \to m$ be a causality graph. Let $V' \subseteq V$ be a subset of nodes closed with respect to paths, i.e. for all $v, w \in V'$: $\text{path}(v, w) \subseteq V'$. Then there exist $k \in \mathbb{N}$ and (B_i, e_i) with $B_i = (V_i, l_i, s_i, \text{out}_i)$ for $i = 1, \dots, 3$ such that $V_2 = V'$, $B = B_1; (\text{id}_k \otimes B_2); B_3$ and $\text{out}_2(o_i) \in V'$ for all i.*

---

[1] The case of non-disjoint sets can be handled by a suitable choice of coproducts.

[2] A function $f\colon A \to B$, where $A = \{a_1, \dots, a_k\}$ is finite, is denoted by $f = [a_1 \mapsto f(a_1), \dots, a_k \mapsto f(a_k)]$. We denote a function with empty domain by $[\,]$.

[3] We apply φ to a sequence of wires, by applying φ pointwise and assuming that $\varphi(i_j) = i_j$ for $1 \le j \le n$.

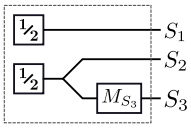


Figure 5 The initial distribution of the CNU from Figure 2 as an MBN.

Thus, given a set of nodes in a BN that contains all nodes on paths between them, we have the induced subnet of the node set and a suitable "context" such that the whole net can be seen as the result of substition of the subnet into the "context".

Modular Bayesian Networks

We will now equip the nodes of causality graphs with matrices, assigning an interpretation to each generator. This fully determines the corresponding matrix of the BN. Note that Bayesian networks as PROPs have earlier been studied in [12, 16, 17].

▶ **Definition 10** (Modular Bayesian network (MBN)). A *modular Bayesian network (MBN)* is a tuple (B, e) where $B = (V, \ell, s, \mathrm{out})$ is a causality graph and e an *evaluation function* that assigns to every generator $g \in G$ with $g\colon n \to 1$ a $2^n \times 2$-matrix $e(g)$. An MBN (B, e) is called an *ordinary Bayesian network (OBN)* whenever B has no inputs (i.e. $B : 0 \to m$), out is a bijection, and every node is associated with a stochastic matrix.

In an OBN every node V corresponds to a random variable and it represents a probability distribution on $\{0,1\}^m$. OBNs are exactly the Bayesian networks considered in [14].

▶ **Example 11.** *Figure 5 gives an example of a BN where* $\mathbf{1/2} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ *and* $M_{S_3} = \begin{pmatrix} 1/3 & 1/2 \\ 2/3 & 1/2 \end{pmatrix}$. *It encodes exactly the probability distribution from Figure 2. Its term representation is* $(g_1 \otimes (g_2; \nabla)); (\mathrm{id}_2 \otimes g_3)$ *where* $e(g_1) = e(g_2) = \mathbf{1/2}$ *and* $e(g_3) = M_{S_3}$.

▶ **Definition 12** (MBN semantics). Let (B, e) be an MBN where the network $B = (V, \ell, s, out)$ is of type $n \to m$. The *MBN semantics* is the matrix $M_e(B)$ with

$$\Big(M_e(B)\Big)(x_1 \dots x_m \mid y_1 \dots y_n) = \sum_{\substack{b\colon W_B \to \{0,1\} \\ b(i_j)=y_j, b(out(o_i))=x_i}} \prod_{v \in V} e\big(\ell(v)\big)\Big(b(v) \;\Big|\; b(s(v))\Big)$$

with $x_1, \dots, x_m, y_1, \dots, y_m \in \{0,1\}$ where b is applied pointwise to sequences.

Intuitively the function b assigns boolean values to wires, in a way that is consistent with the input/output values $(x_1 \dots x_m, y_1 \dots y_n)$. For each such assignment, the corresponding entries in the matrices $\ell(v)$ are multiplied. Finally, we sum over all possible assignments.

▶ Remark. The semantics $M_e(B)$ is compositional. It is the canonical (i.e., free) extension of the evaluation map from single nodes to the causality graph of an MBN (B, e). Here, we rely on two different findings from the literature, namely, the CC-PROP structure of (sub-)stochastic matrices [12] and the characterization of term graphs as the free symmetric monoidal category with graph substition [7]. The formal details can be found in [3].

4 Updating Bayesian Networks

We have introduced MBNs as a compact and compositional representation of distributions on markings of a CNU. Coming back to the scenario of knowledge update, we now describe how success and failure of operations requested by the observer affect the MBN. We will first

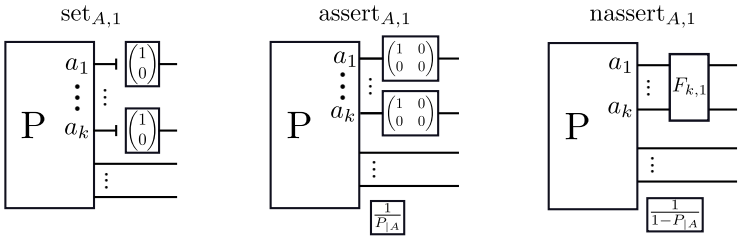


Figure 6 String diagrams of the updated distributions after $\mathrm{set}_{A,1}$, $\mathrm{ass}_{A,1}$, $\mathrm{nas}_{A,1}$ operations were applied to an initial distribution P.

describe how the operations can be formulated as matrix operations that tell us which nodes have to be added to the MBN. We shall see that updated MBNs are in general not OBNs, which makes it harder to interpret and retrieve the encoded distribution. However, we shall show that MBNs can efficiently be reduced to OBNs.

Notation. In this section we will use the following notation: first, we will use variants $\mathrm{id}_n, \nabla_n, \sigma_{n,m}, \top_n$ of the operators/matrices $\mathrm{id}, \nabla, \sigma, \top$, which have a higher arity (see the definitions in Table 1). Furthermore, we will write $\prod_{i=1}^{k} P_i$ for $P_1 \cdot \ldots \cdot P_k$ and $\bigotimes_{i=1}^{k} P_i$ for $P_1 \otimes \cdots \otimes P_k$. By $0 : 1 \to 1$ we denote the 2×2 zero matrix and set $0_k = \bigotimes_{i=1}^{k} 0$. We also introduce $\mathbf{1}_b$ as a notation for the matrix $\begin{pmatrix}1\\0\end{pmatrix}$ if $b = 1$ (respectively $\begin{pmatrix}0\\1\end{pmatrix}$ if $b = 0$).

With $\mathrm{diag}(a_1, \ldots, a_n)$ we denote a square matrix with entries $a_1, \ldots, a_n \in [0, 1]$ on the diagonal and zero elsewhere. In particular, we will need the sub-stochastic matrices $F_{k,b} : k \to k$ where $F_{k,0} = \mathrm{diag}(\underbrace{1, \ldots, 1}_{2^k - 1 \text{ times}}, 0)$ and $F_{k,1} = \mathrm{diag}(0, \underbrace{1, \ldots, 1}_{2^k - 1 \text{ times}})$.

Given a bit-vector $\mathbf{x} \in \{0, 1\}^n$, we will write $\mathbf{x}_{[i]}$ respectively $\mathbf{x}_{[i \ldots j]}$ to denote the i-th entry respectively the sub-sequence from position i to position j. If $A \subseteq \{1, \ldots, n\}$ we define $\mathbf{x}_{[A]} = \{\mathbf{x}_{[i]} \mid i \in A\}$.

CNU Operations on MBNs

In this section we characterize the operations of Definition 3 as stochastic matrices that can be multiplied with the distribution to perform the update. We describe them as compositions of smaller matrices that can easily be interpreted as changes to an MBN. In the following lemmas, $P : 0 \to m$ is always a stochastic matrix representing the distribution of markings of a CNU. Furthermore, $A \subseteq S$ is a set of places and w.l.o.g. we assume that $A = \{1, \ldots, k\}$ for some $k \leq m$ (as otherwise we can use permutations that preceed and follow the operations and switch wires as needed).

Starting with the $\mathrm{set}_{A,b}$ operation (1) recall that it is defined in a way so that the marginal distributions of non-affected places $S \backslash A$ stay the same while the marginals of every single place in A report $b \in \{0, 1\}$ with probability one. The following lemma shows how the matrix for a set operation can be constructed (see Figure 6).

▶ **Lemma 13.** *It holds that* $\mathrm{set}_{A,b}(P) = \left(\bigotimes_{i=1}^{m} T^{\mathrm{set}}_{A,b}(i)\right) \cdot P$ *where* $T^{\mathrm{set}}_{A,b}(i)$ *is* $\mathbf{1}_b \cdot \top$ *if* $i \in A$, *and* id *otherwise. Moreover,* $\bigotimes_{i=1}^{m} T^{\mathrm{set}}_{A,b}(i)$ *is stochastic.*

Next, we deal with the ass operation. Applying it to a distribution P is simply a conditioning of P on non-emptiness of all places A. Intuitively, this means that we keep only entries of P for which the condition is satisfied and set all other entries to zero. However, in

order to keep the updated P a probability distribution, we have to renormalize, which already shows that modelling this operation introduces sub-stochastic matrices to the computation. In the next lemma normalization involves the costly computation of a marginal $P_{|A}$ (the probability that all places in A are set to b), however omitting the normalization factor will give us a sub-stochastic matrix and we will later show how such sub-stochastic matrices can be removed, in many cases avoiding the full costs of a marginal computation.

▶ **Lemma 14.** *It holds that* $\mathrm{ass}_{A,b}(P) = \frac{1}{P_{|A}} \left(\bigotimes_{i=1}^{m} T^{\mathrm{ass}}_{A,b}(i)\right) \cdot P$ *with* $P_{|A} = (\bigotimes_{i=1}^{m} Q_A(i)) \cdot P$ *where* $T^{\mathrm{ass}}_{A,b}(i)$ *is* $F_{1,1-b}$ *if* $i \in A$*, and* id *otherwise. We require that* $P_{|A} \neq 0$*. Furthermore* $Q^{\mathrm{ass}}_{A,b}(i) = \begin{pmatrix} 1 & 0 \end{pmatrix}$ *if* $i \in A$ *and* $\top$ *otherwise.*

In contrast to set and ass, the nas operation couples all involved places in A. Asserting that at least one place has no token means that once the observer learns that e.g. one particular place definitely has a token it affects all the other ones. Thus for updating the distribution we have to pass the wires of places A through another matrix that removes the possibility of all places containing a token and renormalizes.

▶ **Lemma 15.** *The following characterization holds:* $\mathrm{nas}_{A,1}(P) = \frac{1}{P^c_{|A}} (F_{k,1} \otimes \mathrm{id}_{m-k}) \cdot P$ *with* $P^c_{|A} = 1 - P_{|A}$ *($P_{|A}$ is defined as in Lemma 14). We require that* $P^c_{|A} \neq 0$*.*

An analogous result holds for $\mathrm{nas}_{A,0}$ by using $F_{k,0}$.

The previous lemmas determine how to update an MBN (B, e) to incorporate the changes to the encoded distribution stemming from the operations on the CNU. We denote the updated MBN by (B', e') with $B' = (V', \ell', s', \mathrm{out}')$.

For the $\mathrm{set}_{A,b}$ operation Lemma 13 shows that we have to add a new node v_s and a new generator g_s for each $s \in A$. We set $\ell(v_s) = g_s$ and $e'(g_s) = \mathbf{1}_b \cdot \top = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$, $s'(v_s) = \mathrm{out}(o_s)$ and $\mathrm{out}'(o_s) = v_s$. Similarly, this holds for the ass operation with the only difference that the associated matrix for each v_s is $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ (cf. Figure 6).

For the $\mathrm{nas}_{A,b}$ operation Lemma 15 defines a usually larger matrix $F_{k,b} : k \to k$ that intuitively couples the random variables for all places in A. We cannot simply add a node to the MBN which evaluates to $F_{k,b}$ since nodes in the MBN always have to be of type $n \to 1$. However, one can show (see Lemma 18) that for each $F_{k,b}$-matrix, there exists an MBN (B', e') such that $M_{e'}(B')$. This can then be appended to (B, e) which has the same affect as appending a single node with the $F_{k,b}$-matrix.

Simplifying MBNs to OBNs

The characterisations of operations above ensure that updated MBNs correctly evaluate to the updated probability distributions. However, rather than OBNs we obtain MBNs where the complexity of updates is hidden in newly added nodes. Evaluating such MBNs is computationally more expensive because of the additional nodes. Below we show how to simplify the MBN, minimising the number of nodes either after each update or (in a lazy mode) after several updates.

As a first step we provide a lemma that will feature in all following simplifications. It states that every matrix can be expressed by the composition of two matrices.

▶ **Lemma 16** (Decomposition of matrices). *Given a matrix* P *of type* $n \to m$ *and a set of* $k < m$ *outputs – without loss of generality we pick* $\{m-k+1, \ldots, m\}$ *– there exist two matrices* $P^{\vdash} : n \to m-k$ *and* $P^{\dashv} : n+m-k \to k$ *such that*

$$(\mathrm{id}_{m-k} \otimes P^{\dashv}) \cdot ((\nabla_{m-k} \cdot P^{\vdash}) \otimes \mathrm{id}_n) \cdot \nabla_n = P,$$

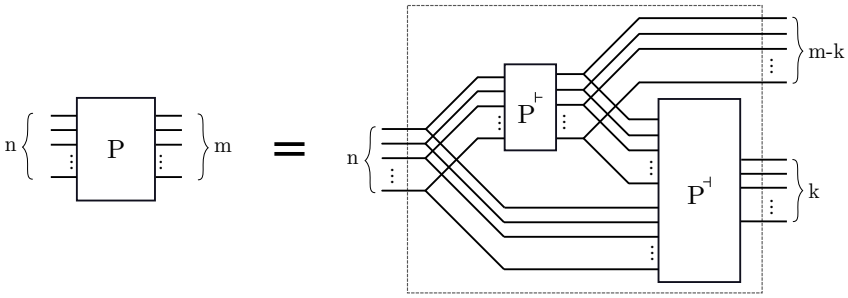


Figure 7 Schematic string diagram depiction of the decomposition of matrices.

which is visualized in Figure 7. Moreover, the matrices can be chosen so that $P^{\dashv}$ is stochastic and $P^{\vdash}$ sub-stochastic. If P is stochastic $P^{\vdash}$ can be chosen to be stochastic as well.

We can now deduce the known special case of arc reversal in OBN, stated e.g. in [4].

▶ **Corollary 17** (Arc reversal in OBNs). *Let (B, e) be an OBN with $B = (V, \ell, s, \mathrm{out})$ and two nodes $u, y \in V$, where u is a direct predecessor of y, i.e. $u \in \mathrm{pred}(y)$. Then there exists an OBN (B', e') with $B' = (V, \ell', s', \mathrm{out})$, evaluating to the same probability distribution, where $\ell'(v) = \ell(v)$, $s'(v) = s(v)$ if $v \neq u$ and $v \neq y$ and $y \in \mathrm{pred}(u)$. Thus the dependency between u and v is reversed.*

Arc reversal comes with a price: as can be seen in the proof, if u is associated with a matrix $P_u \colon n \to 1$ and y with a matrix $P_y \colon m + 1 \to 1$, then we have to create new matrices $P'_u \colon m + n + 1 \to 1$ and $P'_y \colon m + n \to 1$, causing new dependencies and increasing the size of the matrix. Hence arc reversal should be used sparingly.

After arc reversal a node might have duplicated inputs, which can be resolved by multiplying the corresponding matrix with ∇, thus reducing the dimension.

Next, we can use Lemma 16 to show that every matrix can be represented as an MBN. This MBN can always be built in a "minimal" way in that only m nodes are needed to represent a $n \to m$ matrix.

▶ **Lemma 18.** *Let $M : n \to m$ be a (sub-stochastic) matrix. Then there exists an MBN (B, e) with $B = (V, l, s, \mathrm{out})$ such that $M = M_e(B)$, $|V| = m$ and* out *is a bijection. Moreover, if M is stochastic we can guarantee that $e(l(v))$ is stochastic for all $v \in V$. If M is sub-stochastic we can guarantee that v_{front} – the first node in a topological ordering of all nodes V' – is the only node where $e(l(v))$ is sub-stochastic, all other nodes have stochastic matrices.*

▶ **Corollary 19.** *Let (B, e) be an MBN without inputs and assume that $M_e(B)$ is stochastic. Then there exists an OBN (B', e') such that $M_e(B) = M_{e'}(B')$.*

Proof. The result follows trivially from the assumptions because for a stochastic MBN without input ports $M_e(B)$ is simply a column vector holding a probability distribution. It is well known that every probability distribution can be represented by some (ordinary) Bayesian net. Alternatively the result follows directly from Lemma 18. ◀

We just argued that every MBN can be simplified so that it does not contain any unnecessary nodes and at most one sub-stochastic matrix. However, while Lemma 18 shows that these simplifications are always possible it is not helpful in practice: in fact in the proof we take the full matrix represented by an MBN and then split it into (coupled) single nodes. Since we chose to use MBNs in order not to deal with large distribution vectors in the first place, this approach is not practical. Instead, in the following we will describe methods which allow us to simplify an MBN without computing the matrix first.

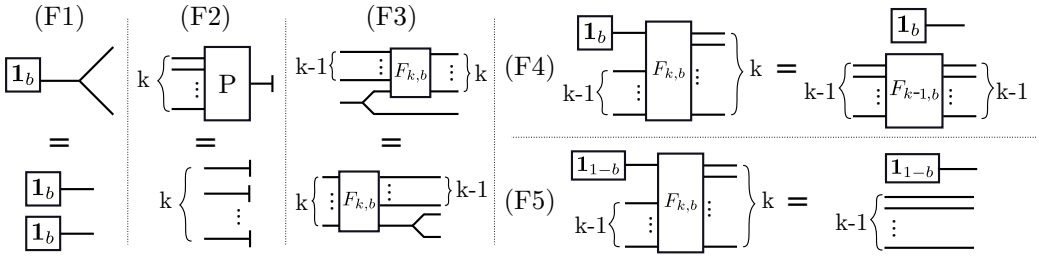


Figure 8 Equalities on sub-stochastic matrices. Note that (F2) holds only if P is stochastic and for (F4) and (F5) we have to assume $k > 1$.

First note that MBNs stemming from CNU operations can contain substructures that can locally be replaced by simpler ones. They are depicted in Figure 8.

▶ **Lemma 20.** *The equalities of Figure 8 hold for (sub-)stochastic matrices.*

As a result, it makes sense to first eliminate all of these substructures. Then there are two issues left to obtain an OBN. First, there are nodes that lost their direct connection with an output port (since output ports were terminated in a set operation or since we added an $F_{k,b}$-matrix). Those have to be merged with other nodes. Second, there are sub-stochastic matrices that have to be eliminated as well. The following lemma states that a node not connected to output ports can be merged with its direct successor nodes. This can introduce new dependencies between these successor nodes, but we remove one node from the network.

▶ **Lemma 21.** *lem Let $B = (V, \ell, s, \mathrm{out})$ be a causality graph, e an evaluation function such that (B, e) is an MBN. Assume that a node $v_0 \in V$ is not connected to an output port, i.e. for all $i \in \{1, \dots, m\} : v_0 \neq \mathrm{out}(o_i)$, and $e(\ell(v_0))$ is stochastic. Then there exists an MBN (B', e') with $B' = (V \backslash \{v_0\}, \ell', s', \mathrm{out})$ such that $M_e(B) = M_{e'}(B')$. Moreover, $e' \circ \ell'|_{\bar{V}} = e \circ \ell|_{\bar{V}}$ and $s'|_{\bar{V}} = s|_{\bar{V}}$ where $\bar{V} = V \backslash (\{v_0\} \cup \mathrm{succ}(v_0))$.*

The conditions on ℓ' and s' mean that the update on B is local as it does not affect the whole network. Only the direct successors of v_0 are affected.

Finally, we have to get rid of sub-stochastic matrices inside the MBN, which have been introduced by the ass and nas operations (we assume that we did not normalize yet). The idea is to exchange nodes labelled with sub-stochastic matrices with the predecessor nodes and move them to the front (as in Lemma 18). Once there, normalization is straightforward by normalizing the vectors associated to these nodes.

▶ **Lemma 22.** *Let $B = (V, \ell, s, \mathrm{out})$ be a causality graph without input ports, i.e. of type $0 \to m$, e an evaluation function such that (B, e) is an MBN. Furthermore we require that there is a one-to-one correspondence between output ports and nodes, i.e., out is a bijection.*

Assume that $V' \subseteq V$ is the set of all nodes equipped with sub-stochastic matrices, i.e. $e(\ell(v))$ is sub-stochastic for all $v \in V'$. Then there exists an OBN (B', e') with $B' = (V, \ell', s', \mathrm{out})$ such that $M_e(B) = M_{e'}(B') \cdot p_B$ where $p_B = \top_m \cdot M_e(B) \leq 1$ is the probability mass of B. Moreover, $e' \circ \ell'|_{\bar{V}} = e \circ \ell|_{\bar{V}}$ and $s'|_{\bar{V}} = s|_{\bar{V}}$ where $\bar{V} = V \backslash (V' \cup \mathrm{pred}^*(V'))$.

Note that $\frac{1}{p_B}$ (whenever $p_B \neq 0$) is the normalization factor that can be obtained by terminating all input ports of B. We do not have to compute p_B explicitly, but it can be derived from the probabilities of the nodes which have been moved to the front (see proof).

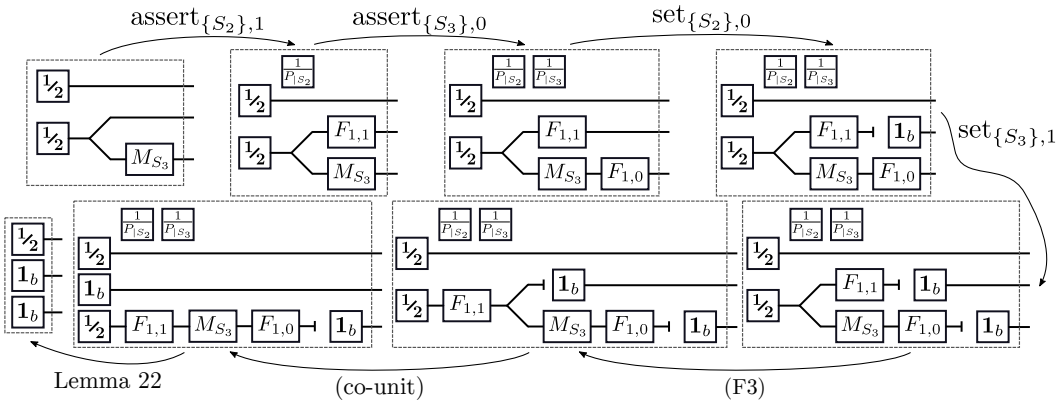


Figure 9 Exemplary update process for the success$_{t_4}$ operation of our running CNU example. Here $\mathbf{1/2} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ and $M_{S_3} = \begin{pmatrix} 1/3 & 1/2 \\ 2/3 & 1/2 \end{pmatrix}$.

▶ **Corollary 23.** *Let $B = (V, \ell, s, \mathrm{out})$ be a causality graph without input ports, i.e. of type $0 \to m$, e an evaluation function such that (B, e) is an OBN. Let $P = M_e(B)$.*

Then we can construct OBNs representing $\mathrm{set}_{A,b}(P), \mathrm{ass}_{A,b}(P), \mathrm{nas}_{A,b}(P)$*, where*

- *the* set *operation modifies only* $\{\mathrm{out}(o_i) \mid i \in A\}$ *and their direct successors* and
- *the* ass *and* nas *operations modify only* $\{\mathrm{out}(o_i) \mid i \in A\}$ *and their predecessors.*

The operations are costly whenever a node has many predecessors or direct successors. In a certain way this is unavoidable because our operations are related to the computation of marginals, which is NP-hard [6]. However, if the Bayesian network has a comparatively "flat" structure, we expect that the efficiency is rather high in the average case, as supported by our runtime results below. Applying the nas operation will introduce dependencies for the random variables corresponding to the pre- and post-conditions of a transition, however this effect is localized if we consider particular classes of Petri nets, such as free-choice nets [9].

▶ **Example 24.** *Figure 9 shows an update process, following a lazy evaluation strategy, for a Bayesian net representing the probability distribution from Figure 2.*

5 Implementation

In order to quantitatively assess the performance of MBNs we developed a prototypical C++ implementation of the concepts in this paper, allowing to read, write, simplify, generate, and visualize MBNs as well as perform operations on CNUs that update an underlying MBN. The implementation is open-source and freely available on GitHub.[4]

As a first means of obtaining runtime results we randomly generated CNs with a range of different parameters: e.g. number of places, number of places in a precondition of a transition, places in the initial marking etc. We then successively picked transitions at random to fire and performed the necessary operations to update the MBN and simplify it to an OBN.

We chose to guarantee a success rate of transition firing of around 1/3. We argue that given the fact that we model an observer with prior knowledge it is realistic to assume a certain rate of successful transitions. A very low sucess rate leads to an accumulation

[4] `https://github.com/bencabrera/bayesian_nets_program`

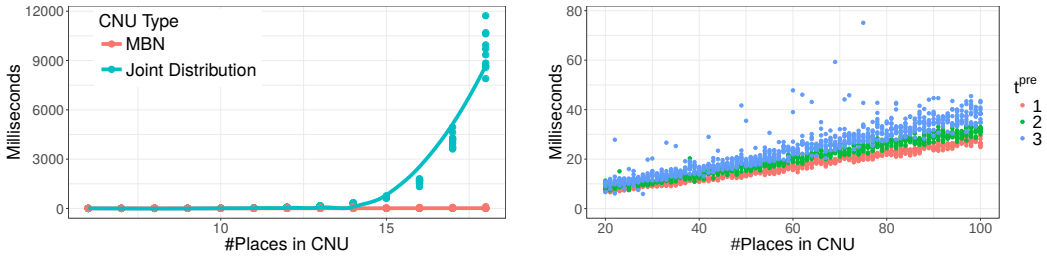


Figure 10 Averaged runtimes for performing 100 CNU operations using joint distributions or MBNs.

of successive $F_{k,b}$ matrices which can only be eliminated using the costly operations on substochastic matrices (see proof of Lemma 22). One could implement effective simplification strategies merging successive $F_{k,b}$ matrices – since composing 0,1 diagonal matrices yields again 0,1 diagonal matrices. However, this is out of scope of this publication.

The plot on the left of Figure 10 shows a comparison between run times when performing CNU operations directly on the joint distribution versus our MBN implementation. One can clearly observe the exponential increase when using the joint distribution while the MBN implementation in this setup stays relatively constant. The plot on the right of Figure 10 hints towards an increase in complexity when CNs – and thus MBNs – are more coupled. When increasing the maximum number of places in the precondition of a transition we observe an increase in run times. The number of outliers with a dramatic increase in run times seem to rise as well.

6 Conclusion

Related work. A concept similar to our nets with uncertainty has been proposed in [18], but without any mechanism for efficiently representing and updating the probability distribution. There are also links to Hidden Markov Models [25] for inferring probabilistic knowledge on hidden states by observing a model.

Bayesian networks were introduced by Pearl in [22] to graphically represent random variables and their dependencies. Our work has some similarities to his probabilistic calculus of actions (do-calculus) [23] which supports the empirical measurement of interventions. However, while Pearl's causal networks model describe true causal relationships, in our case Bayesian networks are just compact symbolic representations of huge probability distributions. There is also a notion of dynamic Bayesian networks [21], where a random variable has a separate instance for each time slice. We instead keep only one instance of every random variable, but update the BN itself. There is substantial work on updating Bayesian networks (for instance [15]) with the orthogonal aim of learning BNs from training data.

PROPs have been introduced in [20], foundations for term-based proofs have been studied in [19] and their graphical language has been developed in [27, 5]. Bayesian networks as PROPs have already been studied in [12] under the name of causal theories, as well as in [17, 16] in order to give a predicate/state transformer semantics to Bayesian networks. However, these papers do not explicitly represent the underlying graph structure and in particular they do not consider updates of Bayesian networks.

We use the results from [7] in order to show that our causality graphs are in fact term graphs, which are freely generated gs-monoidal categories, which in turn are CC-structured PROPs. Although this result is intuitive, it is non-trivial to show: given two terms with

isomorphic underlying graphs, each can be reduced to a normal form which can be converted into each other using the axioms of a CC-structured PROP. Similar results are given in [11, 2] for PROPs with multiplication and unit, in addition to comultiplication and counit.

Future work. We would like to investigate further operations on probability distributions, however it is unclear whether every operation can be efficiently implemented. For instance linear combinations of probability distributions seem difficult to handle.

Van der Aalst [29] showed that all reachable markings in certain free-choice nets can be inferred from their enabled transitions. An unrestricted observer may therefore be in a very strong position. Privacy research often considers statistical queries, such as how many records with certain properties exist in the database [10, 8]. To model such weaker queries we require labelled nets where instead of transitions we observe their labels. To implement this in BNs requires a disjunction of the enabledness conditions of all transitions with the same label. Furthermore we are interested in scenarios where certain transitions are unobservable.

References

1 L. Antova, C. Koch, and D. Olteanu. 10^(10^6) worlds and beyond: efficient representation and processing of incomplete information. *VLDB Journal*, 18(1021), 2009.

2 R. Bruni, F. Gadducci, and U. Montanari. Normal forms for algebras of connections. *Theoretical Computer Science*, 286(2):247–292, 2002.

3 B. Cabrera, T. Heindel, R. Heckel, and B. König. Updating probabilistic knowledge on Condition/Event nets using Bayesian networks, 2018. arXiv:1807.02566. `arXiv:1807.02566`.

4 A.Y.W. Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *Proc. of UAI '97 (Uncertainty in Artificial Intelligence)*, pages 72–79, 1997.

5 B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

6 G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990. `doi:10.1016/0004-3702(90)90060-D`.

7 A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Appl. Categor. Struct.*, 7:299–331, 1999.

8 M.L. Damiani. Location privacy models in mobile applications: conceptual view and research directions. *GeoInformatica*, 18(4):819–842, 2014. `doi:10.1007/s10707-014-0205-7`.

9 J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

10 C. Dwork. Differential privacy: A survey of results. In *Proc. of TAMC '08 (Theory and Applications of Models of Computation)*, pages 1–19. Springer, 2008. LNCS 4978.

11 M. Fiore and M. Devesas Campos. The algebra of directed acyclic graphs. In *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, pages 37–51. Springer, 2013. LNCS 7860.

12 B. Fong. Causal theories: A categorical perspective on Bayesian networks. Master's thesis, University of Oxford, 2012. arXiv:1301.6201.

13 B. Fong and D. I Spivak. Seven Sketches in Compositionality: An Invitation to Applied Category Theory. *ArXiv e-prints*, 2018. `arXiv:1803.05316`.

14 N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

15 N. Friedman and M. Goldszmidt. Sequential update of bayesian network structure. In Dan Geiger and Prakash Shenoy, editors, *Proc. of UAI '97 (Uncertainty in Artificial Intelligence)*, pages 165–174, 1997.

16 B. Jacobs and F. Zanasi. A predicate/state transformer semantics for Bayesian learning. In *Proc. of MFPS*, volume 325 of *ENTCS*, pages 185–200, 2016.

17 B. Jacobs and F. Zanasi. A formal semantics of influence in Bayesian reasoning. In *Proc. of MFCS*, volume 83 of *LIPIcs*, pages 21:1–21:14, 2017.

18 I. Jarkass and M. Rombaut. Dealing with uncertainty on the initial state of a Petri net. In *Proc. of UAI '98 (Uncertainty in Artificial Intelligence)*, pages 289–295, 1998.

19 C. Barry Jay. Languages for monoidal categories. *Journal of Pure and Applied Algebra*, 59(1):61–85, 1989.

20 S. MacLane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71(1):40–106, 1965.

21 K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.

22 J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proc. of the 7th Conference of the Cognitive Science Society*, pages 329–334, 1985. UCLA Technical Report CSD-850017.

23 J. Pearl. A probabilistic calculus of actions. In R. Lopez de Mantaras and D. Poole, editors, *Proc. of UAI '94 (Uncertainty in Artificial Intelligence)*, 1994.

24 J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

25 L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

26 W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.

27 P. Selinger. A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*, pages 289–355. Springer, 2011.

28 D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

29 W.M.P. van der Aalst. Markings in perpetual free-choice nets are fully characterized by their enabled transitions. In *Proc. of PN '18 (Petri Nets)*, pages 315–336. Springer, 2018. LNCS 10877.

30 F. Zanasi. *Interacting Hopf Algebras – the theory of linear systems*. PhD thesis, ENS Lyon, 2015.

Reachability in Timed Automata with Diagonal Constraints

Paul Gastin
LSV, CNRS, ENS Paris-Saclay, Université Paris–Saclay, France
gastin@lsv.fr

Sayan Mukherjee
Chennai Mathematical Institute, India
sayanm@cmi.ac.in

B. Srivathsan
Chennai Mathematical Institute, India
sri@cmi.ac.in

Abstract

We consider the reachability problem for timed automata having diagonal constraints (like $x - y < 5$) as guards in transitions. The best algorithms for timed automata proceed by enumerating reachable sets of its configurations, stored in a data structure called "zones". Simulation relations between zones are essential to ensure termination and efficiency. The algorithm employs a simulation test $Z \preccurlyeq Z'$ which ascertains that zone Z does not reach more states than zone Z', and hence further enumeration from Z is not necessary. No effective simulations are known for timed automata containing diagonal constraints as guards. We propose a simulation relation $\preccurlyeq^d_{LU}$ for timed automata with diagonal constraints. On the negative side, we show that deciding $Z \not\preccurlyeq^d_{LU} Z'$ is NP-complete. On the positive side, we identify a witness for $Z \not\preccurlyeq^d_{LU} Z'$ and propose an algorithm to decide the existence of such a witness using an SMT solver. The shape of the witness reveals that the simulation test is likely to be efficient in practice.

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

Keywords and phrases Timed Automata, Reachability, Zones, Diagonal constraints

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.28

Related Version A full version of the paper is available at [12], `https://arxiv.org/abs/1806.11007`.

Funding Partly supported by UMI RELAX, Infosys foundation (India) and Tata Consultancy Services - Innovation Labs (Pune, India)

1 Introduction

Timed automata [1] are models of real-time systems. They are finite automata equipped with real valued variables called *clocks*. These clocks can be used to constrain the time difference between events: for instance when an event a occurs a clock x can be set to 0 in the transition reading a, and when an event b occurs, the transition reading b can check if $x \leq 4$. These constraints on clocks are called *guards* and clocks which are made 0 in a transition are said to be *reset* in the transition. Guards of the form $x - y > 5$ are called *diagonal constraints*. They are convenient for checking conditions about events in the past: when an event c occurs, we want to check that between events a, b which occurred previously (in the said order), the time gap is at least 5. One can then reset a clock x at a, y at b

and check for $x - y > 5$ at c. It is known that such diagonal constraints do not add to the expressive power: each timed automaton can be converted into an equivalent one with no diagonal guards, that is, a *diagonal-free* automaton [4]. However, this conversion leads to an exponential blowup in the number of states, which is unavoidable in general [6].

State reachability is a basic question in timed automata verification. The problem is to decide if there exists a run of the automaton from the initial state to a given accepting state. This is known to be PSPACE-complete [1]. In practice, the best algorithms for reachability proceed by a forward analysis of the automaton: starting from its initial state, enumerate reachable sets of its configurations stored in the form of a data structure called *zones*. Zones are conjunctions of difference constraints (like $x - y < 6 \ \wedge \ w > 4$) which can be efficiently represented and manipulated using Difference Bound Matrices [11]. *Abstractions* of zones are necessary for termination and efficiency of this enumeration. These abstractions are functions with a finite range mapping each set of configurations to a bigger set. For diagonal free timed automata various implementable abstraction functions are known [2, 16]. For timed automata with diagonal constraints, no such abstraction functions are known and such a forward analysis method does not work. A naïve method would be to analyze the equivalent diagonal free automaton, but then this introduces a (systematic) blowup.

Abstractions of zones can be used in two ways during the forward analysis: explicitly or implicitly. In the explicit case, each time a new zone Z appears, the abstraction function $\mathfrak{a}$ is applied on it and $\mathfrak{a}(Z)$ is stored. Further enumeration starts from $\mathfrak{a}(Z)$. For this explicit method to work, $\mathfrak{a}(Z)$ needs an efficient representation. Hence only abstractions where $\mathfrak{a}(Z)$ is also a zone (also called *convex abstractions*) are used. Extra^+_{LU}[2] is the best known convex abstraction for diagonal free automata and is implemented in the state-of-the-art tool UPPAAL [3]. In the implicit case, zones are not extrapolated and are stored as they are. Each time a new zone Z appears, it is checked if there exists an already visited zone Z' such that $Z \subseteq \mathfrak{a}(Z')$. Intuitively this means that zone Z cannot see more states than Z' and hence the enumeration at Z can stop. Given that $\mathfrak{a}$ has finite range, the computation terminates. Since abstractions of zones are not stored explicitly, there is no restriction for $\mathfrak{a}$ to result in a zone, but an efficient inclusion test $Z \subseteq \mathfrak{a}(Z')$ is necessary as this test is performed each time a new zone appears. For diagonal-free automata, the best known abstraction is $\mathfrak{a}_{\preccurlyeq LU}$ and it subsumes Extra^+_{LU}. The inclusion test $Z \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ can be done in $\mathcal{O}(|X|^2)$ where X is the number of clocks [16]. In both cases - explicit or implicit - it is important to have an abstraction that transforms zones into as big sets as possible, so that the enumeration can terminate with fewer zone visits.

In this paper, we are interested in the implicit method for timed automata with diagonal constraints. Since the abstractions that are usually used are based on simulation relations, the inclusion test $Z \subseteq \mathfrak{a}(Z')$ boils down to a simulation test $Z \preccurlyeq Z'$ between zones. In particular, the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction is based on a simulation relation $\preccurlyeq_{LU}$ [2]. We choose to take this point of view: from the next section, we refrain from using abstractions and present them as simulations instead. We propose a simulation $\preccurlyeq^d_{LU}$ that is sound for diagonal constraints. Contrary to the diagonal free case, we show that the simulation test $Z \not\preccurlyeq^d_{LU} Z'$ is NP-complete. But on the positive side, we give a characterization of a witness for the fact that $Z \not\preccurlyeq^d_{LU} Z'$ and encode the existence of such a witness as the satisfiability of a formula in linear arithmetic. This gives an algorithm for $Z \not\preccurlyeq^d_{LU} Z'$. The shape of the witness shows that in practice the number of potential candidates would be low and the simulation test is likely to be efficient. We have implemented our algorithm in a prototype tool. Preliminary experiments demonstrate that the number of zones enumerated using $\preccurlyeq^d_{LU}$ simulation drastically reduces compared to the number of zones obtained by doing the diagonal free conversion followed by

a forward analysis using $\preccurlyeq_{LU}$. This simulation relation $\preccurlyeq^d_{LU}$ and the associated simulation test also open the door for extending optimizations studied for diagonal free automata [15], to the case of diagonal constraints; and also extending analysis of priced timed automata with diagonal constraints [7, 18].

Related work. Convex abstractions used for diagonal free timed automata had been in use also for diagonal constraints in tools like UPPAAL and KRONOS [19]. It was shown in [5] that this is incorrect: there are automata with diagonal constraints for which using Extra^+_{LU} will give a yes answer to the reachability problem, whereas the accepting state is not actually reachable in the automaton. This is because the extra valuations added during the computation enable guards which were originally not enabled in the automaton, leading to spurious executions. A non convex abstraction for diagonal constraints appears in [5], but the corresponding inclusion test is not known. The current algorithm for diagonal constraints proceeds by an abstraction refinement method [8].

Organization of the paper. Section 2 gives the preliminary definitions. In Section 3, we propose a simulation relation $\preccurlyeq^d_{LU}$ between zones and observe some of its properties. Section 4 gives an algorithm for $Z \not\preccurlyeq^d_{LU} Z'$ via reduction to an SMT formula. Section 5 shows that $Z \not\preccurlyeq^d_{LU} Z'$ is NP-hard by a reduction from 3-SAT. We report some experiments and conclude in Section 6. Missing proofs can be found in the extended version [12].

2 Preliminaries

Let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ the set of integers and $\mathbb{R}_{\geq 0}$ the set of non-negative reals. We denote the power set of a set S by $\mathcal{P}(S)$. A *clock* is a variable that ranges over $\mathbb{R}_{\geq 0}$. Fix a finite set of clocks X. A *valuation* v is a function which maps each clock $x \in X$ to a value in $\mathbb{R}_{\geq 0}$. Let $\Phi(X)$ denote the set of *clock constraints* ϕ formed using the following grammar: $\phi := x \sim c \mid x - y \sim c \mid \phi \wedge \phi$, where $x, y \in X$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$ Constraints of the form $x - y \sim c$ are called *diagonal constraints*. For a clock constraint ϕ, we write $v \models \phi$ if the constraint given by ϕ is satisfied by replacing each clock x in ϕ with $v(x)$. For $\delta \in \mathbb{R}_{\geq 0}$, we write $v + \delta$ for the valuation defined by $(v + \delta)(x) = v(x) + \delta$ for all clocks x. For a set R of clocks, we write $[R]v$ for the valuation obtained by setting each clock $x \in R$ to 0 and each $x \notin R$ to $v(x)$.

▶ **Definition 1** (Timed Automata). A *timed automaton* $\mathcal{A}$ is a tuple (Q, X, Δ, q_0, F) where Q is a finite set of states, X is a finite set of clocks, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of accepting states and $\Delta \subseteq Q \times \Phi(X) \times \mathcal{P}(X) \times Q$ is the transition relation. Each transition in Δ is of the form (q, g, R, q') where $g \in \Phi(X)$ is called the *guard* of the transition and $R \subseteq X$ is the set of clocks that are said to be *reset* at the transition.

Timed automata with no diagonal constraints are called *diagonal-free*. The semantics of timed automata is described as a transition system over the space of its *configurations*. A configuration is a pair (q, v) where $q \in Q$ is a state and v is a valuation. There are two kinds of transitions. *Delay* transitions are given by $(q, v) \to^\delta (q, v + \delta)$ for each $\delta \in \mathbb{R}_{\geq 0}$, and *action* transitions are given by $(q, v) \to^t (q', v')$ for each transition $t \in \Delta$ of the form (q, g, R, q'), if $v \models g$ and $v' = [R]v$. The initial configuration is $(q_0, \mathbf{0})$ where $\mathbf{0}$ denotes the valuation mapping each clock to 0. Note that the above transition system is infinite. A *run* of a timed automaton is an alternating sequence of delay and action transitions starting from the initial configuration: $(q_0, \mathbf{0}) \to^{\delta_0} \to^{t_0} (q_1, v_1) \to^{\delta_1} \to^{t_1} \cdots (q_n, v_n)$. A run of the

above form is said to be accepting if the last state $q_n \in F$. The *reachability problem* for timed automata is the following: given an automaton $\mathcal{A}$, decide if there exists an accepting run. This problem is known to be PSPACE-complete [1]. As the space of configurations is infinite, the main challenge in solving this problem involves computing a finite (and as small as possible) abstraction of the timed automaton semantics. In this section, we recall the reachability algorithm for the diagonal free case. For the rest of the section we fix a timed automaton $\mathcal{A}$.

Instead of working with configurations, standard solutions in timed automata analysis work with sets of valuations. The "successor" operation is naturally extended to the case of sets. For every transition t of $\mathcal{A}$ and every set of valuations W, we have a transition $\Rightarrow^t$ defined as follows: $(q, W) \Rightarrow^t (q', W')$ where $W' = \{v' \mid \exists v \in W,\ \exists \delta \in \mathbb{R}_{\geq 0} :\ (q, v) \rightarrow^t \rightarrow^\delta (q', v')\}$. Note that in the definition we have a $\rightarrow^\delta$ following the $\rightarrow^t$. This ensures that the $\Rightarrow$ successors (where $\Rightarrow = \bigcup_{t \in \Delta} \Rightarrow^t$) are closed under time successors. Moreover, the sets which occur during timed automata analysis using the $\Rightarrow$ relation have a special structure, and are called *zones*. A zone is a set of valuations which can be described using a conjunction of constraints of the form: $x \sim c$ or $x - y \sim c$ where $x, y \in X$ and $c \in \mathbb{N}$. Zones can be efficiently represented using Difference Bound Matrices (DBMs). To each automaton $\mathcal{A}$, we associate a transition system consisting of (state, zone) pairs: the *zone graph* $ZG(\mathcal{A})$ is a transition system whose nodes are of the form (q, Z) where q is a state of $\mathcal{A}$ and Z is a zone. The initial node is (q_0, Z_0) with $Z_0 = \{\mathbf{0} + \delta \mid \delta \geq 0\}$. Transitions are given by $\Rightarrow$.

▶ **Lemma 2.** *The zone graph $ZG(\mathcal{A})$ is sound and complete for reachability [9].*

Although the zone graph is a more succinct representation than the space of configurations, it could still be infinite. The reachability algorithm employs *simulation relations* between zones to obtain a finite abstraction of the zone graph that is sound and complete[1].

We start by defining this notion of simulations at the level of configurations. A *(time-abstract) simulation* between pairs of configurations of $\mathcal{A}$ is a reflexive and transitive relation $(q, v) \preccurlyeq (q', v')$ such that: $q = q'$; for every $(q, v) \rightarrow^\delta (q, v + \delta)$ there exists δ' such that $(q, v') \rightarrow^{\delta'} (q, v' + \delta')$ satisfying $(q, v + \delta) \preccurlyeq (q, v' + \delta')$; and if $(q, v) \rightarrow^t (q_1, v_1)$, then there exists $(q, v') \rightarrow^t (q_1, v_1')$ satisfying $(q_1, v_1) \preccurlyeq (q_1, v_1')$ for the same transition t. We say that (q, v) is simulated by (q', v'). We write $v \preccurlyeq v'$ if $(q, v) \preccurlyeq (q, v')$ for all states q. Simulations can be extended to relate zones in the natural way: we write $Z \preccurlyeq Z'$ if for all $v \in Z$ there exists $v' \in Z'$ such that $v \preccurlyeq v'$. A simulation relation $\preccurlyeq$ is said to be *finite* if there exists $N \in \mathbb{N}$ such that for all $n > N$ and every sequence of zones $\{Z_1, Z_2, \ldots, Z_n\}$, there exists $i < j \leq n$ such that $Z_j \preccurlyeq Z_i$.

Reachability algorithm. The input to the algorithm is a timed automaton $\mathcal{A}$. The algorithm maintains two lists, Passed and Waiting, and makes use of a finite simulation relation $\preccurlyeq$ between zones. The initial node (q_0, Z_0) is added to the Waiting list. Wlog. we assume that q_0 is not accepting. The algorithm repeatedly performs the following steps:

Step 1. If Waiting is empty, then return "$\mathcal{A}$ has no accepting run"; else pick (and remove) a node (q, Z) from Waiting.

Step 2. For each successor $(q, Z) \Rightarrow (q_1, Z_1)$ such that $Z_1 \neq \emptyset$ perform the following operations: if q_1 is accepting, return "$\mathcal{A}$ has an accepting run"; else check if there exists

[1] Existing reachability algorithms make use of what are known as abstraction operators [2, 16], which are based on simulation relations. Instead of abstractions, we choose to present the algorithm directly using simulations between zones.

a node (q_1, Z_1') in Passed or Waiting such that $Z_1 \preccurlyeq Z_1'$: if yes, ignore the node (q_1, Z_1), otherwise add (q_1, Z_1) to Waiting.

Step 3. Add (q, Z) to Passed and proceed to Step 1.

▶ **Theorem 3.** *The reachability algorithm terminates with a correct answer.*

The reachability algorithm relies on an operation $Z_1 \preccurlyeq Z_1'$, where $\preccurlyeq$ is some finite simulation relation as defined earlier. It has been shown that for the simulation relation $\preccurlyeq_{LU}$ of [2] which works for diagonal free automata, checking $Z \preccurlyeq_{LU} Z'$ can be done in time $O(|X|^2)$ [16]. Hence in diagonal free timed automata, this simulation test is as efficient as checking normal inclusion $Z \subseteq Z'$. The successor computation can also be implemented in $O(|X|^2)$ [20] using Difference Bound Matrices. These matrices can also be viewed as graphs. We recall this graph-based representation of zones and some of its properties.

▶ **Definition 4** (Distance graph)**.** A *distance graph* G has clocks as vertices, with an additional special vertex x_0 representing constant 0. Between every two vertices there is an edge with a *weight* of the form $(\lhd, c)$ where $c \in \mathbb{Z}$ and $\lhd \in \{\leq, <\}$ or $(\lhd, c) = (<, \infty)$. An edge $x \xrightarrow{\lhd c} y$ represents a constraint $y - x \lhd c$: or in words, the distance from x to y is bounded by c. We let $[\![G]\!]$ be the set of valuations of clock variables satisfying all the constraints given by the edges of G with the restriction that the value of x_0 is 0.

We will sometimes write 0 instead of x_0 for clarity. An arithmetic over the weights $(\lhd, c)$ can be defined as follows [3].

Equality $(\lhd_1, c_1) = (\lhd_2, c_2)$ if $c_1 = c_2$ and $\lhd_1 = \lhd_2$.

Addition $(\lhd_1, c_1) + (\lhd_2, c_2) = (\lhd, c_1 + c_2)$ where $\lhd = <$ iff either $\lhd_1$ or $\lhd_2$ is $<$.

Total order $(\lhd_1, c_1) < (\lhd_2, c_2)$ if either $c_1 < c_2$ or ($c_1 = c_2$ and $\lhd_1 = <$ and $\lhd_2 = \leq$).

This arithmetic lets us talk about the weight of a path as the sum of the weights of its edges.

A cycle in a distance graph G is said to be *negative* if the sum of the weights of its edges is at most $(<, 0)$. A distance graph is in *canonical form* if there are no negative cycles and the weight of the edge from x to y is the lower bound of the weights of paths from x to y. Given a distance graph, its canonical form can be computed by using an all-pairs shortest paths algorithm like Floyd-Warshall's [3] in time $\mathcal{O}((|X| + 1)^3)$ where $|X|$ is the number of clocks. Note that the number of vertices in the distance graph is $|X| + 1$. A folklore result is that: a distance graph G has no negative cycles iff $[\![G]\!] \neq \emptyset$. Given two distance graphs G_1, G_2 (not necessarily in their canonical form), we define $\min(G_1, G_2)$ to be the distance graph obtained by setting for each $x \to y$ the minimum of the corresponding weights in G_1 and G_2. For two distance graphs G_1 and G_2, we have $[\![\min(G_1, G_2)]\!] = [\![G_1]\!] \cap [\![G_2]\!]$.

A simulation relation for timed automata with diagonal constraints was proposed in [5], but it has not been used in the reachability algorithm since no algorithm for the zone simulation test was known.

3 A new simulation relation in the presence of diagonal constraints

In this section, we introduce a new simulation relation $\preccurlyeq^d_{LU}$ which extends the $\preccurlyeq_{LU}$ simulation of [2]. For this, we first assume that all guards in timed automata are rewritten in the form $x - y \lhd c$ or $x \lhd c$, where $c \in \mathbb{Z}$ and $\lhd \in \{<, \leq\}$. We will also assume that X is a set of clocks including the 0 clock.

▶ **Definition 5** (LU-bounds)**.** An *LU bounds function* is a pair of functions $L : X \times X \mapsto \mathbb{Z} \cup \{\infty\}$ and $U : X \times X \mapsto \mathbb{Z} \cup \{-\infty\}$ mapping each clock difference $x - y$ to a constant or ∞ or $-\infty$ such that the conditions below are satisfied (we write $L(x - y), U(x - y)$ for $L(x, y)$ and $U(x, y)$ respectively):

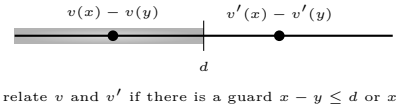


Figure 1 Black dots illustrate the values of $v(x) - v(y)$ and $v'(x) - v'(y)$. The value $v(x) - v(y)$ satisfies the guard $x - y \lhd d$ but $v'(x) - v'(y)$ does not satisfy the same guard.

- either $L(x-y) = \infty$ and $U(x-y) = -\infty$, or $L(x-y) \leq U(x-y)$ for all distinct pairs of clocks $x, y \in X$,
- $L(x-0) = 0$ and $U(0-x) = 0$ for all non zero clocks $x \in X$

The L stands for *lower* and U stands for *upper*. Intuitively, each LU-bounds function corresponds to a set of guards given by $x - y \lhd c$ with $L(x-y) \leq c \leq U(x-y)$. We will now define a simulation relation $\preccurlyeq^d_{LU}$ between valuations parameterized by LU-bounds. The idea is to give a relation $v \preccurlyeq^d_{LU} v'$ such that v' satisfies all guards compatible with the parameter LU that v satisfies. To achieve this, the situation as illustrated in Figure 1 needs to be avoided. This is formalized by the following definition and the subsequent lemma.

▶ **Definition 6** (LU-preorder $\preccurlyeq^d_{LU}$)**.** Let LU be a bounds function. A valuation v' simulates a valuation v with respect to LU, written as $v \preccurlyeq^d_{LU} v'$, if for every pair of distinct clocks $x, y \in X$ the following hold:

- $v'(x) - v'(y) < L(x-y)$ if $v(x) - v(y) < L(x-y)$
- $v'(x) - v'(y) \leq v(x) - v(y)$ if $L(x-y) \leq v(x) - v(y) \leq U(x-y)$

For a valuation v, we write $\langle v \rangle_{LU}$ for the set of all v' such that $v \preccurlyeq^d_{LU} v'$.

▶ **Lemma 7.** *Let x, y be distinct clocks in X, and $x - y \lhd c$ with $c \in \mathbb{Z}$ be a guard. Let LU be a bounds function such that $L(x-y) \leq c \leq U(x-y)$. Then, for every pair of valuations v, v' such that $v \preccurlyeq^d_{LU} v'$, if valuation $v \models x - y \lhd c$ then $v' \models x - y \lhd c$.*

The next lemma shows that time delay preserves $\preccurlyeq^d_{LU}$ from two valuations v and v' with $v \preccurlyeq^d_{LU} v'$. In fact, it is strong in the sense that if we delay δ from v, then the same delay from v' satisfies the LU preorder conditions. The proof of this lemma uses the fact that $L(x-0) = 0$ and $U(0-x) = 0$ for all non-zero clocks x.

▶ **Lemma 8.** *Let LU be a bounds function. For every pair of valuations v and v', if $v \preccurlyeq^d_{LU} v'$, then $v + \delta \preccurlyeq^d_{LU} v' + \delta$ for all $\delta \geq 0$.*

The next lemma shows that resets preserve $\preccurlyeq^d_{LU}$ under certain conditions on LU.

▶ **Lemma 9.** *Let LU be a bounds function satisfying $U(x-0) \geq U(x-y)$ for all $y \in X$ and $L(0-y) \leq L(x-y)$ for all $x \in X$. Then, $v \preccurlyeq^d_{LU} v'$ implies $[R]v \preccurlyeq^d_{LU} [R]v'$ for every $R \subseteq X$.*

The LU preorder can be extended to configurations: $(q, v) \preccurlyeq^d_{LU} (q, v')$ if $v \preccurlyeq^d_{LU} v'$. The above three lemmas give the necessary ingredients to generate an LU bounds function from a timed automaton $\mathcal{A}$ such that the associated LU preorder is a simulation on its space of configurations.

Let $\mathcal{G}$ be a set of constraints. We construct a new set $\overline{\mathcal{G}}$ from $\mathcal{G}$ in the following way:

- Add all the constraints of $\mathcal{G}$ to $\overline{\mathcal{G}}$
- For each clock $x \in X$, add the constraints $x \leq 0$ and $-x \leq 0$ to $\overline{\mathcal{G}}$
- For each constraint $x - y \lhd c \in \mathcal{G}$, add the constraints $x \lhd c$ and $-y \lhd c$ to $\overline{\mathcal{G}}$
- Remove all constraints of the form $x \lhd c_1$ where $c_1 \in \mathbb{R}_{<0}$ and constraints of the form $-x \lhd c_2$ where $c_2 \in \mathbb{R}_{>0}$ from $\overline{\mathcal{G}}$.

We define an LU-bounds function on $\overline{\mathcal{G}}$ in the natural way: for each pair of clocks $x, y \in X$, we set $L(x-y) = \min\{c \mid x - y \lhd c \in \overline{\mathcal{G}}\}$ and $U(x-y) = \max\{c \mid x - y \lhd c \in \overline{\mathcal{G}}\}$. If there are no guards of the form $x - y \lhd c$ in $\overline{\mathcal{G}}$, then we set $L(x-y)$ to be ∞ and $U(x-y)$ to be $-\infty$. Note that since $\overline{\mathcal{G}}$ contains the constraints $x \leq 0$ and has no constraints $x \lhd c$ where $c \in \mathbb{R}_{<0}$, $L(x-0) = 0$ for all $x \in X$. Similarly, $U(0-x) = 0$ for all $x \in X$. For a timed automaton $\mathcal{A}$, let $\mathcal{G}_\mathcal{A}$ be the set of guards present in $\mathcal{A}$. The *LU-bounds of $\mathcal{A}$* is the LU-bounds function defined on $\overline{\mathcal{G}_\mathcal{A}}$. The next theorem follows from Lemmas 7, 8 and 9.

▶ **Theorem 10.** *For every timed automaton $\mathcal{A}$, the relation $\preccurlyeq^d_{LU}$ obtained from the LU-bounds of $\mathcal{A}$ is a simulation relation on its configurations.*

We use this simulation relation extended to zones in the reachability algorithm, as described in Page 4. To do so, we need to give an algorithm for the simulation test $Z \preccurlyeq^d_{LU} Z'$, and show that $\preccurlyeq^d_{LU}$ is finite. Correctness and termination follow from Theorem 3. We first describe the simulation test, and then prove finiteness. Observe that $Z \not\preccurlyeq^d_{LU} Z'$ iff there exists $v \in Z$ such that $\langle v \rangle_{LU} \cap Z' = \emptyset$. We give a distance graph representation for $\langle v \rangle_{LU}$.

▶ **Definition 11** (Distance graph for $\langle v \rangle_{LU}$)**.** Given a valuation v and an LU bounds function, we construct distance graph $G^{LU}_{\langle v \rangle}$ as follows. For every pair of distinct clocks $x, y \in X$, add the edges:

- $y \to x$ with weight $(<, L(x-y))$, if $v(x) - v(y) < L(x-y)$,
- $y \to x$ with weight $(\leq, v(x) - v(y))$, if $L(x-y) \leq v(x) - v(y) \leq U(x-y)$.

Using Definition 6 we can show that $[\![G^{LU}_{\langle v \rangle}]\!]$ equals $\langle v \rangle_{LU}$. The properties of distance graphs as described in Page 5 then lead to the following theorem.

▶ **Theorem 12.** *Let Z, Z' be zones such that Z' is non-empty, and let LU be a bounds function. Let $G_{Z'}$ be the canonical distance graph of Z'. Then, $Z \not\preccurlyeq^d_{LU} Z'$ iff there is a valuation $v \in Z$ and a negative cycle in $\min(G^{LU}_{\langle v \rangle}, G_{Z'})$ in which no two consecutive edges are from $G_{Z'}$.*

A witness to the fact that $Z \not\preccurlyeq^d_{LU} Z'$ is therefore a $v \in Z$ and a negative cycle of a certain shape given by Theorem 12. As explained in Section 4, existence of such a witness can be encoded as satisfiability of a formula in linear arithmetic. This gives an NP procedure. A satisfying assignment to the formula reveals a valuation $v \in Z$ and a corresponding negative cycle across $G^{LU}_{\langle v \rangle}$ and $G_{Z'}$. Although there is no fixed bound on the length of this negative cycle (contrary to the diagonal free case), note that each $y \to x$ edge from $G^{LU}_{\langle v \rangle}$ in the negative cycle needs to have finite $U(x-y)$ and $L(x-y)$ constants **(apart from $x \to 0$ edges)**. If for an automaton, many pairs of clocks have no diagonal constraints (which we believe occurs often in practice) then this simulation test would need to enumerate only a small number of cycles.

The final step is to show that $\preccurlyeq^d_{LU}$ is finite. We make use of a notation: we write $\downarrow Z$ to be the set of valuations u such that $u \preccurlyeq^d_{LU} v$ for some $v \in Z$. Note that $Z \not\preccurlyeq^d_{LU} Z'$ implies $\downarrow Z \neq \downarrow Z'$.

▶ **Theorem 13.** *The simulation relation $\preccurlyeq^d_{LU}$ is finite for every LU bounds function.*

Proof. We will first show that for any zone Z, $\downarrow Z$ is a union of *d-regions* (parameterized by LU) which are defined below. We will subsequently show that there are only finitely many d-regions. The observation that $Z \not\preccurlyeq^d_{LU} Z'$ implies $\downarrow Z \neq \downarrow Z'$ then proves the theorem.

Given a valuation v and LU-bounds function, we define the following relations over pairs of clocks:

- $y \xrightarrow{1} x$ if $v(x) - v(y) < L(x-y)$
- $y \xrightarrow{2} x$ if $L(x-y) \leq v(x) - v(y) \leq U(x-y)$

A d-region R is a set of valuations that satisfies the following:

- all valuations in R have the same $\xrightarrow{1}$ and $\xrightarrow{2}$ relations.
- for every subset $S = \{y_1 \xrightarrow{2} x_1, y_2 \xrightarrow{2} x_2, \ldots, y_k \xrightarrow{2} x_k\}$ of ordered pairs of clocks, every valuation in R satisfies one of the following constraints: either $\left(\sum_{i=1}^{i=k} x_i - y_i = c\right)$ or $c - 1 < \left(\sum_{i=1}^{i=k} x_i - y_i\right) < c$ for an integer c satisfying $\sum_{i=1}^{i=k} L(x_i - y_i) \leq c \leq \sum_{i=1}^{i=k} U(x_i - y_i)$.

We will now show that if a d-region R intersects $\downarrow Z$ then $R \subseteq \downarrow Z$. Let $v \in R$ be such that $v \in \downarrow Z$. Let v' be another valuation in R. Suppose $v' \notin \downarrow Z$. Then $\langle v' \rangle_{LU} \cap Z = \emptyset$. That is, $\min(G^{LU}_{\langle v' \rangle}, G_Z)$ has a negative cycle; let us call it $N_{v'}$. Let N_v be the cycle $N_{v'}$ with the edges coming from $G^{LU}_{\langle v' \rangle}$ replaced with the same edges from $G^{LU}_{\langle v \rangle}$. We want to show that N_v is negative. Since v and v' come from the same region R, we have:

- The weight of a type 1 edge $y_i \xrightarrow{1} x_i$ is $(<, L(x_i - y_i))$ in both N_v and $N_{v'}$. Let $(<, S_1)$ be the sum of the weights of the type 1 edges. This sum is the same in N_v and $N_{v'}$.
- We let $(\leq, S_2) = (\leq, \sum_i v(x_i) - v(y_i))$ and $(\leq, S_2') = (\leq, \sum_i v'(x_i) - v'(y_i))$ be the sum of the weights of type 2 edges $y_i \xrightarrow{2} x_i$ in N_v and $N_{v'}$ respectively. Then, for some integer c, either $S_2 = S_2' = c$ or $c - 1 < S_2 < c$ and $c - 1 < S_2' < c$.

Also the edges coming from G_Z have the same weight in N_v and $N_{v'}$. Call $(\lhd_3, S_3)$ the sum of the weights of the edges coming from G_Z. Finally, let $(\lhd, S = S_1 + S_2 + S_3)$ and $(\lhd, S' = S_1 + S_2' + S_3)$ be the weights of N_v and $N_{v'}$ respectively. Since $N_{v'}$ is negative, $(\lhd, S')$ is at most $(<, 0)$. Now, S_1 and S_3 are integers, and using the relation between S_2 and S_2', we deduce that N_v is also negative. This entails $\langle v \rangle_{LU} \cap Z = \emptyset$, and contradicts the assumption $v \in \downarrow Z$. We get $R \subseteq \downarrow Z$, thereby showing that each $\downarrow Z$ is a union of d-regions.

Each d-region depends only on the $\xrightarrow{1}$ and $\xrightarrow{2}$ relations and the values of c for each subset S of $\xrightarrow{2}$ edges. Since number of clocks is finite, the number of possible relations $\xrightarrow{1}$ and $\xrightarrow{2}$ is finite. For each such relations, the possible values for the constants c is finite. Thus there are only finitely many d-regions. ◀

4 Algorithm for $Z \not\preceq^d_{LU} Z'$

Theorem 12 gives a witness for the fact that $Z \not\preceq^d_{LU} Z'$. In this section, we encode the existence of this witness as an SMT formula over linear arithmetic. For clarity of exposition, we will also restrict to timed automata having no strict constraints as guards, that is, every guard is of the form $x - y \leq c$ or $x \leq c$. This would in particular imply that in the zones obtained during the forward analysis, there will be no strict constraints.

▶ **Definition 14** (Satisfiability modulo Linear Arithmetic)**.** Let Prop be a set of propositional variables, and Vars a set of variables ranging over reals. An atomic term is a constraint of the form $c_1x_1 + c_2x_1 + \cdots + c_kx_k \sim d$ where $c_1, \ldots, c_n, d \in \mathbb{Z}$ and $x_1, x_2, \ldots, x_k \in$ Vars and $\sim \in \{\leq, <, =, >, \geq\}$. A formula in *linear arithmetic* is a boolean combination of propositional variables and atomic terms. Formula ϕ is satisfiable if there exists an assignment of boolean values to propositions in Prop, and real values to variables in Vars such that replacing every occurrence of the variables and propositions by the assigment evaluates ϕ to true.

The next lemma follows from [17].

▶ **Lemma 15.** *Satisfiability of a formula in linear arithmetic is in* NP.

Fix two zones Z, Z' and a bounds function LU. Zones Z and Z' are given by their canonical distance graphs G_Z and $G_{Z'}$. We write c_{yx} for the weight of the edge $y \to x$ in G_Z and c'_{yx} for the weight of $y \to x$ in $G_{Z'}$. Further we assume that the set of clocks is $\{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{n}\}$. The final formula will be obtained by constructing suitable intermediate subformulas as explained below:

Step 1. Guess a $v \in Z$.

Step 2. Guess a subset of edges $y \to x$ which forms a cycle (or a disjoint union of cycles).

Step 3. Guess a colour for each edge $y \to x$ in the cycle: red or blue. No two consecutive edges in the cycle can both be red. Red edges correspond to edges from $G_{Z'}$. Blue edges correspond to edges from $G^{LU}_{\langle v \rangle}$.

Step 4. Assign weights to each edge $y \to x$: if it is coloured red, the weight is c'_{yx} (edge weight of $G_{Z'}$). If the edge $y \to x$ is blue, assign weight according to the following cases:

- $w_{yx} = (<, L(x-y))$ if $v(x) - v(y) < L(x-y)$
- $w_{yx} = (\leq, v(x) - v(y))$ if $L(x-y) \leq v(x) - v(y) \leq U(x-y)$

Add up the weights of all the edges (the comparison $<$ or $\leq$ component of the weight can be maintained using a boolean). If there are no strict edges (that is with weight $<$) in the chosen cycle, check if the sum is < 0. Else, check if the sum is ≤ 0.

Formula for Step 1. We first guess a valuation $v \in Z$. We use real variables $v_0, v_1, \ldots, v_n$ to denote a valuation. These variables should satisfy the constraints given by Z:

$$v_0 = 0 \quad \text{and} \quad \bigwedge_{x,y \in \{0,\ldots,n\}} v_x - v_y \leq c_{yx} \tag{1}$$

Call the above formula $\Phi_1(\bar{v})$ where $\bar{v} = (v_0, \ldots, v_n)$. A satisfying assignment to Φ_1 corresponds to a valuation in Z.

Formula for Step 2. We now need to guess a set of edges of the form $y \to x$ which forms a cycle, or a disjoint union of simple cycles. We will also ensure that no vertex appears in more than one cycle. We will use boolean variables e_{ij} for $i, j \in \{0, \ldots, n\}$ and $i \neq j$.

The cycle must be non-empty.

$$\bigvee_{0 \leq i,j \leq n, j \neq i} e_{ij} \tag{2}$$

If we pick an incoming edge to a clock, then we need to pick an outgoing edge.

$$\bigwedge_{0 \leq i \leq n} \Big(\bigvee_{0 \leq j \leq n, j \neq i} e_{ji} \Big) \implies \Big(\bigvee_{0 \leq j \leq n, j \neq i} e_{ij} \Big) \tag{3}$$

We do not pick more than one outgoing or incoming edges for each clock.

$$\bigwedge_{0 \leq i \leq n} \quad \bigwedge_{\substack{0 \leq j,k \leq n \\ j \neq k, i \neq j, i \neq k}} \neg(e_{ij} \wedge e_{ik}) \ \wedge \ \neg(e_{ji} \wedge e_{ki}) \tag{4}$$

Conjunction of (2, 3, 4) gives a formula $\Phi_2(\bar{e})$ over variables $\bar{e} = \{e_{01}, \ldots, e_{nn-1}\}$.

▶ **Lemma 16.** *Let $\sigma_2 : \bar{e} \mapsto \{\text{true}, \text{false}\}$ be an assignment which satisfies Φ_2. Then the set of edges $\{x \to y\}$ such that $\sigma_2(e_{xy})$ is true forms a vertex-disjoint union of cycles.*

Formula for Step 3. To colour the edges of the cycle formed by e_{ij}, we will use boolean variables r_i for $0 \leq i \leq n$ to color the source of the red edges. Once the red edges are determined, the blue edges are also uniquely determined. Only edges chosen by $\bar{e}$ are coloured red, and no two consecutive edges can be coloured red.

$$\bigwedge_{0\leq i\leq n} \Big(r_i \implies \bigvee_{0\leq j\leq n} e_{ij} \wedge \neg r_j\Big) \tag{5}$$

Then, red edges are edges with corresponding source i satisfying r_i. So for all $i, j \in \{0, \ldots, n\}$ with $i \neq j$ we introduce the macro $\text{red}_{ij} := e_{ij} \wedge r_i$. Blue edges are those that have been chosen for the cycle and have not been coloured red: $\text{blue}_{ij} := e_{ij} \wedge \neg r_i$. Each blue edge should satisfy one of the two conditions mentioned in Definition 11.

$$\bigwedge_{i,j\in\{0,\ldots,n\},i\neq j} \text{blue}_{ij} \implies v_j - v_i \leq U(j-i) \tag{6}$$

Conjunction of (5) and (6) gives formula Φ_3.

▶ **Lemma 17.** *Let σ_3 be an assignment to variables $\bar{v}$, $\bar{e}$ and $\bar{r}$. Suppose σ_3 is a satisfying assignment for $\Phi_1 \wedge \Phi_2 \wedge \Phi_3$. Then, the set of edges with $\sigma_3(e_{ij})$ being true forms a collection of vertex disjoint cycles using edges from $G_{Z'}$ or from $G^{LU}_{\langle v\rangle}$ for some $v \in Z$.*

Formula for Step 4. The last step is to add up weights of the red and blue edges. We make use of real-valued variables w_i for each source i of an edge. We associate weights of red and blue edges.

$$\begin{aligned}\bigwedge_{i,j\in\{0,\ldots,n\},i\neq j} (\text{red}_{ij} \implies w_i = c'_{ij}) \wedge ((\text{blue}_{ij} \wedge \mathit{condition}_1) &\implies w_i = L(j-i)) \\ \wedge ((\text{blue}_{ij} \wedge \mathit{condition}_2) &\implies w_i = v_j - v_i)\end{aligned} \tag{7}$$

where, $\mathit{condition}_1 := v_j - v_i < L(j-i)$ and $\mathit{condition}_2 := L(j-i) \leq v_j - v_i \leq U(j-i)$.

Uncoloured edges take weight 0,

$$\bigwedge_{0\leq i\leq n} \Big(\bigwedge_{0\leq j\leq n, j\neq i} \neg e_{ij}\Big) \implies (w_i = 0) \tag{8}$$

A boolean variable strict is true if one of the blue edges has a weight of the form $(<, c)$.

$$\text{strict} \iff \bigvee_{i,j\in\{0,\ldots,n\},i\neq j} \text{blue}_{ij} \wedge \mathit{condition}_1 \tag{9}$$

The final formula checks if the sum of the weights is at most $(<, 0)$.

$$((\textstyle\sum_{0\leq i\leq n}\ w_i) < 0) \vee [\ \text{strict}\ \wedge ((\textstyle\sum_{0\leq i\leq n}\ w_i) = 0)] \tag{10}$$

Conjunction of (7), (8) and (10) gives formula Φ_4. The final formula is $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4$.

▶ **Theorem 18.** *Formula Φ as constructed above is satisfiable iff $Z \not\preccurlyeq^d_{LU} Z'$.*

Note that there are $\mathcal{O}(n+1)$ real variables v_i, w_i, and $\mathcal{O}((n+1)^2)$ booleans e_{ij}, r_i. Given the representations of Z, Z' and the LU bounds, the entire formula Φ can be computed in $\mathcal{O}((n+1)^3)$, with formula (4) taking the maximum time. This gives an NP procedure for $Z \not\preccurlyeq^d_{LU} Z'$ (c.f. Lemma 15).

5 Checking $Z \not\preccurlyeq^d_{LU} Z'$ is NP-hard

We will consider a special kind of LU bounds, which already turns out to be hard. We say that an LU bounds is *symmetric* if $L(x-y) = -U(y-x)$ for all distinct pairs of clocks x, y. This symmetry gives rise to some nice properties which we will use to show hardness.

▶ **Lemma 19.** *Let v, v' be valuations and LU a symmetric bounds function. Then, $v \preccurlyeq^d_{LU} v'$ iff for all distinct pairs of clocks x, y (denoting $a = v(x) - v(y)$ and $a' = v'(x) - v'(y)$):*

- *either both a' and a are $< L(x-y)$*
- *or $L(x-y) \le a' = a \le U(x-y)$*
- *or both a' and a are $> U(x-y)$.*

Proof. Since $L(x-y) = -U(y-x)$, we deduce that $L(x-y) \le a \le U(x-y)$ iff $L(y-x) \le -a \le U(y-x)$. The rest follows by applying Definition 6 on a, a' and $-a, -a'$. ◀

Thanks to the above lemma, when LU is symmetric: $v \preccurlyeq^d_{LU} v'$ iff $v' \preccurlyeq^d_{LU} v$, and hence $\preccurlyeq^d_{LU}$ an equivalence over valuations. To make this explicit, we will write $v \simeq_{LU} v'$ instead of $v \preccurlyeq^d_{LU} v'$, and $[v]_{LU}$ instead of $\langle v \rangle_{LU}$ for symmetric LU. With this definition, for symmetric LU, we get $Z \not\preccurlyeq^d_{LU} Z'$ iff there exists $v \in Z$ such that for all $v' \simeq_{LU} v$, we have $v' \notin Z'$. Throughout this section, we will fix a symmetric LU bounds function.

The second condition in Lemma 19 constrains the difference between certain pairs of clocks to a constant value for all valuations in an equivalence class of $\simeq_{LU}$. We formalize this notion. Let v be a valuation. Two clocks x and y are said to be *tight in* v if $L(x-y) \le v(x) - v(y) \le U(x-y)$. We denote this by $x \multimap\!\!-\!\!\multimap y$ (can be read as x and y are tied to each other). Notice that $\multimap\!\!-\!\!\multimap$ is symmetric. Let $\multimap\!\!-\!\!\multimap^*$ (can again be read as the tight relation) denote the reflexive and transitive closure of $\multimap\!\!-\!\!\multimap$. The $\multimap\!\!-\!\!\multimap^*$ relation is an equivalence over clocks. For every $v' \in [v]_{LU}$, Lemma 19 gives: $v'(x) - v'(y) = v(x) - v(y)$ when $x \multimap\!\!-\!\!\multimap^* y$ and $v'(x) - v'(y) < L(x-y)$ when $x \not\multimap\!\!-\!\!\multimap^* y$ and $v(x) - v(y) < L(x-y)$. Notice also that the $\multimap\!\!-\!\!\multimap^*$ equivalence classes are identical for v' and v when $v' \simeq_{LU} v$.

Next, we make an observation about zones which do not have strict constraints (like $x - y < c$). We say that a zone Z is *topologically closed* if every edge $y \to x$ in the canonical distance graph of Z has weight of the form $(\le, c)$ with $c \in \mathbb{Z}$, or $(<, \infty)$. A valuation v mapping each x to an integer is said to be an *integral valuation*. The next proposition says that for certain topologically closed zones Z and Z', if $Z \not\preccurlyeq^d_{LU} Z'$ then there is an integral valuation as a witness to this non-simulation. The proof of this proposition makes use of a non-trivial observation on zones. We refer the reader to [12] for more details.

▶ **Proposition 20.** *Let Z be a topologically closed zone s.t. the $\multimap\!\!-\!\!\multimap^*$ equivalence classes of every valuation in Z are the same. Let LU be a symmetric bounds function. Let Z' be a zone with $Z \not\preccurlyeq^d_{LU} Z'$. Then, there exists an integral valuation $u \in Z$ such that $[u]_{LU} \cap Z'$ is empty.*

We now have the necessary ingredients to give the proof of NP-hardness. Consider the decision problem which takes as inputs two zones Z, Z' and outputs whether $Z \not\preccurlyeq^d_{LU} Z'$. We will give a polynomial time reduction from 3-SAT to this decision problem, showing that it is NP-hard.

Notation. Let Var be a finite set of propositional variables. A *literal* is either a variable p or its negation $\neg p$, and a 3-*clause* is a disjunction of three literals $(l_1 \vee l_2 \vee l_3)$. A 3-CNF formula is a conjunction of 3-clauses. For a literal l, we write $\mathrm{Var}(l)$ for the variable corresponding to l. For a 3-CNF formula ϕ, we write $\mathrm{Var}(\phi)$ for the variables present in ϕ. An *assignment* to a 3-CNF formula ϕ is a function from $\mathrm{Var}(\phi)$ to $\{\mathrm{true}, \mathrm{false}\}$. For a clause C and an assignment

σ, we write $\sigma \models C$ if substituting $\sigma(p)$ for each variable p occurring in C evaluates the clause to true. For a formula ϕ and an assignment σ, we write $\sigma \models \phi$ if all clauses of ϕ evaluate to true under σ. A formula ϕ is said to be *satisfiable* if there exists an assignment such that $\sigma \models \phi$. For the rest of the section, fix a 3-CNF formula $\varphi := C_1 \wedge C_2 \wedge \cdots \wedge C_N$. Let Clauses($\varphi$) be the set $\{C_i \mid i \in \{1, \ldots, N\}\}$.

We start with the idea for the reduction. We know that φ is satisfiable iff there *exists* an assignment σ such that *for all* $C \in$ Clauses(φ) : $\sigma \models C$. Correspondingly, we know that $Z \not\preccurlyeq^d_{LU} Z'$ iff there *exists* a $v \in Z$ such that *for all* $v' \simeq_{LU} v : v' \notin Z'$. Given φ, we want to construct two topologically closed zones Z, Z' such that φ is satisfiable iff $Z \not\preccurlyeq^d_{LU} Z'$. We want the (potential) $v \in Z$ for which every $v' \simeq_{LU} v$ satisfies $v' \notin Z'$ to encode the (potential) satisfying assignment for φ. In essence: valuations in Z should encode assignments, the equivalent valuations v' should encode clauses and the fact that $v' \notin Z'$ should correspond to the chosen clause being true. We now proceed with the details of the construction. Figure 2 illustrates the construction on an example. For each literal l^j_i of φ, we add three clocks x^j_i, y^j_i, z^j_i. There are $N+1$ additional clocks $r_0, r_1, \ldots, r_N$, where r_0 is assumed to be the special 0 clock. We will assume that $L(x-y) = -M$, $U(x-y) = M$, $L(0-x) = -M$, $U(x-0) = M$ for all non-zero clocks x, y and an arbitrary constant $M > 3$. This gives a symmetric LU bounds function.

Construction of Z. Zone Z is described by three sets of constraints. The first set of constraints are between clocks of each literal. For every $i \in \{1, \ldots, N\}$ and $j \in \{1, 2, 3\}$:

$$y^j_i - x^j_i \geq 1 \quad \text{and} \quad z^j_i - y^j_i \geq 1 \quad \text{and} \quad z^j_i - x^j_i = 3 \tag{11}$$

The second set of constraints relates the distance between clocks of different literals. In addition, we use the r_i clocks as separators between clauses. For $i \in \{1, \ldots, N\}$:

$$x^1_i - r_{i-1} = 2M - 3 \;\text{ and }\; x^{j+1}_i - z^j_i = 2M - 3 \text{ for } j \in \{1, 2\} \;\text{ and }\; r_i - z^3_i = 2M \tag{12}$$

Constraints (11) and (12) ensure that for every valuation in Z we have the following order of clocks for each $i \in \{1, \ldots, N\}$:

$$r_{i-1} \;<\; x^1_i < y^1_i < z^1_i \;<\; x^2_i < y^2_i < z^2_i \;<\; x^3_i < y^3_i < z^3_i \;<\; r_i \tag{13}$$

In every valuation of Z, we have $x^j_i \multimap y^j_i \multimap z^j_i$ for every literal l^j_i. This is because we have assumed that $M > 3$ and we have restricted the gaps (absolute value of the differences) between x^j_i, y^j_i and y^j_i, z^j_i to be in the interval $[1, 2]$ (c.f. (11)). We do not want any other pair of clocks that are consecutive according to the above ordering to be tight. Hence we choose the rest of the gaps to be strictly more than M (c.f. (12)). Our choice of constraints ensures that each valuation in Z gives a $\multimap^*$ division where $\{x^j_i, y^j_i, z^j_i\}$ forms an equivalence class (let us call it a block), and each $\{r_i\}$ is an equivalence class. Note that for each $v \in Z$, we also have $v(r_i) - v(r_{i-1}) = 8M$ for $i \in \{1, \ldots, N\}$. We will next enforce that literals in φ involving the same variable have the same $y - x$ and $z - y$ values for their corresponding clocks. Without loss of generality, we assume that the three literals corresponding to the same clause have different variables. Therefore this condition is relevant for literals in different clauses, but with the same variable. For every l^j_i and $l^{j'}_{i'}$ such that $\text{Var}(l^j_i) = \text{Var}(l^{j'}_{i'})$ and $i' > i$:

$$y^{j'}_{i'} - y^j_i = (i' - i) \cdot 8M + (j' - j) \cdot 2M \tag{14}$$

Note that from (11) and (12) we can infer that the values of $v(x^{j'}_{i'}) - v(x^j_i)$ and $v(z^{j'}_{i'}) - v(z^j_i)$ are already equal to the right hand side of the above equation, as the x and z clocks are "fixed" and y is "flexible". Constraint (14) then ensures that $v(y^j_i) - v(x^j_i) = v(y^{j'}_{i'}) - v(x^{j'}_{i'})$ and $v(z^j_i) - v(y^j_i) = v(z^{j'}_{i'}) - v(y^{j'}_{i'})$ whenever l^j_i and $l^{j'}_{i'}$ with $i' > i$, have the same variable.

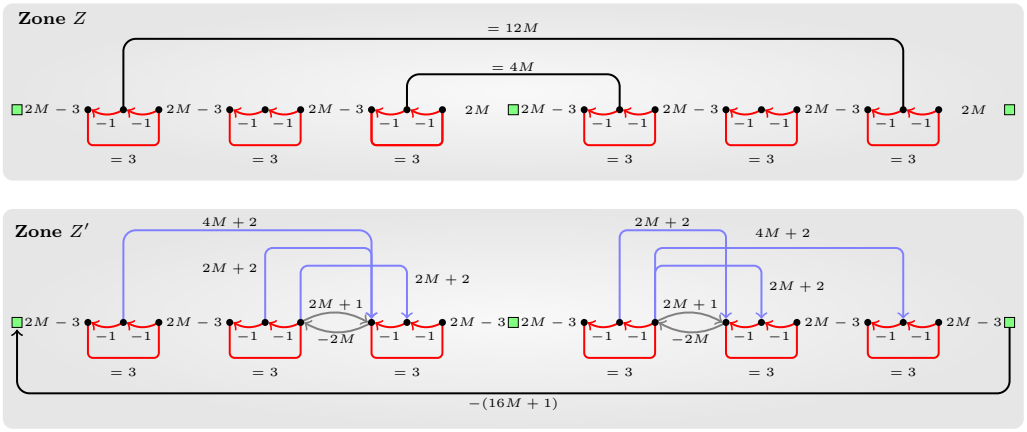


Figure 2 Illustration of the zone Z and Z' for the formula $(p_1 \vee p_2 \vee \neg p_3) \wedge (p_3 \vee \neg p_4 \vee \neg p_1)$. The separator clocks r_0, r_1, r_2 are shown by the green boxes (leftmost box is r_0, middle one is r_1 and the rightmost is r_2). The intermediate literal clocks are shown by the black dots: between r_0 and r_1 are $x_1^1, y_1^1, z_1^1, x_1^2, y_1^2, z_1^2, x_1^3, y_1^3, z_1^3$ in the same sequence. Similarly between r_1 and r_2 are the clocks $x_2^1, \ldots, z_2^3$. An edge of the form $x \xrightarrow{c} y$ simply denotes the constraint $y - x \leq c$, whereas edges $x \xrightarrow{=c} y$ mean that $y - x = c$. When we write c between two consecutive clocks, we mean that the difference between them equals c.

Encoding of assignments. By construction of Z, in every valuation $v \in Z$, we have $v(r_i)$, $v(x_i^j)$ and $v(z_i^j)$ to be fixed integers. The value of $v(y_i^j)$ can vary between $v(x_i^j) + 1$ and $v(x_i^j) + 2$. When this value is in the extremes, either 1 or 2, we get an integral valuation. We encode assignments by such integral valuations. An integral valuation v encodes the assignment σ_v given by: $\sigma_v(\mathrm{Var}(l_i^j)) = \text{true}$ if $v(y_i^j) - v(x_i^j) = 1$ and $\sigma_v(\mathrm{Var}(l_i^j)) = \text{false}$ if $v(y_i^j) - v(x_i^j) = 2$. By (14), the above assignment is well defined. Moreover, the zone Z contains an integral valuation for every possible assignment.

An assignment σ satisfies φ if every clause evaluates to true under σ. From a valuation v encoding this assignment σ, we need a mechanism to check whether each clause is true. This is where we will use the gaps which are not tight (that is the ones $> M$). Clauses will be identified by certain kind of shifts to these unbounded gaps in v. We will introduce some more notation. Let $T := \{(x_i^j, y_i^j, z_i^j) \mid i \in \{1, \ldots, N\} \text{ and } j \in \{1, 2, 3\}\}$ be the triplets of clocks associated with each literal. A literal is said to be *positive* if it is a variable p, and it is *negative* if it is the negation $\neg p$ of some variable p. We will assume that in every clause of φ, the positive literals are written before the negative literals: for example, we write $p_1 \vee p_3 \vee \neg p_2$ instead of $p_1 \vee \neg p_2 \vee p_3$. For each clause C_i, let (e_i, f_i) be the pair of clocks corresponding to C_i in the border between positive and negative literals:

$$(e_i, f_i) := \begin{cases} (r_{i-1}, x_i^1) & \text{if all literals in } C_i \text{ are negative} \\ (z_i^j, x_i^{j+1}) & \text{if for } j \in \{1, 2\},\ l_i^j \text{ is positive and } l_i^{j+1} \text{ is negative} \\ (z_i^3, r_i) & \text{if all literals in } C_i \text{ are positive} \end{cases} \tag{15}$$

Given the formula φ, the above border clocks are fixed. For a valuation $v \in Z$ and $i \in \{1, \ldots, N\}$, define v_i to be the valuation such that:

- $v_i(y) - v_i(x) = v(y) - v(x)$ and $v_i(z) - v_i(y) = v(z) - v(y)$ for all $(x, y, z) \in T$
- $v_i(f_i) - v_i(e_i) = 2M + 1$ and $v_i(f_{i'}) - v_i(e_{i'}) = 2M$ for all $i' \neq i$,
- $v_i(r_0) = 0$ and all other differences between consecutive clocks (according to order given by (13)) is $2M - 3$.

Notice that $v_i \simeq_{LU} v$. Valuation v_i acts as a representative for the clause C_i, through the choice of the difference $2M+1$ in the border of C_i, and $2M$ in the other borders. We want to construct zone Z' such that when C_i is true, the valuation v_i forms a negative cycle with the constraints of Z', via the literal which is true in C_i.

Construction of Z'. Zone Z' is described by five sets of constraints. The first set of constraints are between the clocks of the same literal, and are identical to that in Z:

$$y_i^j - x_i^j \geq 1 \quad \text{and} \quad z_i^j - y_i^j \geq 1 \quad \text{and} \quad z_i^j - x_i^j = 3 \tag{16}$$

The second set of constraints are for border clocks in each clause. For each $i \in \{1, \ldots, N\}$:

$$2M \leq f_i - e_i \leq 2M+1 \tag{17}$$

where e_i and f_i are according to the definition in (15). The third set of constraints fix differences between consecutive blocks not involving border clocks to $2M-3$.

$$\begin{aligned} x_i^1 - r_{i-1} &= 2M-3 \quad \text{if } (r_{i-1}, x_i^1) \neq (e_i, f_i) \text{ and} \\ x_i^{j+1} - z_i^j &= 2M-3 \quad \text{for } j \in \{1,2\} \text{ when } (z_i^j, x_i^{j+1}) \neq (e_i, f_i) \quad \text{and} \\ r_i - z_i^3 &= 2M-3 \quad \text{when } (z_i^3, r_i) \neq (e_i, f_i) \end{aligned} \tag{18}$$

From (16,17,18), we see that for every valuation in Z' the difference between separators, that is $r_i - r_{i-1}$, is between $8M$ and $8M+1$ with the flexibility coming from $f_i - e_i$. The fourth set of constraints ensures that at least one of the $f_i - e_i$ should be bigger than $2M$.

$$r_N - r_0 \geq (8M \cdot N) + 1 \tag{19}$$

So far, the constraints that we have chosen for Z' do not talk about clauses being true or false. Recall that valuation v_i where the border $v_i(f_i) - v_i(e_i) = 2M+1$ represents the choice of C_i for evaluation. The final set of constraints ensure that for every integral valuation v' in Z' which has $v'(f_i) - v'(e_i) = 2M+1$, every literal in C_i evaluates to false under the encoding scheme given in Page 13: that is, if l_i^j is positive then $v'(y_i^j) - v'(x_i^j)$ cannot be 1 and when l_i^j is negative, $v'(y_i^j) - v'(x_i^j)$ cannot be 2. For a positive literal l_i^j let $d_i^j \in \{0,1,2\}$ be the number of (x,y,z) blocks corresponding to positive literals between z_i^j and f_i (does not include j). Similarly, for a negative literal, let $d_i^j \in \{0,1,2\}$ be the number of blocks corresponding to negative literals between e_i and x_i^j (again, excludes j). We add the following constraints:

$$\begin{aligned} f_i - y_i^j &\leq d_i^j \cdot 2M + (2M+2) \quad \text{if } l_i^j \text{ is a positive literal} \\ y_i^j - e_i &\leq d_i^j \cdot 2M + (2M+2) \quad \text{if } l_i^j \text{ is a negative literal} \end{aligned} \tag{20}$$

▶ **Theorem 21.** *Formula φ is satisfiable iff $Z \not\preccurlyeq^d_{LU} Z'$ for the zones Z, Z' and LU bounds function described above.*

Proof sketch. Assume φ is satisfiable. Consider the valuation $v \in Z$ corresponding to the satisfying assignment. Pick an arbitrary $v' \simeq_{LU} v$. If v' were to lie in Z', by (19), at least one of the border differences should be $> 2M$. This forms a contradiction with the literal that is true in clause C_i due to (20).

Assume $Z \not\preccurlyeq^d_{LU} Z'$. As Z and Z' are topologically closed, and the $\multimap^*$ equivalence classes are same for every valuation in Z, by Proposition 20 there is an integral valuation v such that $[v]_{LU} \cap Z'$ is empty. This v gives a satisfying assignment. Mainly, each v_i corresponding to v will form a negative cycle with some literal clocks of C_i, and this literal will be made true by the assignment corresponding to v. ◀

Table 1 Experiments to compare forward analysis with diagonal constraints versus forward analysis on the equivalent diagonal free automaton. "Fischer K" is a model of a communication protocol with K processes as described in [18]. Cex 1 is the automaton in [5] which revealed the bug with the explicit abstraction method. Cex 2 is a similar version with more states, given in [18].

Model		Diagonal constraints + $\preccurlyeq^d_{LU}$		Diagonal free + $\mathfrak{a}_{\preccurlyeq LU}$	
Name	# clocks	# zones	time (in sec.)	# zones	time (in sec.)
Cex 1	4	8	0.08	22	0.02
Cex 2	8	273	30.1	2051	0.1
Fischer 4	8	933	18.3	73677	2.1
Fischer 5	10	4181	132.5	1926991	117.1

Theorem 21 leads to the following result.

▶ **Theorem 22.** *The decision problem $Z \not\preccurlyeq^d_{LU} Z'$ is NP-hard.*

6 Conclusion

In this paper, we have proposed a simulation $\preccurlyeq^d_{LU}$ and a simulation test $Z \preccurlyeq^d_{LU} Z'$ that facilitates a forward analysis procedure for timed automata with diagonal constraints. An abstraction function based on symmetric $\preccurlyeq^d_{LU}$ was already proposed in [5] in the context of forward analysis using explicit abstractions, but it was not used as no efficient storage mechanisms for non-convex abstractions are known. Moreover, no simulation test apart from a brute force check of enumerating over all regions was known either. Here, we provide a more refined simulation test, which in principle gives a more structured way of performing this enumeration. In the diagonal free case, this test can be performed in $O(|X|^2)$ [16]. But, as we show here, in the presence of diagonal constraints, $Z \not\preccurlyeq^d_{LU} Z'$ is NP-complete. Nevertheless, having this forward analysis framework creates the possibility to incorporate recent optimizations studied for diagonal free automata which crucially depend on this inclusion test, and have been indispensable in improving the performance substantially [14, 15]. Moreover, we believe that this framework can be extended to various other problems involving timed automata with diagonal constraints, for instance liveness verification and cost optimal reachability in priced timed automata.

We have implemented reachability for timed automata with diagonal constraints using simulation test $Z \preccurlyeq^d_{LU} Z'$ in a prototype tool T-Checker [13] which has been developed for diagonal free timed automata. The simulation test constructs an SMT formula in linear arithmetic and invokes the Z3 solver [10]. Preliminary experiments on models from [18] are reported in Table 1. For each model $\mathcal{A}$ (with diagonal constraints), the table compares the performance of running the forward analysis approach using $\preccurlyeq^d_{LU}$ on $\mathcal{A}$ (Columns 3 and 4) versus the forward analysis using (diagonal free variant) $\mathfrak{a}_{\preccurlyeq LU}$ [2] on the equivalent diagonal free automaton $\mathcal{A}_{df}$ (Columns 5 and 6). We observe that there is a significant decrease in the number of nodes explored while using $\preccurlyeq^d_{LU}$ on $\mathcal{A}$. The problem with $\mathcal{A}_{df}$ is that each state q of $\mathcal{A}$ has 2^d copies in $\mathcal{A}_{df}$ if d is the number of diagonal constraints (essentially, the states of $\mathcal{A}_{df}$ maintain the information about whether each diagonal is true or false when reaching this state). Therefore a simulation of the form $Z \preccurlyeq^d_{LU} Z'$ arising from (q, Z) and (q, Z') which occurs in the analysis of $\mathcal{A}$ might not be possible while analyzing $\mathcal{A}_{df}$ just because the corresponding paths reach different copies of q, say (q_1, Z) and (q_2, Z'). This prunes

the search faster in $\mathcal{A}$. Indeed, exploiting the conciseness of diagonal constraints could be a valuable tool for modeling and verifying real-time systems. We believe that the performance of our algorithm in terms of time is encouraging: despite the preliminary nature of our implementation, our naïve SMT encoding and the underlying hardness of the simulation test, the time taken is comparable to the diagonal free conversion. Investigating efficient methods for $Z \preccurlyeq^{d}_{LU} Z'$ and comparing our method with other approaches [8] is part of future work.

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 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.

4 Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2,3):145–182, 1998.

5 Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

6 Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

7 Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In *Computer Aided Verification (CAV)*, volume 9779 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2016.

8 Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 3829 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2005.

9 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

10 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

11 David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

12 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. *CoRR*, abs/1806.11007, 2018. `arXiv:1806.11007`.

13 Frédéric Herbreteau. TChecker. `http://www.labri.fr/perso/herbrete/tchecker/index.html`.

14 Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 13 of *LIPIcs*, pages 78–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

15 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 990–1005. Springer, 2013.

16 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016.

17 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

18 Pierre-Alain Reynier. Diagonal constraints handled efficiently in uppaal. In *Research report LSV-07-02*, Laboratoire Spécification et Vérification. ENS Cachan, France, 2007.

19 Sergio Yovine. Kronos: A verification tool for real-time systems. (Kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.

20 Jianhua Zhao, Xuandong Li, and Guoliang Zheng. A quadratic-time dbm-based successor algorithm for checking timed automata. *Information Processing Letters*, 96(3):101–105, 2005.

Parameterized complexity of games with monotonically ordered ω-regular objectives

Véronique Bruyère
Département d'informatique, Université de Mons (UMONS), Mons, Belgium
veronique.bruyere@umons.ac.be

Quentin Hautem[1]
Département d'informatique, Université de Mons (UMONS), Mons, Belgium
quentin.hautem@umons.ac.be

Jean-François Raskin[2]
Département d'informatique, Université Libre de Bruxelles (ULB), Brussels, Belgium
jraskin@ulb.ac.be

Abstract

In recent years, two-player zero-sum games with multiple objectives have received a lot of interest as a model for the synthesis of complex reactive systems. In this framework, Player 1 wins if he can ensure that all objectives are satisfied against any behavior of Player 2. When this is not possible to satisfy all the objectives at once, an alternative is to use some preorder on the objectives according to which subset of objectives Player 1 wants to satisfy. For example, it is often natural to provide more significance to one objective over another, a situation that can be modelled with lexicographically ordered objectives for instance. Inspired by recent work on concurrent games with multiple ω-regular objectives by Bouyer et al., we investigate in detail turned-based games with monotonically ordered and ω-regular objectives. We study the threshold problem which asks whether player 1 can ensure a payoff greater than or equal to a given threshold w.r.t. a given monotonic preorder. As the number of objectives is usually much smaller than the size of the game graph, we provide a parametric complexity analysis and we show that our threshold problem is in FPT for all monotonic preorders and all classical types of ω-regular objectives. We also provide polynomial time algorithms for Büchi, coBüchi and explicit Muller objectives for a large subclass of monotonic preorders that includes among others the lexicographic preorder. In the particular case of lexicographic preorder, we also study the complexity of computing the values and the memory requirements of optimal strategies.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Theory of computation → Algorithmic game theory

Keywords and phrases two-player zero-sum games played on graphs, ω-regular objectives, ordered objectives, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.29

Related Version A full version of the paper is available at `https://arxiv.org/abs/1707.05968`.

Funding The three authors are supported by COST Action GAMENET CA 16228. Véronique

[1] Quentin Hautem is supported by a FRIA fellowship.

[2] Jean-François Raskin is supported by the ERC Starting Grant inVEST (279499), by the ARC project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond" (Fédération Wallonie-Bruxelles), and by the EOS project "Verifying Learning Artificial Intelligence Systems" (FNRS-FWO), and he is Professeur Francqui de Recherche funded by the Francqui foundation.

Bruyère and Jean-François Raskin are both supported by the FNRS PDR project "Subgame perfection in graph games" (T.0088.18).

Acknowledgements We would like to thank Antonia Lechner for useful discussions.

1 Introduction

Two-player zero-sum games played on directed graphs form an adequate framework for the *synthesis of reactive systems* facing an uncontrollable environment [21]. To model properties to be enforced by the reactive system within its environment, games with Boolean objectives and games with quantitative objectives have been studied, for example games with ω-regular objectives [15] and mean-payoff games [23].

Recently, games with *multiple* objectives have received a lot of attention since in practice, a system must usually satisfy several properties. In this framework, the system wins if it can ensure that *all* objectives are satisfied no matter how the environment behaves. For instance, generalized parity games are studied in [11], multi-mean-payoff games in [22], and multidimensional games with heterogeneous ω-regular objectives in [7].

When multiple objectives are conflicting or if there does not exist a strategy that can enforce all of them at the same time, it is natural to consider trade-offs. A general framework for defining trade-offs between n (Boolean) objectives $\Omega_1, \ldots, \Omega_n$ consists in assigning to each infinite path π of the game a payoff $v \in \{0,1\}^n$ such that $v(i) = 1$ iff π satisfies Ω_i, and then to equip $\{0,1\}^n$ with a preorder $\preceq$ to define a preference between pairs of payoffs: $v \preceq v'$ whenever payoff v' is preferred to payoff v. Because the ideal situation would be to satisfy *all* the objectives together, it is natural to assume that the preorder $\preceq$ has the following *monotonicity* property: if v' is such that whenever $v(i) = 1$ then $v'(i) = 1$, then it should be the case that v' is preferred to v.

As an illustration, let us consider a game in which Player 1 strives to enforce three objectives: Ω_1, Ω_2, and Ω_3. Assume also that Player 1 has no strategy ensuring all three objectives at the same time, that is, Player 1 cannot ensure the objective $\Omega_1 \cap \Omega_2 \cap \Omega_3$. Then several options can be considered, see e.g. [6]. First, we could be interested in a strategy of Player 1 ensuring a maximal subset of the three objectives. Indeed, a strategy that enforces both Ω_1 and Ω_3 should be preferred to a strategy that enforces Ω_3 only. This preference is usually called the *subset preorder*. Now, if Ω_1 is considered more important than Ω_2 itself considered more important than Ω_3, then a strategy that ensures the most important possible objective should be considered as the most desirable. This preference is called the *maximize preorder*. Finally, we could also translate the relative importance of the different objectives into a *lexicographic preorder* on the payoffs: satisfying Ω_1 and Ω_2 would be considered as more desirable than satisfying Ω_1 and Ω_3 but not Ω_2. Those three examples are all monotonic preorders.

In this paper, we consider the following threshold problem: given a game graph G, a set of *ω-regular objectives*[3] $\Omega_1, \ldots, \Omega_n$, a monotonic preorder $\preceq$ on the set $\{0,1\}^n$ of payoffs, and a threshold μ, decide whether Player 1 has a strategy such that for all strategies of Player 2, the outcome of the game has payoff v greater than or equal to μ (for the specified preorder), i.e. $\mu \preceq v$. As the number n of objectives is typically much smaller than the size of the game graph G, it is natural to consider a parametric analysis of the complexity of the threshold problem in which the number of objectives and their size are considered to be fixed parameters of the problem. Our main results are as follows.

[3] We cover all classical ω-regular objectives: reachability, safety, Büchi, co-Büchi, parity, Rabin, Streett, explicit Muller, or Muller.

Contributions

First, we provide *fixed parameter tractable solutions* to the threshold problem for *all* monotonic preorders and for *all* classical types of ω-regular objectives. Our solutions rely on the following ingredients:

1. We show that solving the threshold problem is equivalent to *solve a game with a single objective* Ω that is a union of intersections of objectives taken among $\Omega_1, \ldots, \Omega_n$ (Theorem 3). This is possible by *embedding* the monotonic preorder $\precsim$ in the subset preorder and by translating the threshold μ in preorder $\precsim$ into an antichain of thresholds in the subset preorder. A threshold in the subset preorder is naturally associated with a conjunction of objectives, and an antichain of thresholds leads to a union of such conjunctions.
2. We provide a fixed parameter tractable algorithm to solve games with a single objective Ω as described previously for all types of ω-regular objectives $\Omega_1, \ldots, \Omega_n$, leading to a *fixed parameter algorithm for the threshold problem* (Theorem 4). Those results build on the recent breakthrough of Calude et al. that provides a quasipolynomial time algorithm for parity games as well as their fixed parameter tractability [9], and on the fixed parameter tractability of games with an objective defined by a *Boolean combination of Büchi objectives* (Proposition 5).

Second, we consider games with a preorder $\precsim$ having a *compact embedding*, with the main condition that the antichain of thresholds resulting from the embedding in the subset preorder is of *polynomial size*. The maximize preorder, the subset preorder, and the lexicographic preorder, given as examples above, all possess this property. For games with a compact embedding, we go *beyond fixed parameter tractability* as we are able to provide deterministic polynomial time solutions for Büchi, coBüchi, and explicit Muller objectives (Theorem 6). Polynomial time solutions are not possible for the other types of ω-regular objectives as we show that the threshold problem for the *lexicographic preorder* with reachability, safety, parity, Rabin, Streett, and Muller objectives cannot be solved in polynomial time unless $\mathsf{P} = \mathsf{PSPACE}$ (Theorem 7). Finally, we present a *full picture* of the study of the lexicographic preorder for each studied objective. We give the exact complexity class of the threshold problem, show that we can obtain the values from the threshold problem (which thus yields a polynomial algorithm for Büchi, co-Büchi and Explicit Muller objectives, and an FPT algorithm for the other objectives) and provide tight memory requirements for the optimal and winning strategies (Table 2).

Related Work

In [6], Bouyer et al. investigate concurrent games with multiple objectives leading to payoffs in $\{0,1\}^n$ which are ordered using Boolean circuits. While their threshold problem is slightly more general than ours, their games being concurrent and their preorders being not necessarily monotonic, the algorithms that they provide are nondeterministic and guess witnesses whose size depends polynomially not only in the number of objectives but also in the size of the game graph. Their algorithms are sufficient to establish membership to PSPACE for all classical types of ω-regular objectives but they do not provide a basis for the parametric complexity analysis of the threshold problem. In stark contrast, we provide deterministic algorithms whose complexity only depends polynomially in the size of the game graph. Our new deterministic algorithms are thus instrumental to a finer complexity analysis that leads to fixed parameter tractability for all monotonic preorders and all ω-regular objectives. We also provide tighter lower-bounds for the important special case of lexicographic preorder, in particular for parity objectives.

The particular class of games with multiple Büchi objectives ordered with the maximize preorder has been considered in [2]. The interested reader will find in that paper clear practical motivations for considering multiple objectives and ordering them. The lexicographic ordering of objectives has also been considered in the context of quantitative games: lexicographic mean-payoff games in [5], some special cases of lexicographic quantitative games in [8, 16], and lexicographically ordered energy objectives in [12].

In [1] and [19], the authors investigate partially (or totally) ordered specifications expressed in LTL. None of their complexity results leads to the results of this paper since the complexity is de facto much higher with objectives expressed in LTL. Moreover no FPT result is provided in those references.

Structure of the paper

In Section 2, we present all the useful notions about games with monotonically ordered ω-regular objectives. In Section 3, we show that solving the threshold problem is equivalent to solve a game with a single objective that is a union of intersections of objectives (Theorem 3), and we establish the main result of this paper: the fixed parameter complexity of the threshold problem (Theorem 4). Section 4 is devoted to games with a compact embedding and in particular to the threshold problem for lexicographic games. The last section is dedicated to the study of computing the values and memory requirements of optimal strategies in the case of lexicographic games (Table 2). Full paper is available on arXiv.[4]

2 Monotonically ordered ω-regular games

We consider zero-sum turn-based games played by two players, $\mathcal{P}_1$ and $\mathcal{P}_2$, on a finite directed graph. Given *several objectives*, we associate with each play of this game a vector of bits called *payoff*, the components of which indicate the objectives that are satisfied. The set of all payoffs being equipped with a *preorder*, $\mathcal{P}_1$ wants to ensure a payoff greater than or equal to a given threshold against any behavior of $\mathcal{P}_2$. In this section we give all the useful notions and the studied problem.

Preorders

Given some non-empty set P, a *preorder* over P is a binary relation $\preccurlyeq \subseteq P \times P$ that is reflexive and transitive. The *equivalence relation* $\sim$ associated with $\preccurlyeq$ is defined such that $x \sim y$ if and only if $x \preccurlyeq y$ and $y \preccurlyeq x$. The *strict partial order* $\prec$ associated with $\preccurlyeq$ is then defined such that $x \prec y$ if and only if $x \preccurlyeq y$ and $x \not\sim y$. A preorder $\preccurlyeq$ is *total* if $x \preccurlyeq y$ or $y \preccurlyeq x$ for all $x, y \in P$. A set $S \subseteq P$ is *upper-closed* if for all $x \in S$, $y \in P$, if $x \preccurlyeq y$, then $y \in S$. An *antichain* is a set $S \subseteq P$ of pairwise incomparable elements, that is, for all $x, y \in S$, if $x \neq y$, then $x \not\preccurlyeq y$ and $y \not\preccurlyeq x$.

Game structures and strategies

A *game structure* is a tuple $G = (V_1, V_2, E)$ where (V, E) is a finite directed graph, with $V = V_1 \cup V_2$ the set of vertices and $E \subseteq V \times V$ the set of edges such that for each $v \in V$, there exists $(v, v') \in E$ for some $v' \in V$ (no deadlock), and (V_1, V_2) forms a partition of V such that V_i is the set of vertices controlled by player $\mathcal{P}_i$ with $i \in \{1, 2\}$.

[4] See `https://arxiv.org/abs/1707.05968`.

A *play* of G is an infinite sequence of vertices $\pi = v_0 v_1 \ldots \in V^\omega$ such that $(v_k, v_{k+1}) \in E$ for all $k \in \mathbb{N}$. We denote by $\mathsf{Plays}(G)$ the set of plays in G. *Histories* of G are finite sequences $\rho = v_0 \ldots v_k \in V^+$ defined in the same way. Given a play $\pi = v_0 v_1 \ldots$, the set $\mathsf{Occ}(\pi)$ denotes the set of vertices that occur in π, and the set $\mathsf{Inf}(\pi)$ denotes the set of vertices visited infinitely often along π, i.e., $\mathsf{Occ}(\pi) = \{v \in V \mid \exists k \geq 0, v_k = v\}$ and $\mathsf{Inf}(\pi) = \{v \in V \mid \forall k \geq 0, \exists l \geq k,\ v_l = v\}$. Given a set $\Omega \subseteq V^\omega$, we denote by $\overline{\Omega}$ the set $V^\omega \setminus \Omega$.

A *strategy* σ_i for $\mathcal{P}_i$ is a function $\sigma_i \colon V^* V_i \to V$ assigning to each history $\rho v \in V^* V_i$ a vertex $v' = \sigma_i(\rho v)$ such that $(v, v') \in E$. It is *memoryless* if $\sigma_i(\rho v) = \sigma_i(\rho' v)$ for all histories $\rho v, \rho' v$ ending with the same vertex v, that is, if σ_i is a function $\sigma_i \colon V_i \to V$. It is *finite-memory* if $\sigma_i(\rho v)$ only needs finite memory of the history ρv (recorded by a Moore machine). The *size* of σ_i is the size of its Moore machine.

The set of all strategies of $\mathcal{P}_i$ is denoted by Σ_i. Given a strategy σ_i of $\mathcal{P}_i$, a play $\pi = v_0 v_1 \ldots$ of G is *consistent* with σ_i if $v_{k+1} = \sigma_i(v_0 \ldots v_k)$ for all $k \in \mathbb{N}$ such that $v_k \in V_i$. Given an *initial vertex* v_0, and a strategy σ_i of each player $\mathcal{P}_i$, we have a unique play consistent with both strategies σ_1, σ_2, called *outcome* and denoted by $\mathsf{Out}(v_0, \sigma_1, \sigma_2)$.

Single objectives and ordered objectives

An *objective for* $\mathcal{P}_1$ is a set of plays $\Omega \subseteq \mathsf{Plays}(G)$. A *game* (G, Ω) is composed of a game structure G and an objective Ω. A play π is *winning* for $\mathcal{P}_1$ if $\pi \in \Omega$, and losing otherwise. As the studied games are zero-sum, $\mathcal{P}_2$ has the opposite objective $\overline{\Omega}$, meaning that a play π is winning for $\mathcal{P}_1$ if and only if it is losing for $\mathcal{P}_2$. Given a game (G, Ω) and an initial vertex v_0, a strategy σ_1 for $\mathcal{P}_1$ is *winning from* v_0 if $\mathsf{Out}(v_0, \sigma_1, \sigma_2) \in \Omega$ for all strategies σ_2 of $\mathcal{P}_2$. Vertex v_0 is thus called *winning* for $\mathcal{P}_1$. We also say that $\mathcal{P}_1$ is winning from v_0 or that he can *ensure* Ω *from* v_0. Similarly the winning vertices of $\mathcal{P}_2$ are those from which $\mathcal{P}_2$ can ensure his objective $\overline{\Omega}$.

A game (G, Ω) is *determined* if each of its vertices is either winning for $\mathcal{P}_1$ or winning for $\mathcal{P}_2$. Martin's theorem [20] states that all games with Borel objectives are determined. The problem of *solving a game* (G, Ω) means to decide, given an initial vertex v_0, whether $\mathcal{P}_1$ is winning from v_0 (or dually whether $\mathcal{P}_2$ is winning from v_0 when the game is determined).

Instead of a *single* objective Ω, one can consider *several* objectives $\Omega_1, \ldots, \Omega_n$ that are *ordered* with respect to a preorder $\preccurlyeq$ over $\{0,1\}^n$ in the following way. We first define the payoff of a play as a vector[5] of bits the components of which indicate the objectives that are satisfied. Formally, given n objectives $\Omega_1, \ldots, \Omega_n \subseteq \mathsf{Plays}(G)$, the *payoff* function $\mathsf{Payoff} \colon \mathsf{Plays}(G) \to \{0,1\}^n$ assigns a vector of bits to each play $\pi \in \mathsf{Plays}(G)$, where for all $k \in \{1, \ldots, n\}$, $\mathsf{Payoff}_k(\pi) = 1$ if $\pi \in \Omega_k$ and 0 otherwise.

Given the preorder $\preccurlyeq$ over $\{0,1\}^n$, $\mathcal{P}_1$ prefers a play π to a play π' whenever $\mathsf{Payoff}(\pi') \preccurlyeq \mathsf{Payoff}(\pi)$. We call *ordered game* the tuple $(G, \Omega_1, \ldots, \Omega_n, \preccurlyeq)$, the payoff function of which is defined w.r.t. the objectives $\Omega_1, \ldots, \Omega_n$ and its values are ordered with $\preccurlyeq$. In this context, we are interested in the following problem.

▶ **Problem 1.** The *threshold problem* for ordered games $(G, \Omega_1, \ldots, \Omega_n, \preccurlyeq)$ asks, given a threshold $\mu \in \{0,1\}^n$ and an initial vertex $v_0 \in V$, to decide whether $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) has a strategy to ensure the objective $\Omega = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Payoff}(\pi) \succcurlyeq \mu\}$ from v_0 (resp. $\overline{\Omega} = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Payoff}(\pi) \not\succcurlyeq \mu\}$).[6]

[5] Note that in the sequel, we often manipulate equivalently vectors in $\{0,1\}^n$ and sequences of n bits.

[6] Note that when $n = 1$ and $\preccurlyeq$ is the usual order $\leq$ over $\{0,1\}$, we recover the notion of single objective with the threshold $\mu = 1$.

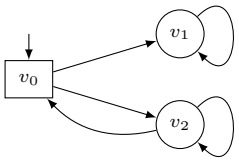


Figure 1 A simple lexicographic game.

In case $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) has such a winning strategy, we also say that he can *ensure* (resp. *avoid*) *a payoff* $\succsim \mu$.

Classical examples of preorders are the following ones [6]. Let $x, y \in \{0,1\}^n$.

- *Counting*: $x \precsim y$ if and only if $|\{j \mid x_j = 1\}| \leq |\{j \mid y_j = 1\}|$. The aim of $\mathcal{P}_1$ is to maximize the number of satisfied objectives.
- *Subset*: $x \precsim y$ if and only if $\{j \mid x_j = 1\} \subseteq \{j \mid y_j = 1\}$. The aim of $\mathcal{P}_1$ is to maximize the subset of satisfied objectives with respect to the inclusion.
- *Maximise*: $x \precsim y$ if and only if $\max\{j \mid x_j = 1\} \leq \max\{j \mid y_j = 1\}$. The aim of $\mathcal{P}_1$ is to maximize the higher index of the satisfied objectives.
- *Lexicographic*: $x \precsim y$ if and only if either $x = y$ or $\exists j \in \{1, \dots, n\}$ such that $x_j < y_j$ and $\forall k \in \{1, \dots, j-1\}$, $x_k = y_k$. The objectives are ranked according to their importance. The aim of $\mathcal{P}_1$ is to maximise the payoff with respect to the induced lexicographic order.

In this article, we *focus on monotonic preorders*. A preorder $\precsim$ is *monotonic* if it is compatible with the subset preorder, i.e. if $\{i \mid x_i = 1\} \subseteq \{i \mid y_i = 1\}$ implies $x \precsim y$. Hence a preorder is monotonic if satisfying more objectives never results in a lower payoff value. This is a *natural property* shared by all the examples of preorders given previously.

▶ **Example 2.** Consider the game structure G depicted on Figure 1, where circle vertices belong to $\mathcal{P}_1$ and square vertices belong to $\mathcal{P}_2$. We consider the ordered game $(G, \Omega_1, \Omega_2, \precsim)$ with $\Omega_i = \{\pi \in \mathsf{Plays}(G) \mid v_i \in \mathsf{Inf}(\pi)\}$ for $i = 1, 2$ and the lexicographic preorder $\precsim$. Therefore the function Payoff assigns value 1 to each play π on the first (resp. second) bit if and only if π visits infinitely often vertex v_1 (resp. v_2). In this ordered game, $\mathcal{P}_1$ has a strategy to ensure a payoff $\succsim 01$ from v_0. Indeed, consider the memoryless strategy σ_1 that loops in v_1 and in v_2. Then, from v_0, $\mathcal{P}_2$ decides to go either to v_1 leading to the payoff 10, or to v_2 leading to the payoff 01. As $10 \succsim 01$, this shows that any play π consistent with σ_1 satisfies $\mathsf{Payoff}(\pi) \succsim 01$. Notice that while $\mathcal{P}_1$ can ensure a payoff $\succsim 01$ from v_0, he has no strategy to enforce the single objective Ω_1 and similarly no strategy to enforce Ω_2.

Homogeneous ω-regular objectives

In this article, given a monotonically ordered game $(G, \Omega_1, \dots, \Omega_n, \precsim)$, we want to study the threshold problem described in Problem 1 for *homogeneous ω-regular objectives*, in the sense that all the objectives $\Omega_1, \dots, \Omega_n$ are of the same type, and taken in the following list of well-known ω-regular objectives.

Given a game structure $G = (V_1, V_2, E)$ and a subset U of V called *target set*:

- The *reachability objective* asks to visit a vertex of U at least once, i.e. $\mathsf{Reach}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Occ}(\pi) \cap U \neq \emptyset\}$.
- The *safety objective* asks to always stay in the set U, i.e. $\mathsf{Safe}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Occ}(\pi) \subseteq U\}$.

- The *Büchi objective* asks to visit infinitely often a vertex of U, i.e. $\mathsf{Buchi}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\pi) \cap U \neq \emptyset\}$.
- The *co-Büchi objective* asks to eventually always stay in the set U, i.e. $\mathsf{CoBuchi}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\pi) \subseteq U\}$.

Given a *family* $\mathcal{F} = (F_i)_{i=1}^k$ of sets $F_i \subseteq V$, and a family of *pairs* $((E_i, F_i)_{i=1}^k)$, with $E_i, F_i \subseteq V$:

- The *explicit Muller objective* asks that the set of vertices seen infinitely often is one among the sets of $\mathcal{F}$, i.e. $\mathsf{ExplMuller}(\mathcal{F}) = \{\pi \in \mathsf{Plays}(G) \mid \exists i \in \{1, \dots, k\}, \mathsf{Inf}(\pi) = F_i\}$.
- The *Rabin objective* asks that there exists a pair (E_i, F_i) such that a vertex of F_i is visited infinitely often while no vertex of E_i is visited infinitely often, i.e. $\mathsf{Rabin}((E_i, F_i)_{i=1}^k) = \{\pi \in \mathsf{Plays}(G) \mid \exists i \in \{1, \dots, k\}, \mathsf{Inf}(\pi) \cap E_i = \emptyset \text{ and } \mathsf{Inf}(\pi) \cap F_i \neq \emptyset\}$.
- The *Streett objective* asks that for each pair (E_i, F_i), a vertex of E_i is visited infinitely often or no vertex of F_i is visited infinitely often, i.e. $\mathsf{Streett}((E_i, F_i)_{i=1}^k) = \{\pi \in \mathsf{Plays}(G) \mid \forall i \in \{1, \dots, k\}, \mathsf{Inf}(\pi) \cap E_i \neq \emptyset \text{ or } \mathsf{Inf}(\pi) \cap F_i = \emptyset\}$.

Given a *coloring* function $p\colon V \to \{0, \dots, d\}$ that associates with each vertex a color, and $\mathcal{F} = (F_i)_{i=1}^k$ a family of subsets F_i of $p(V)$:

- The *parity objective* asks that the minimum color seen infinitely often is even, i.e. $\mathsf{Parity}(p) = \{\pi \in \mathsf{Plays}(G) \mid \min_{v \in \mathsf{Inf}(\pi)} p(v) \text{ is even}\}$.
- The *Muller objective* asks that the set of colors seen infinitely often is one among the sets of $\mathcal{F}$, i.e. $\mathsf{Muller}(p, \mathcal{F}) = \{\pi \in \mathsf{Plays}(G) \mid \exists i \in \{1, \dots, k\}, p(\mathsf{Inf}(\pi)) = F_i\}$.

In the sequel, we make the *assumption* that the considered preorders are monotonic, and by *ordered game*, we always mean monotonically ordered games. When the objectives of an ordered game are of kind X, we speak of an *ordered X game*, or of a $\preccurlyeq$ *X game* if we want to specify the used preorder $\preccurlyeq$. As already mentioned, when $n = 1$, an ordered game (with $\preccurlyeq$ equal to $\leq$) resumes to a game (G, Ω) with a single objective Ω, that is traditionally called an Ω game. For instance, an ordered game $(G, \Omega_1, \dots, \Omega_n, \preccurlyeq)$ where $\Omega_1, \dots, \Omega_n$ are reachability objectives and $\preccurlyeq$ is the lexicographic preorder is called a lexicographic reachability game, and when $n = 1$ (G, Ω_1) is called a reachability game.

Note that given an ordered game with n non-homogeneous ω-regular objectives Ω_i, we can always construct a new equivalent ordered parity game, since each objective Ω_i can be translated into a parity objective [15].

Monotonic preorders embedded in the subset preorder

We here show that solving the threshold problem for an ordered game $(G, \Omega_1, \dots, \Omega_n, \preccurlyeq)$ is equivalent to solving a game (G, Ω) with a single objective Ω equal to the union of intersections of objectives taken in $\{\Omega_1, \dots, \Omega_n\}$. The arguments are the following ones. (1) We consider the set $\{0,1\}^n$ of payoffs ordered with $\preccurlyeq$ as well as ordered with the subset preorder $\subseteq$ (see the example of Figure 2 where $\preccurlyeq$ is the lexicographic preorder). To any payoff $\nu \in \{0,1\}^n$, we associate the set $\delta_\nu = \{i \in \{1, \dots, n\} \mid \nu_i = 1\}$ containing all indices i such that objective Ω_i is satisfied. (2) Consider the set of payoffs $\nu \succcurlyeq \mu$ embedded in the set $\{0,1\}^n$ ordered with $\subseteq$. By monotonicity of $\preccurlyeq$, we obtain an upper-closed set S that can be represented by the antichain of its *minimal elements* (with respect to $\subseteq$), that we denote by $\mathsf{M}(\mu)$. (3) $\mathcal{P}_1$ can ensure a payoff $\succcurlyeq \mu$ if and only if he has a strategy such that any consistent outcome π has a payoff $\nu^* \supseteq \nu$ for some $\nu \in \mathsf{M}(\mu)$, equivalently such that π satisfies (at least) the conjunction of the objectives Ω_i such that $\nu_i = 1$. (4) The objective Ω of $\mathcal{P}_1$ is thus a disjunction (over $\nu \in \mathsf{M}(\mu)$) of conjunctions (over $i \in \delta_\nu$) of objectives Ω_i. This statement is formulated in the next theorem (see again Figure 2).

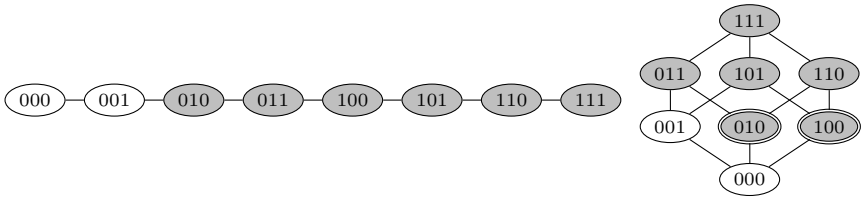


Figure 2 Gray nodes represent the set of payoffs $\nu \succsim \mu = 010$ for the lexicographic preorder and its embedding for the subset preorder. The elements of $\mathsf{M}(\mu) = \{010, 100\}$ are doubly circled.

▶ **Theorem 3.** *Let $(G, \Omega_1, \dots, \Omega_n, \precsim)$ be an ordered game, $\mu \in \{0,1\}^n$ be some threshold, and v_0 be an initial vertex. Then, $\mathcal{P}_1$ can ensure a payoff $\succsim \mu$ from v_0 in $(G, \Omega_1, \dots, \Omega_n, \precsim)$ if and only if $\mathcal{P}_1$ has a winning strategy from v_0 in the game (G, Ω) with the objective $\Omega = \cup_{\nu \in \mathsf{M}(\mu)} \cap_{i \in \delta_\nu} \Omega_i$.*

We end this section by giving some additional notations and terminology. Thanks to Theorem 3, we will prove in Section 3 that the threshold problem is fixed parameter tractable. The proof of this result uses two sizes depending on the number n of objectives:

- the *size* $s(n)$ of $\mathsf{M}(\mu)$. It is upper bounded by 2^n (an antichain of maximum size in the subset preorder over $\{0,1\}^n$ is of exponential size $\binom{n}{\lfloor \frac{n}{2} \rfloor}$).
- the *size* $s'(n)$ defined as follows. In case of Büchi objectives Ω_i, we need to rewrite the objective $\cup_{\nu \in \mathsf{M}(\mu)} \cap_{i \in \delta_\nu} \Omega_i$ in conjunctive normal form $\cap_k \cup_l \Omega'_{k,l}$ with $\Omega'_{k,l} \in \{\Omega_1, \dots, \Omega_n\}$. We denote by $s'(n)$ the size of this conjunction. It is bounded by 2^{2^n}.

In Section 4 we will show that, for several objectives, we can go beyond fixed parameter tractability by providing polynomial time algorithms when the sizes $s(n)$ and $s'(n)$ are polynomial in n. An ordered game $(G, \Omega_1, \dots, \Omega_n, \precsim)$ is said to have a *compact embedding* (in the subset preorder) if both sizes $s(n)$ and $s'(n)$ are polynomial in n. We will also show that lexicographic games have a compact embedding.

3 Fixed parameter complexity of ordered ω-regular games

Parameterized complexity

A *parameterized language* L is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet, the second component being the parameter of the language. It is called *fixed parameter tractable* (FPT) if there is an algorithm that determines whether $(x,t) \in L$ in $f(t) \cdot |x|^c$ time, where c is a constant independent of the parameter t and f is a computable function depending on t only. We also say that L belongs to (the class) FPT. Intuitively, a language is FPT if there is an algorithm running in polynomial time w.r.t the input size times some computable function on the parameter. In this framework, we do not rely on classical polynomial reductions but rather use so called FPT-reductions. An FPT-*reduction* between two parameterized languages $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Sigma'^* \times \mathbb{N}$ is a function $R : L \to L'$ such that

- $(x,t) \in L$ if and only if $(x',t') = R(x,t) \in L'$,
- R is computable by an algorithm that takes $f(t) \cdot |x|^c$ time where c is a constant, and
- $t' \leq g(t)$ for some computable function g.

Moreover, if L' is in FPT, then L is also in FPT. We refer the interested reader to [13] for more details on parameterized complexity.

Our main result states that the threshold problem is in FPT for all the ordered games of this article. Parameterized complexities are given in Table 1.

Table 1 Fixed parameter tractability of ordered games $(G, \Omega_1, \dots, \Omega_n, \preccurlyeq)$: for $i \in \{1, \dots, n\}$, k_i/d_i denotes the number of pairs/colors of each Rabin/Streett/Muller objective Ω_i. Sizes $s(n)$ and $s'(n)$ are resp. upper bounded by 2^n and 2^{2^n}. For $j \in \{1, 2, 3\}$, $M_j = 2^{m_j}$ where $m_1 = \sum_{i=1}^{n} 2 \cdot k_i$, $m_2 = m_3 = \sum_{i=1}^{n} d_i$, and $N_1 = s(n) \cdot \sum_{i=1}^{n} 2 \cdot k_i$, $N_2 = s(n) \cdot \sum_{i=1}^{n} \frac{d_i^2}{2}$, $N_3 = s(n) \cdot \sum_{i=1}^{n} 2^{d_i} \cdot d_i$.

Objectives	Parameters	Threshold problem
Reachability, Safety	n	$O(s(n) \cdot n + 2^n \cdot (\lvert V \rvert + \lvert E \rvert))$
Büchi	n	$O(s(n) \cdot n + s'(n) \cdot \lvert V \rvert^2)$
co-Büchi	n	$O(s(n) \cdot n + s(n) \cdot \lvert V \rvert^2)$
Explicit Muller	n	$O(s(n) \cdot n + (s(n) \cdot \max_i \lvert \mathcal{F}_i \rvert)^3 \cdot \lvert V \rvert^2 \cdot \lvert E \rvert^2)$
Rabin, Streett	$n, k_1, \dots, k_n$	$O((2^{M_1} \cdot N_1 + M_1^{M_1}) \cdot \lvert V \rvert^5)$
Parity	$n, d_1, \dots, d_n$	$O((2^{M_2} \cdot N_2 + M_2^{M_2}) \cdot \lvert V \rvert^5)$
Muller	$n, d_1, \dots, d_n$	$O((2^{M_3} \cdot N_3 + M_3^{M_3}) \cdot \lvert V \rvert^5)$

▶ **Theorem 4.** *The threshold problem is in* FPT *for ordered reachability, safety, Büchi, co-Büchi, explicit Muller, Rabin, Streett, parity, and Muller games.*

The proof of this theorem needs to introduce additional kinds of games (G, Ω) with a single ω-regular objective Ω, like the Boolean Büchi games. It also needs to show that solving the latter games is in FPT.

Parameterized complexity of Boolean Büchi games

Let G be a game structure and $U_1, \dots, U_m$ be m target sets. Let ϕ be a Boolean formula over variables $x_1, \dots, x_m$. We say that a play π satisfies $(\phi, U_1, \dots, U_m)$ if the truth assignment $\{x_i = 1$ if and only if $\mathsf{Inf}(\pi) \cap U_i \neq \emptyset$, and $x_i = 0$ otherwise$\}$ satisfies ϕ. An objective Ω is a *Boolean combination of Büchi objectives*, or shortly a *Boolean Büchi* objective, if $\Omega = \{\pi \in \mathsf{Plays}(G) \mid \pi \text{ satisfies } (\phi, U_1, \dots, U_m)\}$. It is denoted by $\mathsf{BooleanBuchi}(\phi, U_1, \dots, U_m)$.

All operators $\vee$, $\wedge$, $\neg$ are allowed in Boolean Büchi objectives. However we denote by $|\phi|$ the *size* of ϕ equal to the number of disjunctions and conjunctions inside ϕ, and we say that the Boolean Büchi objective $\mathsf{BooleanBuchi}(\phi, U_1, \dots, U_m)$ is *of size* $|\phi|$ *and with* m *variables*. The definition of $|\phi|$ is not the classical one that usually counts the number of operators $\vee, \wedge, \neg$ and variables. This is not a restriction since one can transform any Boolean formula ϕ into one such that negations only apply on variables.

We need to introduce some other kinds of ω-regular objectives with Boolean combinations of objectives that are limited to

- intersections of objectives: like a *generalized reachability* objective or a *generalized Büchi* objective denoted respectively by $\mathsf{GenReach}(U_1, \dots, U_m)$ and $\mathsf{GenBuchi}(U_1, \dots, U_m)$,
- unions of intersections (UI) of objectives: like a *UI reachability* objective, a *UI safety* objective, or a *UI Büchi* objective.

▶ **Proposition 5.** *Solving Boolean Büchi games* (G, Ω) *is in* FPT*, with an algorithm in* $O(2^M \cdot |\phi| + (M^M \cdot |V|)^5)$ *time with* $M = 2^m$ *such that* m *is the number of variables of* ϕ *in the Boolean Büchi objective* Ω*.*

Proof. Let us show the existence of an FPT-reduction from Boolean Büchi games to Muller games. For this purpose, consider a Boolean Büchi game (G, Ω) with the objective $\Omega = \mathsf{BooleanBuchi}(\phi, U_1, \dots, U_m)$, where ϕ is a Boolean formula over variables $x_1, \dots, x_m$, and m is seen as a parameter. We build an adequate Muller game $(G, \mathsf{Muller}(p, \mathcal{F}))$ on the same game structure and parameterized by the number of colors. The coloring function p and the family $\mathcal{F}$ are constructed as follows.

To any vertex $v \in V$, we associate a color $p(v) = \mu$ which is a subset of $\{1, \dots, m\}$ in the following way: $i \in \mu$ if and only if $v \in U_i$.[7] Intuitively, we keep track for all i, whether a vertex belongs to U_i or not. The total number M of colors is thus equal to 2^m. One can notice that $(*)$ a play π visits a vertex $v \in U_i$ if and only if π visits a color μ that contains i.

To any subset F of $p(V)$, we associate the truth assignment $\chi(F) \in \{0,1\}^m$ of variables $x_1, \dots, x_m$ such that for all i, $\chi(F)_i = 1$ if there exists $\mu \in F$ such that $i \in \mu$, and 0 otherwise. The idea (by $(*)$) is that the set F of colors visited infinitely often by a play π corresponds to the set $\mathsf{Inf}(\pi)$ of vertices visited infinitely often such that $\chi(F)_i = 1$ if and only if $\mathsf{Inf}(\pi) \cap U_i \neq \emptyset$. We then define $\mathcal{F} = \{F \subseteq p(V) \mid \chi(F) \models \phi\}$, that is, $\mathcal{F}$ corresponds to the set of all truth assignments satisfying ϕ.

In this way we have the desired FPT-reduction: first, parameter $M = 2^m$ only depends on parameter m. Second, we have that $\mathcal{P}_1$ is winning in $(G, \mathsf{BooleanBuchi}(\phi, U_1, \dots, U_m))$ from an initial vertex v_0 if and only if he is winning in $(G, \mathsf{Muller}(p, \mathcal{F}))$ from v_0. Indeed, a play π satisfies $(\phi, U_1, \dots, U_m)$ if and only if the truth assignment ($x_i = 1$ if and only $\mathsf{Inf}(\pi) \cap U_i \neq \emptyset$, and $x_i = 0$ otherwise) satisfies ϕ. This is equivalent to have that $F = p(\mathsf{Inf}(\pi))$ belongs to $\mathcal{F}$ (by definition of $\chi(F)$), that is, π belongs to $\mathsf{Muller}(p, \mathcal{F})$. Third, the construction of the Muller game is in $O(2^{2^m} \cdot |\phi|)$ time since it requires $O(|V| + |E|)$ time for the game structure, $O(m \cdot |V|)$ time for the coloring function p, and $O(2^{2^m} \cdot |\phi|)$ time for the family $\mathcal{F}$.

From this FPT-reduction and as solving Muller games is in $O((d^d \cdot |V|)^5))$ time where d is the number of colors [9], we have an algorithm solving the Boolean Büchi game in $O(2^M \cdot |\phi| + (M^M \cdot |V|)^5)$ time, where $M = 2^m$. ◀

Proof of FPT membership for ordered games

Thanks to Theorem 3, we provide a proof of Theorem 4 with the parameterized complexities given in Table 1.

Proof of Theorem 4. By Theorem 3, solving the threshold problem for an ordered game $(G, \Omega_1, \dots, \Omega_n, \precsim)$ is equivalent to solving a classical game (G, Ω) with $\Omega = \cup_{\nu \in \mathsf{M}(\mu)} \cap_{i \in \delta_\nu} \Omega_i$. We have $|\mathsf{M}(\mu)| = s(n)$ and $|\delta_\nu| \leq n \ \forall \nu \in \mathsf{M}(\mu)$. Recall that $s(n) \leq 2^n$ and $s'(n) \leq 2^{2^n}$.

We first show that the threshold problem for ordered reachability, safety, Büchi, co-Büchi, and explicit Muller games is in FPT with parameter n. The reduction provided in Theorem 3 is an FPT-reduction as the number of disjunctions/conjunctions in Ω only depends on n. Moreover the construction of the game (G, Ω) is in $O(|V| + |E| + s(n) \cdot n)$ time. In the following items we describe a second FPT-reduction to add to the first one. The sum of the complexities of both FPT-reductions leads to the complexities given in Table 1, rows 2-5.

- If each Ω_i is a reachability (resp. safety) objective, then (G, Ω) is a UI reachability (resp. safety) game that can be reduced to a reachability (resp. safety) game over a game structure of size $2^n \cdot |V|$ [14].[8] The latter is solved in $O(2^n \cdot (|V| + |E|))$ time.
- If Ω is a union of intersections of Büchi objectives, then it can be rewritten as the intersection of unions of Büchi objectives which is a generalized Büchi objective with at most $s'(n)$ target sets. The latter game is solved in $O(s'(n) \cdot |V|^2)$ time by [10]. The

[7] Our definition of color requires μ to be an integer. It suffices to associate with v a vector $\mu^v \in \{0,1\}^m$ such that $\mu^v_i = 1$ if $v \in U_i$ and 0 otherwise, and to define the coloring function $p\colon V \to \{0, \dots, 2^m - 1\}$ that associates with each vertex v the color $p(v)$ such that its binary encoding is equal to μ^v.

[8] This result does not appear explicitly in [14] but can be easily adapted to the case of UI reachability (resp. safety) objectives.

union of intersections of co-Büchi objectives is the complementary of a generalized Büchi objective with at most $s(n)$ target sets, leading to an algorithm in $O(s(n) \cdot |V|^2)$ time.

- If each Ω_i is an explicit Muller objective $\mathsf{ExplMuller}(\mathcal{F}_i)$ then Ω is also an explicit Muller objective. Indeed the intersection (resp. union) of explicit Muller objectives is an explicit Muller objective such that $\cap_i \mathsf{ExplMuller}(\mathcal{F}_i) = \mathsf{ExplMuller}(\mathcal{F})$ with $\mathcal{F} = \cap_i \mathcal{F}_i$ (resp. $\cup_i \mathsf{ExplMuller}(\mathcal{F}_i) = \mathsf{ExplMuller}(\mathcal{F})$ with $\mathcal{F} = \cup_i \mathcal{F}_i$). Thus Ω can be here rewritten as $\mathsf{ExplMuller}(\mathcal{F})$ for some set $\mathcal{F}$ such that $|\mathcal{F}| \leq \sum_{\nu \in \mathsf{M}(\mu)} \min_{j \in \delta_\nu} |\mathcal{F}_j|$. The latter game is solved in $O(|\mathcal{F}| \cdot (|V| \cdot |E| + |\mathcal{F}|)^2) = O((s(n) \cdot \max_i |\mathcal{F}_i|)^3 \cdot |V|^2 \cdot |E|^2)$ time by [17].

We now show that the threshold problem for ordered parity, Rabin, Streett, and Muller games is in FPT thanks to Proposition 5.

- Let us show that the threshold problem for ordered parity games is in FPT with parameters $n, d_1, \ldots, d_n$. If each Ω_i is a parity objective with d_i colors, then each Ω_i is a Boolean Büchi objective of size at most $\frac{d_i^2}{2}$ and using d_i variables. Indeed, as a play is winning for Ω_i if and only there exists an even priority seen infinitely often along the play and no lower priority seen infinitely often. Therefore, Ω is a Boolean Büchi objective Ω' of size $|\phi| \leq s(n) \cdot \sum_{i=1}^{n} \frac{d_i^2}{2}$, and with $m = \sum_{i=1}^{n} d_i$ variables as $\cup_{\nu \in \mathsf{M}(\mu)} \{\Omega_i \mid i \in \delta_\nu\} \subseteq \{\Omega_1, \ldots, \Omega_n\}$. We thus have an FPT-reduction to the game (G, Ω') depending on the parameters $n, d_1, \ldots, d_n$ and with an algorithm in $O(|V|+|E|+|\phi|)$ time. By Proposition 5, solving the game (G, Ω') is in FPT with an algorithm in $O(2^M \cdot |\phi| + (M^M \cdot |V|)^5)$ time with $M = 2^m$. Thus the threshold problem is in FPT with parameters $n, d_1, \ldots, d_n$, with an overall algorithm in $O((2^M \cdot N + M^M) \cdot |V|^5)$ time where $N = 2^n \cdot \sum_{i=1}^{n} \frac{d_i^2}{2}$.
- The arguments are similar for ordered Rabin, Streett, and Muller games. The only differences are the upper bound on size $|\phi|$ and the number m of variables of the related formula ϕ. ◀

4 Ordered games with a compact embedding

In the previous section, we have shown that solving the threshold problem for ordered ω-regular games is in FPT. This result depends on sizes $s(n)$ and $s'(n)$ which vary with the number n of objectives. In this section, we study ordered games with a compact embedding, that is, such that these sizes are polynomial in n.

Beyond fixed parameter tractability

While the threshold problem is in FPT for ordered Büchi, co-Büchi, and explicit Muller games, it becomes polynomial as soon as their preorder has a compact embedding. This is a direct consequence of Table 1, rows 2-4.

▶ **Theorem 6.** *The threshold problem is solved in polynomial time for ordered Büchi, co-Büchi, and explicit Muller games with a compact embedding.*

One can easily prove that ordered games using the subset or the maximize preorder have a compact embedding. We will later prove that this also holds for the lexicographic preorder. Nevertheless it is not the case for the counting preorder. Indeed solving the threshold problem for counting Büchi games is co-NP-complete [6].

Recall that solving the threshold problem for ordered Büchi games reduces to solving some UI Büchi game (by Theorem 3). Whereas solving the latter games is coNP-complete [4], solving the threshold problem for ordered Büchi games is only polynomial when they have a compact embedding (see Theorem 6).

There is no hope to extend Theorem 6 to the other ω-regular objectives studied in this article, unless $\mathsf{P} = \mathsf{PSPACE}$. Indeed, we have PSPACE-hardness of the threshold problem for the following lexicographic games.

▶ **Theorem 7.** *(1) Lexicographic games have a compact embedding and (2) the threshold problem is* PSPACE*-hard for lexicographic reachability, safety, Rabin, Streett, parity, and Muller games.*

The rest of this section is devoted to the proof of Theorem 7.

Lexicographic games

We now focus on the lexicographic preorder $\precsim$. Let us first provide several useful terminology and comments on this preorder. Recall that the lexicographic preorder is monotonic. It is also total, hence $x \sim y$ if and only if $x = y$, and $x \prec y$ if and only if $\neg(y \precsim x)$. Given a vector $x \in \{0,1\}^n$, we denote by $\overline{x}$ the *complement* of x, i.e. $\overline{x}_i = 1 - x_i$, for all $i \in \{1, \dots, n\}$. We denote by $x - 1$ the *predecessor* of $x \neq 0^n$, that is, the greatest vector which is strictly smaller than x. We define the *successor* $x + 1$ of x similarly. In the sequel, as the threshold problem is trivial for $x = 0^n$, we do not consider this threshold. By abuse of notation, we keep writing $x \in \{0,1\}^n$ without mentioning that $x \neq 0^n$. We denote by $\mathsf{Last}_1(x)$ the last index i of x such that $x_i = 1$, i.e. $\mathsf{Last}_1(x) = \max\{i \in \{1, \dots, n\} \mid x_i = 1\}$. Note that $\mathcal{P}_1$ can ensure a payoff $\succsim x \neq 0^n$ if and only if he can ensure a payoff $\succ x - 1$, and when $\mathcal{P}_2$ can avoid a payoff $\succsim x$, we rather say that $\mathcal{P}_2$ can *ensure* a payoff $\prec x$.

We now prove that the lexicographic games have a compact embedding (Part (1) of Theorem 7): we first show that $s(n)$ is polynomial in Proposition 8, and we then show that $s'(n)$ is also polynomial in Proposition 10.

▶ **Proposition 8.** *Let* $x \in \{0,1\}^n$*. Then the set* $\mathsf{M}(x)$ *is equal to* $\{x\} \cup \{y^j \in \{0,1\}^n \mid x_j = 0 \wedge j < \mathsf{Last}_1(x)\}$*, where for all* $j \in \{1, \dots, \mathsf{Last}_1(x) - 1\}$*, we define the vector* $y^j \in \{0,1\}^n$ *as equal to* $x_1 \dots x_{j-1} 10^{n-j}$ *(*x *and* y^j *share the same (possibly empty) prefix* $x_1 \dots x_{j-1}$*). Moreover,* $s(n) = |\mathsf{M}(x)| \leq n$*.*

▶ **Example 9.** Consider the vector $x = 0010100$ such that $\mathsf{Last}_1(x) = 5$. Then, the set $\mathsf{M}(x)$ is equal to $\{x\} \cup \{1000000, 0100000, 0011000\}$.

Proof of Proposition 8. We recall that $\mathsf{M}(x)$ is the set of minimal elements (with respect to the subset preorder $\subseteq$) of the set of payoffs $y \succsim x$ embedded in the set $\{0,1\}^n$ ordered with $\subseteq$. Let us show both inclusions between $\mathsf{M}(x)$ and $M = \{x\} \cup \{y^j \in \{0,1\}^n \mid x_j = 0 \wedge j < \mathsf{Last}_1(x)\}$.

Let $y \in \mathsf{M}(x)$. If $y = x$, then trivially $y \in M$. Otherwise, assume $y \succ x$ and let j be the first index such that $y_j = 1$ and $x_j = 0$. Note that $x_1 \dots x_{j-1} = y_1 \dots y_{j-1}$ since $y \succ x$. We associate with y the vector $y^j = y_1 \dots y_{j-1} 10^{n-j}$. Note that $y^j \succ x$. By minimality of y and by construction of y^j, we obtain $y = y^j$ showing that $y \in M$.

For the second inclusion, as the lexicographic preorder is monotonic, we have $x \in \mathsf{M}(x)$. Now, consider some $y^j \in M$ such that $x_j = 0$ and $j < \mathsf{Last}_1(x)$. Let us show that y^j belongs to $\mathsf{M}(x)$, that is, $y^j \succsim x$ and there is no $y \succsim x$, $y \neq y^j$, such that $y \subset y^j$ (i.e. $\{i \mid y_i = 1\} \subset \{i \mid y^j_i = 1\}$). First, we clearly have $y^j \succsim x$ since $y^j = x_1 \dots x_{j-1} 10^{n-j}$ and $x_j = 0$. Towards a contradiction, assume now that there exists some $y \succsim x$, $y \neq y^j$, such that $y \subset y^j$. Let i be the first index such that $y_i = 0$ and $y^j_i = 1$. As $y \subset y^j$, we have $i \leq j$. If $i < j$, then y has $x_1 \dots x_{i-1} 0$ as prefix, $y^j_i = x_i = 1$, showing that $y \prec x$ in contradiction with $y \succsim x$. If $i = j$, then $y = x_1 \dots x_{j-1} 0^{n-j+1}$, and again $y \prec x$ since $j < \mathsf{Last}_1(x)$ by construction of y^j. ◀

▶ **Proposition 10.** *Let $(G, \Omega_1, \dots, \Omega_n, \precsim)$ be a lexicographic Büchi game and $\mu \in \{0,1\}^n$. Then, $\Omega = \cup_{\nu \in \mathsf{M}(\mu)} \cap_{i \in \delta_\nu} \Omega_i$ can be rewritten in conjunctive normal form with a conjunction of size $s'(n) \leq n$.*

Proof. The proof uses the property that given a lexicographic game $(G, \Omega_1, \dots, \Omega_n, \precsim)$ and a threshold $\mu \in \{0,1\}^n$, $\mathcal{P}_1$ can ensure a payoff $\succsim \mu$ in $(G, \Omega_1, \dots, \Omega_n, \precsim)$ if and only if $\mathcal{P}_1$ can ensure a payoff $\precsim \overline{\mu}$ in the lexicographic game $(G, \overline{\Omega_1}, \dots, \overline{\Omega_n}, \precsim)$. By Theorem 3 and Martin's theorem [20], equivalently, $\mathcal{P}_2$ cannot satisfy the objective $\cup_{\nu \in \mathsf{M}(\overline{\mu}+1)} \cap_{i \in \delta_\nu} \overline{\Omega_i}$. This is equivalent to say that $\mathcal{P}_1$ can satisfy the complement of the latter objective, that is, the objective $\cap_{\nu \in \mathsf{M}(\overline{\mu}+1)} \cup_{i \in \delta_\nu} \Omega_i$. We have $|\mathsf{M}(\overline{\mu}+1)| \leq n$ by Proposition 8. ◀

We finally prove Part (2) of Theorem 7.

Proof of Theorem 7, Part (2). Let us study the complexity lower bounds.

- The PSPACE-hardness of the threshold problem for lexicographic reachability (resp. safety) games is obtained thanks to a polynomial reduction from solving generalized reachability games which is PSPACE-complete [14]. Let (G, Ω) be a generalized reachability game with $\Omega = \mathsf{GenReach}(U_1, \dots, U_n)$. Let $(G, \Omega_1, \dots, \Omega_n, \precsim)$ be the lexicographic reachability (resp. safety) game with $\Omega_i = \mathsf{Reach}(U_i)$ (resp. $\Omega_i = \mathsf{Safe}(V \setminus U_i)$) $\forall i$.
  - Reachability: We have that $\mathcal{P}_1$ is winning in (G, Ω) from v_0 if and only if $\mathcal{P}_1$ can ensure a payoff $\succsim \mu = 1^n$ from v_0 in the lexicographic reachability game $(G, \Omega_1, \dots, \Omega_n, \precsim)$.
  - Safety: We claim that $\mathcal{P}_1$ is winning in (G, Ω) from v_0 if and only if $\mathcal{P}_1$ can ensure a payoff $\succsim \mu = 0^{n-1}1$ from v_0 in the lexicographic safety game. This follows from the determinacy of generalized reachability games, and from the fact that $\mathcal{P}_1$ can ensure a payoff $\succsim \mu$ from v_0 in the lexicographic safety game if and only if $\mathcal{P}_2$ is losing in the generalized reachability game (G, Ω) from v_0.
- The hardness of the threshold problem for lexicographic parity games is obtained thanks to a polynomial reduction from solving games (G, Ω) the objective Ω of which is a union of a Rabin objective and a Streett objective, which is known to be PSPACE-complete [3]. Let $\Omega = \mathsf{Rabin}((E_i, F_i)_{i=1}^{n_1}) \cup \mathsf{Streett}((E_i, F_i)_{i=n_1+1}^{n})$. As any Rabin (resp. Streett) objective is the union (resp. intersection) of parity objectives [11], we can rewrite Ω as $\Omega = \cup_{i=1}^{n_1}(\mathsf{Parity}(p_i)) \cup (\cap_{i=n_1+1}^{n}\mathsf{Parity}(p_i))$, where all p_i are coloring functions. Let $(G, \Omega_1, \dots, \Omega_n, \precsim)$ be the lexicographic parity game where $\Omega_i = \mathsf{Parity}(p_i)$ for all i. We claim that $\mathcal{P}_1$ is winning in the game (G, Ω) from v_0 if and only if $\mathcal{P}_1$ can ensure a payoff $\succsim \mu$ from v_0 in the lexicographic parity game $(G, \Omega_1, \dots, \Omega_n, \precsim)$ where $\mu = 0^{n_1}1^{n-n_1}$. Indeed, if a play π satisfies $\mathsf{Payoff}(\pi) \succsim \mu$ then either $\mathsf{Payoff}(\pi) = \mu$ in which case $\pi \in \cap_{i=n_1+1}^{n}\mathsf{Parity}(p_i)$, i.e. π satisfies the Streett objective, or $\mathsf{Payoff}(\pi) \succ \mu$ in which case there exists $1 \in \{1, \dots, n_1\}$ such that $\pi \in \mathsf{Parity}(p_i)$, i.e. π satisfies the Rabin objective. Conversely, if a play π satisfies the Streett or the Rabin objective then $\mathsf{Payoff}(\pi) \succsim \mu$ since $\mathsf{Payoff}(\pi) \succsim \mu$ (resp. $\succ \mu$) as soon as π satisfies the Streett (resp. Rabin) objective.
- As parity objectives are a special case of Rabin (Streett) objectives, the lower bound follows (from the previous item) for both lexicographic Rabin and Streett games.
- Lexicographic Muller games with $n = 1$ and $\mu = 1$ are a special case of Muller games and solving the latter games is PSPACE-complete [18].

This completes the proof. ◀

5 Values and optimal strategies for lexicographic games

In this section, we first recall the notion of values and optimal strategies. We then show how to compute the values in lexicographic games, and what are the memory requirements for the related optimal strategies. This yields a *full picture* of the study of lexicographic games, see

Table 2 Overview of the results for lexicographic games with ω-regular objectives.

<table>
<tr><th></th><th>Threshold problem</th><th>Value</th><th>$\mathcal{P}_1$ memory</th><th>$\mathcal{P}_2$ memory</th></tr>
<tr><td>Büchi</td><td rowspan="3">P-complete</td><td rowspan="3">polynomial</td><td>linear</td><td>memoryless</td></tr>
<tr><td>Co-Büchi</td><td>memoryless</td><td>linear</td></tr>
<tr><td>Explicit Muller</td><td colspan="2">exponential</td></tr>
<tr><td>Reachability, safety</td><td rowspan="4">PSPACE-complete and FPT</td><td rowspan="4">exponential and FPT</td><td colspan="2" rowspan="4">exponential</td></tr>
<tr><td>Parity</td></tr>
<tr><td>Streett, Rabin</td></tr>
<tr><td>Muller</td></tr>
</table>

Table 2. In this table, the second column indicates the complexity of the threshold problem (Theorems 4, 7 and PSPACE upper bounds follow from results of [6]), the third one indicates the complexity of computing the values and the remaining columns summarize the memory requirements of winning and optimal strategies (Theorem 12 hereafter).

Values and optimal strategies

In a lexicographic game, one can define the best reward that $\mathcal{P}_1$ can ensure from a given vertex, that is, the highest threshold μ for which $\mathcal{P}_1$ can ensure a payoff $\succsim \mu$. Dually, we can also define the worst reward that $\mathcal{P}_2$ can ensure.

In the following definition, the infimum and supremum functions are applied with $\preccurlyeq$.

▶ **Definition 11.** Given a lexicographic game $(G, \Omega_1, \dots, \Omega_n, \preccurlyeq)$, for every vertex $v \in V$, the *upper value* $\overline{\mathsf{Val}}(v)$ and the *lower value* $\underline{\mathsf{Val}}(v)$ are defined as:

$$\overline{\mathsf{Val}}(v) = \inf_{\sigma_2 \in \Sigma_2} \sup_{\sigma_1 \in \Sigma_1} \mathsf{Payoff}(\mathsf{Out}(v, \sigma_1, \sigma_2)) \text{ and } \underline{\mathsf{Val}}(v) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathsf{Payoff}(\mathsf{Out}(v, \sigma_1, \sigma_2)).$$

The lexicographic game $(G, \Omega_1, \dots, \Omega_n, \preccurlyeq)$ is *value-determined* if $\underline{\mathsf{Val}}(v) = \overline{\mathsf{Val}}(v)$ $\forall v \in V$. In this case, we write $\mathsf{Val}(v) = \overline{\mathsf{Val}}(v) = \underline{\mathsf{Val}}(v)$ and we call $\mathsf{Val}(v)$ the *value* of v. Note that the inequality $\underline{\mathsf{Val}}(v) \preccurlyeq \overline{\mathsf{Val}}(v)$ always holds. If $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) can ensure a payoff $\succsim \underline{\mathsf{Val}}(v)$ (resp. $\preccurlyeq \overline{\mathsf{Val}}(v)$) from v, his related winning strategy σ_1^* (resp. σ_2^*) is called *optimal from* v.

Notice that for all lexicographic games such that the objectives $\Omega_1, \dots, \Omega_n$ are Borel sets, we have that these games are value-determined and have optimal strategies by Theorem 3 and Martin's theorem [20]. In the following theorem, we go further by giving time complexities and memory sizes of the optimal strategies.

▶ **Theorem 12.** *(1) The value of each vertex in lexicographic Büchi, co-Büchi, and explicit Muller games can be computed with a polynomial time algorithm, and with an exponential time and an* FPT *algorithm for lexicographic reachability, safety, parity, Rabin, Streett, and Muller games.*

(2) The following assertions hold for both winning strategies of the threshold problem and optimal strategies. Linear memory strategies are necessary and sufficient for $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) while memoryless strategies are sufficient for $\mathcal{P}_2$ (resp. $\mathcal{P}_1$) in lexicographic Büchi (resp. co-Büchi) games. Exponential memory strategies are both necessary and sufficient for both players in lexicographic reachability, safety, explicit Muller, parity, Rabin, Streett, and Muller games.

We only give a sketch of the proof.

First, for the considered lexicographic games, the values can be obtained by solving n times well-chosen threshold problems. Therefore, results of Part (1) of Theorem 12 follows from the second column of Table 2. In addition to give the exact value of a vertex, this

procedure also shows that optimal strategies correspond to winning strategies for specific threshold problems. Therefore, we just have to analyze memory requirements of winning strategies for the threshold problem in lexicographic games to obtain those of optimal strategies. Upper bounds on memory sizes of winning strategies are obtained by analyzing the several reductions done in the proof of Theorem 4 in the case of a preorder with a compact embedding. Lower bounds for lexicographic Büchi and co-Büchi games are obtained thanks to a reduction from generalized Büchi games, and for the other lexicographic games thanks to the reductions proposed in the proof of Theorem 7, Part (2). This yields results of Part (2) of Theorem 12.

References

1 Shaull Almagor and Orna Kupferman. Latticed-LTL synthesis in the presence of noisy inputs. *Discrete Event Dynamic Systems*, 27(3):547–572, 2017. doi:10.1007/s10626-017-0242-0.

2 Rajeev Alur, Aditya Kanade, and Gera Weiss. Ranking automata and games for prioritized requirements. In *CAV Proceedings*, volume 5123 of *LNCS*, pages 240–253. Springer, 2008. doi:10.1007/978-3-540-70545-1_23.

3 Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *CONCUR Proceedings*, volume 2761 of *LNCS*, pages 127–141. Springer, 2003. doi:10.1007/978-3-540-45187-7_8.

4 Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Robustness in the presence of liveness. In *CAV Proceedings*, volume 6174 of *LNCS*, pages 410–424. Springer, 2010. doi:10.1007/978-3-642-14295-6_36.

5 Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV Proceedings*, volume 5643 of *LNCS*, pages 140–156. Springer, 2009. doi:10.1007/978-3-642-02658-4_14.

6 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Concurrent games with ordered objectives. In *FOSSACS Proceedings*, volume 7213 of *LNCS*, pages 301–315. Springer, 2012. doi:10.1007/978-3-642-28729-9_20.

7 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the complexity of heterogeneous multidimensional games. In *CONCUR Proceedings*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.11.

8 Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In *CSL-LICS Proceedings*, pages 26:1–26:26. ACM, 2014. doi:10.1145/2603088.2603109.

9 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *STOC Proceedings*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

10 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Conditionally optimal algorithms for generalized Büchi games. In *MFCS Proceedings*, volume 58 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.25.

11 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In *FOSSACS Proceedings*, volume 4423 of *LNCS*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0_12.

12 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.

13 Rodney G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Publishing Company, Incorporated, 2012.

14 Nathanaël Fijalkow and Florian Horn. Les jeux d'accessibilité généralisée. *Technique et Science Informatiques*, 32:931–949, 2013. doi:10.3166/tsi.32.931-949.

15 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 *LNCS*. Springer, 2002.

16 Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, and Michael Wooldridge. Nash equilibria in concurrent games with lexicographic preferences. In *IJCAI Proceedings*, pages 1067–1073. ijcai.org, 2017. doi:10.24963/ijcai.2017/148.

17 Florian Horn. Explicit Muller games are PTIME. In *FSTTCS Proceedings*, volume 2 of *LIPIcs*, pages 235–243. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1756.

18 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *MFCS Proceedings*, volume 3618 of *LNCS*, pages 495–506. Springer, 2005. doi:10.1007/11549345_43.

19 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. In *EUMAS Proceedings*, volume 8953 of *LNCS*, pages 219–235. Springer, 2014. doi:10.1007/978-3-319-17130-2_15.

20 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

21 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL Proceedings*, pages 179–190. ACM Press, 1989.

22 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.

23 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci., 158*, 959:343–359, 1996.

A Universal Session Type for Untyped Asynchronous Communication

Stephanie Balzer[1]
Carnegie Mellon University, USA

Frank Pfenning[2]
Carnegie Mellon University, USA

Bernardo Toninho[3]
NOVA LINCS, Universidade Nova de Lisboa, Portugal

Abstract

In the simply-typed λ-calculus we can recover the full range of expressiveness of the untyped λ-calculus solely by adding a single recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$. In contrast, in the session-typed π-calculus, recursion alone is insufficient to recover the untyped π-calculus, primarily due to linearity: each channel just has two unique endpoints. In this paper, we show that shared channels with a corresponding sharing semantics (based on the language $\mathsf{SILL_S}$ developed in prior work) are enough to embed the untyped asynchronous π-calculus via a universal shared session type $\mathcal{U}_\mathsf{s}$. We show that our encoding of the asynchronous π-calculus satisfies operational correspondence and preserves observable actions (i.e., processes are weakly bisimilar to their encoding). Moreover, we clarify the expressiveness of $\mathsf{SILL_S}$ by developing an operationally correct encoding of $\mathsf{SILL_S}$ in the asynchronous π-calculus.

2012 ACM Subject Classification Theory of computation → Process calculi, Theory of computation → Linear logic

Keywords and phrases Session types, sharing, π-calculus, bisimulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.30

1 Introduction

Session types [20, 22, 23] prescribe the protocols of message exchange between processes that interact along channels. The recent discovery of a Curry-Howard isomorphism between *linear logic* and the *session-typed π-calculus* [8, 9, 42, 38] has given message-passing concurrency a firm logical foundation. Programming languages [40, 19] building on this isomorphism not only guarantee *session fidelity* (i.e., protocol compliance) but also a form of *global progress*, since the process graph forms a tree and is acyclic by construction.

While the linear logic session framework allows for persistent servers through the exponential modality (i.e., replicated sessions that may be used an arbitrary number of times), it enforces a strict separation between server instances by means of a copying semantics [8, 42]. For instance, interactions between a client and a server cannot affect future client-server interactions. Thus, this session discipline fundamentally excludes programming scenarios that require *sharing* of server resources such as shared databases or shared output devices. This observation triggered the realization that linear session-typed calculi lag behind the untyped

[1] NSF Grant No. CCF-1718267: "Enriching Session Types for Practical Concurrent Programming"
[2] NSF Grant No. CCF-1718267: "Enriching Session Types for Practical Concurrent Programming"
[3] NOVA LINCS (Ref. UID/CEC/04516/2013)

asynchronous π-calculus in expressiveness and the question of whether the full expressiveness of the untyped asynchronous π-calculus could be recovered in such a logical setting [42].

In this paper, we answer this question *positively*. In prior work we have introduced *manifest sharing* [3], a modal-typing discipline that orchestrates the coexistence of linear *and* shared channels while maintaining session fidelity, at the expense of generalized deadlock-freedom. In this work we show that manifest sharing recovers the expressiveness of the untyped asynchronous π-calculus. Given our language $\mathsf{SILL_S}$ [3, 4] that supports manifest sharing, we provide an encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$, showing that our encoding satisfies *operational correspondence* and that π-calculus processes are *weakly bisimilar* to their $\mathsf{SILL_S}$ encodings. To clarify the expressiveness of $\mathsf{SILL_S}$, we moreover develop an encoding in the other direction, embedding $\mathsf{SILL_S}$ into the asynchronous (polyadic) π-calculus and satisfying operational correspondence.

Key to our encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$ is the representation of a π-calculus channel as a recursive *shared* session type $\mathcal{U}_\mathsf{s}$, reminiscent of the encoding of the untyped λ-calculus into the simply-typed λ-calculus via the type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$. While the addition of a single recursive type is sufficient to recover the expressiveness of the untyped λ-calculus in the simply-typed λ-calculus, our result reveals that *both* shared and recursive session types are necessary to achieve the analogous result in the session-typed π-calculus.

The contributions of this paper are:

- A proof that our encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$ is operationally sound and complete and preserves observable actions (i.e., processes are weakly bisimilar to their encoding);[4]
- A formulation of a weak bisimulation between a labelled transition system for the asynchronous π-calculus and a multiset rewriting system for *closed terms* of $\mathsf{SILL_S}$;
- Evidence of the instrumental role shared channels take in the expressiveness of session-typed process calculi;
- An encoding of $\mathsf{SILL_S}$ into the untyped asynchronous polyadic π-calculus, satisfying operational correspondence.

Paper Structure. Section 2 provides the necessary background on $\mathsf{SILL_S}$. Section 3 introduces the encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$ and states and proves operational and observational correspondence (i.e., preservation of reductions and observable actions). Section 4 develops an encoding of $\mathsf{SILL_S}$ into the untyped asynchronous polyadic π-calculus, satisfying operational correspondence. Section 5 summarizes related work, and Section 6 concludes the paper. Proofs are given in a companion technical report.

2 Manifest Sharing with Session Types

In this section, we provide an introduction to *manifest sharing* [3] and the programming language $\mathsf{SILL_S}$ [3, 4], to the extent necessary for the development in this paper. *Session types* [20, 22, 23, 8, 40, 9, 42, 38] prescribe the protocols of message exchange between processes that interact along channels. For example, the recursive linear session type

$$\mathsf{queue}\ A = \&\{\mathsf{enq} : A \multimap \mathsf{queue}\ A,\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : A \otimes \mathsf{queue}\ A\}\}$$

[4] A preliminary version of our encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$ has been published in [3] for illustration purposes, but without proof.

Table 1 Overview of session types in SILL$_S$ together with their operational meaning.

Session type current	cont	Process term current	cont	Description
$c_L : \oplus\{\overline{l : A_L}\}$	$c_L : A_{L_h}$	$c_L.l_h\ ;P$	P	provider sends label l_h along c_L
		$\mathsf{case}\ c_L\ \mathsf{of}\ \overline{l \Rightarrow Q}$	Q_h	client receives label l_h along c_L
$c_L : \&\{\overline{l : A_L}\}$	$c_L : A_{L_h}$	$\mathsf{case}\ c_L\ \mathsf{of}\ \overline{l \Rightarrow P}$	P_h	provider receives label l_h along c
		$c_L.l_h\ ;Q$	Q	client sends label l_h along c_L
$c_L : A_L \otimes B_L$	$c_L : B_L$	$\mathsf{send}\ c_L\ d_L\ ;P$	P	provider sends channel $d_L : A_L$ along c_L
		$y_L \leftarrow \mathsf{recv}\ c_L\ ;Q_{y_L}$	$[d_L/y_L]\,Q_{y_L}$	client receives channel $d_L : A_L$ along c_L
$c_L : A_L \multimap B_L$	$c_L : B_L$	$y_L \leftarrow \mathsf{recv}\ c_L\ ;P_{y_L}$	$[d_L/y_L]\,P_{y_L}$	provider receives channel $d_L : A_L$ along c_L
		$\mathsf{send}\ c_L\ d_L\ ;Q$	Q	client sends channel $d_L : A_L$ along c_L
$c_L : \Pi x{:}A_S.B_L$	$c_L : B_L$	$\mathsf{send}\ c_L\ d_S\ ;P$	P	provider sends channel $d_S : A_S$ along c_L
		$y_S \leftarrow \mathsf{recv}\ c_L\ ;Q_{y_S}$	$[d_S/y_S]\,Q_{y_S}$	client receives channel $d_S : A_S$ along c_L
$c_L : \exists x{:}A_S.B_L$	$c_L : B_L$	$y_S \leftarrow \mathsf{recv}\ c_L\ ;P_{y_S}$	$[d_S/y_S]\,P_{y_S}$	provider receives channel $d_S : A_S$ along c_L
		$\mathsf{send}\ c_L\ d_S\ ;Q$	Q	client sends channel $d_S : A_S$ along c_L
$c_L : \mathbf{1}$	-	$\mathsf{close}\ c_L$	-	provider sends "end" along c_L
		$\mathsf{wait}\ c_L\ ;Q$	Q	provider receives "end" along c_L
$c_L : \downarrow^S_L A_S$	$c_S : A_S$	$c_S \leftarrow \mathsf{detach}\ c_L\ ;P_{x_S}$	$[c_S/x_S]\,P_{x_S}$	provider sends "detach c_S" along c_L
		$x_S \leftarrow \mathsf{release}\ c_L\ ;Q_{x_S}$	$[c_S/x_S]\,Q_{x_S}$	client receives "detach c_S" along c_L
$c_S : \uparrow^S_L A_L$	$c_L : A_L$	$c_L \leftarrow \mathsf{acquire}\ c_S\ ;Q_{x_L}$	$[c_L/x_L]\,Q_{x_L}$	client sends "acquire c_L" along c_S
		$x_L \leftarrow \mathsf{accept}\ c_S\ ;P_{x_L}$	$[c_L/x_L]\,P_{x_L}$	provider receives "acquire c_L" along c_S

defines the protocol of how to interact with a provider of a queue data structure that contains elements of some variable type A. In a session-typed interpretation of intuitionistic linear logic, session types are expressed from the point of view of the providing process, with the channel along which the process provides the session behavior being defined by the session type. This choice avoids the explicit dualization of a session type present in the original presentations of session types [20, 22] and those based on classical linear logic [42]. We adopt an *equi-recursive* [11] interpretation for recursive session types, silently equating a recursive session type with its unfolding and requiring types to be *contractive* [16].

Table 1 provides an overview of SILL$_S$'s session types and their operational reading. For each type constructor, Table 1 lists the points of view of the *provider* and *client* of the given type, in the first and second lines, respectively. For each connective, its session type before the exchange (**Session type current**) and after the exchange (**Session type cont(inuation)**) is given. Likewise, the implementing process term is indicated before the exchange (**Process term current**) and after the exchange (**Process term continuation**). Table 1 shows that the process terms of a provider and a client for a connective come in matching pairs. Both participants' view of the session changes consistently.

For the linear session type queue A specified above, we have the following protocol: a process providing a service of type queue A gives a client the choice to either enqueue (enq) or dequeue (deq) an element of type A. Upon receipt of the label enq, the providing process expects to receive a channel of type A to be enqueued and recurs. Upon receipt of the label deq, the providing process either indicates that the queue is empty (none), in which case it terminates, or that there is a channel stored in the queue (some), in which case it dequeues this element, sends it to the client, and recurs.

Linearity restricts session type queue A to a single client. If we want the queue to be used in a classical consumer-producer scenario, where we have multiple producers and consumers

$$
\begin{array}{l}
q' \leftarrow \mathsf{acquire}\ \mathbf{q}\ ; \\
q'.\mathsf{enq}\ ; \\
\mathsf{send}\ q'\ \mathbf{x}\ ; \\
\mathbf{q} \leftarrow \mathsf{release}\ q'
\end{array}
$$

Figure 1 A client of a shared queue.

accessing the queue, we can use the following *shared session type* instead:

$$
\begin{array}{l}
\mathbf{queue}\ \mathbf{A_S} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \& \{\mathsf{enq} : \Pi x : \mathbf{A_S}.\downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{queue}\ \mathbf{A_S}, \\
\qquad\qquad\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{queue}\ \mathbf{A_S},\ \mathsf{some} : \exists x : \mathbf{A_S}.\downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{queue}\ \mathbf{A_S}\}\}
\end{array}
$$

For ease of reading, we typeset shared session types and channels in programs in red and **bold** font as opposed to linear session types and channels, which we typeset in black and regular font. Session type $\mathsf{queue}\ A_{\mathsf{S}}$ now describes the session offered by a shared process. Since a shared process can have multiple clients that refer to the process by a *shared channel*, state-altering communication with a shared process must only happen once exclusive access to the process has been obtained. Otherwise, session fidelity would be endangered. To this end, $\mathsf{SILL}_{\mathsf{S}}$ imposes an *acquire-release* discipline on shared processes, where an acquire yields exclusive access to a shared process, if the process is available, and a release relinquishes exclusive access. As a result, processes can alternate between linear and shared, where a successful acquire of a shared process turns the process into a linear one, and conversely, a release of a linear process turns the process into a shared one.

A potential producer process can now interact with a process that implements session type $\mathsf{queue}\ A_{\mathsf{S}}$ according to Figure 1, assuming that q is of type $\mathsf{queue}\ A_{\mathsf{S}}$ and x is of type A_{S}. The statement $q' \leftarrow \mathsf{acquire}\ q$, yields, if successful, the queue's linear channel q' along which the producer process can enqueue the element. The statement $q \leftarrow \mathsf{release}\ q'$ releases the now linear queue process providing along q', giving turn to another producer or consumer process, and yields the queue's shared channel q. As indicated by Table 1, there exist the dual notions of an *accept* and *detach* for an acquire and release, respectively, denoting the matching statements by a provider.

A key contribution of manifest sharing is not only to support acquire-release as a programming primitive but also to make it manifest in the type system. Generalizing the idea of type *stratification* [35, 6, 36], session types are stratified into a linear and shared layer with two *adjoint modalities* going back and forth between them:

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid \exists x : A_{\mathsf{S}}.B_{\mathsf{L}} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \Pi x : A_{\mathsf{S}}.B_{\mathsf{L}} \mid \&\{\overline{l : A_{\mathsf{L}}}\} \mid \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}
\end{array}
$$

The modal operator $\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}$ shifting *down* from the shared to the linear layer is then interpreted as a *release* (and, dually, detach) and the operator $\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$ shifting *up* from the linear to the shared layer as an *acquire* (and, dually, accept). As a result, we obtain a type system where a session type dictates any form of synchronization, including the acquisition and release of a shared process.

Returning to the shared session type $\mathsf{queue}\ A_{\mathsf{S}}$ defined above, we can see that any exchange of labels or channels with the queue is now guarded by a preceding acquire, and that the queue must be released before it recurs. The shared session type further deviates from its linear version in that it contains shared elements, as the entire queue is shared, and by recurring in the empty case of a dequeueing request, as there are now multiple clients.

We briefly discuss the typing and the dynamics of acquire-release. The typing and the dynamics of the residual linear connectives are standard. As is usual for an intuitionistic

$$\mathsf{proc}(c_{\mathsf{L}},\, x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{S}},\, x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\, P_{x_{\mathsf{L}}}) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\, [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \mathsf{unavail}(a_{\mathsf{S}})$$

$$\mathsf{proc}(c_{\mathsf{L}},\, x_{\mathsf{S}} \leftarrow \mathsf{release}\ a_{\mathsf{L}}\ ;\, Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\, x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ;\, P_{x_{\mathsf{S}}}),\ \mathsf{unavail}(a_{\mathsf{S}}) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{S}},\, [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}})$$

Figure 2 Multiset rewriting rules for acquire-release.

interpretation, each connective gives rise to a left and a right rule, denoting the use and provision, respectively, of a session of the given type:

$$(\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathsf{L})\quad \frac{\Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}};\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ;\, Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}$$

$$(\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathsf{R})\quad \frac{\Gamma;\ \cdot \vdash_\Sigma P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ;\, P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})}$$

$$(\text{T-}\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathsf{L})\quad \frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ;\, Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}$$

$$(\text{T-}\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathsf{R})\quad \frac{\Gamma \vdash_\Sigma P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\ ;\, P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}})}$$

The typing judgments $\Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ and $\Gamma;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ indicate that process P provides a service of session type A along channel x, given the typing of services provided by processes along the channels in typing contexts Γ (and Δ). Γ and Δ consist of hypotheses on the typing of shared and linear channels, respectively, where Γ is a structural and Δ a linear context. To allow for recursive process definitions, the typing judgment depends on a signature Σ that is populated with all process definitions prior to type-checking. The adjoint formulation forbids a shared process from depending on linear channels [3, 35]. Thus, when a shared session accepts an acquire and shifts to linear, it starts with an empty linear context.

Operationally, the dynamics of $\mathsf{SILL}_{\mathsf{S}}$ is captured by *multiset rewriting rules* [10], which denote computation in terms of state transitions between configurations of processes. Multiset rewriting rules are local in that they only mention the parts of a configuration they rewrite. For acquire-release we have the rules of Figure 2.

Configuration states are defined by the predicates $\mathsf{proc}(c_m,\, P)$ and $\mathsf{unavail}(a_{\mathsf{S}})$. The former denotes a process with process term P providing along channel c_m, the latter a placeholder for a shared process providing along channel a_{S} that is currently not available. The above rule exploits the invariant that a process' providing channel a can come at one of two modes, a linear one, a_{L}, and a shared one, a_{S}. While the process is linear, it provides along a_{L}, while it is shared, along a_{S}. When a process shifts between modes, it switches between the two modes of its offering channel. This channel at the appropriate mode is substituted for the variables occurring in process terms.

3 Recovering the Untyped Asynchronous π-calculus in $\mathsf{SILL}_{\mathsf{S}}$

We now detail our encoding of the asynchronous π-calculus into $\mathsf{SILL}_{\mathsf{S}}$, show that it satisfies *operational correspondence* and that processes are *weakly bisimilar* to their $\mathsf{SILL}_{\mathsf{S}}$ encodings.

3.1 Encoding the Untyped Asynchronous π-calculus in $\mathsf{SILL}_{\mathsf{S}}$

The essence of linear session-typed process calculi – treating channels as stateful resources – is fundamental in facilitating reasoning about session-typed programs and to guarantee strong properties, such as session fidelity and possibly deadlock-freedom. However, where channels in linear session-typed process calculi connect exactly one *sending* process with one *receiving* process, in the untyped π-calculus they may connect *multiple* sending and

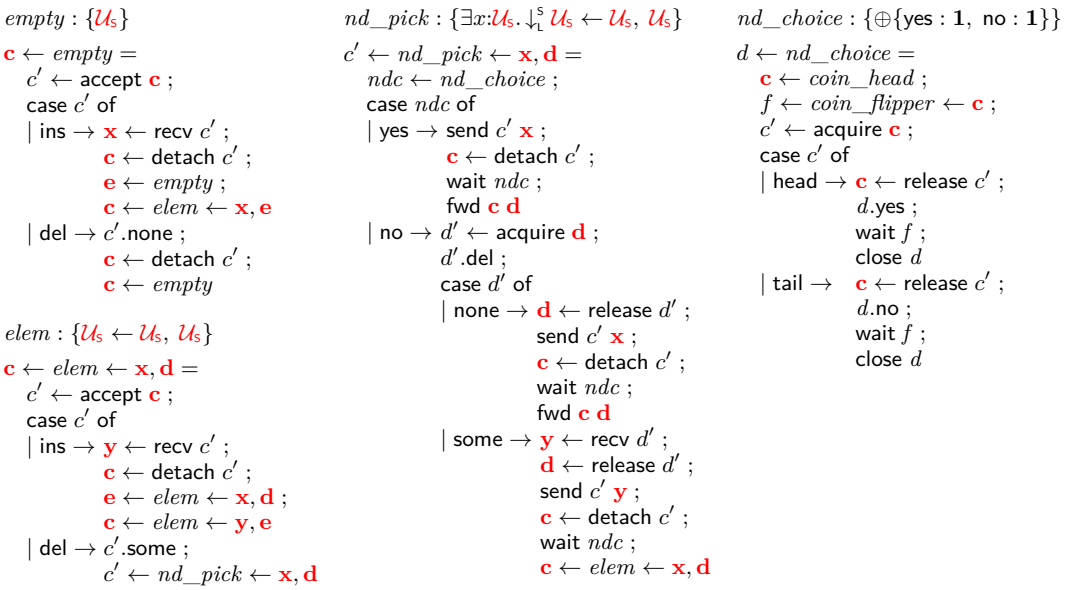


Figure 3 Processes *empty* and *elem* implementing a π-calculus channel with auxiliary processes. See Figure 4 for processes *coin_head*, *coin_tail*, and *coin_flipper* and session type coin.

receiving processes, giving rise to *non-determinism*. For example, the π-calculus process $c(x).P \mid \overline{c}\langle a\rangle \mid c(y).Q$, made up of three parallel components, where the first and third seek to input along channel c and the second outputs the name a along c, may reduce to either $[a/x]\,P \mid c(y).Q$ or $c(x).P \mid [a/y]\,Q$.

In purely linear session-typed process calculi, on the other hand, message exchange is completely *deterministic*, even in the presence of replicated or persistent sessions (this argument is made precise through a typed contextual equivalence for intuitionistic linear logic sessions in [34]). The addition of sharing to session-typed calculi – and with it non-determinism – suggests that it should now be possible to faithfully encode the untyped π-calculus. In previous work we have postulated this conjecture by providing an encoding of the untyped asynchronous π-calculus into SILL_S [3], without any further proof. We now refine the encoding and prove it operationally and behaviorally correct.

The basic idea of our encoding is to represent a π-calculus *process* by a *linear* SILL_S process and a π-calculus *channel* by a *shared* SILL_S process. Reminiscent of the encoding of the untyped λ-calculus into the typed λ-calculus, we type π-calculus channels with a *universal recursive shared* session type $\mathcal{U}_\mathsf{s}$:

$$\mathcal{U}_\mathsf{s} = \uparrow^\mathsf{s}_\mathsf{L} \&\{\mathsf{ins} : \Pi x{:}\mathcal{U}_\mathsf{s}.\downarrow^\mathsf{s}_\mathsf{L} \mathcal{U}_\mathsf{s},\ \mathsf{del} : \oplus\{\mathsf{none} : \downarrow^\mathsf{s}_\mathsf{L} \mathcal{U}_\mathsf{s},\ \mathsf{some} : \exists x{:}\mathcal{U}_\mathsf{s}.\downarrow^\mathsf{s}_\mathsf{L} \mathcal{U}_\mathsf{s}\}\}$$

Similar to the type $\mathsf{queue}\ A_\mathsf{s}$ of Section 2, the type $\mathcal{U}_\mathsf{s}$ represents a buffer that stores elements, but with the elements being of type $\mathcal{U}_\mathsf{s}$ themselves and *without* maintaining any order. Figure 3 shows the processes *empty* and *elem* that implement session type $\mathcal{U}_\mathsf{s}$. In SILL_S, we declare the type of a defined process X with $X : \{A \leftarrow A_1, \ldots, A_n\}$, indicating that the process provides a service of type A, using channels of type $A_1, \ldots, A_n$. The definition of the process is then given by $x \leftarrow X \leftarrow y_1, \ldots, y_n = P$, where P is the body of the process with occurrences of channels $y_1 : A_1, \ldots, y_n : A_n$. A new process X providing along channel x is spawned with an expression of the form $x \leftarrow X \leftarrow y_1, \ldots, y_n\ ;\ Q_x$, where Q_x is the continuation binding x. We refer to Table 1 for the meaning of the process terms.

$\mathbf{coin} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \oplus \{\mathsf{head} : \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{coin},\ \mathsf{tail} : \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{coin}\}$

$coin\_head : \{\mathbf{coin}\}$
$\mathbf{c} \leftarrow coin\_head =$
  $c' \leftarrow \mathsf{accept}\ \mathbf{c}$;
  $c'.\mathsf{head}$;
  $\mathbf{c} \leftarrow \mathsf{detach}\ c'$;
  $\mathbf{c} \leftarrow coin\_tail$

$coin\_tail : \{\mathbf{coin}\}$
$\mathbf{c} \leftarrow coin\_tail =$
  $c' \leftarrow \mathsf{accept}\ \mathbf{c}$;
  $c'.\mathsf{tail}$;
  $\mathbf{c} \leftarrow \mathsf{detach}\ c'$;
  $\mathbf{c} \leftarrow coin\_head$

$coin\_flipper : \{\mathbf{1} \leftarrow \mathbf{coin}\}$
$d \leftarrow coin\_flipper \leftarrow \mathbf{c} =$
  $c' \leftarrow \mathsf{acquire}\ \mathbf{c}$;
  $\mathsf{case}\ c'\ \mathsf{of}$
  $\mid \mathsf{head} \rightarrow\ \mathbf{c} \leftarrow \mathsf{release}\ c'$;
    $\mathsf{close}\ d$
  $\mid \mathsf{tail} \rightarrow\ \mathbf{c} \leftarrow \mathsf{release}\ c'$;
    $\mathsf{close}\ d$

Figure 4 Processes *coin_head*, *coin_tail*, and *coin_flipper* and session type coin, upon which process *nd_choice* in Figure 3 relies.

The buffer is implemented as a sequence of *elem* processes, ending in an *empty* process. The recursive process *elem* provides a buffer sequence along channel c and uses a channel $x : \mathcal{U}_{\mathsf{s}}$ (the buffer element at the current position in the sequence) as well as a channel $d : \mathcal{U}_{\mathsf{s}}$ (the next *elem* of the sequence). Process *empty*, on the other hand, provides an empty buffer sequence along channel c, without using any other channels. Both processes insert the received element at the head of the buffer sequence in the ins case, but handle the del case differently. Whereas process *empty* responds with label none, process *elem* responds with label some, followed by sending and deleting an *arbitrary* element from the buffer. Process *elem* achieves arbitrary deletion by recurring as process *nd_pick*. Process *nd_pick*, in turn, uses process *nd_choice* to nondeterministically choose between sending and deleting the element at the current position in the sequence (case yes) or, possibly recursively, propagating the deletion request to the next element in the sequence (case no). While linear session-typed calculi are deterministic, *non-determinism* arises in $\mathsf{SILL}_{\mathsf{S}}$ from the acquisition of shared channels, since it is unknown which client among all those competing to acquire a shared process will succeed. Process *nd_choice* uses this fact and achieves non-determinism by reading a coin that it shares with process *coin_flipper* (see Figure 4). Both processes then try to acquire the coin concurrently, which switches sides when read, with the result that the value read by *nd_choice* depends on the order in which the coin is acquired.

Given the buffer abstraction, encoded π-calculus processes in $\mathsf{SILL}_{\mathsf{S}}$ simply amount to "producers" and "consumers" of shared channels of type $\mathcal{U}_{\mathsf{s}}$. Any such process can communicate along a π-calculus channel by acquiring the corresponding $\mathsf{SILL}_{\mathsf{S}}$ channel of universal type. We are now ready to give the encoding of the untyped asynchronous monadic π-calculus [30, 37] into $\mathsf{SILL}_{\mathsf{S}}$. The syntax of the asynchronous π-calculus is [5]:

$$P \quad \triangleq \quad 0 \ \mid\ \overline{c}\langle a\rangle \ \mid\ c(x).P \ \mid\ \nu c\, P \ \mid\ P_1 \mid P_2 \ \mid\ !P$$

0 denotes an inactive process. $\overline{c}\langle a\rangle$ represents an asynchronous send of channel a along channel c. $c(x).P$ amounts to a guarded input, where the channel received along c is bound to x in the continuation P. $\nu c\, P$ introduces a new channel c that is bound in P. $P_1 \mid P_2$ denotes parallel composition of P_1 and P_2, and $!P$ replication of P (i.e., an unbounded number of copies of P in parallel). We assume a standard reduction and labelled transition semantics, but where replication involves an explicit reduction (and τ transition) instead of expansion through structural congruence: $!P \longrightarrow P \mid !P$. Moreover, we enforce that structural congruence is only applied at the top-level of processes.

Our encoding, shown in Figure 5, yields for each π-calculus process P a corresponding linear process $[\![P]\!]$ in $\mathsf{SILL}_{\mathsf{S}}$, satisfying the typing judgment: $\Gamma_{\mathcal{F}}; \Gamma_{\mathcal{B}}; \Gamma_{\mathcal{I}}; \cdot \vdash_{\Sigma} [\![P]\!] :: (\cdot)$. We use an empty succedent to denote that the process does not provide any session. Since all communication is going to happen along π-calculus channels, i.e., the shared $\mathsf{SILL}_{\mathsf{S}}$ processes of type $\mathcal{U}_{\mathsf{s}}$, the linear $\mathsf{SILL}_{\mathsf{S}}$ processes representing π-calculus processes merely become clients

$$
\begin{array}{lcl}
[\![0]\!] & = & \cdot \\
[\![\overline{c}\langle a\rangle]\!] & = & x \leftarrow \mathit{snd} \leftarrow \mathbf{c}\ ; \\
 & & \mathsf{send}\ x\ \mathbf{a}\ ; \\
 & & \mathsf{wait}\ x\ ; \cdot \\
[\![c(x).P]\!] & = & y \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c}\ ; \\
 & & \mathbf{z} \leftarrow \mathsf{recv}\ y\ ; \\
 & & \mathsf{wait}\ y\ ; \\
 & & [\mathbf{z}/x]\,[\![P]\!] \\
[\![\nu x\, P]\!] & = & \mathbf{y} \leftarrow \mathit{empty}\ ; \\
 & & [\mathbf{y}/x]\,[\![P]\!] \\
[\![P_1 \mid P_2]\!] & = & \_ \leftarrow [\![P_1]\!]\ ; \\
 & & [\![P_2]\!] \\
[\![!P]\!] & = & \mathit{Rec}_{!P} \quad \text{where} \\
\mathit{Rec}_{!P} & = & \_ \leftarrow [\![P]\!]\ ; \\
 & & \mathit{Rec}_{!P}
\end{array}
$$

$$
\begin{array}{l}
\mathit{snd} : \{(\Pi x{:}\mathcal{U}_{\mathsf{s}}.\,\mathbf{1}) \leftarrow \mathcal{U}_{\mathsf{s}}\} \\
d \leftarrow \mathit{snd} \leftarrow \mathbf{c} = \\
\quad \mathbf{x} \leftarrow \mathsf{recv}\ d\ ; \\
\quad c' \leftarrow \mathsf{acquire}\ \mathbf{c}\ ; \\
\quad c'.\mathsf{ins}\ ; \\
\quad \mathsf{send}\ c'\ \mathbf{x}\ ; \\
\quad \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\quad \mathsf{close}\ d
\end{array}
$$

$$
\begin{array}{l}
\mathit{poll\_rcv} : \{(\exists x{:}\mathcal{U}_{\mathsf{s}}.\,\mathbf{1}) \leftarrow \mathcal{U}_{\mathsf{s}}\} \\
d \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c} = \\
\quad c' \leftarrow \mathsf{acquire}\ \mathbf{c}\ ; \\
\quad c'.\mathsf{del}\ ; \\
\quad \mathsf{case}\ c'\ \mathsf{of} \\
\quad \mid \mathsf{none} \rightarrow \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\qquad\qquad\quad d \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c} \\
\quad \mid \mathsf{some} \rightarrow \mathbf{x} \leftarrow \mathsf{recv}\ c'\ ; \\
\qquad\qquad\quad \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\qquad\qquad\quad \mathsf{send}\ d\ \mathbf{x}\ ; \\
\qquad\qquad\quad \mathsf{close}\ d
\end{array}
$$

Figure 5 Translation of untyped asynchronous π-calculus processes into $\mathsf{SILL}_{\mathsf{S}}$ and auxiliary processes *snd* and *poll_recv* (*empty* : $\{\mathcal{U}_{\mathsf{s}}\}$ is defined in Figure 3).

of those processes, without providing any behavior outright. In our earlier encoding [3], we have translated π-calculus processes into linear $\mathsf{SILL}_{\mathsf{S}}$ processes of type $\mathbf{1}$, since the notion of a non-providing linear process is not present in $\mathsf{SILL}_{\mathsf{S}}$. Our current encoding avoids the spurious exchange of wait messages required by type $\mathbf{1}$ and constitutes a return to the original interpretation of linear logic [8], where processes terminate silently. In the above typing judgment, we moreover subdivide the context Γ into three parts, to keep track of the *free* ($\Gamma_{\mathcal{F}}$) and *bound* ($\Gamma_{\mathcal{B}}$) π-calculus channels as well as of channels that are only used *internally* to the encoding ($\Gamma_{\mathcal{I}}$). When an encoded process reduces, new linear channels may be generated, for example, the providing channel of process *nd_choice*, which are all internal to the encoding.

The inactive process 0 is encoded as the empty $\mathsf{SILL}_{\mathsf{S}}$ process. The encoding of an output $[\![\overline{c}\langle a\rangle]\!]$ is implemented by spawning a new linear $\mathsf{SILL}_{\mathsf{S}}$ process *snd* of type $\Pi x{:}\mathcal{U}_{\mathsf{s}}.\,\mathbf{1}$ with access to the buffer implementing channel c. The encoding then sends the channel a to the spawned process *snd*, waiting for *snd* to acquire the buffer c, insert a, and terminate. The encoding of an input $[\![c(x).P]\!]$ is implemented by spawning a new linear $\mathsf{SILL}_{\mathsf{S}}$ process *poll_rcv* of type $\exists x{:}\mathcal{U}_{\mathsf{s}}.\,\mathbf{1}$ with access to the buffer implementing channel c. The encoding then waits for the spawned process *poll_rcv* to send back a channel and terminate, after which it continues at P, substituting the received channel for x. Process *poll_rcv* repeatedly checks, in a potentially infinite loop, if the buffer c contains an element. If so, it deletes it from the buffer, passes it on, and terminates. New name creation ($[\![\nu x\, P]\!]$) simply spawns a new buffer, offering on some fresh name x. Parallel ($[\![P_1 \mid P_2]\!]$) composition is embodied by a spawning of the processes P_1 in parallel with the executing process P_2. Finally, replication ($[\![!P]\!]$) is implemented by a loop that spawns copies of the replicated process.

To make our encoding more tangible, we derive the initial $\mathsf{SILL}_{\mathsf{S}}$ configuration obtained from translating the process $[\![\overline{c}\langle a\rangle \mid c(x).0]\!]$ according to the rules in Figure 5:

$$
\begin{array}{l}
a_{\mathsf{s}}, c_{\mathsf{s}}\ ;\ \cdot\ ;\ \cdot\ ;\ \cdot \vDash_{\Sigma} \mathsf{proc}(\_,\ y_{\mathsf{L}} \leftarrow \mathit{poll\_rcv} \leftarrow c_{\mathsf{s}};\ z_{\mathsf{s}} \leftarrow \mathsf{recv}\ y_{\mathsf{L}};\ \mathsf{wait}\ y_{\mathsf{L}};\ \cdot), \\
\qquad\qquad\qquad\quad \mathsf{proc}(\_,\ y_{\mathsf{L}} \leftarrow \mathit{snd} \leftarrow c_{\mathsf{s}};\ \mathsf{send}\ y_{\mathsf{L}}\ a_{\mathsf{s}};\ \mathsf{wait}\ y_{\mathsf{L}};\ \cdot), \\
\qquad\qquad\qquad\quad \mathsf{buf}(a_{\mathsf{s}} \mid y_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{s}};\ P_{y_{\mathsf{L}}}),\ \ \mathsf{buf}(c_{\mathsf{s}} \mid y_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{s}};\ P_{y_{\mathsf{L}}})
\end{array}
$$

To the left, we list the contents of the contexts $\Gamma_{\mathcal{F}}$; $\Gamma_{\mathcal{B}}$; $\Gamma_{\mathcal{I}}$; Δ, to the right the process configuration. For readability we use the short-form $\mathsf{buf}(a \mid P_a)$ to represent a sequence of *empty*-terminated *elem* processes denoting an entire buffer, with P_a standing for the next

statements to be executed. The above configuration will reduce, according to the semantics of SILL_S, until it halts in a state that consists of buffers representing the π-calculus channels, *coin_head* processes for any nondeterministic choices made, and unavail predicates for any shared channels that are not available. On the other hand, any linearly spawned processes that are internal to the encoding and not part of a buffer will have terminated.

Asynchrony of π-calculus outputs is achieved in our encoding by the introduction of the buffers, which temporarily store outputs until there is a process that is willing to receive. As a matter of fact, our buffers can be thought of manifestations of the "ether" to which asynchronous outputs are sent in the untyped asynchronous π-calculus! Our encoding is thus reminiscent of the encoding of the untyped asynchronous π-calculus into an untyped synchronous π-calculus with bags [5]. In fact, unlike the π-calculus where synchronous and asynchronous calculi have different expressive power [33], in the session-typed setting we can easily and selectively implement one in the other either by using double shifts to force acknowledgments [35] or by spawning single-message processes to achieve asynchrony [3]. The only significant point in SILL_S is that acquire/accept interactions must be a synchronization point. As we discuss in Section 3.3, crucial to the correctness of our encoding is also the removal of buffer elements non-deterministically. This guarantees that at no point in a reduction is the order between outputs determined. The use of nondeterministic deletion is another improvement over our earlier encoding [3], which uses non-deterministic insertion.

An interested reader may wonder whether asynchronous messages could not be encoded directly as processes, rather than storing them temporarily in a buffer until their receipt. After all, this is exactly what the syntax of the asynchronous π-calculus enforces! Non-determinism would then be achieved by the operational dynamics of the multiset rewriting rules, eliminating the need for the explicit encoding of non-deterministic buffers. Since every π-calculus channel c is mapped to a shared SILL_S channel c_s, this hypothetical encoding would require the ability to have multiple processes offering along the same shared channel (either the sender or the receiver sides of the communication). This is not allowed by the typing discipline, which crucially enforces that every process offers along a unique channel. Thus, an explicit representation of buffers is key, which then requires the encoding of non-deterministic bags to mimic the semantics of asynchrony in a precise way.

3.2 Operational Correspondence

We now develop an operational correspondence result for our encoding of the untyped asynchronous π-calculus. Operational correspondence results are standard *desiderata* for encodings of process calculi [18], showing that the computational features of the source language are preserved by the encoding in a precise sense. Following the terminology of [18], we aim to establish operational *completeness* (i.e., that π-calculus reductions are mimicked by the encoding) and *soundness* (i.e., that computations of encoded processes can be mapped back to those of the source terms) of our encoding.

As is the case in most encodings, some of the computation steps in the image of our encoding are purely administrative artifacts, and thus may not have a counterpart in the source. Specifically, the encoding of π-calculus channels as buffers introduces quite a few such "spurious" steps. Rather than relating source and image of the encoding at every step [5, 18], we introduce the notion of an *administrative* transition, and then state operational correspondence modulo such administrative transitions.

Given the nature of the asynchronous π-calculus, in which outputs are sent into the "ether" and synchronization only happens upon receipt, we deem the interactions leading to the insertion into a buffer as administrative and only the removal itself *relevant*. This treatment

is consistent with the existing literature. In the encoding of the untyped asynchronous π-calculus into an untyped synchronous π-calculus with bags [5], output prefixes are equated with one-element bags, and synchronization amounts to directly reading from these bags. We define relevant and administrative transitions in the image of our encoding as follows:

▶ **Definition 1** (Relevant and Administrative Transitions of Encoding)**.** We say that a relevant transition, written $\longrightarrow_r$, is a standard transition between SILL configurations such that: Ω, $\mathsf{proc}(d_{\mathsf{L}}, x_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ;\ Q_{x_{\mathsf{S}}}) \longrightarrow \Omega'$, $\mathsf{proc}(d_{\mathsf{L}}, [a_{\mathsf{S}}/x_{\mathsf{S}}]\ Q_{x_{\mathsf{S}}})$, for some $\Omega, \Gamma_{\mathcal{F}}, \Gamma_{\mathcal{B}}, \Gamma_{\mathcal{I}}, a_{\mathsf{S}}, c_{\mathsf{S}}$, and d_{L} such that $a_{\mathsf{S}} \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$, $c_{\mathsf{S}} \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$, and $d_{\mathsf{S}} \in \Gamma_{\mathcal{I}}$.

An administrative transition, written $\longrightarrow_a$, is a transition defined by the standard transition relation between SILL configurations, but excluding a relevant transition. We write $\Longrightarrow_a$ for the reflexive transitive closure of $\longrightarrow_a$, and write $\Longrightarrow_r$ for $\Longrightarrow_a \longrightarrow_r \Longrightarrow_a$.

Inspecting our encoding (Figure 3 and Figure 5), we can see that a relevant transition amounts to the receive action in the some branch in process *poll_rcv*, which synchronizes with the buffer to receive a channel. The parameters of the above definition uniquely identify this synchronization point: process *poll_rcv* is a linear process providing along a linear channel d_{L} that is internal to the encoding ($d_{\mathsf{L}} \in \Gamma_{\mathcal{I}}$), and both the received channel a_{S} and the offering channel c_{S} of the buffer are either free our bound names of the original π-calculus process ($a_{\mathsf{S}} \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$ and $c_{\mathsf{S}} \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$).

Equipped with these two notions of transition, we can establish operational soundness and completeness. Their statements rely on the definition $[\![fn(P)]\!]$, which stand for a *configuration* of *empty* buffer processes of the form $\mathsf{buf}(c_{\mathsf{S}_1} \mid y_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{S}_1};\ Q_{y_{\mathsf{L}}}), \ldots, \mathsf{buf}(c_{\mathsf{S}_n} \mid y_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{S}_n};\ Q_{y_{\mathsf{L}}})$, where $fn(P) = \{c_1, \ldots, c_n\}$ denotes the set of free names in P. The definition allows us to compose an encoded π-calculus process with the appropriate buffer representations for all its free channel names.

▶ **Theorem 2** (Operational Correspondence)**.**

Completeness. *For all $P \longrightarrow P'$, there exists Ω_1, Ω_2 such that $[\![fn(P)]\!]$, $\mathsf{proc}(\_, [\![P]\!]) \Longrightarrow_r \Omega_1, \Omega_2$ or $[\![fn(P)]\!]$, $\mathsf{proc}(\_, [\![P]\!]) \Longrightarrow_a \Omega_1, \Omega_2$, with $[\![fn(P')]\!]$, $\mathsf{proc}(\_, [\![P']\!]) \Longrightarrow_a \Omega_2$.*

Soundness. *For all P and $[\![fn(P)]\!]$, $\mathsf{proc}(\_, [\![P]\!]) \Longrightarrow_r \Omega$, there exists a P', Ω_1, Ω_2 such that $P \longrightarrow P'$ and $\Omega = \Omega_1, \Omega_2$ and $[\![fn(P')]\!]$, $\mathsf{proc}(\_, [\![P']\!]) \Longrightarrow_a \Omega_2$.*

For operational completeness, we identify each individual π-calculus reduction with either one relevant transition (possibly preceded or followed by several administrative transitions), or, for the π-calculus reduction corresponding to forking a parallel replica (i.e., $!P \longrightarrow P \mid !P$), with one administrative transition. For operational soundness, we match relevant transitions of encoded processes with one process reduction. In both settings we identify the artifacts of the encoding (coin processes and unavail channels) through the configuration Ω_1.

We note that the encodings of continuations eventually "catch up" (via administrative transitions) with the configuration that results from the relevant transition, instead of having a more immediate identification through the encoding. This treatment is due to the distinction between processes (static entities) and configurations (runtime entities) in $\mathsf{SILL}_{\mathsf{S}}$, a distinction not present in the π-calculus, where processes *are* the runtime entities. For instance, parallel composition in $\mathsf{SILL}_{\mathsf{S}}$ is achieved via an explicit spawning construct, whose semantics is to administratively *transition* to a configuration with the spawned process executing in parallel.

3.3 Observational Correspondence

In the previous section we have established that our encoding preserves reductions in the π-calculus in a strong sense, by identifying precisely the transitions in the operational semantics of $\mathsf{SILL_S}$ that correspond to reductions in the π-calculus processes in a way that is consistent with standard results on the nature of asynchrony of the untyped asynchronous π-calculus.

We now go further and relate *observable actions* (i.e., labelled transitions) in the π-calculus with their corresponding observables in $\mathsf{SILL_S}$ configuration rewrites. The key challenge here is to identify what those observables in $\mathsf{SILL_S}$ are because of the significant differences between the semantic frameworks of the π-calculus and $\mathsf{SILL_S}$. Whereas the π-calculus adopts an *open-world* view of observable actions with an unspecified environment (the "ether"), $\mathsf{SILL_S}$ adopts a *closed-world* view of a configuration of processes that are composed to form a complete program that can be run.

To clarify, consider the π-calculus process $\overline{c}\langle a\rangle \mid c(x).P$, where both c and a are free names. This process can interact with the environment through its free names by taking any of the following three observable actions: the output along c, the input along c, or the τ-action, corresponding to the synchronization between these dual actions. Now consider the $\mathsf{SILL_S}$ encoding of $[\![\overline{c}\langle a\rangle \mid c(x).P]\!]$. It results in a complete configuration consisting of the encoding of the process together with an *explicit* encoding of the free names c and a in terms of the buffers offering along c and a. Given this setup, any potential action on the π-calculus side will result in a series of actual computational steps on the $\mathsf{SILL_S}$ side, affecting the buffers as prescribed by the protocol of type $\mathcal{U}_\mathsf{s}$. In such a closed-world setting, trying to exactly mimic potential actions seems unnatural, if not impossible.

However, it is still the case that we want to relate π-calculus behavior with $\mathsf{SILL_S}$ behavior in a precise sense. To reconcile the open-world view of a labelled transition semantics with the closed-world view of computational steps, we note that the encoding already accounts for this issue by essentially *implementing* "the environment" through the channel encodings that must be composed with the processes at the top-level. Thus, what we deem to be *observable* when we consider a configuration made up of encoded π-calculus processes and corresponding channel encodings are precisely the inputs and outputs to and from buffers. Conversely, any steps in a $\mathsf{SILL_S}$ configuration that do not involve any inputs or outputs to and from buffers, we deem to be *unobservable*.

▶ **Definition 3** (Unobservable Transitions of Configuration)**.** Given a configuration Ω we say that there is an unobservable transition from Ω to Ω', written $\Omega \longrightarrow_{un} \Omega'$, iff $\Omega \longrightarrow \Omega'$ where the transition does *not* involve any of the two reductions below:

$$\begin{aligned} \Omega_0,\, \mathsf{proc}(d_\mathsf{L},\, x_\mathsf{s} \leftarrow \mathsf{recv}\ c_\mathsf{L}\ ;\ P_{x_\mathsf{s}}) &\longrightarrow \Omega_0',\, \mathsf{proc}(d_\mathsf{L},\, [a_\mathsf{s}/x_\mathsf{s}]\, P_{x_\mathsf{s}}) \\ \Omega_0,\, \mathsf{proc}(d_\mathsf{L},\, \mathsf{send}\ c_\mathsf{L}\ e_\mathsf{s}\ ;\ P) &\longrightarrow \Omega_0',\, \mathsf{proc}(d_\mathsf{L},\, P) \end{aligned}$$

for some $\Omega_0, \Omega_0', \Gamma_\mathcal{F}, \Gamma_\mathcal{B}, \Gamma_\mathcal{I}, a_\mathsf{s}, c_\mathsf{s}, d_\mathsf{L}$ and e_s such that $a_\mathsf{s}, e_\mathsf{s}, c_\mathsf{s} \in \Gamma_\mathcal{F} \cup \Gamma_\mathcal{B}$, and $d_\mathsf{s} \in \Gamma_\mathcal{I}$. We write $\Omega \Longrightarrow_{un} \Omega'$ to stand for the reflexive transitive closure of $\longrightarrow_{un}$.

▶ **Definition 4** (Observable Transitions of Configuration)**.** Given a configuration Ω we define a notion of an observable transition $\Omega \xrightarrow{\alpha} \Omega'$, stating that configuration Ω performs action α and transitions to configuration Ω', with $\alpha ::= \overline{c}\langle a\rangle \mid c(a) \mid (\nu a)\overline{c}\langle a\rangle \mid \tau$ as follows:

- $\Omega \xrightarrow{\overline{c}\langle a\rangle} \Omega'$ if $c, a \in \Gamma_\mathcal{F}$, $\Omega = \Omega_1, \mathsf{proc}(d_\mathsf{L},\, \mathsf{send}\ c\ a; P), \Omega_2$, for some Ω_1, P, Ω_2 and $d_\mathsf{s} \in \Gamma_\mathcal{I}$ and $\Omega \longrightarrow \Omega'$ with $\Omega' = \Omega_1', \mathsf{proc}(d_\mathsf{L},\, P), \Omega_2'$, for some Ω_1', Ω_2'.
- $\Omega \xrightarrow{(\nu a)\overline{c}\langle a\rangle} \Omega'$ if $c \in \Gamma_\mathcal{F}$, $a \in \Gamma_\mathcal{B}$, $\Omega = \Omega_1, \mathsf{proc}(d_\mathsf{L},\, \mathsf{send}\ c\ a; P), \Omega_2$, for some Ω_1, P, Ω_2 and $d_\mathsf{s} \in \Gamma_\mathcal{I}$ and $\Omega \longrightarrow \Omega'$ with $\Omega' = \Omega_1', \mathsf{proc}(d_\mathsf{L},\, P), \Omega_2'$, for some Ω_1', Ω_2'.

- $\Omega \xrightarrow{c(a)} \Omega'$ if $c \in \Gamma_{\mathcal{F}}$, $a \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$, $\Omega = \Omega_1, \mathsf{proc}(d_{\mathsf{L}},\ x \leftarrow \mathsf{recv}\ c; P_x), \Omega_2$, for some Ω_1, P, Ω_2 and d, with $d_{\mathsf{S}} \in \Gamma_{\mathcal{I}}$ and $\Omega \longrightarrow \Omega'$ with $\Omega' = \Omega_1', \mathsf{proc}(d_{\mathsf{L}},\ [a/x]\, P_x), \Omega_2'$, for some Ω_1', Ω_2'.
- $\Omega \xrightarrow{\tau} \Omega'$ if all of the following:
  1. $c \in \Gamma_{\mathcal{B}}$, $a \in \Gamma_{\mathcal{F}} \cup \Gamma_{\mathcal{B}}$, $\Omega = \Omega_1, \mathsf{proc}(d_{\mathsf{L}},\ \mathsf{send}\ c\ a; P), \Omega_2$, for some Ω_1, P, Ω_2 and $d_{\mathsf{S}} \in \Gamma_{\mathcal{I}}$ and $\Omega \longrightarrow \Omega''$ with $\Omega'' = \Omega_1', \mathsf{proc}(d_{\mathsf{L}},\ P), \Omega_2'$, for some $\Omega_1', \Omega_2', \Omega''$;
  2. $\Omega'' \Longrightarrow_{un} \Omega'''$ with $\Omega''' = \Omega_1'', \mathsf{proc}(d_{\mathsf{L}},\ x \leftarrow \mathsf{recv}\ c; Q_x), \Omega_2''$, for some $\Omega_1'', Q, \Omega_2''$ and d, with $d_{\mathsf{S}} \in \Gamma_{\mathcal{I}}$ and $\Omega''' \longrightarrow \Omega'$ with $\Omega' = \Omega_1''', \mathsf{proc}(d_{\mathsf{L}},\ [a/x]\, Q_x), \Omega_2'''$, for some Ω_1''', Ω_2'''.

We write $\Omega \overset{\alpha}{\Longrightarrow} \Omega'$ for $\Omega \Longrightarrow_{un} \Omega'' \xrightarrow{\alpha} \Omega''' \Longrightarrow_{un} \Omega'$.

The several observable transitions mirror the π-calculus labelled transitions, but where the role of the environment is replaced with the respective channel implementations. The first three cases define, respectively, output of a free name, output of a bound name, and input of a name (using the techniques of Definition 1 to track names). To account for synchronizations (τ-actions) in the π-calculus, we model the three steps that are required to perform a full communication in the encoding: an output action of a free or bound name to a buffer, followed by some sequence of unobservable transitions (needed to complete the several intermediate stages of the encoding), and an input action from the same buffer. With the right definition of observable in place, we define the natural notion of (weak) bisimulation between a π-calculus process and a $\mathsf{SILL}_{\mathsf{S}}$ configuration.

▶ **Definition 5** (Weak Bisimulation)**.** A relation $\mathcal{R}$ between asynchronous π-calculus processes and $\mathsf{SILL}_{\mathsf{S}}$ configurations is a weak bisimulation if and only if, whenever $P \mathcal{R} \Omega$:

- If $P \xrightarrow{\alpha} P'$ and $\alpha \neq \tau$ then $\Omega \overset{\alpha}{\Longrightarrow} \Omega'$ and $P' \mathcal{R} \Omega'$
- If $P \xrightarrow{\tau} P'$ then $\Omega \Longrightarrow_{un} \Omega'$ or $\Omega \overset{\tau}{\Longrightarrow} \Omega'$ and $P' \mathcal{R} \Omega'$

plus the symmetric cases. We say that P is weakly bisimilar to Ω, written $P \approx \Omega$ iff there exists a weak bisimulation $\mathcal{R}$ such that $P \mathcal{R} \Omega$.

▶ **Theorem 6** (Observational Correspondence)**.** *Let P be an asynchronous π-calculus process. We have that $P \approx \Omega$, $\mathsf{proc}(\_, [\![P]\!])$, where Ω is a configuration made up of process encodings for the free names of P, with (non-empty)* arbitrary *contents.*

The expert reader may wonder how our use of a weak bisimulation captures asynchrony in the appropriate way, noting that a weak *asynchronous* bisimulation is necessary to accurately relate the asynchronous π-calculus and synchronous π-calculus with bags [5]. Would it then not be the case that we could use queues or stacks as buffers and replicate our bisimulation argument? Our argument holds *precisely* because of the non-deterministic (i.e., bag-like) nature of our buffer implementations. Otherwise, out-of-order message reception – a defining characteristic of asynchrony – would *not* be simulated correctly by our encoding. In this sense, our bisimulation is implicitly asynchronous by implementing the environment in terms of buffers that enforce non-deterministic removals.

4 Simulating Shared Session Types in the π-calculus

In this section, we close the loop and provide an encoding of $\mathsf{SILL}_{\mathsf{S}}$ process terms into the asynchronous *polyadic* π-calculus. The extension to the polyadic π-calculus is necessary to send along with the actual channel a fresh continuation channel that must be used for the next exchange in the protocol. This continuation-passing-style encoding (similar to that of Dardha et al. [13]) ensures that messages are received in the order specified by the protocol.

The resulting encoding is summarised in Figure 6. To simplify our encoding, we use a type-directed expansion of forwarding corresponding to the standard identity expansion in the sequent calculus. The resulting programs no longer use forwarding as a primitive, but implement it by processes that forward messages from client to provider and vice versa. Observational correctness of this expansion has been shown for the linear fragment [7] and with recursive types [17]. The strong logical underpinnings lead us to conjecture that observational correctness extends to sharing as well.

The general pattern of the encoding is to translate a positive type [35] to an output and a negative type [35] to an input with matching bindings. In case of a linear output or input, a fresh continuation channel is provided in addition to the actual channel to be sent or received, respectively. This channel is then used in the process continuation (in parallel) in place of the original channel, guaranteeing that the session discipline is not disturbed by out-of-order messages. To encode the acquire-release discipline of $\mathsf{SILL_S}$, we must preserve the shared mode of a channel throughout the translation. To this end, we indicate a linear $\mathsf{SILL_S}$ channel by a pair $\ulcorner x_\mathsf{L}, x_\mathsf{S} \urcorner$, where the left and right projections yield the linear mode x_L and shared mode x_S, respectively. A release then restores the session to the shared channel. To ensure a *blocking* semantics for an acquire, the encoding of an acquire and accept forces synchronization via the channel w. The encoding of choice makes use of a selection channel per choice, used to indicate the choice outcome and unlock the appropriate continuation. For simplicity, and without loss of generality, we limit the encoding to binary internal and external choice. Process definitions are encoded as top-level replicated processes:

For each $(x_\mathsf{L} \leftarrow p \leftarrow \overline{y_{\mathsf{L}_i}}, \overline{w_{\mathsf{S}_j}} = P_{x_\mathsf{L}, \overline{y_{\mathsf{L}_i}}, \overline{w_{\mathsf{S}_j}}}) \in \Sigma$:

$$!(p(\overline{y_{\mathsf{L}_i}}, \overline{y_{\mathsf{S}_i}}, \overline{w_{\mathsf{S}_j}}, z).\nu x_\mathsf{L}, x_\mathsf{S}\, (\overline{z}\langle x_\mathsf{L}, x_\mathsf{S}\rangle \mid [\![[\overline{\ulcorner y_{\mathsf{L}_i}, y_{\mathsf{S}_i} \urcorner}/\overline{y_{\mathsf{L}_i}}, \ulcorner x_\mathsf{L}, x_\mathsf{S} \urcorner / x_\mathsf{L}] P_{x_\mathsf{L}, \overline{y_{\mathsf{L}_i}}, \overline{w_{\mathsf{S}_j}}}]\!]))$$

For each $(x_\mathsf{S} \leftarrow p \leftarrow \overline{y_{\mathsf{S}_i}} = P_{x_\mathsf{S}, \overline{y_{\mathsf{S}_i}}}) \in \Sigma$: $!(p(\overline{y_{\mathsf{S}_i}}, z).\nu x_\mathsf{S}\, (\overline{z}\langle x_\mathsf{S}\rangle \mid [\![P_{x_\mathsf{S}, \overline{y_{\mathsf{S}_i}}}]\!]))$

The name of the definition is used as a channel that the encoding of the spawn construct uses to access new instances of the definition (generated via replication). The process receives the sessions that are needed to execute the definition and a channel z, used to send back the pair of (fresh) channels x_S and x_L used by the encoding of the definition body.

Operational Correspondence. To establish the operational correctness of our encoding, we consider an *asynchronous* semantics for $\mathsf{SILL_S}$. While operational completeness would not be affected by a synchronous semantics, soundness would require reasoning up-to observational equivalence. Since the expressiveness of $\mathsf{SILL_S}$ has been shown to be orthogonal to the choice of synchrony or asynchrony, we opt for the latter for the sake of simplicity. The semantics spawns single-message outputting processes using a continuation-passing style to achieve type-safe asynchrony [3].

Recalling that in $\mathsf{SILL_S}$ static entities are distinct from runtime entities, we lift the encoding to configurations, where the channels along which processes offer their session behavior are represented as bound names:

$$[\![\cdot]\!] = \mathbf{0} \qquad [\![\mathsf{proc}(c,\, P),\, \Omega]\!] = (\nu c_\mathsf{S}, c_\mathsf{L})([\![P]\!] \mid [\![\Omega]\!]) \qquad [\![\mathsf{unavail}(c_\mathsf{S}),\, \Omega]\!] = [\![\Omega]\!]$$

We can now show that $\mathsf{SILL_S}$ transitions are always matched by a synchronization in the π-calculus (and vice-versa) rather straightforwardly, given the direct nature of the encoding.

▶ **Theorem 7** (Operational Correspondence). *Let $\longrightarrow^+$ be the transitive closure of $\longrightarrow$:*

Completeness. *If P is a well-typed, forwarding-free $\mathsf{SILL_S}$ process and $\mathsf{proc}(a,\, P) \longrightarrow^+ \Omega$ then $[\![P]\!] \longrightarrow^+ [\![\Omega]\!]$.*

Soundness. *For all well-typed, forwarding-free $\mathsf{SILL_S}$ configurations Ω such that $[\![\Omega]\!] \longrightarrow^+ Q$, there exists a configuration Ω' such that $\Omega \longrightarrow^+ \Omega'$ and $Q \Longrightarrow [\![\Omega']\!]$.*

$$
\begin{array}{lcl}
[\![x_{\mathsf{L}} \leftarrow p \leftarrow \overline{\ulcorner y_{\mathsf{L}_i}, y_{\mathsf{S}_i} \urcorner}, \overline{w_{\mathsf{S}_j}};\ Q_{x_{\mathsf{L}}}]\!]^{\text{SPAWN}_{\mathrm{LL}}} & = & \nu z\,(\overline{p}\langle \overline{y_{\mathsf{L}_i}}, \overline{y_{\mathsf{S}_i}}, \overline{w_{\mathsf{S}_j}}, z\rangle \mid z(x_{\mathsf{L}}, x_{\mathsf{S}}).[\![[\ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}]\!]) \\
[\![x_{\mathsf{S}} \leftarrow p \leftarrow \overline{y_{\mathsf{S}_i}};\ Q_{x_{\mathsf{S}}}]\!]^{\text{SPAWN}_{\mathrm{LL/SS}}} & = & \nu z\,(\overline{p}\langle \overline{y_{\mathsf{S}_i}}, z\rangle \mid z(x_{\mathsf{S}}).[\![Q_{x_{\mathsf{S}}}]\!]) \\
[\![y_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}};\ Q_{y_{\mathsf{L}}}]\!]^{\uparrow^{\mathsf{S}}_{\mathrm{L}}} & = & \nu y_{\mathsf{L}}, w\,(\overline{x_{\mathsf{S}}}\langle y_{\mathsf{L}}, x_{\mathsf{S}}, w\rangle \mid w(\ ).[\![[\ulcorner y_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / y_{\mathsf{L}}]\, Q_{y_{\mathsf{L}}}]\!]) \\
[\![y_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}};\ P_{y_{\mathsf{L}}}]\!]^{\uparrow^{\mathsf{S}}_{\mathrm{R}}} & = & x_{\mathsf{S}}(y_{\mathsf{L}}, y_{\mathsf{S}}, w).(\overline{w}\langle\ \rangle \mid [\ulcorner y_{\mathsf{L}}, y_{\mathsf{S}} \urcorner / y_{\mathsf{L}}]\, [\![P_{y_{\mathsf{L}}}]\!]) \\
[\![y_{\mathsf{S}} \leftarrow \mathsf{release}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner;\ Q_{y_{\mathsf{S}}}]\!]^{\downarrow^{\mathsf{S}}_{\mathrm{L}}} & = & x_{\mathsf{L}}(y_{\mathsf{S}}).[\![Q_{y_{\mathsf{S}}}]\!] \\
[\![x_{\mathsf{S}} \leftarrow \mathsf{detach}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner;\ P]\!]^{\downarrow^{\mathsf{S}}_{\mathrm{R}}} & = & \overline{x_{\mathsf{L}}}\langle x_{\mathsf{S}}\rangle \mid [\![P]\!] \\
[\![\mathsf{wait}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner;\ Q]\!]^{\mathbf{1}_{\mathrm{L}}} & = & x_{\mathsf{L}}(\ ).[\![Q]\!] \\
[\![\mathsf{close}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\!]^{\mathbf{1}_{\mathrm{R}}} & = & \overline{x_{\mathsf{L}}}\langle\ \rangle \\
[\![y_{\mathsf{L}} \leftarrow \mathsf{recv}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner;\ P_{y_{\mathsf{L}}}]\!]^{\otimes_{\mathrm{L}}/\multimap_{\mathrm{R}}} & = & x_{\mathsf{L}}(y_{\mathsf{L}}, y_{\mathsf{S}}, z_{\mathsf{L}}, z_{\mathsf{S}}).[\![[\ulcorner z_{\mathsf{L}}, z_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner, \ulcorner y_{\mathsf{L}}, y_{\mathsf{S}} \urcorner / y_{\mathsf{L}}] P_{y_{\mathsf{L}}}]\!] \\
[\![\mathsf{send}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner\ \ulcorner y_{\mathsf{L}}, y_{\mathsf{S}} \urcorner;\ P]\!]^{\otimes_{\mathrm{R}}/\multimap_{\mathrm{L}}} & = & \nu z_{\mathsf{L}}\,(\overline{x_{\mathsf{L}}}\langle y_{\mathsf{L}}, y_{\mathsf{S}}, z_{\mathsf{L}}, x_{\mathsf{S}}\rangle \mid [\![[\ulcorner z_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, P]\!]) \\
[\![y_{\mathsf{S}} \leftarrow \mathsf{recv}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner;\ P_{y_{\mathsf{S}}}]\!]^{\exists_{\mathrm{L}}/\Pi_{\mathrm{R}}} & = & x_{\mathsf{L}}(y_{\mathsf{S}}, z_{\mathsf{L}}, z_{\mathsf{S}}).[\![[\ulcorner z_{\mathsf{L}}, z_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, P_{y_{\mathsf{S}}}]\!] \\
[\![\mathsf{send}\ \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner\ y_{\mathsf{S}};\ P]\!]^{\exists_{\mathrm{R}}/\Pi_{\mathrm{L}}} & = & \nu z_{\mathsf{L}}\,(\overline{x_{\mathsf{L}}}\langle y_{\mathsf{S}}, z_{\mathsf{L}}, x_{\mathsf{S}}\rangle \mid [\![[\ulcorner z_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, P]\!]) \\
[\![\ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner.\mathsf{case}(P, Q)]\!]^{\oplus_{\mathrm{L}}/\&_{\mathrm{R}}} & = & \nu y_{\mathsf{inl}}, y_{\mathsf{inr}}\,(\overline{x_{\mathsf{L}}}\langle y_{\mathsf{inl}}, y_{\mathsf{inr}}\rangle \mid \\
 & & \quad y_{\mathsf{inl}}(z_{\mathsf{L}}, z_{\mathsf{S}}).[\![[\ulcorner z_{\mathsf{L}}, z_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, P]\!] \mid \\
 & & \quad y_{\mathsf{inr}}(z_{\mathsf{L}}, z_{\mathsf{S}}).[\![[\ulcorner z_{\mathsf{L}}, z_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, Q]\!]) \\
[\![\ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner.\mathsf{inl}; P]\!]^{\oplus_{\mathrm{R}_1}/\&_{\mathrm{L}_1}} & = & \nu z_{\mathsf{L}}\,(x_{\mathsf{L}}(y_{\mathsf{inl}}, y_{\mathsf{inr}}).\overline{y_{\mathsf{inl}}}\langle z_{\mathsf{L}}, x_{\mathsf{S}}\rangle \mid [\![[\ulcorner z_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, P]\!]) \\
[\![\ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner.\mathsf{inr}; Q]\!]^{\oplus_{\mathrm{R}_2}/\&_{\mathrm{L}_2}} & = & \nu z_{\mathsf{L}}\,(x_{\mathsf{L}}(y_{\mathsf{inl}}, y_{\mathsf{inr}}).\overline{y_{\mathsf{inr}}}\langle z_{\mathsf{L}}, x_{\mathsf{S}}\rangle \mid [\![[\ulcorner z_{\mathsf{L}}, x_{\mathsf{S}} \urcorner / \ulcorner x_{\mathsf{L}}, x_{\mathsf{S}} \urcorner]\, Q]\!])
\end{array}
$$

Figure 6 Translation of $\mathsf{SILL}_{\mathsf{S}}$ process terms into the asynchronous, polyadic π-calculus.

5 Related Work

Encodings of Asynchrony. Encodability results are a standard benchmark for expressiveness of π-calculi [18]. For the asynchronous π-calculus [21], encodings into various formulations of synchronous π-calculi exist [5], as well as impossibility results [33] regarding the ability to adequately encode certain forms of choice in an asynchronous setting.

Our encoding of the asynchronous π-calculus is reminiscent of the encoding of the asynchronous π-calculus in a π-calculus with bags by Beauxis et al. [5], shown to be in tight correspondence via an asynchronous bisimilarity. Their framework considers buffers as primitives in the target calculus, whereas we encode the bag-like behavior of buffers explicitly as $\mathsf{SILL}_{\mathsf{S}}$ processes that adhere to a particularly typed protocol, making our encoding more primitive, but adding several administrative reductions to encoded processes due to the sharing discipline and the implementation of nondeterminism when reading from a buffer. This fact, combined with the restrictive (typed) usage of buffers in our setting allows us to reason using a weak bisimilarity rather than a more involved *asynchronous* bisimilarity. Beauxis et al. also consider an encoding of their calculus with bags in the asynchronous π-calculus. The general structure of the encoding is similar to our encoding of $\mathsf{SILL}_{\mathsf{S}}$ in the asynchronous π-calculus, modulo the richer syntax of $\mathsf{SILL}_{\mathsf{S}}$, which introduces more communication actions in the image of the encoding. We note that our encoding is greatly simplified by linearity and by the fact that $\mathsf{SILL}_{\mathsf{S}}$ does not employ *mixed* choice [31].

Linear Logic and Session Types. The propositions-as-types correspondence between linear logic and session types introduced by Caires and Pfenning [8, 9] initiated an ongoing line of research exploring the logical reading of sessions along various axes [42, 24, 34, 35, 2]. Starting with [8], which translates the linear session language into a π-calculus (which is more expressiveness than the source language), various works on encodings in this logical setting have been proposed [39, 41, 29, 28]. These study encodings between session-typed processes and functional languages, since the considered session languages are not powerful enough to express general π-calculus behaviors. Recent works [2, 12] attempt to address these limitations in expressiveness by allowing composed processes to share more than one linear channel, but still do not allow for the sharing available in SILL_S, crucial to our encoding. We also highlight the work of Dardha and Pérez [14] comparing session-typed processes arising from linear logic and those from the Kobayashi-style typings [26, 25, 32] for the π-calculus. They observe that the *degree of sharing* determines an expressiveness hierarchy for typed processes and develop encodings from the latter into the former (not preserving the degree of sharing). In this sense, our encoding of asynchronous π-calculus completely preserves the sharing of channels, at the cost of allowing deadlocks when acquiring shared channels.

Session-Typed Behavioral Theory. The behavioral theory of session-typed processes has been studied in both the multiparty [27] and the linear logic settings [7, 34, 1]. Our notion of observation is related to the observed communication semantics of Atkey [1], which must also address the challenge of observing actions within a "closed-world" framework. However, their system is based on classical linear logic and does not have sharing, making the precise relationship with our formulation of observable on shared names unclear.

Substructural Logical Reasoning. The work of Deng et al. [15] studies a natural notion of logical preorder between linear logic contexts using process calculi techniques such as simulation preorders. While the study of the relationship between contexts can be seen as a study of multiset rewriting of configurations, the process calculus induced by their reading of linear logic is a fairly different formalism from SILL_S. For instance, their labelled transition system cannot be reasonably used as a labelled transition system for SILL_S since it cannot represent the equivalent of channel passing, nor does it make use of the deep inspection of multiset rewriting terms needed for our semantics and reasoning.

6 Concluding Remarks

In this paper, we gave an encoding of the untyped asynchronous π-calculus into SILL_S via a universal shared session type $\mathcal{U}_\mathsf{s}$, proving its operational and observational correctness. This result shows that the full expressiveness of the untyped asynchronous π-calculus can be recovered in session-typed process calculi. We also provide an operationally correct encoding in the other direction to simulate shared session types in the π-calculus. Given their universality, session-typed calculi with manifest sharing become strong competitors over traditional approaches since they not only guarantee protocol compliance in the presence of non-determinism but also make sharing explicit in the type structure. For future work, we wish to investigate a general behavioral theory of manifest sharing, as well as study techniques to establish deadlock-freedom in the presence of shared channels.

References

1 Robert Atkey. Observed communication semantics for classical processes. In *European Symposium on Programming (ESOP)*, pages 56–82, 2017.

2 Robert Atkey, Sam Lindley, and J. Garrett Morris. Conflation confers concurrency. In S. Lindley et al., editor, *Wadler Festschrift*, pages 32–55. Springer LNCS 9600, 2016.

3 Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. *Proceedings of the ACM on Programming Languages (PACMPL)*, 1(ICFP):37:1–37:29, 2017.

4 Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. Technical Report CMU-CS-17-106, Carnegie Mellon University, March 2017.

5 Romain Beauxis, Catuscia Palamidessi, and Frank D. Valencia. On the asynchronous nature of the asynchronous π-calculus. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 473–492. Springer, 2008.

6 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *8th International Workshop on Computer Science Logic (CSL)*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 1994. An extended version appeared as Technical Report UCAM-CL-TR-352, University of Cambridge.

7 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *European Symposium on Programming (ESOP)*, pages 330–349. Springer, 2013.

8 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *21st International Conference on Concurrency Theory (CONCUR)*, pages 222–236. Springer, 2010.

9 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016.

10 Iliano Cervesato and Andre Scedrov. Relating state-based and process-based concurrency through linear logic. *Information and Computation*, 207(10):1044–1077, 2009.

11 Karl Crary, Robert Harper, and Sidd Puri. What is a recursive module? In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 50–63, 1999.

12 Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 91–109, 2018.

13 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *Principles and Practice of Declarative Programming (PPDP)*, pages 139–150, 2012.

14 Ornela Dardha and Jorge A. Pérez. Comparing deadlock-free session typed processes. In *EXPRESS/SOS*, pages 1–15, 2015.

15 Yuxin Deng, Robert J. Simmons, and Iliano Cervesato. Relating reasoning methodologies in linear logic and process algebra. *Mathematical Structure in Computer Science*, 26(5):868–906, 2016.

16 Simon J. Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2–3):191–225, 2005.

17 Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-typed concurrent contracts. In A. Ahmed, editor, *European Symposium on Programming (ESOP'18)*, pages 771–798, Thessaloniki, Greece, 2018. Springer LNCS 10801.

18 Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

19 Dennis Griffith and Frank Pfenning. SILL. `https://github.com/ISANobody/sill`, 2015.

20 Kohei Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory (CONCUR)*, pages 509–523. Springer, 1993.

21 Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *5th European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science, pages 133–147. Springer, 1991.

22 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *7th European Symposium on Programming (ESOP)*, pages 122–138. Springer, 1998.

23 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 273–284. ACM, 2008.

24 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 582–594, 2016.

25 Naoki Kobayashi. A type system for lock-free processes. *Inf. Comput.*, 177(2):122–159, 2002.

26 Naoki Kobayashi. A new type system for deadlock-free processes. In *International Conference on Concurrency Theory (CONCUR)*, pages 233–247, 2006.

27 Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. *Logical Methods in Computer Science*, 10(4), 2014.

28 Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In *European Symposium On Programming (ESOP)*, pages 560–584, 2015.

29 Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In *International Colloquium on Functional Progrmaming (ICFP)*, pages 434–447, 2016.

30 Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

31 Uwe Nestmann. What is a "good" encoding of guarded choice? *Inf. Comput.*, 156(1-2):287–319, 2000.

32 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In *Computer Science Logic – Logic in Computer Science (CSL-LICS)*, pages 72:1–72:10, 2014.

33 Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

34 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014.

35 Frank Pfenning and Dennis Griffith. Polarized substructural session types. In *18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 3–22. Springer, 2015.

36 Jason Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, January 2009. URL: `http://www.cs.cmu.edu/~jcreed/papers/jdml.pdf`.

37 Davide Sangiorgi and David Walker. *The π-Calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.

38 Bernardo Toninho. *A Logical Foundation for Session-based Concurrent Computation*. PhD thesis, Carnegie Mellon University and New University of Lisbon, 2015.

39 Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In *15th International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, pages 346–360. Springer, 2012.

40 Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: a monadic integration. In *22nd European Symposium on Programming (ESOP)*, pages 350–369. Springer, 2013.

41 Bernardo Toninho and Nobuko Yoshida. On polymorphic sessions and functions - A tale of two (fully abstract) encodings. In *European Symposium On Programming (ESOP)*, pages 827–855, 2018.

42 Philip Wadler. Propositions as sessions. In *17th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 273–286. ACM, 2012.

Verification of Immediate Observation Population Protocols

Javier Esparza[1]
Technische Universität München, Munich, Germany
esparza@in.tum.de
https://orcid.org/0000-0001-9862-4919

Pierre Ganty[2]
IMDEA Software Institute, Madrid, Spain
pierre.ganty@imdea.org
https://orcid.org/0000-0002-3625-6003

Rupak Majumdar[3]
MPI-SWS, Kaiserslautern, Germany
rupak@mpi-sws.org

Chana Weil-Kennedy[4]
Technische Universität München, Munich, Germany
chana.wk@gmail.com

Abstract

Population protocols (Angluin et al., *PODC*, 2004) are a formal model of sensor networks consisting of identical mobile devices. Two devices can interact and thereby change their states. Computations are infinite sequences of interactions satisfying a strong fairness constraint.

A population protocol is well-specified if for every initial configuration C of devices, and every computation starting at C, all devices eventually agree on a consensus value depending only on C. If a protocol is well-specified, then it is said to compute the predicate that assigns to each initial configuration its consensus value.

In a previous paper we have shown that the problem whether a given protocol is well-specified and the problem whether it computes a given predicate are decidable. However, in the same paper we prove that both problems are at least as hard as the reachability problem for Petri nets. Since all known algorithms for Petri net reachability have non-primitive recursive complexity, in this paper we restrict attention to immediate observation (IO) population protocols, a class introduced and studied in (Angluin et al., *PODC*, 2006). We show that both problems are solvable in exponential space for IO protocols. This is the first syntactically defined, interesting class of protocols for which an algorithm not requiring Petri net reachability is found.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Population protocols, Immediate Observation, Parametrized verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.31

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.06071`.

[1] Supported by ERC Advanced Grant (787367: PaVeS).
[2] Supported by Madrid Regional Government project S2013/ICE-2731, N-Greens Software – Next-GeneRation Energy-EfficieNt Secure Software, the Spanish Ministry of Economy and Competitiveness project No. TIN2015-71819-P, RISCO – RIgorous analysis of Sophisticated COncurrent and distributed systems, and by a Ramón y Cajal fellowship RYC-2016-20281.
[3] supported by the ERC Synergy award (IMPACT).
[4] Part of this work was done during a visit at the IMDEA Software Institute.

1 Introduction

Population protocols [2, 3] are a model of distributed, concurrent computation by anonymous, identical finite-state agents. They capture the essence of distributed computation in different areas. In particular, even though they were introduced to model networks of passively mobile sensors, they are also being studied in the context of natural computing [12, 7]. They also exhibit many common features with Petri nets, another fundamental model of concurrency.

A protocol has a finite set of states Q and a set of transitions of the form $(q, q') \mapsto (r, r')$, where $q, q', r, r' \in Q$. If two agents are in states, say, q_1 and q_2, and the protocol has a transition of the form $(q_1, q_2) \mapsto (q_3, q_4)$, then the agents can interact and simultaneously move to states q_3 and q_4. Since agents are anonymous and identical, the global state of a protocol is completely determined by the number of agents at each local state, called a *configuration*. A protocol *computes* a boolean value for a given initial configuration if in all fair executions starting at it, all agents eventually agree to this value[5] – so, intuitively, population protocols compute by reaching a stable consensus. Observe that a protocol may compute no value for some initial configuration, in which case it is deemed not *well-specified* [2].

Population protocols are parameterized systems. Every initial configuration yields a different finite-state instance of the protocol, and the specification is a global property of the infinite family of protocol instances so generated. More precisely, the specification is a predicate $P(x)$ stipulating the boolean value $P(C)$ that the protocol must compute from the initial configuration C.

Initial verification efforts for verifying population protocols studied the problem of checking if $P(x)$ is correctly computed for a *finite* set of initial configurations, a task within the reach of finite-state model checkers. In 2015 we obtained the first positive result on parameterized verification [9]. We showed that the problem of deciding if a given protocol is well-specified for all initial configurations is decidable. The same result holds for the correctness problem: given a protocol and a predicate, deciding if the protocol is well-specified and computes the predicate. Unfortunately, we also showed [9, 10] that both problems are as hard as the reachability problem for Petri nets. Since all known algorithms for Petri net reachability run in non-primitive recursive time in the worst case, the applicability of this result is limited.

In this paper we initiate the investigation of subclasses of protocols with a more tractable well specification and correctness problems. We focus on the subclass of *immediate observation* protocols (IO protocols), introduced and studied by Angluin et al. [4]. These are protocols whose transitions have the form $(q_1, q_2) \mapsto (q_1, q_3)$. Intuitively, in an IO protocol an agent can change its state from q_2 to q_3 by *observing* that another agent is in state q_1. This yields an elegant model of protocols in which agents interact through *sensing*: If an agent in state q_2 senses the presence of another agent in state q_1, then it can change its state to q_3. The other agent typically does not even know that it has been sensed, and so it keeps its current state. They also capture the notion of catalysts in chemical reaction networks.

Angluin et al. focused on the expressive power of IO protocols. Our main result is that for IO protocols, both the well specification and correctness problems can be solved in EXPSPACE (we also show the problem is PSPACE-hard). This is the first time that the verification problems of a substantial class of protocols are proved to be solvable in elementary time. To ensure elementary time, our proof uses techniques significantly different from previous results

[5] An execution is fair if it is finite and cannot be extended, or it is infinite and satisfies the following condition: if C appears infinitely often in the execution, then every step enabled at C is taken infinitely often in the execution.

[9]. The key to our result is the use of *counting constraints* to symbolically represent possibly infinite (but not necessarily upward-closed) sets of configurations. A counting constraint is a boolean combination of atomic threshold constraints of the form $x_i \geq k$. We prove that, contrary to the case of arbitrary protocols, the set of configurations reachable from a counting set (the set of solutions of a counting constraint) is again a counting set and we characterize the complexity of representing this set. We believe that this result can be of independent interest for other parameterized systems.

Angluin et al. [4] proved that IO protocols compute exactly the predicates represented by counting constraints. Our main theorem yields a new proof of this result as a corollary. But it also goes further. Using our complexity results, we can provide a lower bound on the state complexity of IO protocols, i.e., on the number of states necessary to compute a given predicate. These results complement recent bounds obtained for arbitrary protocols [5].

2 Immediate Observation Population Protocols

2.1 Preliminaries

A *multiset* on a finite set E is a mapping $C\colon E \to \mathbb{N}$, thus, for any $e \in E$, $C(e)$ denotes the number of occurrences of element e in C. Operations on $\mathbb{N}$ like addition, subtraction, or comparison, are extended to multisets by defining them component wise on each element of E. Given $e \in E$, we denote by $\boldsymbol{e}$ the multiset consisting of one occurrence of element e, that is, the multiset satisfying $\boldsymbol{e}(e) = 1$ and $\boldsymbol{e}(e') = 0$ for every $e' \neq e$. Given $E' \subseteq E$ define $C(E') \stackrel{\text{def}}{=} \sum_{e \in E'} C(e)$. Given a total order $e_1 \prec e_2 \prec \cdots \prec e_n$ on E, a multiset C can be equivalently represented by the vector $(C(e_1), \ldots, C(e_n)) \in \mathbb{N}^n$.

2.2 Protocol Schemes

A *protocol scheme* $\mathcal{A} = (Q, \Delta)$ consists of a finite non-empty set Q of states and a set $\Delta \subseteq Q^4$. If $(q_1, q_2, q_1', q_2') \in \Delta$, we write $(q_1, q_2) \mapsto (q_1', q_2')$ and call it a *transition*.

Confugurations of a protocol scheme $\mathcal{A}$ are given by *populations*. A population P is a multiset on Q with at least two elements, i.e., $P(Q) \geq 2$. The set of all populations is denoted $\mathrm{Pop}(Q)$. Intuitively, a configuration $C \in \mathrm{Pop}(Q)$ describes a collection of identical finite-state *agents* with Q as set of states, containing $C(q)$ agents in state q.

Pairs of agents *interact* using transitions from Δ. Formally, given two configurations C and C' and a transition $\delta = (q_1, q_2) \mapsto (q_1', q_2')$, we write $C \xrightarrow{\delta} C'$ if

$$C \geq (\boldsymbol{q}_1 + \boldsymbol{q}_2) \text{ holds, and } C' = C - (\boldsymbol{q}_1 + \boldsymbol{q}_2) + (\boldsymbol{q}_1' + \boldsymbol{q}_2') \ .$$

(Recall that $\boldsymbol{q}$ is the multiset consisting only of one occurrence of q.) From the definition of interaction, it is easily seen that, inside the tuple $(q_1, q_2, q_1', q_2') \in \Delta$, the ordering between q_1 and q_2 and between q_1' and q_2' is irrelevant. We write $C \xrightarrow{w} C'$ for a sequence $w = \delta_1 \ldots \delta_k$ of transitions if there exists a sequence $C_0, \ldots, C_k$ of configurations satisfying $C = C_0 \xrightarrow{\delta_1} C_1 \cdots \xrightarrow{\delta_k} C_k = C'$. We also write $C \to C'$ if $C \xrightarrow{\delta} C'$ for some transition $\delta \in \Delta$, and call $C \to C'$ an *interaction*. We say that C' is *reachable from* C if $C \xrightarrow{w} C'$ for some (possibly empty) sequence w of transitions.

Note that transitions are enabled only when there are at least two agents. This is why we assume that populations have at least two elements.

An *execution* of $\mathcal{A}$ is a finite or infinite sequence of configurations $C_0, C_1, \ldots$ such that $C_i \to C_{i+1}$ for each $i \geq 0$. An execution $C_0, C_1, \ldots$ is *fair* if it is finite and cannot be extended, or it is infinite and for every step $C \to C'$, if $C_i = C$ for infinitely many indices

$i \geq 0$, then $C_j = C$ and $C_{j+1} = C'$ for infinitely many indices $j \geq 0$ [2, 3]. Informally, if C appears infinitely often in a fair execution, then every step enabled at C is taken infinitely often in the execution.

Given a set S of configurations and a transition t of a protocol scheme (Q, Δ), we define:

- $post[t](S) \stackrel{\text{def}}{=} \{C' \mid C \xrightarrow{t} C' \text{ for some } C \in S\}$ and $post(S) \stackrel{\text{def}}{=} \bigcup_{t \in \Delta} post[t](S)$.
- $post^0(S) \stackrel{\text{def}}{=} S$; $post^{i+1}(S) \stackrel{\text{def}}{=} post(post^i(S))$ for every $i \geq 0$; and $post^*(S) \stackrel{\text{def}}{=} \bigcup_{i \geq 0} post^i(S)$.

We also define $pre[t](S) \stackrel{\text{def}}{=} \{C' \mid C' \xrightarrow{t} C \text{ for some } C \in S\}$. The sets $pre(S)$ and $pre^*(S)$ are defined as above for $post$.

2.2.1 Immediate Observation Protocol Schemes

A protocol scheme is *immediate observation* (IO) if all its transitions are immediate observation. A transition $(q_1, q_2) \mapsto (q_1', q_2')$ is immediate observation iff $\{q_1, q_2\} \cap \{q_1', q_2'\} \neq \emptyset$. Consider, for instance, a transition (q_s, q_o, q_d, q_o) where q_s, q_o and q_d are all distinct. Observe that the transition is immediate observation since $\{q_s, q_o\} \cap \{q_d, q_o\} = \{q_o\} \neq \emptyset$. Intuitively, in an interaction specified by an immediate observation transition, one agent observes the state of another and updates it own state, but the observed agent remains as it was (and its state, unmodified by the interaction, is given by $\{q_1, q_2\} \cap \{q_1', q_2'\}$). Other typical examples of immediate observation transitions are (q_o, q_o, q_d, q_o), (q_s, q_o, q_o, q_o) (q_s, q_o, q_s, q_o) and (q_o, q_o, q_o, q_o) where q_s, q_o and q_d are all distinct. Note that in the last two cases, the state of two agents are the same before and after interacting.

2.3 Population Protocols

As Angluin et al. [2], we consider population protocols as a computational model, computing predicates $\Pi\colon \mathrm{Pop}(\Sigma) \to \{0, 1\}$, where Σ is a non-empty, finite set of *input variables*.

An *input mapping* for a protocol scheme $\mathcal{A}$ is a function $I\colon \mathrm{Pop}(\Sigma) \to \mathrm{Pop}(Q)$ that maps each input population $X \in \mathrm{Pop}(\Sigma)$ to a configuration of $\mathcal{A}$. The set of *initial configurations* is $\mathcal{I} = \{I(X) \mid X \in \mathrm{Pop}(\Sigma)\}$. An input mapping I is *Presburger* if the set of pairs $(X, C) \in \mathrm{Pop}(\Sigma) \times \mathrm{Pop}(Q)$ such that $C = I(X)$ is definable in Presburger arithmetic. An input mapping I is *simple* if there is an injective map $\nu\colon \Sigma \to Q$ such that $I(X) = \sum_{\sigma \in \Sigma} X(\sigma)\boldsymbol{\nu(\sigma)}$. That is, each input variable is assigned a (distinct) state, and a population X over Σ is assigned the initial configuration consisting of $X(\sigma)$ agents in the state $\nu(\sigma)$ and no other agents. Unless otherwise specified, we restrict our attention to the class of *simple* input mappings.

An *output mapping* for a protocol scheme is a function $O\colon Q \to \{0, 1\}$ that associates to each state q of $\mathcal{A}$ an output value in $\{0, 1\}$. The output mapping induces the following properties on configurations: a configuration C is a

- *b-consensus* for $b \in \{0, 1\}$ if $\sum_{p \in O^{-1}(1-b)} C(p) = 0$ and a *consensus* if it is a b-consensus for some b;
- *dissensus* if it is a b-consensus for no b (that is C is a dissensus if $\sum_{p \in O^{-1}(b)} C(p) > 0$ and $\sum_{p \in O^{-1}(1-b)} C(p) > 0$).

A *population protocol* is a triple $(\mathcal{A}, I, O)$, where $\mathcal{A}$ is a protocol scheme, I is a simple input mapping, and O is an output mapping. The population protocol is *immediate observation* (IO) if $\mathcal{A}$ is immediate observation.

An execution $C_0, C_1, \ldots$ *stabilizes to* b for a given $b \in \{0, 1\}$ if there exists $n \in \mathbb{N}$ such that C_m is a b-consensus for every $m \geq n$ (if the execution is finite, then this means for every m between n and the length of the execution). Notice that there may be many different

executions from a given configuration C_0, each of which may stabilize to 0 or to 1 or not stabilize at all (by visiting infinitely many dissensus or infinitely many 0 and 1 consensus).

A population protocol $(\mathcal{A}, I, O)$ is *well-specified* if for every input configuration $C_0 \in \mathcal{I}$, every fair execution of $\mathcal{A}$ starting at C_0 stabilizes to the same value $b \in \{0, 1\}$. Otherwise, it is *ill-specified*. The *well specification problem* asks if a given population protocol is well-specified?

Finally, a population protocol $(\mathcal{A}, I, O)$ *computes* a predicate $\Pi\colon \mathrm{Pop}(\Sigma) \to \{0, 1\}$ if for every $X \in \mathrm{Pop}(\Sigma)$, every fair execution of $\mathcal{A}$ starting at $I(X)$ stabilizes to $\Pi(X)$. It follows easily from the definitions that a protocol computes a predicate iff it is well-specified. The *correctness* problem asks, given a population protocol and a predicate whether the protocol computes the predicate.

3 Counting Constraints and Counting Sets

▶ **Definition 1.** Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and let $x \in X$. A constraint of the form $l \leq x$, where $l \in \mathbb{N}$, is a *lower bound*, and a constraint of the form $x \leq u$, where $u \in \mathbb{N} \cup \{\infty\}$, is an *upper bound*. A *literal* is a lower bound or an upper bound.

A *counting constraint* is a boolean combination of literals. A counting constraint is in *counting normal form* (CoNF) if it is a disjunction of conjunctions of literals, where each conjunction, called a *counting minterm*, contains exactly two literals for each variable, one of them an upper bound and the other a lower bound. We often write a counting constraint in CoNF as the set of its counting minterms.

The semantics of a counting constraint is a *counting set*, a set of vectors in $\mathbb{N}^n$ or, equivalently, a set of valuations to the variables in X. The semantics is defined inductively on the structure of a counting constraint, as expected. Define $[\![l \leq x]\!] = \{x \mapsto m \in \mathbb{N} \mid m \geq l\}$ ($[\![\infty \leq x]\!] = \emptyset$) and $[\![x \leq u]\!] = \{x \mapsto m \in \mathbb{N} \mid m \leq u\}$. Disjunction, conjunction, and negation of counting constraints translates into union, intersection, and complement of counting sets.

The following proposition follows easily from the definition of counting sets and the disjunctive normal form for propositional logic.

▶ **Proposition 2.**
1. *Counting sets are closed under Boolean operations.*
2. *Every counting constraint is equivalent to a counting constraint in CoNF.*

Proof Sketch. **1.** Proof is easy. **2.** Put the constraint in disjunctive normal form. Remove negations in front of literals using $[\![\neg(x_i \leq c)]\!] = [\![x_i \geq c + 1]\!]$ if $c \in \mathbb{N}$ and remove the enclosing minterm otherwise; and $[\![\neg(x_i \geq c)]\!] = [\![x_i \leq c - 1]\!]$ if $c \in \mathbb{N} \setminus \{0\}$ and remove the enclosing minterm otherwise. Remove minterms containing unsatisfiable literals $l \leq x_i \wedge x_i \leq u$ with $l > u$. Remove redundant bounds, e.g., replace $(l_1 \leq x \wedge l_2 \leq x)$ by $\max\{l_1, l_2\} \leq x$. If a minterm does not contain a lower bound (upper bound) for x_i, add $0 \leq x_i$ ($x_i \leq \infty$). ◀

Next, we introduce a representation of CoNF-constraints used in the rest of the paper.

▶ **Definition 3** (Representation of CoNF-constraints)**.** We represent a counting minterm by a pair $M \stackrel{\text{def}}{=} (L, U)$ where $L\colon X \to \mathbb{N}$ and $U\colon X \to \mathbb{N} \cup \{\infty\}$ assign to each variable its lower and upper bound, respectively. We represent a CoNF-constraint Γ as the set of representations of its minterms: $\Gamma = \{M_1, \ldots, M_m\}$.

▶ **Definition 4** (Measures of counting constraints)**.** The *L-norm* of a counting minterm $M = (L, U)$ is $\|M\|_l \stackrel{\text{def}}{=} \sum_{x \in X} L(x)$, and its *$U$-norm* is $\|M\|_u \stackrel{\text{def}}{=} \sum_{\substack{x \in X \\ U(x) < \infty}} U(x)$ (and 0 if

$U(x) < \infty$ for no x). The L- and U-norms of a CoNF-constraint $\Gamma = \{M_1, \dots, M_m\}$ are $\|\Gamma\|_l \stackrel{\text{def}}{=} \max_{i\in[1,m]}\{\|M_i\|_l\}$ and $\|\Gamma\|_u \stackrel{\text{def}}{=} \max_{i\in[1,m]}\{\|M_i\|_u\}$.

▶ **Proposition 5.** *Let Γ_1, Γ_2 be CoNF-constraints over n variables.*

- *There exists a CoNF-constraint Γ with $[\![\Gamma]\!] = [\![\Gamma_1]\!] \cup [\![\Gamma_2]\!]$ such that $\|\Gamma\|_u \le \max\{\|\Gamma_1\|_u, \|\Gamma_2\|_u\}$ and $\|\Gamma\|_l \le \max\{\|\Gamma_1\|_l, \|\Gamma_2\|_l\}$.*
- *There exists a CoNF-constraint Γ with $[\![\Gamma]\!] = [\![\Gamma_1]\!] \cap [\![\Gamma_2]\!]$ such that $\|\Gamma\|_u \le \|\Gamma_1\|_u + \|\Gamma_2\|_u$ and $\|\Gamma\|_l \le \|\Gamma_1\|_l + \|\Gamma_2\|_l$.*
- *There exists a CoNF-constraint Γ with $[\![\Gamma]\!] = \mathbb{N}^n \setminus [\![\Gamma_1]\!]$ such that $\|\Gamma\|_u \le n\|\Gamma_1\|_l$ and $\|\Gamma\|_l \le n\|\Gamma_1\|_u + n$.*

Proof. Remember that a CoNF constraint for m minterms in dimension n is a m-disjunction of n-conjunctions, and that the L-norm (respectively U-norm) is the maximum sum of lower (resp. upper) bounds in one conjunction. The union of two counting sets Γ_1, Γ_2 with CoNF constraints is represented by the disjunction of the two constraints, and it is still CoNF so the result follows. The intersection is represented by a conjunction of the two constraints and so is not CoNF and needs to be rearranged as in Proposition 2. The new n-conjunctions of literals (i.e. the new minterms) mix unmodified bounds from Γ_1 and Γ_2, so the result follows. The complement is represented by the negation of the original constraint, which we rearrange into CoNF using $\neg(l \le x \le u) \equiv (0 \le x \le l-1) \vee (u+1 \le x \le \infty)$. We obtain n-conjunctions with lower bounds of the form $u+1$, with $u \le \|\Gamma_1\|_u$ an upper bound in a minterm of the original constraint. This yields $\|\Gamma\|_l \le n\|\Gamma_1\|_u + n$ and the reasoning is similar for the U-norm. ◀

▶ Remark 6. The counting sets contain the finite, upward-closed and downward-closed sets:

- Every finite subset of $\mathbb{N}^n$ is a counting set. Indeed, $\{(k_1, \dots, k_n)\} = [\![(L, U)]\!]$ with $L(x_i) = k_i = U(x_i)$ for every $x_i \in X$, and so finite sets are counting sets too.
- A set $S \subseteq \mathbb{N}^n$ is upward-closed if whenever $v \in S$ and $v \le_\times v'$, we have $v' \in S$, where we write $v \le_\times v'$ if the ordering holds pointwise (meaning $v(x) \le v'(x)$ for every $x \in X$). Upward-closed sets are counting sets. Indeed, by Dickson's lemma, every upward-closed set has a finite set $\{v_1, \dots, v_k\}$ of minimal elements with respect to $\le_\times$, and so the set is $[\![\{(L_1, U), \dots, (L_k, U)\}]\!]$ where $L_i(x_j) = v_i(j)$ and $U(x_j) = \infty$ for every $1 \le j \le n$.
- A set $S \subseteq \mathbb{N}^n$ is downward-closed if whenever $v \in S$ and $v' \le_\times v$, we have $v' \in S$. Since a set is downward-closed iff its complement is upward-closed, every downward-closed set is a counting set. Further, it is easy to see that downward-closed sets are represented by counting constraints $\{(L, U_1), \dots, (L, U_k)\}$ where $L(x_j) = 0$ for every $1 \le j \le n$.

Next, we define a well-quasi-ordering on counting sets. For two counting minterms M_1 and M_2, we write $M_1 \preceq M_2$ if $[\![M_1]\!] \supseteq [\![M_2]\!]$. For CoNF-constraints Γ_1 and Γ_2, define the ordering $\Gamma_1 \sqsubseteq \Gamma_2$ if for each counting minterm $M_2 \in \Gamma_2$ there is a counting minterm $M_1 \in \Gamma_1$ such that $M_1 \preceq M_2$. Note that $\Gamma_1 \sqsubseteq \Gamma_2$ implies $[\![\Gamma_1]\!] \supseteq [\![\Gamma_2]\!]$.

▶ **Theorem 7.** *For every $u \ge 0$, the ordering $\sqsubseteq$ on counting sets represented by CoNF-constraints of U-norm at most u is a well-quasi-order.*

Proof. We first prove that counting minterms with $\preceq$ form a better quasi order. For two counting minterms M_1 and M_2, we write $M_1 \preceq M_2$ if $[\![M_1]\!] \supseteq [\![M_2]\!]$. Let $\mathcal{M} = M_1, M_2, \dots$ be an infinite sequence of counting minterms of U-norm at most u, where $M_i = (L_i, U_i)$. Since there are only finitely many mappings $U\colon X \to \mathbb{N} \cup \{\infty\}$ of norm at most u, the sequence $\mathcal{M}$ contains an infinite subsequence $\mathcal{M}'$ such that every minterm M_i of $\mathcal{M}'$ satisfies $U_i = U$ for some mapping U. So $\mathcal{M}'$ is of the form $(L_1, U), (L_2, U) \dots$ By Dickson's lemma, there

are $i < j$ such that $L_i \leq_\times L_j$, and so $[\![(L_i, U)]\!] \supseteq [\![(L_j, U)]\!]$. Hence, defining C be the set of all counting minterms of U-norm at most u we find that $(C, \preceq)$ is a well-quasi-order. In fact, standard arguments show that this is a better-quasi-order [1]. Hence, the ordering $\sqsubseteq$ is a better quasi order on counting constraints [1], implying it is also a well-quasi-order. ◀

4 Reachability Sets of IO Population Protocols

We show that if S is a counting set, then $post^*(S)$ and $pre^*(S)$ are also counting sets. First we show that we can restrict ourselves to IO protocols in a certain normal form.

4.1 A Normal Form for Immediate Observation Protocols

An IO protocol is in *normal form* if $q_s \neq q_o$ for every transition $(q_s, q_o) \mapsto (q_o, q_d)$, i.e., the state of the observed agent is different from the source state of the observer.

Given an IO population protocol $\mathcal{P} = (\mathcal{A}, I, O)$ we define an IO protocol in normal form $\mathcal{P}' = (\mathcal{A}', I', O')$ which is well-specified iff $\mathcal{P}$ is well-specified. Further, the number of states and transitions of $\mathcal{P}'$ is linear in the number of states and transitions of $\mathcal{P}$. The mapping I' is a Presburger mapping even if I is simple, but this does not affect our results.

$\mathcal{P}'$ is defined adding transition and states to $\mathcal{P}$. First we add a state r. Then, we replace each transition $t = (q, q) \mapsto (q, q_d)$ of $\mathcal{P}$ by a transition $t' = (q', q) \mapsto (q', q_d)$, where q' is a primed copy of q, and add two further transitions $(q, r) \mapsto (r, q')$ and $(q', r) \mapsto (r, q)$.

It remains to define the output function of the new states as well as the input mapping I' of $\mathcal{P}'$. We define I' to be a Presburger initial mapping which coincides with I on the state of $\mathcal{P}$ and such that $I(X)(r) = 1$ for all X and $I(X)(q') = 0$ for all X and primed state q'. The output of primed copies is the same as their unprimed version, that is $O(q') = O(q)$. The only technical difficulty is the definition of the output of state r. Because of the way in which we have defined the transitions involving r, the agent initially in state r cannot leave r. Therefore, whatever the output $O(r)$ we assign to r, the protocol $\mathcal{P}'$ can never reach consensus $1 - O(r)$, and so $\mathcal{P}'$ may not be well-specified even if $\mathcal{P}$ is. To solve this problem, we add a primed copy r' of r such that r and r' have distinct outputs. Every transition with r as observer is duplicated but this time with r' as observed state. Finally, for every state q of $\mathcal{P}$, if $O(q) = O(r')$ we add the transition $(q, r) \mapsto (q, r')$, and otherwise we add the transition $(q, r') \mapsto (q, r)$. After adding these states, the agent initially in r switches between r and r', and finally stabilizes to the same value the other agents stabilize to.

4.2 The Functions pre^* and $post^*$ Preserve Counting Sets

We show that if S is a counting set, then $post^*(S)$ and $pre^*(S)$ are also counting sets. Further, given a CoNF-constraint Γ representing S, we show how to construct a CoNF-constraint representing $post^*(S)$ and $pre^*(S)$. In the following, we abbreviate $post([\![\Gamma]\!])$ to $post(\Gamma)$, and similarly for other notations involving $post$ and pre, like $post[t](\Gamma)$, $post^*(\Gamma)$, etc.

We start with some simple examples. First, we observe that the result does not hold for arbitrary population protocols. Consider the protocol with four distinct states $\{q_1, q_2, q_3, q_4\}$ and one single transition $(q_1, q_2) \mapsto (q_3, q_4)$. Let $M = [\![0 \leq x_3 \leq 0 \wedge 0 \leq x_4 \leq 0]\!]$. Then $post^*(M) = [\![x_3 = x_4]\!]$, which is not a counting set. Intuitively, the reason is that the transitions links the number of agents in states x_3 and x_4. However, this is only possible because the transition is not IO. Indeed, consider now the protocol $\mathcal{P}_1$ with states $\{q_1, q_2, q_3\}$ and one single IO transition $(q_1, q_2) \mapsto (q_1, q_3)$. Table 1 lists some typical constraints for M, and gives constraints for $post^*(M)$.

Table 1 The set $post^*[t](M)$ for two IO transitions and counting minterm M. For conciseness and clarity we use equality constraints instead of two inequalities.

M	$\|M\|_l$	$\|M\|_u$	$\Gamma \stackrel{\text{def}}{=} post^*[t](M)$ where $t \stackrel{\text{def}}{=} (q_1, q_2) \mapsto (q_1, q_3)$	$\|\Gamma\|_l$	$\|\Gamma\|_u$
$x_1 = 0 \wedge x_2 \geq 2 \wedge x_3 = 1$	3	1	$x_1 = 0 \wedge x_2 \geq 2 \wedge x_3 = 1$	3	1
$x_1 = 1 \wedge x_2 = 2 \wedge x_3 \geq 1$	4	3	$(x_1 = 1 \wedge x_2 = 2 \wedge x_3 \geq 1)$ $\vee (x_1 = 1 \wedge x_2 = 1 \wedge x_3 \geq 2)$ $\vee (x_1 = 1 \wedge x_2 = 0 \wedge x_3 \geq 3)$	4	3
$x_1 = 1 \wedge x_2 \geq 1 \wedge x_3 = 2$	4	3	$(x_1 = 1 \wedge x_2 \geq 1 \wedge x_3 = 2)$ $\vee (x_1 = 1 \wedge x_2 \geq 0 \wedge x_3 \geq 3)$	4	3
$x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_3 \geq 2$	3	0	$(x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_3 \geq 2)$ $\vee (x_1 \geq 1 \wedge x_2 \geq 0 \wedge x_3 \geq 3)$	4	0
M	$\|M\|_l$	$\|M\|_u$	$\Gamma \stackrel{\text{def}}{=} post^*[t](M)$ where $t \stackrel{\text{def}}{=} (q_1, q_2) \mapsto (q_2, q_2)$	$\|\Gamma\|_l$	$\|\Gamma\|_u$
$x_1 \geq 1 \wedge x_2 = 0$	1	0	$x_1 \geq 1 \wedge x_2 = 0$	1	0
$x_1 = 1 \wedge x_2 \geq 2$	3	1	$(x_1 = 1 \wedge x_2 \geq 2) \vee (x_1 = 0 \wedge x_2 \geq 3)$	3	1
$x_1 \geq 2 \wedge x_2 = 1$	3	1	$(x_1 \geq 2 \wedge x_2 \geq 1) \vee (x_1 \geq 1 \wedge x_2 \geq 2)$ $\vee (x_1 \geq 0 \wedge x_2 \geq 3)$	3	0

Given a minterm (L, U), we syntactically define a CoNF-constraint $(L, U)_{t^*}$ for the set:

$$post^*[t](L, U) \stackrel{\text{def}}{=} \{C' \mid \exists k \geq 0 \exists C \in [\![(L, U)]\!] \text{ such that } C \xrightarrow{t^k} C'\} \ .$$

That is, $(L, U)_{t^*}$ captures the set of all configurations that can be obtained from (L, U) by firing transition t an arbitrary number of times.

▶ **Definition 8.** Let (L, U) be a minterm and let $t = (q_s, q_o) \mapsto (q_d, q_o)$ be an IO transition. Define $(L, U)_{t^*}$ to be the set given by (L, U) and all the minterms (L', U') such that all the following conditions hold:

1. $[\![(L'', U)]\!] \neq \emptyset$ where $[\![L'']\!] = [\![L]\!] \cap [\![x_s \geq 1 \wedge x_o \geq 1]\!]$.
2. $U'(x) = U(x)$ and $L'(x) = L''(x)$ for every $x \in X \setminus \{x_s, x_d\}$.
3. If $U(x_s) < \infty$, then there exists $1 \leq k \leq U(x_s)$ such that $U'(x_s) = U(x_s) - k$, $L'(x_s) = \max\{0, L''(x_s) - k\}$, $U'(x_d) = U(x_d) + k$ and $L'(x_d) = L''(x_d) + k$.
4. If $U(x_s) = \infty$, then $U'(x_s) = U'(x_d) = \infty$ and there exists $1 \leq k \leq L''(x_s)$ such that $L'(x_s) = L''(x_s) - k$ and $L'(x_d) = L''(x_d) + k$.

Given a CoNF-constraint $\Gamma = \{M_1, \ldots, M_m\}$, we define $\Gamma_{t^*} = \bigcup_{i=1}^{m} M_{it^*}$.

▶ **Lemma 9.** *Let $\mathcal{P}$ be an IO protocol and let Γ be a CoNF-constraint. Then $\Gamma_{t^*} = post^*[t](\Gamma)$. Further, $\|\Gamma_{t^*}\|_u \leq \|\Gamma\|_u$.*

Proof. It suffices to prove that for every minterm (L, U) and for every transition t we have $post^*[t](L, U) = (L, U)_{t^*}$ and $\|(L, U)_{t^*}\|_u \leq \|(L, U)\|_u$. The rest follows easily from the definitions of $post^*$ and of a counting constraint.

Condition (1) holds iff some vector in $[\![(L, U)]\!]$ enables t, hence $[\![(L'', U)]\!]$ is the set $[\![(L, U)]\!]$ of vectors minus those disabling t. If no vector enables t then $(L, U)_{t^*}$ is the singleton $\{(L, U)\}$. Condition (2) states that the number of agents in states other than q_s and q_d does not change. Condition (3–4) defines the result of firing t one or more times.

The inequality $\|(L, U)_{t^*}\|_u \leq \|(L, U)\|_u$ follows immediately from (1–4). Observe that $\|(L, U)_{t^*}\|_u < \|(L, U)\|_u$ may hold if $U(x_s) = \infty$ and $U(x_d) < \infty$. ◀

To prove the main theorem of the section, we introduce the following definition.

▶ **Definition 10.** Given a protocol $\mathcal{P}$, let S be a set of configurations and let Γ be a CoNF-constraint.

- Define: $post_a(S) \stackrel{\text{def}}{=} \bigcup_{t\in\Delta} post^*[t](S)$; $post_a^0(S) \stackrel{\text{def}}{=} S$ and $post_a^{i+1}(S) \stackrel{\text{def}}{=} post_a(post_a^i(S))$ for every $i \geq 0$; $post_a^*(S) \stackrel{\text{def}}{=} \bigcup_{i\geq 0} post_a^i(S)$.
- Similarly, define in the constraint domain: $post_a(\Gamma) \stackrel{\text{def}}{=} \bigcup_{t\in\Delta} \Gamma_{t^*}$; $post_a^0(\Gamma) \stackrel{\text{def}}{=} \Gamma$ and $post_a^{i+1}(\Gamma) \stackrel{\text{def}}{=} post_a(post_a^i(\Gamma))$ for every $i \geq 0$.

The a-subscript stands for "accelerated." Observe that we cannot define $post_a^*(\Gamma)$ directly as the infinite union $\bigcup_{i\geq 0} post_a^i(\Gamma)$ because constraints are only closed under finite unions.

▶ **Theorem 11.** *Let $\mathcal{P}$ be an IO protocol and let S be a counting set. Then both $post^*(S)$ and $pre^*(S)$ are counting sets.*

Proof. We first prove that $post^*(S)$ is a counting set. It follows from Definition 10 that $post^i(S) \subseteq post_a^i(S)$ but $post_a^i(S) \subseteq post^*(S)$ for every $i \geq 0$, hence $post_a^*(S) = post^*(S)$, and so it suffices to prove that $post_a^*(S)$ is a counting set.

Let Γ be a CoNF-constraint such that $[\![\Gamma]\!] = S$. By Lemma 9, $post_a^i(\Gamma)$ is a counting set and $\|post_a^i(\Gamma)\|_u \leq \|\Gamma\|_u$ for every $i \geq 0$. By Theorem 7, there exist indices $i < j$ such that $post_a^j(\Gamma) \subseteq post_a^i(\Gamma)$, hence $post_a^j(\Gamma) = post_a^i(\Gamma)$ since $\Gamma' \subseteq post_a(\Gamma')$ for all Γ', and finally $post_a^*(\Gamma) = \bigcup_{k=1}^{j} post_a^k(\Gamma)$. Since counting sets are closed under finite union, $post_a^*(S)$ is a counting set.

Finally we show that $pre^*(S)$ is also a counting set. Consider the protocol $\mathcal{P}_r$ obtained by "reversing" the transitions of $\mathcal{P}$, i.e., $\mathcal{P}_r$ has a transition $(q_1, q_2) \mapsto (q_3, q_4)$ iff $\mathcal{P}$ has a transition $(q_3, q_4) \mapsto (q_1, q_2)$. Then $pre^*(S)$ in $\mathcal{P}$ is equal to $post^*(S)$ in $\mathcal{P}_r$. ◀

4.3 Bounding the Size of $post^*(\Gamma)$

Given a CoNF-constraint Γ, we obtain an upper bound on the size of a CoNF-constraint denoting $post^*(\Gamma)$ and $pre^*(\Gamma)$. More precisely, we obtain bounds on the L-norm and U-norm of a constraint for $post^*(\Gamma)$ as a function of the same parameters for Γ.

We first recall a theorem of Rackoff [14] recast in the terminology of population protocols.

▶ **Theorem 12** ([14, 6])**.** *Let $\mathcal{P}$ be a population protocol with set of states Q and let C be a configuration of $\mathcal{P}$. For every configuration C', if there exists C'' such that $C' \xrightarrow{*} C'' \geq_\times C$, then there exists σ and C''' such that $C' \xrightarrow{\sigma} C''' \geq_\times C$ and $|\sigma| \leq (3 + C(Q))^{(3|Q|)!+1} \in C(Q)^{2^{\mathcal{O}(|Q|\log|Q|)}}$. (Recall that $C(Q) \stackrel{def}{=} \sum_{q\in Q} C(q)$ and $C(Q) \geq 2$.)*

Observe that the bound on the length of σ depends only on C and $\mathcal{P}$, but not on C'. Using this theorem we can already obtain an upper bounds for $pre^*(\Gamma)$ when $[\![\Gamma]\!]$ is upward-closed. The bound is valid for arbitrary population protocols.

Recall that if $[\![\Gamma]\!]$ is upward-closed we can assume $\|\Gamma\|_u = 0$ (see Remark 6).

▶ **Proposition 13.** *Let $\mathcal{P}$ be population protocol with n states. Let S be an upward-closed set of configurations and let Γ be a CoNF-constraint with $\|\Gamma\|_u = 0$ such that $[\![\Gamma]\!] = S$. There exists a CoNF constraint Γ' such that $[\![\Gamma']\!] = pre^*(\Gamma)$ and $\|\Gamma'\|_u = 0$, $\|\Gamma'\|_l \in (\|\Gamma\|_l)^{2^{\mathcal{O}(n\log n)}}$.*

Proof. It is well known that if S is upward-closed, then so is $pre^*(S)$. (This follows from Lemma 9, but is also an easy consequence of the fact that $C \xrightarrow{*} C'$ implies $C + C'' \xrightarrow{*} C' + C''$ for every C''). Let $K \stackrel{\text{def}}{=} (3 + \|\Gamma\|_l)^{(3n)!+1}$. By Theorem 12, for every configuration C, if $C \in pre^*(S)$ then $C \in \bigcup_{i=0}^{K} pre^i(S)$, and so $pre^*(S) = \bigcup_{i=0}^{K} pre^i(S) = pre_a^K(S)$. Let $\Gamma' = pre_a^K(\Gamma)$. Then $[\![\Gamma']\!] = pre^*(S)$. Further, we have $\|\Gamma'\|_u = 0$ by Lemma 9 (the Lemma proves the result for $post^*$, but exactly the same proof works for pre^* by reversal of

transitions). To prove the bound for the L-norm, observe that by the definition of $(L,U)_{t^*}$ we have $\|(L,U)_{t^*}\|_l \leq \|(L,U)\|_l + 1$, as we are always in case 4. of Definition 8 (because S is upward-closed). Since $pre_a(\Gamma) = \bigcup_{t\in\Delta_r} \Gamma_{t^*}$ and the L-norm of a union is the maximum of the L-norms, we get $\|pre_a(\Gamma)\|_l \leq \|\Gamma\|_l + 1$. By induction, $\|pre_a^K(\Gamma)\|_l \leq \|\Gamma\|_l + K$, and the result follows. ◀

In the rest of the section we obtain a bound valid not only for upward-closed sets, but for arbitrary counting sets. The price to pay is a restriction to IO protocols. We start with some miscellaneous notations that will be useful.

- Given a mapping $f\colon X \to \mathbb{N}$ and $Y \subseteq X$ we write $f(Y)$ for $\sum_{x\in Y} f(x)$, and $f|_Y$ for the projection of f onto Y.
- Given a transition sequence σ, we denote by $c(\sigma)$ the "compression" of σ as the shortest regular expression $r = t_1^* \dots t_m^*$ such that $\sigma \in L(r)$, and denote $|c(\sigma)| = m$. While σ induces a sequence of $pre[t]$ or $post[t]$, $c(\sigma)$ induces a sequence of $pre^*[t]$ or $post^*[t]$.

For the rest of the section we fix an IO protocol $\mathcal{P}$ with a set of states Q and $|Q| = n$. We say that C *covers* C' if $C \geq_\times C'$. We introduce a relativization.

▶ **Definition 14.** Let $E \subseteq Q$. A configuration C *E-covers* C', denoted $C \geq_E C'$, if $C(q) = C'(q)$ for every $q \in E$ and $C(q) \geq C'(q)$ for every $q \in Q \setminus E$. $\mathcal{P}$ is *E-increasing* if for every transition $(q_s, q_o) \mapsto (q_d, q_o)$ either $q_s \notin E$ or $q_d \in E$.

Observe that $\mathcal{P}$ is vacuously $\emptyset$-increasing and Q-increasing. Intuitively, if $\mathcal{P}$ is E-increasing then the total number of agents in the states of E cannot decrease. Indeed, for that we would need a transition that removes agents from E without replacing them, i.e., a transition such that $q_s \in E$ and $q_d \notin E$. So, by induction, we have:

▶ **Lemma 15.** *If $\mathcal{P}$ is E-increasing and $C' \xrightarrow{*} C$ then $C'(E) \leq C(E)$.*

Now we give a result bounding the length of E-covering sequences for E-increasing protocols.

▶ **Lemma 16.** *Let $\mathcal{P} = (Q, \Delta)$ be an IO protocol scheme, let C be a configuration of $\mathcal{P}$, and let $E \subseteq Q$ such that $\mathcal{P}$ is E-increasing. For every configuration C', if there exists C'' such that $C' \xrightarrow{*} C'' \geq_E C$, then there exists σ and C''' such that $C' \xrightarrow{\sigma} C''' \geq_E C$ and $|\sigma| \in C(Q)^{2^{\mathcal{O}(n \log n)}}$, where the constant in the Landau symbol is independent of $\mathcal{P}$ and C.*

Proof. We use a theorem of Bozzelli and Ganty [6] that generalizes Rackoff's theorem to Vector Addition Systems with States (VASS). Recall that a d-VASS is a pair (P, Δ) where P is a set of control points and $\Delta \subseteq P \times \mathbb{Z}^d \times P$ is a finite set of transitions. The number d is called the dimension. A configuration of a d-VASS is a pair (p, v), where $p \in P$ and $v \in \mathbb{N}^d$. Intuitively, the VASS acts on d counters that can only take non-negative values. Formally, we have $(p, v) \to (p', v')$ if there is a transition (p, v'', p') such that $v + v'' = v'$, i.e., the machine moves from p to p' by updating the counters with v''. Given two configurations (p, v) and (p', v'), we write $(p, v) \geq_\times (p', v')$ if $p = p'$ and $v \geq_\times v'$. It is shown [6] in Theorem 1 that given a d-VASS (P, Δ) and a configuration C, for each configuration C', if there exists C'' such that $C' \xrightarrow{*} C'' \geq_\times C$, then there exists σ and C''' such that $C' \xrightarrow{\sigma} C''' \geq_\times C$ and $|\sigma| \leq |P| \cdot (\|\Delta\|_1 + \|C\|_1 + 2)^{(3d)!+1}$, where $\|\Delta\|_1$ and $\|C\|_1$ denote the maximal components of Δ and C, respectively.

Let $n = |Q|$. We construct a VASS $V_{\mathcal{P},E}$ that simulates the protocol $\mathcal{P}$, and then apply Bozzelli and Ganty's theorem. We do not give all the formal details of the construction.

Intuitively, given a configuration C of $\mathcal{P}$, we split it into $(C|_E, C|_{Q\setminus E})$. Since $\mathcal{P}$ is E-increasing, every configuration $(C'|_E, C'|_{Q\setminus E})$ from which we can reach $(C|_E, C|_{Q\setminus E})$ satisfies $C'|_E(E) \leq C|_E(E)$ (Lemma 15), and so there are only finitely many (at most $(C(E)+1)^n$) possibilities for $C'|_E$. The control points of the VASS $V_{\mathcal{P},E}$ correspond to these finitely many possibilities. Formally, the set of control points of $V_{\mathcal{P},E}$ is the set of all mappings $M\colon E \to \mathbb{N}$ such that $M(E) \leq C(E)$, plus some auxiliary control points (see below). The dimension, or number of counters, is $|Q \setminus E|$. The transitions of $V_{\mathcal{P},E}$ simulate the transitions of $\mathcal{P}$. For example, assume $t = (q_o, q_s) \mapsto (q_o, q_d)$ is a transition of $\mathcal{P}$ such that $q_s, q_o \notin E$ and $q_d \in E$. Then for every control point M of $V_{\mathcal{P},E}$ the VASS has a transition t_1 leading from M to an auxiliary control point $\langle M, t\rangle$, and a transition t_2 leading from $\langle M, t\rangle$ to the control point M' given by $M'(q_d) = M(q_d) + 1$ and $M'(q) = M(q)$ for every other $q \in E$. Transition t_1 decrements the counter of q_s and q_o by 1, leaving all other counters untouched, and transition t_2 increments the counters q_o, leaving all other counters untouched.

It follows that there is an execution $C' \xrightarrow{*} C'' \geq_E C$ in $\mathcal{P}$ iff there is an execution $(C'|_E, C'|_{Q\setminus E}) \xrightarrow{*} (C''|_E, C''|_{Q\setminus E}) \geq_\times (C|_E, C|_{Q\setminus E})$ in $V_{\mathcal{P},E}$ of at most twice the length.

Applying Bozzelli and Ganty's theorem, we obtain that the length of σ is bounded by $|P| \cdot (\|\hat{\Delta}\|_1 + \|C\|_1 + 2)^{(3d)!+1}$, where $|P|$, $\hat{\Delta}$, and d are now the set of control points, transitions, and dimension of $V_{\mathcal{P},E}$. We have $|P| \leq (C(E)+1)^n + |\Delta|(C(E)+1)^n$, $d = |Q \setminus E| \leq n$, $\|\hat{\Delta}\|_1 = 2$. Further, we have $\|C\|_1 \leq C(Q \setminus E)$, which leads to a bound of $(1+|\Delta|)(C(E)+1)^n \cdot (C(Q\setminus E)+4)^{(3n)!+1} \in C(Q)^{2^{\mathcal{O}(n \log n)}}$. ◀

Next we prove a double exponential bound on the length of E-covering sequences. The result is similar to Lemma 16 with two important changes: the restriction to E-increasing protocols is dropped, and we consider the bound on the length of $c(\sigma)$ instead of σ.

▶ **Theorem 17.** *Let $\mathcal{P}$ be an IO protocol with a set Q of n states, and let C be a configuration of $\mathcal{P}$. For every $E \subseteq Q$ and for every configuration C_0, if there exists τ and C' such that $C_0 \xrightarrow{\tau} C' \geq_E C$, then there exists σ and C'' such that $C_0 \xrightarrow{\sigma} C'' \geq_E C$ and $|c(\sigma)| \in C(Q)^{2^{\mathcal{O}(n^2 \log n)}}$, where the constant in the Landau symbol is independent of $\mathcal{P}$, C, and C_0.*

Proof. We prove by induction on $|E|$ that the result holds with $|c(\sigma)| \in C(Q)^{2^{e\mathcal{O}(n \log n)}}$, where $e \stackrel{\text{def}}{=} \max\{1, |E|\}$, and then apply $e \leq n$.

Base: $|E| = 0$. Then $\mathcal{P}$ is vacuously E-increasing, and the result follows from Lemma 16.

Step: $|E| > 0$. We use the following notation: Given a transition sequence ρ, we denote $\mathcal{P}_\rho$ the restriction of $\mathcal{P}$ to the transitions that occur in ρ.

If $\mathcal{P}_\tau$ is E-increasing, then we can apply Lemma 16, and we are done. Else, the definition of E-increasing shows there exist C_1 and C_2 and a decomposition $\tau = \tau_1\, t\, \tau_2$ such that

$$C_0 \xrightarrow{\tau_1} C_1 \xrightarrow{t} C_2 \xrightarrow{\tau_2} C' \geq_E C \ .$$

The protocol $\mathcal{P}_{\tau_2}$ is E-increasing, but $\mathcal{P}_{t\tau_2}$ is not E-increasing (observe that possibly $\tau_2 = \epsilon$). By Lemma 16 applied to $\mathcal{P}_{\tau_2}$, there exists σ_2 and $\tilde{C}''$ such that

$$C_0 \xrightarrow{\tau_1} C_1 \xrightarrow{t} C_2 \xrightarrow{\sigma_2} \tilde{C}'' \geq_E C \quad \text{and} \quad |\sigma_2| \in C(Q)^{2^{\mathcal{O}(n \log n)}} \ .$$

Since σ_2 can remove at most $|\sigma_2|$ agents from a state, there exist C_1', C_2', C'' such that

$$C_0 \xrightarrow{\tau_1} C_1 \geq_E C_1' \xrightarrow{t} C_2' \xrightarrow{\sigma_2} C'' \geq_E C \quad \text{and } C_1'(Q) \in C(Q)^{2^{\mathcal{O}(n \log n)}} \ .$$

Indeed, it suffices to define

- $C_1'(q) = \min\{C_1(q), |\sigma_2| + C(q)\}$ for every $q \in Q \setminus E$ and $C_1'(q) = C_1(q)$ for every $q \in E$,

- $C_2'(q) = \min\{C_2(q), |\sigma_2| + C(q)\}$ for every $q \in Q \setminus (E \cup \{q_d\})$, $C_2'(q) = C_2(q)$ for every $q \in E$ and $C_2'(q_d) = \min\{C_2(q_d), 1 + |\sigma_2| + C(q)\}$ where $t = (q_o, q_s) \mapsto (q_o, q_d)$.

Recall that $\mathcal{P}_{t\tau_2}$ is not E-increasing, and so $t = (q_o, q_s) \mapsto (q_o, q_d)$ for states q_s, q_d such that $q_s \in E$ and $q_d \notin E$. (Intuitively, the occurrence of t "removes agents" from E.) Let $E' \stackrel{\text{def}}{=} E \setminus \{q_s\}$. Since $C_0 \xrightarrow{\tau_1} C_1 \geq_E C_1'$, we also have $C_0 \xrightarrow{\tau_1} C_1 \geq_{E'} C_1'$. By induction hypothesis, there exists σ_1 and C_1'' such that $C_0 \xrightarrow{\sigma_1} C_1'' \geq_{E'} C_1'$ and

$$|c(\sigma_1)| \in C_1'(Q)^{2^{e'\mathcal{O}(n\log n)}} \in \left(C(Q)^{2^{\mathcal{O}(n\log n)}}\right)^{2^{e'\mathcal{O}(n\log n)}} \in C(Q)^{2^{\mathcal{O}(n\log n)}\cdot 2^{e'\mathcal{O}(n\log n)}}$$
$$\in C(Q)^{2^{\mathcal{O}(n\log n)+e'\mathcal{O}(n\log n)}} \in C(Q)^{2^{e\mathcal{O}(n\log n)}} .$$

(Observe that $C_1'' \geq_{E'} C_1'$ holds, but $C_1'' \geq_E C_1'$ may not hold, we may have $C_1''(q_s) > C_1'(q_s)$.) To sum up, we have configurations C_1', C_1'', C_2', C'' and transition sequences σ_1, σ_2 such that

$$C_0 \xrightarrow{\sigma_1} C_1'' \geq_{E'} C_1' \xrightarrow{t} C_2' \xrightarrow{\sigma_2} C'' \geq_E C \quad \text{and} \quad |c(\sigma_1\, t\, \sigma_2)| \in C(Q)^{2^{e\mathcal{O}(n\log n)}} .$$

Claim. There exist C_2'' and C''' such that

$$C_0 \xrightarrow{\sigma_1} C_1'' \xrightarrow{t^{C_1''(q_s)-C_1'(q_s)+1}} C_2'' \xrightarrow{\sigma_2} C''' \geq_E C .$$

Proof of the claim. Since $C_1'' \geq_{E'} C_1'$ and C_1' enables t, so does C_1''. Since $\mathcal{P}$ is an IO protocol (a hypothesis we had not used so far), C_1'' enables not only t, but also the sequence $t^{C_1''(q_s)-C_1'(q_s)+1}$. So there indeed exists a configuration C_2'' such that

$$C_0 \xrightarrow{\sigma_1} C_1'' \xrightarrow{t^{C_1''(q_s)-C_1'(q_s)+1}} C_2'' .$$

It remains to prove that $C_2'' \xrightarrow{\sigma_2} C''' \geq_E C$ holds for some configuration C'''. First we show $C_2'' \geq_E C_2'$, which amounts to proving $C_2'' \geq_{E'} C_2'$ and $C_2''(q_s) = C_2'(q_s)$.

The first part, i.e., $C_2'' \geq_{E'} C_2'$, follows from: $C_1'' \xrightarrow{t^{C_1''(q_s)-C_1'(q_s)+1}} C_2''$, $C_1'' \geq_{E'} C_1'$, $C_1' \xrightarrow{t} C_2'$, $q_d \notin E$, which implies $q_d \notin E'$, and the fact that t move agents from q_s to q_d (thus increasing their number in q_d). The second part, $C_2''(q_s) = C_2'(q_s)$, is proved by

$$C_2''(q_s) = C_1''(q_s) - (C_1''(q_s) - C_1'(q_s) + 1) = C_1'(q_s) - 1 = C_2'(q_s) .$$

So indeed we have $C_2'' \geq_E C_2'$. Now, since C_2' enables σ_2 and $C_2'' \geq_E C_2'$, the configuration C_2'' enables σ_2 too. So there exists a configuration C''' such that $C_2'' \xrightarrow{\sigma_2} C'''$. Further,

$$\begin{array}{ccc} C_1'' & \xrightarrow{t^{C_1''(q_s)-C_1'(q_s)+1}} & C_2'' \xrightarrow{\sigma_2} C''' \\ \geq_{E'} & & \geq_E \end{array} \qquad \begin{array}{ccc} C_1'' & \xrightarrow{t^{C_1''(q_s)-C_1'(q_s)+1}} & C_2'' \xrightarrow{\sigma_2} C''' \\ \geq_{E'} & & \geq_E \quad \geq_E \end{array}$$

since $C_1' \xrightarrow{t} C_2' \xrightarrow{\sigma_2} C'' \geq_E C$ holds, we have $C_1' \xrightarrow{t} C_2' \xrightarrow{\sigma_2} C'' \geq_E C$. So $C''' \geq_E C'' \geq_E C$, and the claim is proved. ◀

By the claim we have $C_0 \xrightarrow{\sigma_1\, t^{C_1''(q_s)-C_1'(q_s)+1}\, \sigma_2} C''' \geq_E C$. Let $\sigma = \sigma_1 t^{C_1''(q_s)-C_1'(q_s)+1}\sigma_2$. While $C_1''(q_s) - C_1'(q_s)$ can be arbitrarily large, we have $c(\sigma) = c(\sigma_1\, t\, \sigma_2)$, and so we conclude $C_0 \xrightarrow{\sigma} C''' \geq_E C \quad \text{and} \quad |c(\sigma)| \in C(Q)^{2^{e\mathcal{O}(n\log n)}}$. ◀

Theorem 17 allows to derive the promised bounds on a constraint for $pre^*(\Gamma)$ and $post^*(\Gamma)$.

▶ **Theorem 18.** *Let $\mathcal{P}$ be an IO population protocol with n states, and let Γ be a CoNF-constraint. There exists a CoNF-constraint Γ' satisfying $[\![\Gamma']\!] = pre^*(\Gamma)$, $\|\Gamma'\|_u \leq \|\Gamma\|_u$ and $\|\Gamma'\|_l \in \|\Gamma\|_u \left(\|\Gamma\|_l + \|\Gamma\|_u\right)^{2^{\mathcal{O}(n^2\log n)}}$. Further, Γ' can be constructed in $(2 + \|\Gamma\|_u)^n \cdot \|\Gamma\|_u \left(\|\Gamma\|_l + \|\Gamma\|_u\right)^{2^{\mathcal{O}(n^2\log n)}}$ time and space. Further, the same holds for $post^*(\Gamma)$.*

Proof. The bound on $\|\Gamma'\|_u$ follows from Lemma 9. The bound on $\|\Gamma'\|_l$ is proved in a similar way to Proposition 13, but using Theorem 17 instead of Theorem 12. Let (L,U) be a counting minterm in Γ. We define the set of states $E_{(L,U)} = \{q_i \mid U(x_i) < \infty\}$ and $\mathcal{C}^{\min}_{(L,U)} = \{C \mid \forall q_i \in Q \backslash E_{(L,U)}, L(x_i) \leq C(q_i) \leq U(x_i) \text{ and } \forall q_i \in E_{(L,U)}, C(q_i) = L(x_i)\}$ the configurations of (L,U) minimal over $Q \backslash E_{(L,U)}$. Notice that a configuration is in (L,U) if and only if it covers a configuration in $\mathcal{C}^{\min}_{(L,U)}$. By applying Theorem 17 to every $C \in \mathcal{C}^{\min}_{(L,U)}$ and to $E_{(L,U)}$, we get $pre^*(L,U) = \bigcup_{i=0}^{K} pre_a^i(L,U)$ for K the bound in Theorem 17 but with $\left(\sum_{q_i \in Q \backslash E} L(x_i) + \sum_{q_i \in E} U(x_i)\right)$ instead of $C(Q)$. Now since Γ is the union of such minterms (L,U), and by definition of the L and U-norms, $pre^*(\Gamma) = \bigcup_{i=0}^{K} pre_a^i(\Gamma)$ for $K \in (\|\Gamma\|_l + \|\Gamma\|_u)^{2^{\mathcal{O}(n^2 \log n)}}$. By Definition 8, we have $\|(L,U)_{t^*}\|_l \leq \|(L,U)\|_l + (\|(L,U)\|_u - 1)$. Using $\|\Gamma_{t^*}\|_u \leq \|\Gamma\|_u$, we reason by induction and get $\|pre_a^i(\Gamma)\|_l \leq \|\Gamma\|_l + i(\|\Gamma\|_u - 1)$ for all i, and the result on the L-norm follows.

The algorithm needs linear time and space in the number of minterms of Γ'. An upper bound on the number of minterms (L,U) is computed as follows. Since $\|\Gamma'\|_l \in \|\Gamma\|_u (\|\Gamma\|_l + \|\Gamma\|_u)^{2^{\mathcal{O}(n^2 \log n)}}$, there are at most $(1 + \|\Gamma'\|_l)^n \in \|\Gamma\|_u (\|\Gamma\|_l + \|\Gamma\|_u)^{2^{\mathcal{O}(n^2 \log n)}}$ possibilities for L, and since $\|\Gamma'\|_u \leq \|\Gamma\|_u$ at most $(2 + \|\Gamma\|_u)^n$ possibilities for U. ◀

The following result characterizes the size of counting constraints.

▶ **Corollary 19.** *Let $\mathcal{P}$ be an IO protocol with n states. Given $c \geq 2, d \geq 1$, let $\mathcal{G}(c,d)$ be the class of CoNF-constraints Γ such that $\|\Gamma\|_l, \|\Gamma\|_u \leq c^{2^{d \cdot (n^2 \log n)}}$. There exists a constant k that does not depend on n or $\mathcal{P}$ such that :*

1. *for every $\Gamma_1, \Gamma_2 \in \mathcal{G}(c,d)$, there exists $\Gamma \in \mathcal{G}(c,d)$ such that $[\![\Gamma]\!] = [\![\Gamma_1]\!] \cup [\![\Gamma_2]\!]$.*
2. *for every $\Gamma_1, \Gamma_2 \in \mathcal{G}(c,d)$, there exists $\Gamma \in \mathcal{G}(c,d+1)$ such that $[\![\Gamma]\!] = [\![\Gamma_1]\!] \cap [\![\Gamma_2]\!]$.*
3. *for every $\Gamma_1 \in \mathcal{G}(c,d)$, there exists $\Gamma \in \mathcal{G}(c,d+1)$ such that $[\![\Gamma]\!] = \mathbb{N}^n \setminus [\![\Gamma_1]\!]$.*
4. *for every $\Gamma_1 \in \mathcal{G}(c,d)$, there exists $\Gamma \in \mathcal{G}(c,d+k+2)$ such that $[\![\Gamma]\!] = pre^*([\![\Gamma_1]\!])$.*
5. *for every $\Gamma_1 \in \mathcal{G}(c,d)$, there exists $\Gamma \in \mathcal{G}(c,d+k+2)$ such that $[\![\Gamma]\!] = post^*([\![\Gamma_1]\!])$.*

The first three bounds follow from Prop 5. For the last two, the constant k is the one from the Landau symbol in Theorem 18.

5 An Algorithm for Deciding Well Specification

We show that the well-specification and correctness problems can be solved in exponential space for IO protocols, improving on the result for general protocols stating that they are at least as hard as the reachability problem for Petri nets [9]. We first introduce some notions.

▶ **Definition 20.** Given a population protocol $\mathcal{P}$, a configuration C is a *stable b-consensus* if C is a b-consensus and so is C' for every C' reachable from C. Let $\mathcal{C}_b$ and $\mathcal{ST}_b$ denote the sets of b-consensus and stable b-consensus configurations of $\mathcal{P}$. Observe that $\mathcal{ST}_b = \overline{pre^*(\mathcal{C}_b)}$.

Next, we characterize the well-specified protocols starting with the following lemma.

▶ **Lemma 21.** *Let $\mathcal{P}$ be a population protocol, let $C_0, C_1, C_2, \ldots$ be a fair execution of $\mathcal{P}$, and let S be a set of configurations. If S is reachable from C_i for infinitely many indices $i \geq 0$, then $C_j \in S$ for infinitely many indices $j \geq 0$.*

Proof. Let n be the number of states of $\mathcal{P}$ and let m be the number of agents of C_0. Then there are at most $K \stackrel{\text{def}}{=} (m+1)^n$ configurations reachable from C_0. So for infinitely many indices $i \geq 0$ we have $C_i \in \cup_{i \leq K} pre^i(S)$. We proceed by induction on K. If $K = 0$, then

$C_i \in S$ and we are done. If $K > 0$, then by fairness there exist infinitely many indices $j \geq 0$ such that $C_j \in \cup_{i \leq K-1} pre^i(S)$, and we conclude by induction hypothesis. ◀

▶ **Proposition 22.** *A population protocol $\mathcal{P}$ is well-specified iff the following hold:*
1. $post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$ *(or, equivalently,* $post^*(\mathcal{I}) \cap \overline{pre^*(\mathcal{ST}_0)} \cap \overline{pre^*(\mathcal{ST}_1)} = \emptyset$ *);*
2. $pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$.

Proof. We start with $\mathcal{ST}_b$ which is defined (Definition 20) as the set of configurations C such that C is a b-consensus and so is C' for every C' reachable from C.

By definition, $\mathcal{P}$ is *well-specified* if for every input configuration $C_0 \in \mathcal{I}$, every fair execution of $\mathcal{P}$ starting at C_0 stabilizes to the same value $b \in \{0, 1\}$. Equivalently, $\mathcal{P}$ is *well-specified* if every input configuration $C_0 \in \mathcal{I}$ satisfies the following two conditions:
(a) every fair execution starting at C_0 stabilizes to some value; and
(b) no two fair executions starting at C_0 stabilize to different values (i.e., to 0 and to 1).
We claim that **(a)** is equivalent to:

$$\text{for every } C \in post^*(\mathcal{I}) \text{ there exists } C' \text{ such that } C \xrightarrow{*} C' \text{ and } C' \in \mathcal{ST}_0 \cup \mathcal{ST}_1. \quad \textbf{(A)}$$

Assume **(a)** holds, and let $C \in post^*(\mathcal{I})$. Then $C_0 \xrightarrow{*} C$ for some $C_0 \in \mathcal{I}$. Extend $C_0 \xrightarrow{*} C$ to a fair execution. By **(a)**, the execution stabilizes to some value b. So $\mathcal{ST}_b$ is reachable from every configuration of the execution. By Lemma 21, the execution reaches a configuration $C' \in \mathcal{ST}_b$. For the other direction, assume **(A)** holds, and consider a fair execution starting at $C_0 \in \mathcal{I}$. By Lemma 21, the execution reaches a configuration of $\mathcal{ST}_b$ for $b \in \{0, 1\}$. By the definition of $\mathcal{ST}_b$, all successor configurations also belong to $\mathcal{ST}_b$, and so the execution stabilizes to b. Now we claim that **(b)** is equivalent to:

$$\text{no configuration } C \in post^*(\mathcal{I}) \text{ can reach both } \mathcal{ST}_0 \text{ and } \mathcal{ST}_1. \quad \textbf{(B)}$$

Assume **(B)** does not hold, i.e., there is $C \in post^*(\mathcal{I})$ and configurations $C_0 \in \mathcal{ST}_0$ and $C_1 \in \mathcal{ST}_1$ such that $C \xrightarrow{*} C_0$ and $C \xrightarrow{*} C_1$. These two executions can be extended to fair executions, and by the definition of $\mathcal{ST}_0$ and $\mathcal{ST}_1$ these executions stabilize to 0 and 1, respectively. So **(b)** does not hold.

Assume now that **(b)** does not hold. Then two fair executions starting at C_0 stabilize to different values. So C_0 can reach both $\mathcal{ST}_0$ and $\mathcal{ST}_1$, and **(B)** does not hold.

So **(a)** and **(b)** are equivalent to **(A)** and **(B)**. Since **(A)** is equivalent to $post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$, and **(B)** is equivalent to $pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$, we are done. ◀

▶ **Theorem 23.** *The well specification problem for IO protocols is in* EXPSPACE *and is* PSPACE*-hard.*

Proof. Let $\mathcal{P}$ be an IO protocol with n states. Recall that $\mathcal{ST}_b$ is given by $\overline{pre^*(\overline{\mathcal{C}_b})}$ where $\mathcal{C}_b$, for $b \in \{0, 1\}$, can be represented by the CoNF-constraint of single minterm defined by $x_i = 0$ for all $q_i \in O^{-1}(1 - b)$ and $0 \leq x_i \leq \infty$ otherwise. By Corollary 19, there exists a constant d, independent of $\mathcal{P}$, and a CoNF constraint $\Gamma \in \mathcal{G}(2, d)$ such that $[\![\Gamma]\!]$ is given by $post^*(\mathcal{I}) \cap \overline{pre^*(\mathcal{ST}_0)} \cap \overline{pre^*(\mathcal{ST}_1)}$.

In order to falsify condition **1.** of Proposition 22 it suffices to exhibit, following the previous reasoning, a "small" configuration C, such that $C(Q) \leq c^{2^{d \cdot (n^2 \log n)}}$, in the intersection. Note that C can be written in EXPSPACE. The EXPSPACE decision procedure follows the following steps: **1.** Guess a "small" configuration C. **2.** Check that C belongs to $post^*(\mathcal{I})$. **3.** Check that C belongs to $\overline{pre^*(\mathcal{ST}_b)}$, for $b = 0, 1$.
Algorithm for **2.**: Guess a at most double exponential sequence of minterms such that the first one is a minterm of $\mathcal{I}$, and every pair of consecutive minterms is related by $post^*[t]$

(given by Definition 8) for some t. Observe that we keep track of the last computed element and the number of steps performed so far in exponential space. Then, check that C belongs to the resulting minterm.
Algorithm for **3.**: It follows from EXPSPACE = coEXPSPACE that it is equivalent to check $C \in pre^*(\mathcal{ST}_b)$ is in EXPSPACE. Our algorithm is divided in two steps.
Step 1. Let c, d be such that $\mathcal{ST}_b \in \mathcal{G}(c,d)$. Guess a minterm M in $\mathcal{G}(c,d)$ and proceed similarly to Algorithm for **2.** to compute a minterm of $pre^*(M)$ and then check that C belongs to the resulting minterm.
Step 2. Verify that M does indeed belong to $\mathcal{ST}_b$. Formally, we rely on the following equivalences: $[\![M]\!] \subseteq \mathcal{ST}_b$ iff $[\![M]\!] \subseteq \overline{pre^*(\overline{\mathcal{C}_b})}$ iff $[\![M]\!] \cap pre^*(\overline{\mathcal{C}_b}) = \emptyset$. Using EXPSPACE = coEXPSPACE we now show that $[\![M]\!] \cap pre^*(\overline{\mathcal{C}_b}) \neq \emptyset$ belongs to EXPSPACE. We nondeterministically choose a minterm in $\overline{\mathcal{C}_b}$ and as previously explained guess a minterm in $pre^*(\overline{\mathcal{C}_b})$. Finally, we check whether it intersects with $[\![M]\!]$.

We use a similar reasoning for checking in EXPSPACE condition **2.** of Proposition 22.

The proof for PSPACE-hardness reduces from the acceptance problem for deterministic Turing machines running in linear space [13]. The proof follows the structure of analogous proofs for 1-safe Petri nets [11] (and also [8]) and will be provided in the full version. ◀

5.1 Consequences

In this section we list some consequences of Theorem 18 and Theorem 23.

In [4], Angluin *et al.* showed that IO protocols can compute exactly the counting predicates, i.e., the predicates that can be expressed by counting constraints. This is also a consequence of the proof of Theorem 23. Moreover, our results allow us to go further, and provide a bound on the number of states required to compute a predicate.

▶ **Corollary 24.** *IO population protocols compute exactly the counting predicates, i.e., the predicates corresponding to counting constraints.*

Proof. Let $\mathcal{P}$ be a well-specified IO protocol. The sets $\mathcal{I} \cap \overline{pre^*(\overline{pre^*(\mathcal{ST}_b)})}$ for $b \in \{0,1\}$ are the sets of initial configurations from which $\mathcal{P}$ stabilizes to $b = 0, 1$. Theorem 18 shows that they are counting sets. ◀

▶ **Corollary 25.** *Let $\mathcal{P}$ be an IO protocol computing a counting predicate $P(x_1, \ldots, x_k)$ of U-norm u and L-norm ℓ. Then there exists a constant c, independent of $\mathcal{P}$, such that $\mathcal{P}$ has at least $g \log\log(\max\{u, \ell\})$ states, where g denotes the inverse of the function $n \mapsto c \cdot (n^2 \log n)$.*

Proof. The set $\mathcal{I} \cap \overline{pre^*(\overline{pre^*(\mathcal{ST}_1)})}$ describes the initial configurations that stabilize to 1, i.e., the initial configurations for which the predicate computed by the protocol is true. By Corollary 19 (using a reasoning similar to that of Theorem 23), if $\mathcal{P}$ has n states, then the U-norm and L-norm of $\mathcal{I} \cap \overline{pre^*(\overline{pre^*(\mathcal{ST}_1)})}$ are bounded by the function $f(n) = 2^{2^{\mathcal{O}(n^2 \log n)}}$. Therefore, for a certain constant c, $\log\log\max\{u, \ell\} \leq c \cdot (n^2 \log n)$ and the number of states of a protocol computing a predicate of U-norm u and L-norm ℓ is at least $g \log\log(\max\{u, \ell\})$, where $g(x)$ is the inverse function of $x \mapsto c \cdot (x^2 \log x)$. ◀

Finally, we can show that the correctness problem for IO protocols is also in EXPSPACE.

▶ **Corollary 26.** *Let $\mathcal{P}$ be an IO population protocol with n states and k input states, and let $P(x_1, \ldots, x_k)$ be a counting predicate, expressed as a CoNF-constraint. The correctness problem for $\mathcal{P}$ and P, i.e., the problem of deciding if $\mathcal{P}$ computes P, is in* EXPSPACE.

Proof Sketch. We give a nondeterministic, exponential space algorithm for the complement of the correctness problem. The algorithm guesses nondeterministically a minterm of $\mathcal{I} \cap \overline{pre^*(\overline{pre^*(\mathcal{ST}_1)})}$, and checks that $\mathcal{I} \cap \overline{pre^*(\overline{pre^*(\mathcal{ST}_1)})}$ contains a configuration that does not satisfy P. The algorithm does a similar check for $\mathcal{ST}_0$ and a configuration that does satisfy P. The minterm can be constructed in exponential space by Theorem 23, and the check whether a minterm implies a CoNF-constraint can be done in polynomial time. ◀

References

1 Parosh A. Abdulla and Aletta Nylén. Better is better than well: on efficient verification of infinite-state systems. In *LICS '00*. IEEE Comput. Soc, 2000. doi:10.1109/lics.2000.855762.

2 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC '04*, pages 290–299. ACM, 2004. doi:10.1145/1011767.1011810.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006. doi:10.1007/s00446-005-0138-3.

4 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.

5 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: on the minimal size of population protocols. In *STACS '18*, volume 96, pages 16:1–16:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

6 Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for vass. In *RP '11*, volume 6945 of *LNCS*, pages 96–109. Springer, 2011. doi:10.1007/978-3-642-24288-5_10.

7 Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Algorithmic verification of population protocols. In *SSS '10*, volume 6366 of *LNCS*, pages 221–235. Springer, 2010. doi:10.1007/978-3-642-16023-3_19.

8 Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1&2):117–136, 1995. doi:10.1016/0304-3975(94)00231-7.

9 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. In *CONCUR '15*, volume 42 of *LIPIcs*, pages 470–482. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.470.

10 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Model checking population protocols. In *FSTTCS '16*, volume 65. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/lipics.fsttcs.2016.27.

11 Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4(3):277–299, 1977. doi:10.1016/0304-3975(77)90014-7.

12 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2014. doi:10.1145/2678280.

13 Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

14 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.

The Satisfiability Problem for Unbounded Fragments of Probabilistic CTL

Jan Křetínský
Technical University of Munich, Germany
jan.kretinsky@tum.de
https://orcid.org/0000-0002-8122-2881

Alexej Rotar
Technical University of Munich, Germany
alexej.rotar@tum.de

Abstract

We investigate the satisfiability and finite satisfiability problem for probabilistic computation-tree logic (PCTL) where operators are not restricted by any step bounds. We establish decidability for several fragments containing quantitative operators and pinpoint the difficulties arising in more complex fragments where the decidability remains open.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases temporal logic, probabilistic verification, probabilistic computation tree logic, satisfiability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.32

Related Version A full version of the paper is available at [24], `https://arxiv.org/abs/1806.11418`.

Funding This research was funded in part by the Czech Science Foundation grant No. P202/12/G061, TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project 383882557 "Statistical Unbounded Verification".

1 Introduction

Temporal logics are a convenient and useful formalism to describe behaviour of dynamical systems. Probabilistic CTL (PCTL) [17, 16] is the probabilistic extension of the branching-time logic CTL [12], obtained by replacing the existential and universal path quantifiers with the probabilistic operators, which allow us to quantify the probability of runs satisfying a given path formula. At first, the probabilities used were only 0 and 1 [17], giving rise to the *qualitative PCTL (qPCTL)*. This has been extended to any values from [0, 1] in [16], yielding the *(quantitative) PCTL* (onwards denoted just *PCTL*). More precisely, the syntax of these logics is built upon atomic propositions, Boolean connectives, temporal operators such as $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"), and the probabilistic quantifier $\bowtie q$ where $\bowtie$ is a numerical comparison such as $\leq$ or $>$, and $q \in [0,1] \cap \mathbb{Q}$ is a rational constant. A simple example of a PCTL formula is $ok\mathbf{U}_{=1}(\mathbf{X}_{\geq 0.9}finish)$, which says that on almost all runs we reach a state where there is 90% chance to *finish* in the next step and up to this state *ok* holds true. PCTL formulae are interpreted over Markov chains [26] where each state is assigned a subset of atomic propositions that are valid in a given state.

In this paper, we study the *satisfiability problem*, asking whether a given formula has a *model*, i.e. whether there is a Markov chain satisfying it. If a model does exist, we also want to construct it. Apart from being a fundamental problem, it is a possible tool for checking consistency of specifications or for reactive synthesis. The problem has been shown EXPTIME-complete for qPCTL in the setting where we quantify over finite models (*finite satisfiability*) [17, 7] as well as over generally countable models (*infinite satisfiability*) [7]. The problem for (the general quantitative) PCTL remains open for decades. We address this question on fragments of PCTL. The considered fragments are not of primary interst themselves. Rather they illustrate the techniques we develop and how far we can push decidability results when applying only those. In order to get a better understanding of this ultimate problem, we answer the problem for several fragments of PCTL that are

- quantitative, i.e. involving also probabilistic quantification over arbitrary rational numbers (not just 0 and 1),
- step unbounded, i.e. not imposing any horizon for the temporal operators.

Besides, we consider models with unbounded size, i.e. countable models or finite models, but with no a priori restriction on the size of the state space. These are the three distinguishing features, compared to other works. The closest are the following. Firstly, solutions for the qPCTL have been given in [17, 7] and for a more general logic PCTL* in [25, 21]. Secondly, [9] shows decidability for *bounded PCTL* where the scope of the operators is restricted by a step bound to a given time horizon. Thirdly, the *bounded satisfiability problem* is to determine, whether there exists a model of a given size for a given formula. This problem has been solved by encoding it into an SMT problem [4]. There is an important implication of this result. Namely, if we are able to determine a maximum required model size for some formula, then it follows that the satisfiability of that formula can also be determined. We take this approach in some of our proofs. Additionally, we use the result of [7] that the branching degree (number of successors) for a model of a formula ϕ can be bounded by $|\phi| + 2$, where $|\phi|$ is the length of ϕ.

Our contribution is as follows:

- We show decidability of the (finite and infinite) satisfiability problem for several quantitative unbounded fragments of PCTL, focusing on future- and globally-operators ($\mathbf{F}$,$\mathbf{G}$).
- We investigate the relationship between finite and infinite satisfiability on these fragments.
- We identify a fundamental issue preventing us from extending our techniques to the general case. We demonstrate this on a formula enforcing a more complicated form of its models. This allows us to identify the "smallest elegant" fragment where the problem remains open and the solution requires additional techniques.

Due to space constraints, the proofs are sketched and then worked out in detail in [24].

1.1 Further related work

As for the *non-probabilistic* predecessors of PCTL, the satisfiability problem is known to be EXPTIME-complete for CTL [12] as well as the more general modal μ-calculus [3, 15]. Both logics have the small model property [12, 20], more precisely, every satisfiable formula ϕ has a finite-state model whose size is exponential in the size of ϕ. The complexity of the satisfiability problems has been investigated also for fragments of CTL [23] and the modal μ-calculus [18].

The satisfiability problem for qPCTL and qPCTL* was investigated already in the early 80's [25, 21, 17], together with the existence of sound and complete axiomatic systems. The decidability for qPCTL over countable models also follows from these general results for

qPCTL*, but the complexity was not examined until [7], showing it is also EXPTIME-complete, both for finite and infinite satisfiability.

While the decidability of satisfiability is open, there are only few negative results. [7] proves *undecidability* of the problem whether for a given PCTL formula there exists a model with a branching degree that is bounded by a given integer, where the branching degree is the number of successors of a state. However, the authors have not been able to extend their proof and show the undecidability for the general problem.

The PCTL *model checking problem* is the task to determine, whether a given system satisfies a given formula, i.e. whether it is a model of the formula. This problem has been studied both for finite and infinite Markov chains and decision processes, see e.g. [10, 19, 14, 13, 8]. The PCTL *strategy synthesis* problem asks whether the non-determinism in a given Markov decision process can be resolved so that the resulting Markov chain satisfies the formula [1, 22, 5, 6].

2 Preliminaries

In this section, we recall basic notions related to (discrete-time) Markov chains [26] and the probabilistic CTL [16]. Let $\mathcal{A}$ be a finite set of atomic propositions.

2.1 Markov chains

▶ **Definition 1** (Markov chain). A *Markov chain* is a tuple $M = (S, P, s_0, L)$ where S is a countable set of *states*, $P : S \times S \to [0, 1]$ is the *probability transition matrix* such that, for all $s \in S$, $\sum_{t \in S} P(s, t) = 1$, $s_0 \in S$ is the *initial* state, and $L : S \to 2^{\mathcal{A}}$ is a labeling function.

Whenever we write M, we implicitly mean a Markov chain (S, P, s_0, L). The semantics of a Markov chain M, is the probability space $(Runs_M, \mathcal{F}_M, \mathbb{P}_M)$, where $Runs_M = S^\omega$ is the set of *runs* of M, $\mathcal{F}_M \subseteq 2^{S^\omega}$ is the σ-algebra generated by the set of cylinders of the form $Cyl_M(\rho) = \{\pi \in S^\omega \mid \rho \text{ is a prefix of } \pi\}$ and the probability measure is uniquely determined [2] by $\mathbb{P}_M(Cyl_M(\rho_0 \cdots \rho_n)) := \prod_{0 \le i < n} P(\rho_i, \rho_{i+1})$ if $\rho_0 = s_0$ and 0 otherwise.

We say that a state is *reached* on a run if it appears in the sequence; a set of states is reached if some of its states are reached. The immediate successors of a state s are denoted by $post_M(s) := \{t \in S \mid P(s, t) > 0\}$ and the set of states reachable with positive probability is the reflexive and transitive closure $post^*_M(s)$. We will write $\mathbb{P}(\cdot), post(\cdot)$, and $post^*(\cdot)$, if M is clear from the context.

The *unfolding* of a Markov chain M is the Markov chain $T_M := (S^+, P', s_0, L')$ with the form of an infinite tree given by $P'(\rho s, \rho s s') = P(s, s')$ and $L'(\rho s) = L(s)$. Each state of T_M maps naturally to a state of M (the last one in the sequence), inducing an equivalence relation $\rho s \sim \rho s'$ iff $s = s'$. Consequently, each run of T_M maps naturally to a run of M and the unfolding preserves the measure of the respective events.

For a Markov chain M, a set $T \subseteq S$ is called *strongly connected* if for all $s, t \in T$, $t \in post^*(s)$; it is a *strongly connected component (SCC)* if it is maximal (w.r.t. inclusion) with this property. If, moreover, $post^*(t) \subseteq T$ for all $t \in T$ then it is a *bottom SCC (BSCC)*. A classical result, see e.g. [2], states that the set of states visited infinitely often is almost surely, i.e. with probability 1, a BSCC:

▶ **Lemma 2.** *In every finite Markov chain, the set of BSCCs is reached almost surely. Further, conditioning on runs reaching a BSCC C, every state of C is reached infinitely often almost surely.*

2.2 Probabilistic Computational Tree Logic

The definition of probabilistic CTL (PCTL) [16] is usually based on the next- and until-operators ($\mathbf{X}$, $\mathbf{U}$). In this paper, we restrict our attention to the future- and globally operators ($\mathbf{F}$, $\mathbf{G}$), which can be derived from the until-operator. Further, w.l.o.g. we impose the negation normal form and the lower-bound-comparison normal form; for the respective transformations see, e.g., [2].

▶ **Definition 3** (PCTL($\mathbf{F}$,$\mathbf{G}$) syntax and semantics)**.** The *formulae* are given by the following syntax:

$$\Phi ::= a \mid \neg a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathbf{F}_{\rhd q}\Phi \mid \mathbf{G}_{\rhd q}\Phi$$

where $q \in [0,1]$, $\rhd \in \{\geq, >\}$, and $a \in \mathcal{A}$ is an atomic proposition. Let M be a Markov chain and $s \in S$ its state. We define the modeling relation $\models$ inductively as follows

M1 $M, s \models a$ iff $a \in L(s)$
M2 $M, s \models \neg a$ iff $a \notin L(s)$
M3 $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$
M4 $M, s \models \phi \vee \psi$ iff $M, s \models \phi$ or $M, s \models \psi$
M5 $M, s \models \mathbf{F}_{\rhd q}\varphi$ iff $\mathbb{P}_{M(s)}(\{\pi \mid \exists i \in \mathbb{N}_0 : M, \pi[i] \models \varphi\}) \rhd q$
M6 $M, s \models \mathbf{G}_{\rhd q}\varphi$ iff $\mathbb{P}_{M(s)}(\{\pi \mid \forall i \in \mathbb{N}_0 : M, \pi[i] \models \varphi\}) \rhd q$
where $M(s)$ is M with s being the initial state, and $\pi[i]$ is the ith element of π. We say that M is a *model* of φ if $M, s_0 \models \varphi$.

We will denote the set of literals by $\mathcal{L} := \mathcal{A} \cup \{\neg a \mid a \in \mathcal{A}\}$. Instead of the constraint ≥ 1, we often write $= 1$. Further, we define the set of all subformulae. This definition slightly deviates from the usual definition of subformulae, e.g. the one in [7], in that $\neg a \in sub(\phi)$ does not necessarily imply $a \in sub(\phi)$.

▶ **Definition 4** (Subformulae)**.** The set $sub(\phi)$ is recursively defined as follows
- $\phi \in sub(\phi)$
- if $\psi \wedge \xi \in sub(\phi)$ or $\psi \vee \xi \in sub(\phi)$, then $\psi, \xi \in sub(\phi)$
- if $\mathbf{F}_{\rhd q}\psi \in sub(\phi)$ or $\mathbf{G}_{\rhd q}\psi \in sub(\phi)$ then $\psi \in sub(\phi)$

Next, we introduce the satisfiability problems, which are the main topic of the paper.

▶ **Definition 5** (The satisfiability problems)**.** A formula ϕ is called *(finitely) satisfiable*, if there is a (finite) model for ϕ. Otherwise, it is (finitely) unsatisfiable. The (finite) satisfiability problem is to determine whether a given formula is (finitely) satisfiable.

Instead of simply writing "satisfiable" we sometimes stress the absence of "finitely" and write "generally satisfiable" for satisfiablity on countable, i.e. finite or countably infinite, models. For some proofs, it is more convenient to consider the unfolding of a Markov chain instead of the original one. As we mentioned already, the measure of events is preserved in the unfolding of a chain. Hence, we can state the following lemma.

▶ **Lemma 6.** *If M is a model of ϕ then its unfolding T_M is a model of ϕ.*

We say that formulae ϕ, ψ are (finitely) equivalent if they have the same set of (finite) models, written $\phi \equiv \psi$ ($\phi \equiv_{fin} \psi$); that they are (finitely) equisatisfiable if they are both (finitely) satisfiable or both (finitely) unsatisfiable; and that $\phi \Rightarrow \psi$ if every model of ϕ is also a model of ψ.

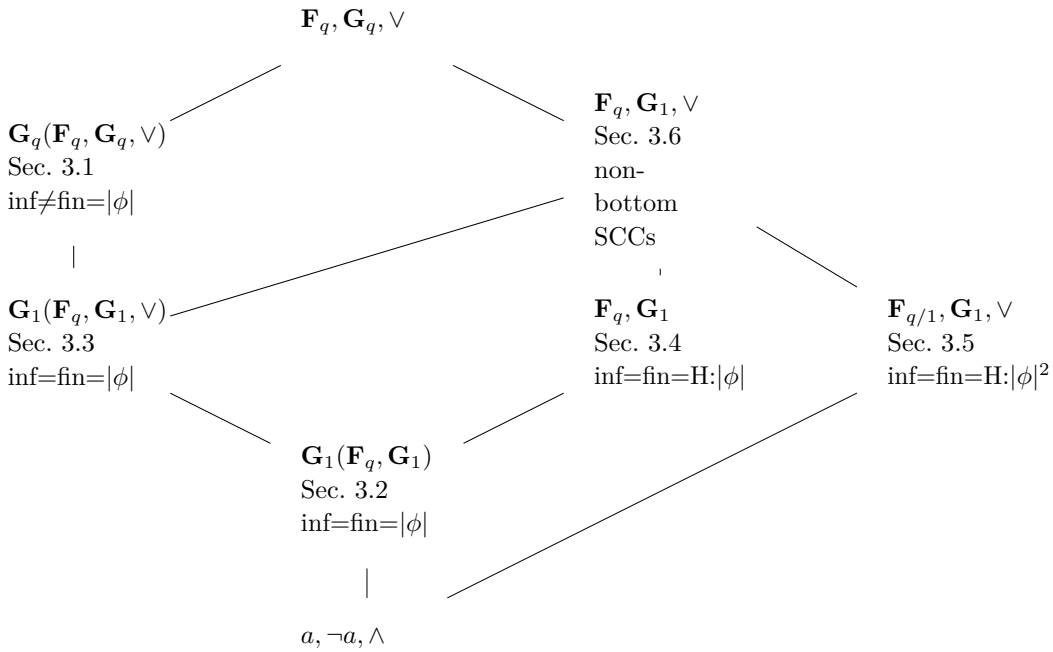


Figure 1 Hasse diagram summarizing the satisfiability results for the considered fragments of PCTL($\mathbf{F}$, $\mathbf{G}$), all containing literals and conjunctions, and some form of quantitative comparisons. The fragments are described by the list of operators they allow (excluding the constructs of the minimal fragment). The subscript denotes the possible constraints on probabilistic operators. $\mathbf{G}_x(list)$ denotes formulae in the fragment described by *list* with $\mathbf{G}$-operators at the top-level. fin and inf abbreviate finite and general satisfiability, respectively. fin=inf denotes that the problems are equivalent. H:x denotes that the height of a tree model can be bounded by x. By $= x$ we denote that the model size can be bounded by x. The $\mathbf{F}_q, \mathbf{G}_1, \vee$-fragment might require non-bottom SCCs in finite models.

3 Results

In this section we present our results. A summary is schematically depicted in Fig. 1. We briefly describe the considered fragments; the full formal definitions can be found in the respective sections. Since already the satisfiability for propositional logic in negation normal form has nontrivial instances only when all the constructs $a, \neg a$ and conjunction are present, we only consider fragments with all three included; see the bottom of the Hasse diagram. The fragments are named by the list of constructs they use, where we omit the three constructs above to avoid clutter. Here 1 stands for ≥ 1 and q stands for $\triangleright q$ for all $q \in [0,1] \cap \mathbb{Q}$. Further, $\mathbf{G}_x(list)$ denotes the sub-fragment of *list* where the topmost operator is $\mathbf{G}_x$. Finally, $\mathbf{F}_{q/1}$ denotes the use of $\mathbf{F}_q$ with the restriction that inside $\mathbf{G}$ only $q = 1$ can be used.

The fragments are investigated in the respective sections. We examine the problems of the general satisfiability ("inf") and the finite satisfiability ("fin"); equality denotes the problems are equivalent. We use two results to prove decidability of the problems. Firstly, [4] shows that given a formula ϕ and an integer n, one can determine whether or not there is a model for ϕ that has at most n states. Consequently, we obtain the decidability result whenever we establish an upper bound on the size of smallest models. Here "$|\phi|$" denotes the satisfiability of a given ϕ on models of size $\leq |\phi|$. Secondly, [7] establishes that for any satisfiable PCTL formula there is a model with branching bounded by $|\phi| + 2$. Consequently,

we obtain the decidability result whenever it is sufficient to consider trees of a certain height H (with back edges) since the number of their nodes is then bounded by $(|\phi|+2)^H$. Here "H:n" denotes that the models can be limited to a height $H \leq n$.

While we obtain decidability in the lower part of the diagram, the upper part only treats finite satisfiability, and in particular for $\mathbf{F}_q, \mathbf{G}_1, \vee$, we only demonstrate that models with more complicated structure are necessary. Namely, the models may be of unbounded sizes for structurally same formulae – i.e. formulae which only differ in the constraints on the temporal operators – or require presence of non-bottom SCCs, see Section 3.6 and the discussion in Section 4.

3.1 Finite satisfiability for $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$

This section treats $\mathbf{G}$-formulae of the $\mathbf{F}_q, \mathbf{G}_q$-fragment, i.e. of PCTL($\mathbf{F}, \mathbf{G}$). In particular, it includes $\mathbf{G}_{>0}$-formulae. In general, formulae in this fragment (even without quantified $\mathbf{F}$ and $\mathbf{G}$-operators) can enforce rather complicated behaviour [7]. Therefore, we will focus on finitely satisfiable formulae. We will see that they can be satisfied by rather simple models.

▶ **Definition 7.** $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$-formulae are given by the grammar

$$\Phi ::= \mathbf{G}_{\triangleright q}\Psi$$
$$\Psi ::= a \mid \neg a \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \mathbf{F}_{\triangleright q}\Psi \mid \mathbf{G}_{\triangleright q}\Psi$$

The main result of this section is that finitely satisfiable formulae in this fragment can be satisfied by models of size linear in $|\phi|$.

▶ **Theorem 8.** *Let ϕ be a finitely satisfiable $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$-formula. Then ϕ has a model of size at most $|\phi|$.*

Intuitively, we obtain the result from the fact that some BSCC is reached almost surely and every state in a BSCC is reached almost surely, once we have entered one. In infinite models, BSCCs are not reached almost surely and therefore the proofs cannot be extended to general satisfiability. The following lemma and its proof demonstrate how we can make use of the BSCC properties in order to obtain an equisatisfiable formula in a simpler fragment.

▶ **Lemma 9.** *Let ϕ be a $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$-formula. Then, ϕ is finitely equisatisfiable to a $\mathbf{G}_1(\mathbf{F}_1, \mathbf{G}_1)$-formula ϕ', such that $\phi' \Rightarrow \phi$.*

Proof Sketch. Write ϕ as $\mathbf{G}_{\triangleright q}\psi$. Assume that we have a finite model M for $\mathbf{G}_{\triangleright q}\psi$. Intuitively, we can select a BSCC that satisfies $\mathbf{G}_{=1}\psi$. We know that there is a BSCC because we are dealing with a finite model. We also know that there is at least one BSCC satisfying our formula, for otherwise M would not be a model for it. In a BSCC, every state is reached almost surely from every other state by Lemma 2. Hence, we can select exactly one state for each $\mathbf{F}$-subformula which satisfies that formula's argument. Then we can create a new BSCC from these states, arranging them, e.g., in a circle. This BSCC models $\mathbf{G}_{=1}\hat{\psi}$, where $\hat{\psi}$ replaces all probabilistic operators with their "almost surely" version. Hence, we have created a model for a $\mathbf{G}_1(\mathbf{F}_1, \mathbf{G}_1)$ formula from a model for $\mathbf{G}_{\triangleright q}\psi$. The opposite direction follows from the fact that $\mathbf{G}_{=1}\hat{\psi} \Rightarrow \mathbf{G}_{\triangleright q}\psi$. ◀

Note that the transformation does not produce an equivalent formula. Hence, we cannot replace an occurrence of such a formula in a more complex formula. For instance, the formula $\mathbf{G}_{\geq 1/2}\neg a \wedge \mathbf{F}_{\geq 1/2}a$ is satisfiable, whereas $\mathbf{G}_{=1}\neg a \wedge \mathbf{F}_{\geq 1/2}a$ is not. The proof does not work for equality because we are selecting one BSCC while ignoring the rest. This example

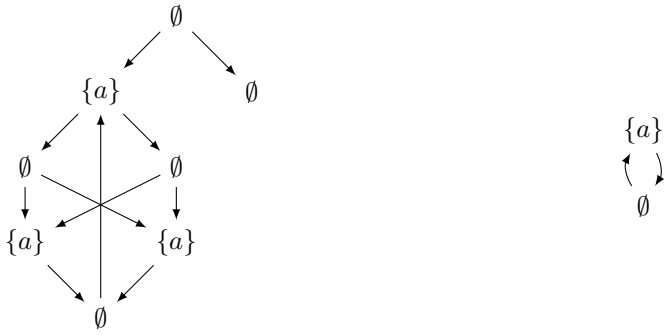


Figure 2 A large and a small model for Formula (1).

demonstrates why we cannot ignore certain BSCCs in general. Using the above result, it is easy to prove Theorem 8.

Proof Sketch. [Proof Sketch of Theorem 8] This follows immediately from the proof of Lemma 9. The BSCC that we have created has at most as many states as there are $\mathbf{F}$-subformulae, which is bounded by $|\phi|$. ◀

▶ **Example 10.** Consider the formula

$$\phi := \mathbf{G}_{\geq 1/2}(\mathbf{F}_{\geq 1/3} a \wedge \mathbf{F}_{\geq 1/3} \neg a). \tag{1}$$

The large Markov chain in Figure 2 models ϕ. Unlabeled arcs indicate a uniform distribution over all successors. It is clear that the model is unnecessarily complicated. After reducing it according to Lemma 9, we obtain the smaller Markov chain on the right.

The example below shows that satisfiability is not equivalent to finite satisfiability for this fragment, and that the proposed transformation does not preserve equisatisfiability over general models. The decidability of the general satisfiability thus remains open here.

▶ **Example 11.** Note that we made use of the BSCC properties for the proofs of this subsection, such as that some BSCC is reached almost surely. Since this is only the case for finite Markov chains, our transformation only holds for finite satisfiability. If we consider the general satisfiability problem, then the equivalent of Lemma 9 is not true. For instance, the formula

$$\phi := \mathbf{G}_{>0}(\neg a \wedge \mathbf{F}_{>0} a) \tag{2}$$

is satisfiable, but requires infinite models, as pointed out in [7]. One such model is given in Figure 3. Observe that the single horizontal run has measure greater than 0. Now consider

$$\hat{\phi} := \mathbf{G}_{=1}(\neg a \wedge \mathbf{F}_{=1} a)$$

Obviously, this formula unsatisfiable. Hence, in this case ϕ is not equisatisfiable to $\hat{\phi}$.

3.2 Satisfiability for $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$

This section treats $\mathbf{G}$-formulae of a fragment where $\mathbf{G}_{\triangleright q}$ only appears with $q = 1$ and there is no disjunction. The results are later utilized in a richer fragment in Section 3.4. In fact, the main result of this section is an immediate consequence of the main theorem of Section 3.3. Still, the results are interesting themselves as they show some properties of models for formulae in this fragment which do not apply in the generalized case.

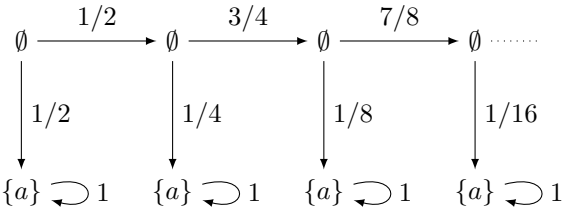


Figure 3 An infinite model for Formula (2).

▶ **Definition 12.** $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formulae are given by the grammar

$$\Phi ::= \mathbf{G}_{=1}\Psi$$
$$\Psi ::= a \mid \neg a \mid \Psi \wedge \Psi \mid \mathbf{F}_{\rhd q}\Psi \mid \mathbf{G}_{=1}\Psi$$

We prove that satisfiable formulae of this fragment are satisfiable by models of linear size and thus also finitely satisfiable.

▶ **Theorem 13.** *Let ϕ be a satisfiable $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formula. Then ϕ has a model of size at most $|\phi|$.*

The idea here is that we can find a state which behaves similarly to a BSCC (even in infinite models) in that it satisfies all $\mathbf{G}$-subformulae. We can then use this state's successors to construct a small model. The outline of the proof is roughly as follows: First we show that from every state and for every subformula we can find a successor that satisfies this subformula. Using this, we can show that there is a state that satisfies all $\mathbf{G}$-subformulae.

▶ **Lemma 14.** *Let ϕ be a satisfiable $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formula and M its model. Then, for every $\psi \in sub(\phi)$, and $s \in S$, there is a state $t \in post^*(s)$, such that $M, t \models \psi$.*

Proof Sketch. This follows from the fact that we do not allow disjunctions in this fragment. We apply induction over the depth of a subformula ψ. If the formula is ϕ itself, then there is nothing to show. Otherwise, the induction hypothesis yields that the higher-level subformulae are satisfied at some state s. From this, we can easily see that in all possible cases the claim follows: If the higher-level formula is a conjunction, then ψ is one of its conjuncts. Since both conjuncts must be satisfied by s, in particular ψ must be satisfied at s. A similar argument applies to $\mathbf{G}$-formulae. If it is of the form $\mathbf{F}_{\rhd q}\xi$, then we know that there must be a reachable state where ξ holds. ◀

This concludes the first part of the proof. We continue with the second part and prove that we can find a state which satisfies all $\mathbf{G}$-subformulae.

▶ **Lemma 15.** *Let ϕ be a satisfiable $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formula, M its model, and let $G := \{\psi \in sub(\phi) \mid \psi = \mathbf{G}_{=1}\xi$ for some $\xi\}$. Then there is a state $s \in S$ such that $M, s \models \psi$ for all $\psi \in G$.*

Proof Sketch. It is clear that after encountering a $\mathbf{G}$-formula at some state, all successors will also satisfy it. Therefore, the set of satisfied $\mathbf{G}$-formulae is monotonically growing and bounded. Hence, we can apply induction over the number of yet unsatisfied $\mathbf{G}$-formulae. In every step, we are looking for the next state to satisfy an additional $\mathbf{G}$-formula. This is always possible (as long as there are still unsatisfied ones), due to Lemma 14. ◀

Now, we can prove Theorem 13.

Proof Sketch of Theorem 13. By Lemma 15, we can find a state that satisfies all $\mathbf{G}$-subformulae. In some sense, this state's subtree resembles a BSCC. We can include exactly one state for each $\mathbf{F}$-subformula and create a BSCC out of those states, e.g., arrange them in a circle. We apply induction over $\psi \in sub(\phi)$. The satisfaction of literals and conjunctions is straightforward. Since every state is reached almost surely, every $\mathbf{F}$-formula will be satisfied that way. The satisfaction of the $\mathbf{G}$-formulae follows from the fact that all states used to satisfy all $\mathbf{G}$-formulae in the original model, and from the induction hypothesis. ◀

For the case of finite satisfiability, we also present an alternative proof, which sheds more light on this fragment and its super-fragments. For details, see [24, Appendix B]. Let $\equiv_{fin}$ denote equivalence of PCTL formulae over finite models.

▶ **Theorem 16.** *Let ϕ be a $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formula. Then, the following equivalence holds:*

$$\mathbf{G}_{=1}\phi \equiv_{fin} \mathbf{G}_{=1}(\bigwedge_{l\in A} l \wedge \mathbf{F}_{=1}\mathbf{G}_{=1}\bigwedge_{l\in B} l \wedge \bigwedge_{i\in I}\mathbf{F}_{=1}\bigwedge_{l\in C_i} l)$$

for appropriate $I \subset \mathbb{N}$, and $A, B, C_i \subset \mathcal{L}$.

Proof Sketch. The proof is based on the following auxiliary statements

$$\mathbf{G}_{=1}\mathbf{G}_{=1}\phi \equiv \mathbf{G}_{=1}\phi \tag{3}$$

$$\mathbf{G}_{=1}\mathbf{F}_{\triangleright q}\phi \equiv_{fin} \mathbf{G}_{=1}\mathbf{F}_{=1}\phi \tag{4}$$

$$\mathbf{F}_{=1}\mathbf{F}_{\triangleright q}\phi \equiv \mathbf{F}_{\triangleright q}\phi \tag{5}$$

and follows by induction.

The second statement is the most interesting one. Intuitively, it is a zero-one law, stating that infinitely repeating satisfaction with a positive probability ensures almost sure satisfaction. Notably, this only holds if the probabilities are bounded from below, hence for finite models, not necessarily for infinite models. ◀

It is an easy corollary of this theorem that a satisfiable formula has a model of a circle form with A and B holding in each state and each element in each C_i holding in some state. In general the models can be of a lasso shape where the initial (transient) part only has to satisfy A, allowing for easy manipulation in extensions of this fragment.

▶ Remark. Note that the equivalence does not hold over infinite models. Indeed, consider as simple a formula as $\mathbf{G}_{=1}\mathbf{F}_{>0}a$, which is satisfied on the Markov chain of Fig. 3 [7], while this does not satisfy the transformed $\mathbf{G}_{=1}\mathbf{F}_{=1}a$. Crucially, equivalence (4) does not hold already for this tiny fragment. Interestingly, when we build a model for the transformed formula, which is equisatisfiable but not equivalent, it turns out to be a model of the original formula. If, moreover, we consider $\neg a, \wedge, \mathbf{G}_{>0}$ then finite and general satisfiability start to differ.

Before we move on to the next fragment, we will prove another consequence of Lemma 14. It is a statement about the BSCCs of models for formulae in this fragment and will be used later for the proof of Theorem 21.

▶ **Corollary 17.** *Let $\mathbf{G}_{=1}\phi$ be a satisfiable $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1)$-formula and M its model. Then, for every BSCC $T \subseteq post^*(s_0)$ of M, the following holds*

1. *For all $\psi \in sub(\mathbf{G}_{=1}\phi)$, there is a state $t \in T$, such that $M, t \models \psi$.*
2. *For all $\mathbf{G}_{=1}\psi \in sub(\mathbf{G}_{=1}\phi)$, and for all states $t \in T$, $M, t \models \mathbf{G}_{=1}\psi$.*

Proof. Point 1 follows from the fact that every reachable BSCC must satisfy $\mathbf{G}_{=1}\phi$, and from Lemma 14. Point 2 follows immediately from point 1. ◀

Note that we did not assume finite satisfiability here, so the model might not contain a single BSCC. In that case, the claim is trivially true. However, Theorem 13 allows us to focus on finitely satisfiable formulae in this fragment.

3.3 Satisfiability for $\mathbf{G_1(F_q, G_1, \vee)}$

This section treats $\mathbf{G}$-formulae of the fragment where $\mathbf{G}_{\triangleright q}$ only appears with $q = 1$. We thus lift a restriction of the previous fragment and allow for disjunctions. We generalize the obtained results to this larger fragment. We mentioned earlier that some of the results of the previous fragment do not apply here. Concretely, Lemma 14 does not hold here; that is, there might be subformulae which are not satisfied almost surely. Therefore, there is not necessarily a state that satisfies all $\mathbf{G}$-subformulae. For example, consider $\mathbf{G}_{=1}(\mathbf{F}_{>0}\mathbf{G}_{=1}a \vee \mathbf{F}_{>0}\mathbf{G}_{=1}\neg a)$. There cannot be a single BSCC to satisfy both disjuncts. Although this is not a problem for the results of this section, it will turn out to be a fundamental problem when dealing with arbitrary formulae of the $\mathbf{F}_q, \mathbf{G}_1, \vee$ fragment.

▶ **Definition 18.** $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1, \vee)$-formulae are given by the grammar

$$\Phi ::= \mathbf{G}_{=1}\Psi$$
$$\Psi ::= a \mid \neg a \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \mathbf{F}_{\triangleright q}\Psi \mid \mathbf{G}_{=1}\Psi$$

We prove that satisfiable formulae of this fragment are satisfiable by models of linear size and thus also finitely satisfiable.

▶ **Theorem 19.** *Let ϕ be a satisfiable $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1, \vee)$-formula. Then ϕ has a model of size at most $|\phi|$.*

Proof Sketch. The proof for this theorem works essentially the same as it did for Theorem 13. Recall that we looked for a state to satisfy all $\mathbf{G}$-formulae. Though we will not necessarily find a state that does so in this fragment, we can look for a state that satisfies maximal subsets of satisfied $\mathbf{G}$-formulae. Then, we can continue in a similar way as we did in the simpler setting. ◀

3.4 Satisfiability for $\mathbf{F_q, G_1}$

This section treats general formulae of the fragment with no disjunction and where $\mathbf{G}_{\triangleright q}$ only appears with $q = 1$.

▶ **Definition 20.** $\mathbf{F}_q, \mathbf{G}_1$-formulae are given by the grammar

$$\Phi ::= a \mid \neg a \mid \Phi \wedge \Phi \mid \mathbf{F}_{\triangleright q}\Phi \mid \mathbf{G}_{=1}\Phi$$

In Section 3.2 we discussed a special case of this fragment, where the top-level operator is $\mathbf{G}_{=1}$. Two results are particularly interesting for this section: Firstly, the construction of models for such formulae as explained in the proof of Theorem 13, and secondly, the properties of BSCCs in models for such formulae as stated in Corollary 17. We will use those in order to simplify models in this generalized setting. We say a Markov chain has *height* h if it is a tree with back edges of height h.

▶ **Theorem 21.** *A satisfiable $\mathbf{F}_q, \mathbf{G}_1$-formula ϕ has a model of height $|\phi|$.*

Proof Sketch. Our aim is to transform a given, possibly infinite model into a tree-like shape. To do so, we first construct a tree by considering all **F**-formulae which are not nested in **G**s (in the following simply non-nested **F**-formulae). Each path in this tree will satisfy each of these **F**-formulae at most once. At the end of each path, we will then insert BSCCs satisfying the **G**-formulae, in the spirit of Theorem 13.

The collapsing procedure from a state s is as follows: We first determine which of the **F**-formulae that are satisfied at s are relevant. Those are the formulae which are not nested in other temporal formulae and have not yet been satisfied on the current path. Once we have determined this set, say I, we need to find the successors which are required to satisfy the formulae in I. For this, we construct the set $sel(s)$. Informally, $sel(s)$ contains all states t s.t. (i) t satisfies at least one formula $\psi \in I$, and (ii) there is no state on the path between s and t satisfying ψ. Formally, $sel(s) := \{t \in post^*(s) \mid \exists \mathbf{F}_{\rhd r}\psi \in I.\ M,t \models \psi \wedge \forall t' \in post^*(s) \cap pre^*(t).M,t \not\models \psi\}$. Then, we connect s to every state in $sel(s)$ directly; i.e. for $t \in sel(s)$, we set $P(s,t) := P^*(s,t)$, and for all states $t \notin sel(s)$, we set $P(s,t) := 0$.[1] A simple induction on the length of a path yields that every state that is reachable from s in the constructed MC is reached with at least the probability as in the original one. From this, one can easily see that every non-nested **F**-formula is satisfied. The new set $post(s)$ might be infinite. However, we know that we can prune most of the successors and limit the branching degree to $|\phi| + 2$ [7]. Then, we repeat the procedure from each of the successors. Since the number of non-nested **F**-formulae decreases with every step, we will reach states which do not have to satisfy non-nested **F**-formulae at all on every branch. The number of steps we need to reach such states, is bounded by the number of non-nested **F**-formulae in ϕ. At those states, we can use Theorem 13 to obtain models for the respective **G**-formulae. Those are of size linear in the size of the **G**-formulae. The overall height is then bounded by $|\phi|$. The fact that the resulting MC is a model can be easily proved by induction over $|\phi|$. ◀

The models that we construct have a quite regular shape: They start as a tree and in every step ensure satisfaction of one of the **F**-formulae. As soon as they have satisfied all outer **F**-formulae, on every branch a model of circle shape for the respective **G**-formula follows. Since the branching degree is at most $|\phi| + 2$ and the number of steps before we repeat a state is bounded by $|\phi|$, the overall size is bounded to $(|\phi| + 2)^{2|\phi|+1}$.

▶ **Example 22.** Let $\phi^p := \mathbf{F}_{\geq p}\mathbf{G}_{=1}a$, and $\psi^p := \mathbf{F}_{\geq p}\mathbf{G}_{=1}\neg a$. The large Markov chain of Figure 4 is a model for $\phi^{1/2} \wedge \psi^{1/2}$. The grayed states illustrate the set $sel(s_0)$. The other boxes show the sets $sel(.)$ of the respective grayed state. Everything in between is omitted. The smaller Markov chain is the reduced version of the original model.

3.5 Satisfiability for $\mathbf{F}_{q/1}, \mathbf{G}_1, \vee$

In the previous section we have been able to construct simple models for formulae of the $\mathbf{F}_q, \mathbf{G}_1$-fragment by exploiting the nature of **G**-formulae thereof as presented in Section 3.2. This works because every formula nested within a **G** is satisfied in every BSCC. Hence, we can simply postpone the satisfaction of those until we reach a BSCCs. In the $\mathbf{F}_q, \mathbf{G}_1, \vee$-fragment, this is not the case anymore, as discussed in Section 3.3. This can cause some complications, which are discussed in more detail in Section 3.6. In order to be able to apply similar techniques as in the previous section, we can simplify the fragment and enforce the property that **F**-formulae occur only with $q = 1$ within **G**s.

[1] In fact, we need to scale $P(s,t)$ in order to obtain a Markov chain, in general, as the probability to reach $sel(s)$ might be less than 1. For details, refer to the formal proof in [24, Appendix A].

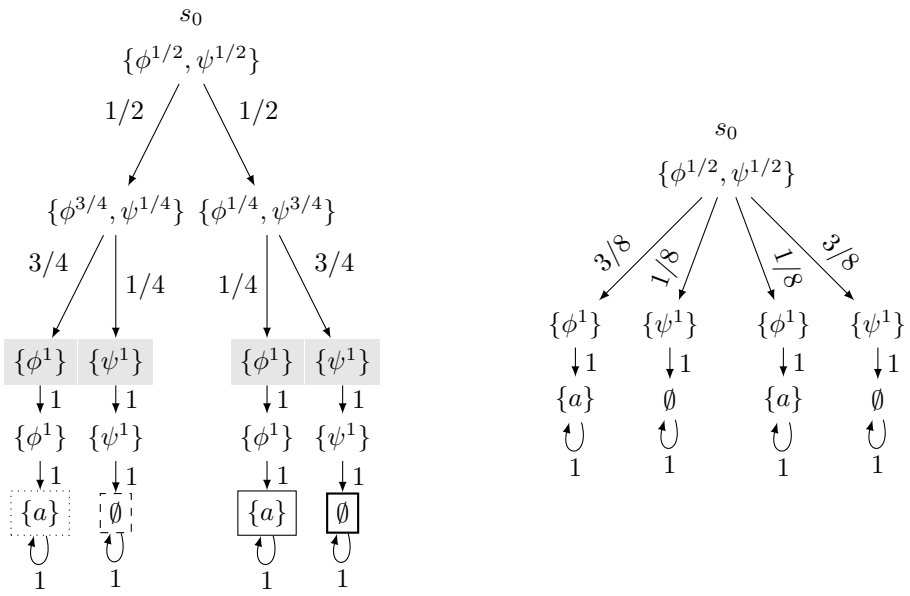


Figure 4 Example of a reduction for a $\mathbf{F}_q, \mathbf{G}_1$-formula.

▶ **Definition 23.** $\mathbf{F}_{q/1}, \mathbf{G}_1, \vee$-formulae are given by the grammar

$$\Phi ::= a \mid \neg a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathbf{F}_{\rhd q}\Phi \mid \mathbf{G}_{=1}\Psi$$
$$\Psi ::= a \mid \neg a \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \mathbf{F}_{=1}\Psi \mid \mathbf{G}_{=1}\Psi$$

Again, we show that the necessary minimal height of models can be bounded.

▶ **Theorem 24.** *A satisfiable $\mathbf{F}_{q/1}, \mathbf{G}_1, \vee$-formula ϕ has a model of height $|\phi|^2$.*

Proof Sketch. As the first step, we apply the same procedure as in the proof of Theorem 21. The outer $\mathbf{F}$-formulae are then satisfied for the same reason as in the setting without disjunctions. However, the $\mathbf{G}$-nested $\mathbf{F}$-formulae might not be satisfied anymore because the BSCCs do not necessarily satisfy each of them. Since the $\mathbf{G}$-nested $\mathbf{F}$-formulae appear only with $q = 1$, we know that once a state of the original model satisfies such a formula, almost every path satisfies the respective path formula. Let s be a state of the reduced chain, and $t \in post(s)$. In the original model, there might be states in between s and t. If some $\mathbf{G}$-nested $\mathbf{F}$-formula (say $\mathbf{F}_{=1}\psi$) which is satisfied at s is also satisfied at t, we do not need to take care of it. If this is not the case, then we know for sure that some of the states between s and t must satisfy ψ. We can determine such states for each $\mathbf{G}$-nested $\mathbf{F}$-formula. We include exactly one of those for each such formula. Then, preserving the order, we chain them in such a way that each one has a unique successor. The last one's unique successor is t. Let s' be the first one. Then, we set $P(s, s') := P(s, t)$. We repeat this procedure for each state of the reduced chain.

However, we must be careful when picking the states that we want to include. For instance, if we always pick the first state to satisfy some formula, then we might in the end violate ϕ. Say we have two formulae $\mathbf{F}_{=1}\psi$ and $\mathbf{F}_{=1}\xi$ and states u, v, w occuring in this order, such that $M, u \models \psi$, $M, v \models \xi$, and $M, w \models \psi$. If $M, v \models \mathbf{F}_{=1}\psi$ in the original model, we might violate it if we omit w. Therefore, we shall rather omit u und pick w. In general, we pick the last possible state for each formula. In a sense, we look backwards starting from t. This way, it is guaranteed that whenever we have not yet satisfied some $\mathbf{F}$-formula, it will be propagated to t.

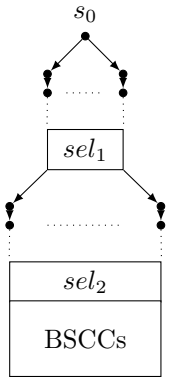


Figure 5 Reduction of models for $\mathbf{F_{q/1}}, \mathbf{G_1}, \vee$.

This way, we preserve the reachability probabilities and therefore the satisfaction of the outer **F**-formulae. The newly added states guarantee the satisfaction of nested **F**-formulae. An induction over ϕ shows that the constructed MC is again a model. Since we add at most $|\phi|$ new states between the states of the reduced MC which is of height at most $|\phi|$, we obtain the claimed bound on the height. ◀

Figure 5 illustrates the transformation of the models as described in the proof sketch. sel_1 and sel_2 are the selections. In the $\mathbf{F}_q, \mathbf{G}_1$-fragment, we directly connected those sets. Here, we insert simple chains between the selections. The construction guarantees that we have at most one state per **F**-formula to satisfy. This is obtained by postponing the satisfaction until the last possible moment before sel.

▶ Remark. Note that the resulting models have a tree shape with BSCCs. There are no non-bottom SCCs. Even if the original model did have such, they are removed by this construction: The reduction algorithm takes care that every non-nested **F**-formula occurs at most once on every path. The inserted chains contain at most one state per nested **F**-formula and do not introduce cycles.

▶ **Example 25.** Consider the formula

$$\phi := \mathbf{F}_{\geq 1/2}(\mathbf{G}_{=1}a) \wedge \mathbf{G}_{=1}(\mathbf{F}_{=1}\neg a \vee \mathbf{F}_{=1}b).$$

Figure 6 (a) shows a model for ϕ. The boxes illustrate the selection of s_0. Figure 6 (b) shows the corresponding reduced chain. However, it is not a model for ϕ. The reason is that neither states satisfying $\neg a$ nor such that satisfy b are reached almost surely from s_0. By including additional states, the chain in Figure 6 (c) corrects this, and thereby we obtain a model of ϕ.

3.6 Finite Models for $\mathbf{F}_q, \mathbf{G_1}, \vee$

In this section, we discuss PCTL($\mathbf{F}, \mathbf{G}$) where **G** appears only with the contraint $q = 1$. Previously for the $\mathbf{F}_q, \mathbf{G}_1$-fragment, i.e. without disjunctions, we started from a model which was unfolded for a number of steps; we simplified such a model by dropping states (including all non-bottom SCCs) and then we inserted simple chains that guarantee the satisfaction of the **G**-nested **F**-formulae. The resulting model thus (i) does not contain any non-bottom SCCs and (ii) the size only depends on the structure of the formula, not on the constraints of the **F**-formulae. However, in the general $\mathbf{F}_q, \mathbf{G}_1, \vee$-fragment, we cannot insert such simple

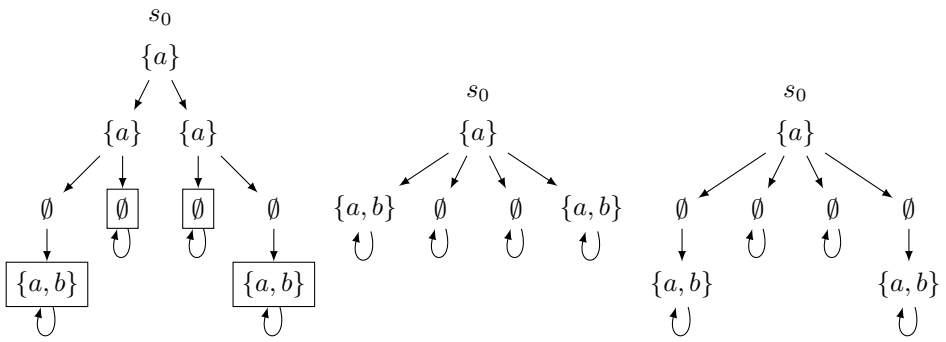


Figure 6 Example of a reduction in the $\mathbf{F_{q/1}}, \mathbf{G_1}, \vee$ fragment: (a) original model, (b) reduced model, (c) corrected model.

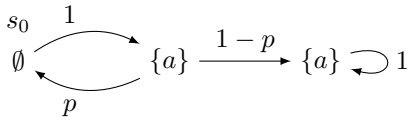


Figure 7 Example model for ϕ.

chains to satisfy nested $\mathbf{F}$s. Instead, we may have to branch at several places. Intuitively, the reason for such complications is the presence of a *repeated, controlled choice*. This enables us to find a formula which requires more complicated models, namely models which either have non-bottom SCCs or are of size which also depends on the constraints and not only on the structure of the formula, or can even be infinite.

▶ **Example 26.** Consider $\phi := \mathbf{G}_{=1}(\mathbf{F}_{=1}(a \wedge \mathbf{F}_{>0}\neg a) \vee a) \wedge \mathbf{F}_{=1}\mathbf{G}_{=1}a \wedge \neg a$. We can try to construct a model for ϕ as follows: Firstly, we have to start at a state which satisfies $\neg a$. This enforces the satisfaction of the first disjunct in the $\mathbf{G}$-formula. Therefore, almost all paths must lead to a state satisfying $a \wedge \mathbf{F}_{>0}\neg a$. This state must eventually reach a state that satisfies $\neg a$ again with positive probability. Hence, we find ourselfs in the same situation as was the case in the initial state. So, we need to either create an SCC of alternating states that satisfy a and $\neg a$, or create an infinite model. If we create an SCC, the side constraint $\mathbf{F}_{=1}\mathbf{G}_{=1}a$ enforces us to eventually leave this SCC. Hence, it is a non-bottom SCC. The MC in Figure 7 is a possible model. From s_0 it models ϕ, for any $p \in (0, 1)$.

Note that the formula given in the example is qualitative. For this fragment, it is known how to solve the satisfiability problem already [7]. However, we can easily adapt the formula to be quantitative. In this case, we might still be able to obtain a model for the quantitative version by keeping its shape and only adapting the probabilties of the model for the qualitative version. The question arises, whether this is possible in general or not. This question remains open and might be interesting for future work.

4 Discussion, Conclusion, and Future Work

We have identified the pattern of the *controlled repeated choice*, i.e. formulae of the form

$$\mathbf{G}_{=1}(\phi_1 \vee \cdots \vee \phi_n)$$

where at least one of the ϕ_i contains an $\mathbf{F}$-formula that has a constraint other than $q = 1$. Additionally, we have "controlling" side constraints, as in Example 26. We have seen that the

presence of this pattern enforces more complicated structure of models even in the qualitative setting. This pattern is expressible in the $\mathbf{F}_q, \mathbf{G}_1, \vee$-fragment. Whenever we

(a) drop the side constraints, keeping only the $\mathbf{G}_{=1}$-part, i.e. consider the $\mathbf{G}_1(\mathbf{F}_q, \mathbf{G}_1, \vee)$-fragment, or

(b) drop the disjunction for the choice and consider the $\mathbf{F}_q, \mathbf{G}_1$-fragment, or

(c) drop the quantity of the choice and consider the $\mathbf{F}_{q/1}, \mathbf{G}_1, \vee$-fragment,

the structure is simpler and we obtain decidability. For these fragments, we have even shown that the general satisfiability problem is equivalent to the finite satisfiability problem.

Further, adding quantities to $\mathbf{G}$-constraints obviously also makes the satisfiability problem more complicated. Already for the qualitative $\mathbf{G}_{>0}(\mathbf{F}_{>0})$-fragment, satisfiability and finite satisfiability differ. Nevertheless, we established the decidability of finite satisfiability even for the $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$-fragment.

Consequently, instead of attacking the whole quantitative PCTL or even just PCTL$(\mathbf{F}, \mathbf{G})$, we suggest two easier tasks, which should lead to a fundamental increase in understanding the general problem, namely:

- finite (and also general) satisfiability of the $\mathbf{F}_q, \mathbf{G}_1, \vee$-fragment, i.e. PCTL$(\mathbf{F}, \mathbf{G})$ where $\mathbf{G}$ is limited to the $= 1$ constraint, and
- infinite satisfiability of the $\mathbf{G}_q(\mathbf{F}_q, \mathbf{G}_q, \vee)$-fragment, i.e. $\mathbf{G}$-formulae of PCTL$(\mathbf{F}, \mathbf{G})$.

While the former omits issues stemming from $\mathbf{G}_{>0}$ [7] and only deals with the repeated choice, the latter generalizes the qualitative results for $\mathbf{G}_{>0}, \mathbf{G}_{=1}$ [17, 7] in the presence of general quantitative $\mathbf{F}$'s.

Further, potentially more straight-forward directions include the generalization of the results obtained in this paper to the until- and release-operators instead of future- and globally-operators, respectively, or the introduction of the next-operator.

References

1 C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *Proceedings of IFIP TCS'2004*, pages 493–506. Kluwer, 2004.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

3 B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, volume 398 of *LNCS*, pages 62–74. Springer, 1987.

4 Nathalie Bertrand, John Fearnley, and Sven Schewe. Bounded satisfiability for PCTL. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 92–106, 2012.

5 T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Stochastic games with branching-time winning objectives. In *Proceedings of LICS 2006*, pages 349–358. IEEE, 2006.

6 T. Brázdil, V. Forejt, and A. Kučera. Controller synthesis and verification for Markov decision processes with qualitative branching time objectives. In *Proceedings of ICALP 2008*, LNCS. Springer, 2008.

7 T. Brázdil, V. Forejt, J. Křetínský, and A. Kučera. The satisfiability problem for probabilistic CTL. In *LICS*, pages 391–402. IEEE, 2008. doi:10.1109/LICS.2008.21.

8 T. Brázdil, A. Kučera, and O. Stražovský. On the decidability of temporal properties of probabilistic pushdown automata. In Diekert and Durand [11], pages 145–157.

9 Souymodip Chakraborty and Joost-Pieter Katoen. On the satisfiability of some simple probabilistic logics. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 56–65. ACM, 2016.

10 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *JACM*, 42(4):857–907, 1995.

11 Volker Diekert and Bruno Durand, editors. *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*. Springer, 2005.

12 E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of STOC'82*, pages 169–180. ACM, 1982.

13 J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2, 2006. doi:10.2168/LMCS-2(1:2)2006.

14 K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. In Diekert and Durand [11], pages 340–352.

15 M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *JCSS*, 18:194–211, 1979.

16 H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *FAC*, 6:512–535, 1994.

17 S. Hart and M. Sharir. Probabilistic propositional temporal logics. *Information and Control*, 70(2/3):97–155, 1986.

18 T. Henzinger, O. Kupferman, and R. Majumdar. On the universal and existential fragments of the μ-calculus. *TCS*, 354(2):173–186, 2006.

19 M. Huth and M.Z. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of LICS'97*, pages 111–122. IEEE, 1997.

20 D. Kozen. A finite-model theorem for the propositional μ-calculus. *Studia Logica*, 47(3):233–241, 1988.

21 S. Kraus and D. J. Lehmann. Decision procedures for time and chance (extended abstract). In *FOCS*, pages 202–209. IEEE, 1983.

22 A. Kučera and O. Stražovský. On the controller synthesis for finite-state Markov decision processes. In *Proceedings of FST&TCS 2005*, volume 3821 of *LNCS*, pages 541–552. Springer, 2005.

23 O. Kupferman and M.Y. Vardi. An automata-theoretic approach to modular model checking. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22:87–128, 2000.

24 Jan Křetínský and Alexej Rotar. The satisfiability problem for unbounded fragments of probabilistic CTL. Technical report, arXiv.org, 2018. arXiv:1806.11418.

25 D. J. Lehmann and S. Shelah. Reasoning with time and chance (extended abstract). In *ICALP*, volume 154 of *LNCS*, pages 445–457. Springer, 1983.

26 J.R. Norris. *Markov Chains*. Cambridge, 1998.

Automatic Analysis of Expected Termination Time for Population Protocols

Michael Blondin[1]
TU Munich, Germany
blondin@in.tum.de
https://orcid.org/0000-0003-2914-2734

Javier Esparza[2]
TU Munich, Germany
esparza@in.tum.de
https://orcid.org/0000-0001-9862-4919

Antonín Kučera[3]
Masaryk University, Brno, Czech Republic
kucera@fi.muni.cz
https://orcid.org/0000-0002-6602-8028

Abstract

Population protocols are a formal model of sensor networks consisting of identical mobile devices. Two devices can interact and thereby change their states. Computations are infinite sequences of interactions in which the interacting devices are chosen uniformly at random.

In well designed population protocols, for every initial configuration of devices, and for every computation starting at this configuration, all devices eventually agree on a consensus value. We address the problem of automatically computing a parametric bound on the expected time the protocol needs to reach this consensus. We present the first algorithm that, when successful, outputs a function $f(n)$ such that the expected time to consensus is bound by $\mathcal{O}(f(n))$, where n is the number of devices executing the protocol. We experimentally show that our algorithm terminates and provides good bounds for many of the protocols found in the literature.

2012 ACM Subject Classification Theory of computation → Distributed computing models, Theory of computation → Probabilistic computation, Theory of computation → Logic and verification

Keywords and phrases population protocols, performance analysis, expected termination time

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.33

Related Version Extended version of the paper: [9], `https://arxiv.org/abs/1807.00331`.

Supplement Material Source code: `https://github.com/blondimi/pp-time-analysis`.

[1] Supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT).
[2] Supported by ERC Advanced Grant (787367: PaVeS).
[3] Supported by the Czech Science Foundation, grant No. P202/12/G061. The presented results were achieved during the author's stay at TU München supported by the Friedrich Wilhelm Bessel Research Award (Alexander von Humboldt Foundation).

1 Introduction

Population protocols are a model of distributed computation in which agents with very limited computational resources randomly interact in pairs to perform computational tasks [3, 4]. They have been used as an abstract model of wireless networks, chemical reactions, and gene regulatory networks, and it has been shown that they can be implemented at molecular level (see, e.g., [23, 21, 11, 20]).

Population protocols compute by reaching a stable consensus in which all agents agree on a common output (typically a Boolean value). The output depends on the distribution of the initial states of the agents, called the initial *configuration*, and so a protocol computes a predicate that assigns a Boolean value to each initial configuration. For example, a protocol in which all agents start in the same state computes the predicate $x \geq c$ if the agents agree to output 1 when there are at least c of them, and otherwise agree to output 0. A protocol with two initial states computes the majority predicate $x \geq y$ if the agents agree to output 1 exactly when the initial number of agents in the first state is greater than or equal to the initial number of agents in the second state.

In previous work, some authors have studied the automatic verification of population protocols. Since a protocol has a finite state space for each initial configuration, model checking algorithms can be used to verify that the protocol behaves correctly for *a finite number* of initial configurations. However, this technique cannot prove that the protocol is correct for *every* configuration. In [16] it was shown that the problem of deciding whether a protocol computes some predicate, and the problem of deciding whether it computes a given predicate, are both decidable and at least as hard as the reachability problem for Petri nets.

In practice, protocols should not only correctly compute a predicate, but also do it fast. The most studied quantitative measure is the expected number of pairwise interactions needed to reach a stable consensus. The measure is defined for the stoichiometric model in which the pair of agents of the next interaction are picked uniformly at random. A derived measure is the *parallel time*, defined as the number of interactions divided by the number of agents. The first paper on population protocols already showed that every predicate can be computed by a protocol with expected total number of interactions $\mathcal{O}(n^2 \log n)$, where n is the number of agents [3, 4]. Since then, there has been considerable interest in obtaining upper and lower bounds on the number of interactions for some fundamental tasks, like *leader election* and *majority*, and there is also much work on finding trade-offs between the speed of a protocol and its number of states (see, e.g., [14, 1, 6] and the references therein). However, none of these works addresses the verification [10] problem: given a protocol, determine its expected number of interactions.

As in the qualitative case, probabilistic model checkers can be used to compute the expected number of interactions for a given configuration. Indeed, in this case the behaviour of the protocol is captured by a finite-state Markov chain, and the expected number of interactions can be computed as the expected number of steps until a bottom strongly connected component of the chain is reached. This was the path followed in [12], using the PRISM probabilistic model checker. However, as in the functional case, this technique cannot give a bound valid for *every* configuration.

This paper presents the first algorithm for the automatic computation of an upper bound on the expected number of interactions. The algorithm takes advantage of the hierarchical structure of population protocols where an initial configuration reaches a stable consensus by passing through finitely many "stages". Entering a next stage corresponds to entering a configuration where some behavioral restrictions become permanent (for example, some

interactions become permanently disabled, certain states will never be populated again, etc.). The algorithm automatically identifies such stages and computes a finite acyclic *stage graph* representing the protocol evolution. If all bottom stages of the graph correspond to stabilized configurations, the algorithm proceeds by deriving bounds for the expected number of interactions to move from one stage to the next, and computes a bound for the expected number of interactions by taking an "asymptotic maximum" of these bounds. In unsuitable cases, the resulting upper bound can be higher than the actual expected number of interactions. We report on an implementation of the algorithm and its application to case studies.

Related work. To the best of our knowledge, we present the first algorithm for the automatic quantitative verification of population protocols. In fact, even for sequential randomized programs, the automatic computation of the expected time is little studied. After the seminal work of Flajolet *et al.* in [17], there is recent work by Kaminski *et al.* [19] on the computation of expected runtimes using weakest preconditions, by Chatterje *et al.* on the automated analysis of recurrence relations for expected time [10], by Van Chan Ngo *et al.* [22] on the automated computation of bounded expectations using amortized resource analysis, and by Batz *et al.* [5, 22] on the computation of sampling times for Bayesian networks. These works are either not targeted to distributed systems like population protocols, or do not provide the same degree of automation as ours.

Structure of the paper. In Section 2, we introduce population protocols and a simple modal logic to reason about their behaviours. In Section 3, we introduce stage graphs and explain how they allow to prove upper bounds on the expected number of interactions of population protocols. We then give a dedicated algorithm for the computation of stage graphs in Section 4, analyze the bounds derived by this algorithm in Section 5, and report on experimental results in Section 6. Finally, we conclude in Section 7. Due to space constraints, we defer proofs of some statements to a full version of this paper [9].

2 Population protocols

In this section, we introduce population protocols and their semantics. We assume familiarity with basic notions of probability theory, such as probability space, random variables, expected value, etc. When we say that some event happens *almost surely*, we mean that the probability of the event is equal to one. We use $\mathbb{N}$ to denote the set of non-negative integers.

A *population* consists of n agents with states from a finite set $Q = \{A, B, \ldots\}$ interacting according to a directed *interaction graph* $\mathcal{G}$ (without self-loops) over the agents. The interaction proceeds in a sequence of steps, where in each step an edge of the interaction graph is selected uniformly at random, and the states (A, B) of the two chosen agents are updated according to a transition function containing rules of the form $(C, D) \mapsto (E, F)$. We assume that for each pair of states (C, D), there is at least one rule $(C, D) \mapsto (E, F)$. If there are several rules with the same left-hand side, then one is selected uniformly at random. The unique agent identifiers are not known to the agents and not used by the protocol.

Usually, $\mathcal{G}$ is considered as a *complete* graph, and this assumption is adopted also in this work. Since the agent identifiers are hidden and $\mathcal{G}$ is complete, a population is fully determined by the number of agents in each state. Formally, a *configuration* is a vector $\mathcal{C} \in \mathbb{N}^Q$, where $\mathcal{C}(A)$ is the number of agents in state A. For every $A \in Q$, we use $\mathbf{1}_A$ to

denote the vector satisfying $\mathbf{1}_A(A) = 1$ and $\mathbf{1}_A(B) = 0$ for all $B \neq A$. Note that there is no difference between transitions $(A, B) \mapsto (C, D)$ and $(A, B) \mapsto (D, C)$, because both of them update a given configuration in the same way.

Most of the population protocols studied for complete interaction graphs have a symmetric transition function where pairs (A, B) and (B, A) are updated in the same way. For the sake of simplicity, we restrict our attention to symmetric protocols.[4] Then, the transitions can be written simply as $AB \mapsto CD$, because the ordering of states before/after the $\mapsto$ symbol is irrelevant. Formally, AB and CD are understood as elements of $Q^{\langle 2 \rangle}$, i.e., multisets over Q with precisely two elements.

▶ **Definition 1.** A *population protocol* is a tuple $\mathcal{P} = (Q, T, \Sigma, I, O)$ where

- Q is a non-empty finite set of *states*;
- $T : Q^{\langle 2 \rangle} \times Q^{\langle 2 \rangle}$ is a total *transition relation*;
- Σ is a non-empty finite *input alphabet*,
- $I : \Sigma \to Q$ is the *input function* mapping input symbols to states,
- $O : Q \to \{0, 1\}$ is the *output function*.

We write $AB \mapsto CD$ to indicate that $(AB, CD) \in T$. When defining the set T, we usually specify the outgoing transitions only for some subset of $Q^{\langle 2 \rangle}$. For the other pairs AB, there (implicitly) exists a single *idle* transition $AB \mapsto AB$. We also write $I(\Sigma)$ to denote the set $\{q \in Q \mid q = I(\sigma) \text{ for some } \sigma \in \Sigma\}$.

2.1 Executing population protocols

A transition $AB \mapsto CD$ is *enabled* in a configuration $\mathcal{C}$ if $\mathcal{C} - \mathbf{1}_A - \mathbf{1}_B \geq \mathbf{0}$. A transition $AB \mapsto CD$ enabled in $\mathcal{C}$ can *fire* and thus produce a configuration $\mathcal{C}' = \mathcal{C} - \mathbf{1}_A - \mathbf{1}_B + \mathbf{1}_C + \mathbf{1}_D$. The *probability* of executing a transition $AB \mapsto CD$ enabled in $\mathcal{C}$ is defined by

$$\mathbb{P}[\mathcal{C}, AB \mapsto CD] = \begin{cases} \frac{\mathcal{C}(A) \cdot (\mathcal{C}(A) - 1)}{(n^2 - n) \cdot |\{EF \in Q^{\langle 2 \rangle} : AA \mapsto EF\}|} & \text{if } A = B\,, \\ \frac{2 \cdot \mathcal{C}(A) \cdot \mathcal{C}(B)}{(n^2 - n) \cdot |\{EF \in Q^{\langle 2 \rangle} : AB \mapsto EF\}|} & \text{if } A \neq B\,. \end{cases}$$

where n is the size of $\mathcal{C}$. Note that $2 \cdot \mathcal{C}(A) \cdot \mathcal{C}(B)$ is the number of directed edges connecting agents in states A and B (when $A \neq B$), and $n^2 - n$ is the total number of directed edges in a complete directed graph without self-loops with n vertices. If a pair of agents in states A and B is selected, one of the outgoing transitions of AB is chosen uniformly at random.

We write $\mathcal{C} \to \mathcal{C}'$ to indicate that $\mathcal{C}'$ is obtained from $\mathcal{C}$ by firing some transition, and we use $\mathbb{P}[\mathcal{C} \to \mathcal{C}']$ to denote the probability of executing a transition enabled in $\mathcal{C}$ producing $\mathcal{C}'$. Note that there can be several transitions enabled in $\mathcal{C}$ producing $\mathcal{C}'$, and $\mathbb{P}[\mathcal{C} \to \mathcal{C}']$ is the total probability of executing some of them.

An *execution* initiated in a given configuration $\mathcal{C}$ is a finite sequence $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$ of configurations such that $\ell \in \mathbb{N}$, $C_0 = \mathcal{C}$, and $\mathcal{C}_i \to \mathcal{C}_{i+1}$ for all $i < \ell$. A configuration $\mathcal{C}'$ is *reachable* from a configuration $\mathcal{C}$ if there is an execution initiated in $\mathcal{C}$ ending in $\mathcal{C}'$. A *run* is an infinite sequence of configurations $\omega = \mathcal{C}_0, \mathcal{C}_1, \ldots$ such that every finite prefix of ω is an execution. The configuration $\mathcal{C}_i$ of a run ω is also denoted by ω_i. For a given execution $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$, we use $\mathit{Run}(\mathcal{C}_0, \ldots, \mathcal{C}_\ell)$ to denote the set of all runs starting with $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$.

[4] All of the presented results can easily be extended to non-symmetric population protocols. The only technical difference is the way of evaluating/estimating the probability of executing a given transition in a given configuration.

For every configuration $\mathcal{C}$, we define the probability space $(Run(\mathcal{C}), \mathcal{F}, \mathbb{P}_{\mathcal{C}})$, where $\mathcal{F}$ is the σ-algebra generated by all $Run(\mathcal{C}_0, \ldots, \mathcal{C}_\ell)$ such that $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$ is an execution initiated in $\mathcal{C}$, and $\mathbb{P}_{\mathcal{C}}$ is the unique probability measure satisfying $\mathbb{P}_{\mathcal{C}}(Run(C_0, \ldots, C_\ell)) = \prod_{i=0}^{\ell-1} \mathbb{P}[\mathcal{C}_i \to \mathcal{C}_{i+1}]$.

2.2 A simple modal logic for population protocols

To specify properties of configurations, we use a qualitative variant of the branching-time logic EF. Let $AP_{\mathcal{P}} = Q \cup \{A! \mid A \in Q$ such that there is a non-idle transition $AA \mapsto BC\}$. The formulae of our qualitative logic are constructed in the following way, where a ranges over $AP_{\mathcal{P}} \cup \{Out_0, Out_1\}$:

$$\varphi \quad ::= \quad a \mid \neg\varphi \mid \varphi_0 \wedge \varphi_1 \mid \Box\varphi \mid \Diamond\varphi.$$

The semantics is defined inductively:

$\mathcal{C} \models A$	iff	$\mathcal{C}(A) > 0$,
$\mathcal{C} \models A!$	iff	$\mathcal{C}(A) = 1$,
$\mathcal{C} \models Out_0$	iff	$O(A) = 0$ for all $A \in Q$ such that $\mathcal{C}(A) > 0$,
$\mathcal{C} \models Out_1$	iff	$O(A) = 1$ for all $A \in Q$ such that $\mathcal{C}(A) > 0$,
$\mathcal{C} \models \neg\varphi$	iff	$\mathcal{C} \not\models \varphi$,
$\mathcal{C} \models \varphi_0 \wedge \varphi_1$	iff	$\mathcal{C} \models \varphi_0$ and $\mathcal{C} \models \varphi_1$,
$\mathcal{C} \models \Box\varphi$	iff	$\mathbb{P}_{\mathcal{C}}(\{\omega \in Run(\mathcal{C}) \mid \omega_i \models \varphi$ for all $i \in \mathbb{N}\}) = 1$,
$\mathcal{C} \models \Diamond\varphi$	iff	$\mathbb{P}_{\mathcal{C}}(\{\omega \in Run(\mathcal{C}) \mid \omega_i \models \varphi$ for some $i \in \mathbb{N}\}) = 1$.

Note that $\mathcal{C} \models \Box\varphi$ iff all configurations reachable from $\mathcal{C}$ satisfy φ, and $\mathcal{C} \models \Diamond\varphi$ iff a run initiated in $\mathcal{C}$ visits a configuration satisfying φ almost surely (i.e., with probability one). We also use `tt`, `ff`, and other propositional connectives whose semantics is defined in the standard way. Furthermore, we occasionally interpret a given set of configurations $\mathcal{B}$ as a formula where $\mathcal{C} \models \mathcal{B}$ iff $\mathcal{C} \in \mathcal{B}$.

For every formula φ, we define a random variable $Steps_\varphi$ assigning to every run $\mathcal{C}_0, \mathcal{C}_1, \ldots$ either the least $\ell \in \mathbb{N}$ such that $\mathcal{C}_\ell \models \varphi$, or ∞ if there is no such ℓ. For a given configuration $\mathcal{C}$, we use $\mathbb{E}_{\mathcal{C}}[Steps_\varphi]$ to denote the expected value of $Steps_\varphi$ in the probability space $(Run(\mathcal{C}), \mathcal{F}, \mathbb{P}_{\mathcal{C}})$.

2.3 Computable predicates, interaction complexity

Every *input* $\mathcal{X} \in \mathbb{N}^\Sigma$ is mapped to the configuration $\mathcal{C}_\mathcal{X}$ such that

$$\mathcal{C}_\mathcal{X}(q) = \sum_{\substack{\sigma \in \Sigma \\ I(\sigma) = q}} \mathcal{X}(\sigma) \qquad \text{for every } q \in Q.$$

An *initial* configuration is a configuration of the form $\mathcal{C}_\mathcal{X}$ where $\mathcal{X}$ is an input. A configuration $\mathcal{C}$ is *stable* if $\mathcal{C} \models Stable$, where $Stable \equiv (\Box Out_0) \vee (\Box Out_1)$. We say that a protocol $\mathcal{P}$ *terminates* if $\mathcal{C} \models \Diamond Stable$ for every initial configuration $\mathcal{C}$. A protocol $\mathcal{P}$ *computes* a unary predicate Λ on inputs if it terminates and every stable configuration $\mathcal{C}'$ reachable from an initial configuration $\mathcal{C}_\mathcal{X}$ satisfies $\mathcal{C}' \models Out_x$, where x is either 1 or 0 depending on whether $\mathcal{X}$ satisfies Λ or not, respectively.

The *interaction complexity* of $\mathcal{P}$ is a function $InterComplexity_\mathcal{P}$ assigning to every $n \geq 1$ the *maximal* $\mathbb{E}_{\mathcal{C}}[Steps_{Stable}]$, where $\mathcal{C}$ ranges over all initial configurations of size n. Since several interactions may be running in parallel, the *time complexity* of $\mathcal{P}$ is defined as

$InterComplexity_{\mathcal{P}}(n)$ divided by n. Hence, asymptotic bounds on interaction complexity immediately induce the corresponding bounds on time complexity.

2.4 Running examples

A well-studied predicate for population protocols is *majority*. Here, $\Sigma = \{A, B\}$, $I(A) = A$, $I(B) = B$, and the protocol computes whether there are at least as many agents in state B as there are in state A. As running examples, we use two different protocols for computing majority, taken from [15] and [18].

▶ **Example 2** (majority protocol of [15])**.** We have that $Q = \{A, B, a, b\}$, $O(A) = O(a) = 0$, $O(B) = O(b) = 1$, and the transitions are the following: $AB \mapsto ab$, $Ab \mapsto Aa$, $Ba \mapsto Bb$ and $ba \mapsto bb$.

▶ **Example 3** (majority protocol of [18])**.** Here, $Q = \{A, B, C, a, b\}$, $O(A) = O(a) = 0$, $O(B) = O(b) = O(C) = 1$, and the transitions are the following: $AB \mapsto bC$, $AC \mapsto Aa$, $BC \mapsto Bb$, $Ba \mapsto Bb$, $Ab \mapsto Aa$ and $Ca \mapsto Cb$.

3 Stages of population protocols

Most of the existing population protocols are designed so that each initial configuration passes through finitely many "stages" before reaching a stable configuration. Entering a next stage corresponds to performing some additional non-reversible changes in the structure of configurations. Hence, the transition relation between stages is acyclic, and each configuration in a non-terminal stage eventually enters one of the successor stages with probability one. This intuition is formalized in our next definition.

▶ **Definition 4.** Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. A *stage graph* for $\mathcal{P}$ is a triple $\mathcal{G} = (\mathbb{S}, \hookrightarrow, [\![\cdot]\!])$ where $\mathbb{S}$ is a finite set of *stages*, $\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$ is an acyclic transition relation, and $[\![\cdot]\!]$ is a function assigning to each $S \in \mathbb{S}$ a set of configurations $[\![S]\!]$ such that the following conditions are satisfied:

(a) For every initial configuration $\mathcal{C}$ there is some $S \in \mathbb{S}$ such that $\mathcal{C} \in [\![S]\!]$.

(b) For every $S \in \mathbb{S}$ with at least one successor under $\hookrightarrow$, and for every $\mathcal{C} \in [\![S]\!]$, we have that[5] $\mathcal{C} \models \Diamond Succ(S)$, where $Succ(S) \equiv \bigvee_{S \hookrightarrow S'} [\![S']\!]$.

Note that a stage graph for $\mathcal{P}$ is not determined uniquely. Even a trivial graph with one stage S and no transitions such that $[\![S]\!]$ is the set of all configurations is a valid stage graph by Definition 4. To analyze the interaction complexity of $\mathcal{P}$, we need to construct a stage graph so that the expected number of transitions needed to move from stage to stage can be determined easily, and all terminal stages consist only of stable configurations (see Lemma 6 below).

Formally, a stage S is *terminal* if it does not have any successors, i.e., there is no S' satisfying $S \hookrightarrow S'$. Let $\mathcal{T}$ be the set of all terminal stages, and let $Term \equiv \bigvee_{S \in \mathcal{T}} [\![S]\!]$. It follows directly from Definition 4(b) that $\mathcal{C} \models \Diamond Term$ for every initial configuration $\mathcal{C}$. Let $ReachTerminal_{\mathcal{G}}$ be a function assigning to every $n \geq 1$ the maximal $\mathbb{E}_{\mathcal{C}}[Steps_{Term}]$, where $\mathcal{C}$ ranges over all initial configurations of size n. Furthermore, for every $S \in \mathbb{S}$, we define a

[5] Recall that sets of configurations can be interpreted as formulae of the modal logic introduced in Section 2.2.

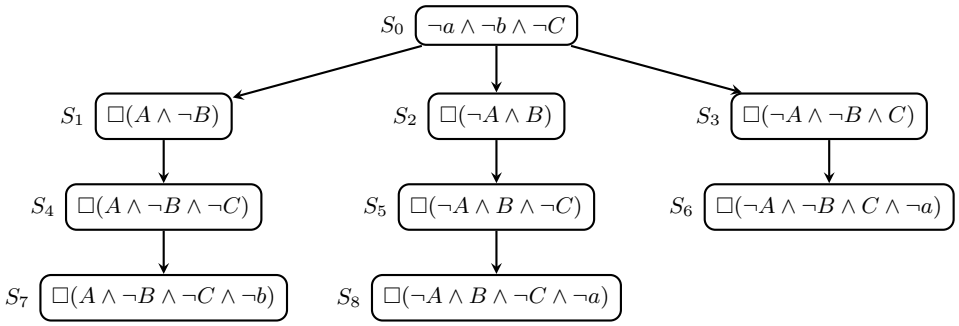


Figure 1 A stage graph for the majority protocol of Example 3.

function $\mathit{ReachNext}_S$ assigning to every $n \geq 1$ the maximal $\mathbb{E}_{\mathcal{C}}[\mathit{Steps}_{\mathit{Succ}(S)}]$, where $\mathcal{C}$ ranges over all configurations of $[\![S]\!]$ of size n (if $[\![S]\!]$ does not contain any configuration of size n, we put $\mathit{ReachNext}_S(n) = 0$).

An asymptotic upper bound for $\mathit{ReachTerminal}_{\mathcal{G}}$ can be obtained by developing an asymptotic upper bound for all $\mathit{ReachNext}_S$, where $S \in \mathbb{S}$. Even though such a bound on $\mathit{ReachTerminal}_{\mathcal{G}}$ depends on $|\mathbb{S}|$, the latter is a constant since it is independent from the number of agents. Therefore, the following holds:

▶ **Lemma 5.** *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol and $\mathcal{G} = (\mathbb{S}, \hookrightarrow, [\![\cdot]\!])$ a stage graph for $\mathcal{P}$. Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that $\mathit{ReachNext}_S \in \mathcal{O}(f)$ for all $S \in \mathbb{S}$. Then $\mathit{ReachTerminal}_{\mathcal{G}} \in \mathcal{O}(f)$.*

Observe that if every terminal stage S satisfies $[\![S]\!] \subseteq \mathit{Stable}$, then $\mathit{InterComplexity}_{\mathcal{P}} \leq \mathit{ReachTerminal}_{\mathcal{G}}$ (pointwise). Thus, we obtain the following:

▶ **Lemma 6.** *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol and $\mathcal{G} = (\mathbb{S}, \hookrightarrow, [\![\cdot]\!])$ a stage graph for $\mathcal{P}$ such that $[\![S]\!] \subseteq \mathit{Stable}$ for every terminal stage S. Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that $\mathit{ReachNext}_S \in \mathcal{O}(f)$ for all $S \in \mathbb{S}$. Then $\mathit{InterComplexity}_{\mathcal{P}} \in \mathcal{O}(f)$.*

3.1 An example of a stage graph

In this section, we give an example of a stage graph $\mathcal{G}$ for the majority protocol $\mathcal{P}$ of Example 3, and we show how to analyze the interaction complexity of $\mathcal{P}$ using $\mathcal{G}$.

The stage graph $\mathcal{G}$ of Fig. 1 is a simplified version of the stage graph computed by the algorithm of the forthcoming Section 4. Intuitively, the hierarchy of stages corresponds to "disabling more and more states" along runs initiated in initial configurations. For each stage S_i of $\mathcal{G}$, the set $[\![S_i]\!]$ consists of all configurations satisfying the associated formula shown in Fig. 1. Since $[\![S_0]\!]$ is precisely the set of all initial configurations, Condition (a) of Definition 4 is satisfied. For every $\mathcal{C}_0 \in [\![S_0]\!]$, transition $AB \mapsto bC$ can be executed in all configurations reachable from $\mathcal{C}_0$ until A or B disappears. Furthermore, the number of A's and B's can only decrease along every run initiated in $\mathcal{C}_0$. Hence, $\mathcal{C}_0$ almost surely reaches a configuration $\mathcal{C}$ where A or B (or both of them) disappear. Note that if, e.g., $\mathcal{C}(A) = 0$ and $\mathcal{C}(B) > 0$, then this property is "permanent", i.e., every successor $\mathcal{C}'$ of $\mathcal{C}$ also satisfies $\mathcal{C}'(A) = 0$ and $\mathcal{C}'(B) > 0$. Thus, we obtain the stages S_1, S_2, and S_3. Observe that if A and B disappear simultaneously (which happens iff the initial configuration $\mathcal{C}_0$ satisfies $\mathcal{C}_0(A) = \mathcal{C}_0(B)$), then the configuration $\mathcal{C}$ will contain at least one copy of C which cannot be removed.

In all configurations of $[\![S_1]\!]$, the only potentially executable transitions are the following: $AC \mapsto Aa$, $Ab \mapsto Aa$, $Ca \mapsto Cb$. Since A appears in all configurations reachable from configurations of $[\![S_1]\!]$, the transition $AC \mapsto Aa$ stays enabled in all of these configurations

until C disappears. Hence, every configuration of $[\![S_1]\!]$ almost surely reaches a configuration of $[\![S_4]\!]$. Similarly, we can argue that all configurations of $[\![S_4]\!]$ almost surely reach a configuration of $[\![S_7]\!]$, etc. Hence, Condition (b) of Definition 4 is also satisfied.

Let $\mathcal{C}_0 \in [\![S_0]\!]$ be an initial configuration of size n, and let $\mathcal{C}$ be a configuration reachable from $\mathcal{C}_0$ such that $m = \min\{\mathcal{C}(A), \mathcal{C}(B)\} > 0$. The probability of firing $AB \mapsto bC$ stays larger than m^2/n^2 in all configurations reached from $\mathcal{C}$ by executing a finite sequence of transitions *different* from $AB \mapsto bC$. This means that $AB \mapsto bC$ is fired after at most n^2/m^2 trials on average. Since $\min\{\mathcal{C}_0(A), \mathcal{C}_0(B)\} \leq n/2$, we obtain

$$\mathit{ReachNext}_{S_0}(n) \quad \leq \quad \sum_{i=1}^{n/2} \frac{n^2}{i^2} \quad \leq \quad n^2 \cdot \sum_{i=1}^{n} \frac{1}{i^2} \quad \leq \quad n^2 \cdot \mathcal{H}_{n,2} \quad \in \quad \mathcal{O}(n^2)\,.$$

Here, $\mathcal{H}_{n,2}$ is the n-th Harmonic number of order 2. As $\lim_{n\to\infty} \mathcal{H}_{n,2} = c < \infty$, we have that $n^2 \cdot \mathcal{H}_{n,2} \in \mathcal{O}(n^2)$.

Now, let us analyze $\mathit{ReachNext}_{S_1}(n)$. Let $\mathcal{C} \in [\![S_1]\!]$ be a configuration of size n. We need to fire the transition $AC \mapsto Aa$ repeatedly until all C's disappear. Let $\mathcal{C}'$ be a configuration reachable from $\mathcal{C}$ such that $\mathcal{C}'(C) = m$. Since $\mathcal{C} \models \Box(A \wedge \neg B)$, we have that $\mathcal{C}'(A) > 0$, and hence the probability of firing $AC \mapsto Aa$ in $\mathcal{C}'$ is at least m/n^2. Thus, we obtain

$$\mathit{ReachNext}_{S_1}(n) \quad \leq \quad \sum_{i=1}^{n} \frac{n^2}{i} \quad \leq \quad n^2 \cdot \sum_{i=1}^{n} \frac{1}{i} \quad \leq \quad n^2 \cdot \mathcal{H}_n \quad \in \quad \mathcal{O}(n^2 \log(n))\,.$$

Here $\mathcal{H}_n$ denotes the n-th Harmonic number (of order 1). Since $\lim_{n\to\infty} \mathcal{H}_n = c \cdot \log(n)$ where c is a constant, we get $n^2 \cdot \mathcal{H}_n \in \mathcal{O}(n^2 \log(n))$.

Similarly, we can show that $\mathit{ReachNext}_{S_i}(n) \in \mathcal{O}(n^2 \log(n))$ for every stage S_i of the considered stage graph. Since all configurations associated to terminal stages are stable, we can apply Lemma 6 and conclude that $\mathit{InterComplexity}_{\mathcal{P}} \in \mathcal{O}(n^2 \log(n))$. Let us note that the algorithm of the forthcoming Section 4 can derive this result fully automatically in less than a second.

4 Computing a stage graph

In this section, we give an algorithm computing a stage graph for a given population protocol. Intuitively, the algorithm tries to identify a subset of transitions which will be simultaneously and permanently disabled in the future with probability one, and also performs a kind of "case analysis" how this can happen. The resulting stage graph admits computing an upper asymptotic bounds on $\mathit{ReachNext}_S$ for every stage S, which allows to compute an asymptotic upper bound on the interaction complexity of the protocol by applying Lemma 6.

For the rest of this section, we fix a population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$. A *valuation* is a *partial* function $\nu : AP_{\mathcal{P}} \to \{\mathtt{tt}, \mathtt{ff}\}$ such that $\nu(A!) = \mathtt{tt}$ implies $\nu(A) = \mathtt{tt}$ whenever $A!, A \in \mathit{Dom}(\nu)$, where $\mathit{Dom}(\nu)$ is the domain of ν. Slightly abusing our notation, we also denote by ν the propositional formula

$$\bigwedge_{\substack{p \in \mathit{Dom}(\nu) \\ \nu(p) = \mathtt{tt}}} p \quad \wedge \quad \bigwedge_{\substack{p \in \mathit{Dom}(\nu) \\ \nu(p) = \mathtt{ff}}} \neg p$$

Hence, by writing $\mathcal{C} \models \nu$ we mean that $\mathcal{C}$ satisfies the above formula.

For every *transition head* $AB \in Q^{\langle 2 \rangle}$, let ξ_{AB} be either the formula $\neg A \vee \neg B$ or the formula $\neg A \vee A!$, depending on whether $A \neq B$ or $A = B$, respectively. Hence, the formulae ξ_{AB} and $\neg\xi_{AB}$ say that all transitions of the form $AB \mapsto CD$ are disabled and enabled,

respectively. For a given set $\mathcal{T} \subseteq Q^{\langle 2 \rangle}$, consider the propositional formula $\Psi_{\mathcal{T}} \equiv \bigwedge_{AB \in \mathcal{T}} \xi_{AB}$. To simplify our notation, we write just $\mathcal{T}$ instead of $\Psi_{\mathcal{T}}$, i.e., $\mathcal{C} \models \mathcal{T}$ iff all transitions specified by $\mathcal{T}$ are disabled in $\mathcal{C}$.

▶ **Definition 7.** Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. A *$\mathcal{P}$-stage* is a triple $S = (\Phi, \pi, \mathcal{T})$ where

- Φ is a propositional formula over $AP_{\mathcal{P}}$,
- π is a valuation, called the *persistent valuation*,
- $\mathcal{T} \subseteq Q^{\langle 2 \rangle}$ is a set of transition heads, called the *permanently disabled transition heads*.

For every $\mathcal{P}$-stage $S = (\Phi, \pi, \mathcal{T})$, we put $[\![S]\!] = \{\mathcal{C} \mid \mathcal{C} \models \Phi \wedge \Box\pi \wedge \Box\mathcal{T}\}$.

Our algorithm computes a stage graph for $\mathcal{P}$ gradually by adding more and more $\mathcal{P}$-stages. It starts by inserting the *initial* $\mathcal{P}$-stage $S_0 = (\Phi, \emptyset, \emptyset)$, where

$$\Phi \;\equiv\; \left(\bigvee_{A \in I(\Sigma)} A \right) \wedge \bigwedge_{A \in Q \setminus I(\Sigma)} \neg A \,.$$

Note that $[\![S_0]\!]$ is precisely the set of all initial configurations (the empty conjunction is interpreted as *true*). Then, the algorithm picks an unprocessed $\mathcal{P}$-stage in the part of the stage graph constructed so far, and computes its immediate successors. This goes on until all $\mathcal{P}$-stages become either internal or terminal. Since the total number of constructed $\mathcal{P}$-stages can be exponential in the size of $\mathcal{P}$, the worst-case complexity of our algorithm is exponential. However, as we shall see in Section 6, protocols with hundreds of states and transitions can be successfully analyzed even by our prototype implementation.

Let $S = (\Phi, \pi, \mathcal{T})$ be a non-terminal $\mathcal{P}$-stage, and let $AP_S \subseteq AP_{\mathcal{P}}$ be the set of all atomic propositions appearing in the formula Φ. The successor $\mathcal{P}$-stages of S are constructed as follows. First, the algorithm computes the set Val_S consisting of all valuations ν with domain AP_S such that ν satisfies Φ when the latter is interpreted over AP_S. Intuitively, this corresponds to dividing $[\![S]\!]$ into disjoint "subcases" determined by different ν's (as we shall see, Φ always implies the formula $\pi \wedge \mathcal{T}$, so ν cannot be in conflict with the information represented by π and $\mathcal{T}$; furthermore, we have $Dom(\pi) \subseteq Dom(\nu)$). Then, for each $\nu \in Val_S$, a $\mathcal{P}$-stage S_ν is constructed, and S_ν may or may not become a successor of S. If none of these S_ν becomes a successor of S, then S is declared as terminal.

Let us fix some $\nu \in Val_S$. In the rest of this section, we show how to compute the $\mathcal{P}$-stage $S_\nu = (\Phi_\nu, \pi_\nu, \mathcal{T}_\nu)$, and how to determine whether or not S_ν becomes a successor of S. An explicit pseudocode for constructing S_ν is given in [9].

4.1 Computing the valuation π_ν

The valuation π_ν is obtained by extending π with the "permanent part" of ν. Intuitively, we try to identify $A \in Q$ such that $\nu(A) = \texttt{tt}$ (or $\nu(A) = \texttt{ff}$) and all transitions containing A on the left-hand (or the right-hand) side are permanently disabled. Furthermore, we also try to identify $A \in Q$ such that $\nu(A!) = \texttt{tt}$ and the number of A's cannot change by firing transitions which are not permanently disabled. Technically, this is achieved by a simple fixed-point computation guaranteed to terminate quickly. The details are given in [9].

4.2 Computing the set $\mathcal{T}_\nu$ and the formula Φ_ν

In some cases, the constructed persistent valuation π_ν already guarantees that a configuration satisfying $\pi_\nu \wedge \mathcal{T}$ is stable or cannot evolve (fire non-idle transitions) any further. Then, we in

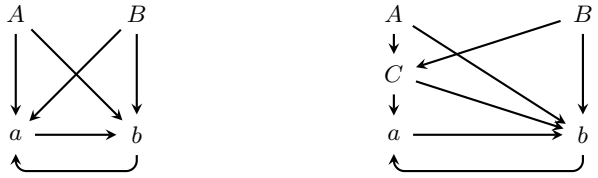


Figure 2 Transformation graphs of Example 8.

fact identified a subset of configurations belonging to $[\![S]\!]$ which does not require any further analysis. Hence, we put $\mathcal{T}_\nu = \mathcal{T}$, $\Phi_\nu = \pi_\nu$, and the configuration S_ν becomes a successor $\mathcal{P}$-stage of S declared as terminal.

Formally, we say that $(\pi_\nu, \mathcal{T})$ is *stable* if there is $x \in \{0, 1\}$ such that for all states $A \in Q$ where $\pi_\nu(A) = \mathtt{tt}$ or $A \notin \mathit{Dom}(\pi_\nu)$ we have that $\mathit{Out}(A) = x$, and for every transition $CD \mapsto EF$ where $\mathit{Out}(E) \neq x$ or $\mathit{Out}(F) \neq x$, the formula $(\pi_\nu \wedge \mathcal{T}) \Rightarrow \xi_{CD}$ is a propositional tautology. Furthermore, we say that $(\pi_\nu, \mathcal{T})$ is *dead* if it is not stable and for every non-idle transition $CD \mapsto EF$ we have that the formula $(\pi_\nu \wedge \mathcal{T}) \Rightarrow \xi_{CD}$ is a propositional tautology.

If S_ν is *not* stable or dead, we use π_ν and $\mathcal{T}$ to compute the *transformation graph* G_ν, and then analyze G_ν to determine $\mathcal{T}_\nu$ and Φ_ν.

4.2.1 The transformation graph

The vertices of the transformation graph G_ν are the states which have *not* yet been permanently disabled according to π_ν, and the edges are determined by a set of transitions whose heads have not yet been permanently disabled according to π_ν and $\mathcal{T}$. Formally, we put $G_\nu = (V, \rightarrow)$ where the set of vertices V consists of all $A \in Q$ such that either $A \notin \mathit{Dom}(\pi_\nu)$ or $\pi_\nu(A) = \mathtt{tt}$, and the set of edges is determined as follows: Let $AB \mapsto CD$ be a non-idle transition such that $(\pi_\nu \wedge \mathcal{T}) \Rightarrow \xi_{AB}$ is *not* a tautology.

- If the sets $\{A, B\}$ and $\{C, D\}$ are disjoint, then the transition generates the edges $A \rightarrow C$, $A \rightarrow D$, $B \rightarrow C$, $B \rightarrow D$. Intuitively, both A and B can be "transformed" into C or D.
- Otherwise, the transition has the form $AB \mapsto AD$ for $B \neq D$. In this case it generates the edge $B \rightarrow D$. Intuitively, B can be "transformed" into D in the context of A.

▶ **Example 8.** Consider the protocol of Example 2 and its initial stage $S = (\Phi, \pi, \mathcal{T})$ where $\Phi = (A \vee B) \wedge \neg a \wedge \neg b$ and $\pi = \mathcal{T} = \emptyset$. Three valuations satisfy Φ; in particular the valuation ν which sets to $\mathtt{tt}$ precisely the variables A and B. Since both A and B can disappear in the future, and both a and b can become populated, the "permanent part" of ν, i.e., the valuation π_ν, has the empty domain. The transformation graph G_ν is shown in Fig. 2 (left).

Consider now the majority protocol of Example 3 with initial stage $(\Phi, \emptyset, \emptyset)$ (where Φ says there are only A's and B's), and a valuation ν which sets to $\mathtt{tt}$ precisely the variables A and B. The domain of π_ν is again the empty set, and the transformation graph G_ν is shown in Fig. 2 (right).

A key observation about transformation graphs is that all transitions generating edges connecting two *different* strongly connected components (SCCs) of G_ν become simultaneously disabled in the future almost surely. More precisely, let Exp_ν be the set of all $AB \in Q^{\langle 2 \rangle}$ such that there exists a transition $AB \mapsto CD$ generating an edge of G_ν connecting two different SCCs of G_ν. We have the following:

▶ **Lemma 9.** *Let G_ν be a transformation graph, and let $\mathcal{C}$ be a configuration such that $\mathcal{C} \models \Box\pi_\nu \wedge \Box\mathcal{T}$. Then $\mathcal{C} \models \Diamond \mathit{Exp}_\nu$. Furthermore, $\mathcal{C} \models \Diamond\Box \mathit{Exp}_\nu$.*

However, there is a subtle problem. When the transitions specified by Exp_ν become simultaneously disabled *for the first time*, they may be disabled only *temporarily*, i.e., $\mathcal{C}$ does *not* have to satisfy the formula $\Box(Exp_\nu \Rightarrow \Box Exp_\nu)$. As we shall see in Section 5, it is relatively easy to obtain an upper bound on the expected number of transitions needed to visit a configuration satisfying Exp_ν. However, it is harder to give an upper bound on the expected number of transitions needed to reach a configuration satisfying $\Box Exp_\nu$ (i.e., entering the next stage) unless $\mathcal{C} \models \Box(Exp_\nu \Rightarrow \Box Exp_\nu)$. This difficulty is addressed in the next section.

▶ **Example 10.** We continue with Example 8. For the transformation graph of Fig. 2 (left), we have $Exp_\nu = \{AB\}$. For the transformation graph of Fig. 2 (right), we have $Exp_\nu = \{AB, AC, BC\}$. Hence, according to Lemma 9, every initial configuration of the majority protocol of Example 2 almost surely reaches a configuration satisfying $\neg A \vee \neg B$, and every initial configuration of the majority protocol of Example 3 almost surely reaches a configuration satisfying $(\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg C)$. Furthermore, in both cases $\mathcal{C} \models \Box(Exp_\nu \Rightarrow \Box Exp_\nu)$ for every initial configuration $\mathcal{C}$.

4.2.2 Computing $\mathcal{T}_\nu$ and Φ_ν: Case $Exp_\nu \neq \emptyset$

Let $\Gamma_\nu \equiv \nu \wedge \Box\pi_\nu \wedge \Box\mathcal{T}$, and let $\mathcal{C}$ be a configuration satisfying Γ_ν. A natural idea to construct $\mathcal{T}_\nu$ is to enrich $\mathcal{T}$ by Exp_ν. However, Exp_ν can be empty, i.e., the transformation graph G_ν may consist just of disconnected SCCs. For this reason we first consider the case where Exp_ν is nonempty.

Computing $\mathcal{T}_\nu$. As discussed in Section 4.2.1, the fact that $\mathcal{C} \models \Diamond\Box Exp_\nu$ does not necessarily imply $\mathcal{C} \models \Box(Exp_\nu \Rightarrow \Box Exp_\nu)$ complicates the interaction complexity analysis. Therefore, after computing Exp_ν we try to compute a non-empty subset $\mathcal{J}_\nu \subseteq Exp_\nu$ such that $\mathcal{C} \models \Box(\mathcal{J}_\nu \Rightarrow \Box\mathcal{J}_\nu)$ for all configurations $\mathcal{C}$ satisfying Γ_ν. If we succeed, we put $\mathcal{T}_\nu = \mathcal{T} \cup \mathcal{J}_\nu$. Otherwise, $\mathcal{T}_\nu = \mathcal{T} \cup Exp_\nu$. Intuitively, the set $\mathcal{J}_\nu$ is the largest subset M of Exp_ν such that every element of M can be re-enabled only by firing a transition which has been identified as permanently disabled. This again leads to a simple fixed-point computation, which is detailed in [9].

A proof of the next lemma is straightforward.

▶ **Lemma 11.** *For every configuration $\mathcal{C}$ such that $\mathcal{C} \models \Gamma_\nu$ we have that*
(a) $\mathcal{C} \models \Diamond\Box(\pi_\nu \wedge \mathcal{T} \wedge Exp_\nu)$
(b) $\mathcal{C} \models \Box(\mathcal{J}_\nu \Rightarrow \Box\mathcal{J}_\nu)$
If $\mathcal{J}_\nu \neq \emptyset$, we put $\mathcal{T}_\nu = \mathcal{T} \cup \mathcal{J}_\nu$. Otherwise, we put $\mathcal{T}_\nu = \mathcal{T} \cup Exp_\nu$.

Computing Φ_ν. We say that a configuration $\mathcal{C}$ is *S_ν-entering* if $\mathcal{C} \models \Box\pi_\nu \wedge \Box\mathcal{T}_\nu$ and there is an execution $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$ such that $\mathcal{C}_0 \models \Gamma_\nu$, $\mathcal{C}_\ell = \mathcal{C}$, and $\mathcal{C}_j \not\models \Box\pi_\nu \wedge \Box\mathcal{T}_\nu$ for all $j < \ell$. An immediate consequence of Lemma 11 is the following:

▶ **Lemma 12.** *Almost all runs initiated in a configuration satisfying Γ_ν visit an S_ν-entering configuration.*

The formula Φ_ν specifies the properties of S_ν-entering configurations. The formula Φ_ν always implies $\pi_\nu \wedge \mathcal{T}_\nu$, but it can also be more detailed if $\mathcal{J}_\nu \neq \emptyset$. More precisely, we say that $\mathcal{J}_\nu$ is *ν-disabled* if $\mathcal{J}_\nu \neq \emptyset$ and for all $AB \in \mathcal{J}_\nu$ we have that $\nu \Rightarrow \xi_{AB}$ is a propositional tautology (i.e., all transitions specified by $\mathcal{J}_\nu$ are disabled in all configurations satisfying ν). Similarly, $\mathcal{J}_\nu$ is *ν-enabled* if $\mathcal{J}_\nu \neq \emptyset$ and there exists $AB \in \mathcal{J}_\nu$ such that $\nu \Rightarrow \neg\xi_{AB}$ is a tautology (i.e., some transition specified by $\mathcal{J}_\nu$ is enabled in all configurations satisfying ν).

Observe that if $\mathcal{J}_\nu$ is ν-disabled, then all transitions specified by $\mathcal{J}_\nu$ are simultaneously disabled in every configuration $\mathcal{C}$ satisfying Γ_ν. Hence, all S_ν-entering configurations satisfy Γ_ν (see Lemma 11 (b)). Now suppose that $\mathcal{J}_\nu$ is ν-enabled, and let Q_ν be the set of all $A \in Q$ such that $AB \in \mathcal{J}_\nu$ for some $B \in Q$. Since for every configuration $\mathcal{C}$ satisfying Γ_ν there is a transition specified by $\mathcal{J}_\nu$ enabled in $\mathcal{C}$, the last transition executed before visiting an S_ν-entering configuration must be a transition "transforming" some $A \in Q_\nu$, i.e., a transition of the form $AB \mapsto CD$ generating an edge $A \to C$ of G_ν. Let $\mathcal{K}_\nu$ be the set of all right-hand sides of all such transitions. The formula Φ_ν is defined as follows:

$$\Phi_\nu \quad \equiv \quad \begin{cases} \pi_\nu \wedge \mathcal{T}_\nu \ \wedge \ \nu & \text{if } \mathcal{J}_\nu \text{ is } \nu\text{-disabled,} \\ \pi_\nu \wedge \mathcal{T}_\nu \ \wedge \ \Big(\bigvee\limits_{CD \in \mathcal{K}_\nu} \neg \xi_{CD} \Big) & \text{if } \mathcal{J}_\nu \text{ is } \nu\text{-enabled,} \\ \pi_\nu \wedge \mathcal{T}_\nu & \text{otherwise.} \end{cases}$$

It is easy to check that every S_ν-entering configuration satisfies the formula Φ_ν. The constructed $\mathcal{P}$-stage $S_\nu = (\Phi_\nu, \pi_\nu, \mathcal{T}_\nu)$ becomes a successor of the $\mathcal{P}$-stage S.

4.2.3 Computing $\mathcal{T}_\nu$ and Φ_ν: Case $Exp_\nu = \emptyset$

In this case G_ν is a collection of disconnected SCCs. We put $\mathcal{T}_\nu = \mathcal{T}$. In the rest of the section we show how to construct the formula Φ_ν.

We say that an edge $A \to B$ of G_ν is *stable* if there is a transition $AC \mapsto BD$ generating $A \to B$ such that $\pi_\nu(C) = \texttt{tt}$. Let $\mathcal{I}_\nu$ be the union of all non-bottom SCCs of the directed graph obtained from G_ν by considering only the stable edges of G_ν.

▶ **Lemma 13.** *For every configuration $\mathcal{C}$ such that $\mathcal{C} \models \Gamma_\nu$ we have that $\mathcal{C} \models \Diamond\big(\bigwedge_{A \in \mathcal{I}_\nu} \neg A\big)$.*

Similarly as above, we say that $\mathcal{C}$ is *S_ν-entering* if $\mathcal{C} \models \Box\pi_\nu \wedge \Box\mathcal{T}_\nu \wedge \bigwedge_{A \in \mathcal{I}_\nu} \neg A$ and there is an execution $\mathcal{C}_0, \ldots, \mathcal{C}_\ell$ such that $\mathcal{C}_0 \models \Gamma_\nu$, $\mathcal{C}_\ell = \mathcal{C}$, and $\mathcal{C}_j$ does *not* satisfy the above formula for all $j < \ell$.

Observe that if $\nu(A) = \texttt{ff}$ for all $A \in \mathcal{I}_\nu$, then ν implies $\bigwedge_{A \in \mathcal{I}_\nu} \neg A$ and hence every configuration $\mathcal{C}$ satisfying Γ_ν is S_ν-entering. Further, if $\nu(A) = \texttt{tt}$ for some $A \in \mathcal{I}_\nu$, then the last transition executed before visiting an S_ν-entering configuration is a transition $EF \mapsto CD$ generating a stable edge $E \to C$ of G_ν where $E \in \mathcal{I}_\nu$ and $C \notin \mathcal{I}_\nu$. Let $\mathcal{L}_\nu$ be the set of all right-hand sides of all such transitions. We put

$$\Phi_\nu \equiv \begin{cases} \pi_\nu \wedge \mathcal{T}_\nu \wedge \Big(\bigwedge\limits_{A \in \mathcal{I}_\nu} \neg A \Big) \wedge \Big(\bigvee\limits_{CD \in \mathcal{L}_\nu} \neg \xi_{CD} \Big) & \text{if } \nu(A) = \texttt{tt} \text{ for some } A \in \mathcal{I}_\nu, \\ \pi_\nu \wedge \mathcal{T}_\nu \wedge \ \nu & \text{if } \nu(A) = \texttt{ff} \text{ for all } A \in \mathcal{I}_\nu, \\ \pi_\nu \ \wedge \ \mathcal{T}_\nu \ \wedge \Big(\bigwedge\limits_{A \in \mathcal{I}_\nu} \neg A \Big) & \text{otherwise.} \end{cases}$$

We say that the constructed $\mathcal{P}$-stage $S_\nu = (\Phi_\nu, \pi_\nu, \mathcal{T}_\nu)$ is *redundant* if there is a $\mathcal{P}$-stage $S' = (\Phi', \pi', \mathcal{T}')$ on the path from the initial stage S_0 to S such that $\pi_\nu = \pi'$, $\mathcal{T}_\nu = \mathcal{T}'$, and Φ' implies Φ_ν. The $\mathcal{P}$-stage S_ν becomes a successor of S iff S_ν is not redundant. This ensures termination of the algorithm even for poorly designed population protocols.

5 Computing the interaction complexity

We show how to compute an upper asymptotic bounds on $\mathit{ReachNext}_S$ for every stage S in the stage graph constructed in Section 4.

For the rest of this section, we fix a population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$, a $\mathcal{P}$-stage $S = (\Phi, \pi, \mathcal{T})$, and its successor $S_\nu = (\Phi_\nu, \pi_\nu, \mathcal{T}_\nu)$. Recall the formula Γ_ν, the graph $G_\nu = (V, \rightarrow)$, and the sets Exp_ν, $\mathcal{J}_\nu$ defined in Section 4. We show how to compute an asymptotic upper bound on the function Reach_{S,S_ν} that assigns to every $n \geq 1$ the maximal $\mathbb{E}_{\mathcal{C}}[\mathit{Steps}_{\mathit{Enter}(S_\nu)}]$, where $\mathit{Enter}(S_\nu)$ is a fresh atomic proposition satisfied precisely by all S_ν-entering configurations, and $\mathcal{C}$ ranges over all configurations of size n satisfying Γ_ν (if there is no such configuration of size n, we put $\mathit{Reach}_{S,S_\nu}(n) = 0$). Observe that $\max_{S_\nu}\{\mathit{Reach}_{S,S_\nu}\}$, where S_ν ranges over all successor stages of S, is then an asymptotic upper bound on $\mathit{ReachNext}_S$.

Let us note that if $\mathcal{P}$ terminates, then $\mathit{InterComplexity}_{\mathcal{P}} \in 2^{2^{\mathcal{O}(n)}}$. This trivial bound follows by observing that the number of all configurations of size n is $2^{\mathcal{O}(n)}$, and the probability of reaching a stable configuration in $2^{\mathcal{O}(n)}$ transitions is $2^{-2^{o(n)}}$; this immediately implies the mentioned upper bound on $\mathit{InterComplexity}_{\mathcal{P}}$. As we shall see, the worst asymptotic bound on Reach_{S,S_ν} is $2^{\mathcal{O}(n)}$, and in many cases, our results allow to derive even a polynomial upper bound on Reach_{S,S_ν}.

Recall that if $(\pi_\nu, \mathcal{T})$ is stable or dead, we have that $\mathit{Reach}_{S,S_\nu}(n) = 0$ for all $n \in \mathbb{N}$ (in this case, we define S_ν-entering configurations are the configurations satisfying $\Box(\pi_\nu \wedge \mathcal{T})$). Now suppose $(\pi_\nu, \mathcal{T})$ is not stable or dead. Furthermore, let us first assume $\mathit{Exp}_\nu = \emptyset$. Then, the upper bound on Reach_{S,S_ν} is singly exponential in n.

▶ **Theorem 14.** *If $\mathit{Exp}_\nu = \emptyset$, then $\mathit{Reach}_{S,S_\nu} \in 2^{\mathcal{O}(n)}$.*

Now assume $\mathit{Exp}_\nu \neq \emptyset$. Let $\mathcal{U} \subseteq Q$ be the set of all states appearing in some non-bottom SCC of G_ν. We start with some auxiliary definitions.

▶ **Definition 15.** For every $A \in \mathcal{U}$, let $\mathit{Exp}_\nu[A]$ be the set of all $B \in Q$ such that $AB \in \mathit{Exp}_\nu$. We say that S_ν is *fast* if, for every $A \in \mathcal{U}$, the formula $\big(\pi_\nu \wedge \mathcal{T} \wedge \neg \mathit{Exp}_\nu \wedge A\big) \Rightarrow \big(\bigvee_{B \in \mathit{Exp}_\nu[A]} \neg \xi_{AB}\big)$ is a propositional tautology.

▶ **Definition 16.** For every $A \in V$, let $[A]$ be the SCC of G_ν containing A. We say that S_ν is *very fast* if every transition $AB \mapsto CD$ such that $AB, CD \in V^{\langle 2 \rangle}$ and $\{A, B, C, D\} \cap \mathcal{U} \neq \emptyset$ satisfies one of the following conditions:

- The formula $\big(\pi_\nu \wedge \mathcal{T}\big) \Rightarrow \xi_{AB}$ is a propositional tautology.
- $[C] \neq [A] \neq [D]$ and $[C] \neq [B] \neq [D]$.

▶ **Theorem 17.** *If $\mathit{Exp}_\nu \neq \emptyset$ and $\mathcal{J}_\nu \neq \emptyset$, then*

- *$\mathit{Reach}_{S,S_\nu} \in \mathcal{O}(n^3)$.*
- *If S_ν is fast, then $\mathit{Reach}_{S,S_\nu} \in \mathcal{O}(n^2 \cdot \log(n))$.*
- *If S_ν is very fast, then $\mathit{Reach}_{S,S_\nu} \in \mathcal{O}(n^2)$.*

Computing an asymptotic upper bound on Reach_{S,S_ν} when $\mathit{Exp}_\nu \neq \emptyset$ and $\mathcal{J}_\nu = \emptyset$ is more complicated. We show that a *polynomial* upper bound always exists, and that the degree of the polynomial is computable. However, our proof does not yield an efficient algorithm for computing/estimating the degree.

▶ **Theorem 18.** *If $\mathit{Exp}_\nu \neq \emptyset$ and $\mathcal{J}_\nu = \emptyset$, then $\mathit{Reach}_{S,S_\nu} \in \mathcal{O}(n^c)$ for some computable constant c.*

Table 1 Results of the experimental evaluation where $|Q|$, $|T|$ and $|\mathbb{S}|$ correspond respectively to the number of states and transitions of the protocol, and the number of nodes of its stage graph.

Protocol predicate / params.	$\lvert Q\rvert$	$\lvert T\rvert$	$\lvert\mathbb{S}\rvert$	Bound	Time
$x_1 \vee \ldots \vee x_n$ [12]	2	1	5	$n^2 \cdot \log n$	0.1
$x \geq y$ (Example 3)	5	6	13	$n^2 \cdot \log n$	0.4
$x \geq y$ [7]	4	3	9	$n^2 \cdot \log n$	0.2
$x \geq y$ (Example 2)	4	4	11	$\exp(n)$	0.3
Flocks-of-bird protocol [4]: $x \geq c$					
$c = 5$	6	21	26	n^3	0.8
$c = 10$	11	66	46	n^3	4.0
$c = 15$	16	136	66	n^3	12.1
$c = 20$	21	231	86	n^3	28.9
$c = 25$	26	351	106	n^3	58.0
$c = 30$	31	496	126	n^3	118.9
$c = 35$	36	666	146	n^3	222.3
$c = 40$	41	861	166	n^3	366.2
$c = 45$	46	1081	186	n^3	495.3
$c = 50$	51	1326	206	n^3	952.8
$c = 55$	56	1596	—	—	T/O
Logarithmic flock-of-birds protocol[8]: $x \geq c$					
$c = 15$	8	23	66	n^3	2.6
$c = 31$	10	34	130	n^3	6.1
$c = 63$	12	47	258	n^3	13.9
$c = 127$	14	62	514	n^3	39.4
$c = 255$	16	79	1026	n^3	81.0
$c = 1023$	20	119	4098	n^3	395.7
$c = 2047$	22	142	8194	n^3	851.9
$c = 4095$	24	167	—	—	T/O

Protocol predicate / params.	$\lvert Q\rvert$	$\lvert T\rvert$	$\lvert\mathbb{S}\rvert$	Bound	Time
Flocks-of-bird protocol [12]: $x \geq c$					
$c = 5$	6	9	54	n^3	2.5
$c = 7$	8	13	198	n^3	11.3
$c = 10$	11	19	1542	n^3	83.9
$c = 13$	14	25	12294	n^3	816.4
$c = 15$	16	29	—	—	T/O
Average-and-conquer protocol[9] [2]: $x \geq y$ with params. m and d					
$m = 3, d = 1$	6	21	41	$n^2 \cdot \log n$	2.0
$m = 3, d = 2$	8	36	1948	$n^2 \cdot \log n$	98.7
$m = 5, d = 1$	8	36	1870	n^3	80.1
$m = 5, d = 2$	10	55	—	—	T/O
$m = 7, d = 1$	10	55	—	—	T/O
Remainder protocol [8]: $\sum_{1 \leq i < m} i \cdot x_i \equiv 0 \pmod m$					
$m = 3$	5	12	27	$n^2 \cdot \log n$	0.8
$m = 5$	7	25	225	$n^2 \cdot \log n$	12.5
$m = 7$	9	42	1351	$n^2 \cdot \log n$	88.9
$m = 9$	11	63	7035	$n^2 \cdot \log n$	544.0
$m = 10$	12	75	—	—	T/O
Threshold protocol [4]: $\sum_{1 \leq i \leq k} a_i \cdot x_i < c$					
$-x_1 + x_2 < 0$	12	57	21	n^3	3.0
$-x_1 + x_2 < 1$	20	155	131	n^3	30.3
$-x_1 + x_2 < 2$	28	301	—	—	T/O
$-2x_1 - x_2 + x_3 + 2x_4 < 0$	20	155	1049	n^3	166.3
$-2x_1 - x_2 + x_3 + 2x_4 < 1$	20	155	1049	n^3	155.2
$-2x_1 - x_2 + x_3 + 2x_4 < 2$	28	301	—	—	T/O

6 Experimental results

We have implemented our approach as a tool[6] that takes a population protocol as input and follows the procedure of Section 4 to construct a stage graph together with an upper bound on $\mathit{InterComplexity}_{\mathcal{P}}$. Our tool is implemented in Python 3, and uses the SMT solver Microsoft Z3 [13] to test for tautologies and to obtain valid valuations.

We tested our implementation on multiple protocols drawn from the literature: a simple broadcast protocol [12], the majority protocols of Example 2, Example 3 and [2], various flock-of-birds protocols [4, 12, 7], a remainder protocol [8] and a threshold protocol [4]. Most of these protocols are parametric, i.e. they are a family of protocols depending on some parameters. For these protocols, we increased their parameters until reaching a timeout. In particular, for the logarithmic flock-of-birds protocol computing $x \geq c$, we used thresholds of the form $c = 2^i - 1$ as they essentially consist the most complicated case of the protocol.

All tests were performed on the same computer equipped with eight Intel® Core™ i5-8250U 1.60 GHz CPUs, 8 GB of memory and Ubuntu Linux 17.10 (64 bits). Each test had a timeout of 1000 seconds (∼16.67 minutes). The duration of each test was evaluated as the sum of the `user` time and `sys` time reported by the Python time library.

The results of the benchmarks are depicted in Table 1, where the *bound* column refers to the derived upper bound on $\mathit{InterComplexity}_{\mathcal{P}}$. In particular, the tool derived exponential and $n^2 \cdot \log n$ bounds for the protocols of Example 2 and Example 3 respectively. The generated trees across all instances grow in width but not much in height: the maximum height between the roots and the leaves varies between 2 and 5, and most nodes are leaves.

[6] The tool and its benchmarks are available at `https://github.com/blondimi/pp-time-analysis`.
[7] Protocol of Example 2 without the tie-breaking rule $ba \mapsto bb$ (only correct if $x \neq y$).
[8] An adapted version of the protocol of [7, Sect. 3] without so-called k-way transitions.
[9] The protocol is only correct assuming $x \neq y$.

It is worth noting that the $n^2 \log n$ bounds obtained in Table 1 for the *average-and-conquer* and *remainder* protocols are tight with respect to the best known bounds [2, 4]. However, some of the obtained bounds are not tight, e.g. we report n^3 for the *threshold protocol* but an $n^2 \log n$ upper bound was shown in [4]. Moreover, it seems possible to decrease the n^3 bound to n^2 for the *flocks-of-bird protocol* of [4]. We are unsure of the precise bounds for the remaining protocols.

7 Conclusion

We have presented the first algorithm for quantitative verification of population protocols able to provide asymptotic bounds valid for any number of agents. The algorithm is able to compute good bounds for many of the protocols described in the literature.

The algorithm is based on the notion of stage graph, a concept that can be of independent value. In particular, we think that stage graphs can be valuable for fault localization and perhaps even automatic repair of ill designed protocols.

An interesting question is whether our algorithm is "weakly complete", meaning that for every predicate there exists a protocol for which our algorithm can compute the exact time bound. We know that this is the case for protocols with leaders, and conjecture that the result extends to all protocols, but currently we do not have a proof.

Another venue for future research is the automatic computation of lower bounds. Here, while stage graphs will certainly be useful, they do not seem to be enough, and will have to be complemented with other techniques.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proc.* 28^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 47–56, 2015. doi:10.1145/2767386.2767429.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc.* 23^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

4 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, jan 2006. doi:10.1007/s00446-005-0138-3.

5 Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. How long, O bayesian network, will I sample thee? - A program analysis perspective on expected sampling times. In *Proc.* 27^{th} *European Symposium on Programming (ESOP)*, pages 186–213. Springer, 2018. doi:10.1007/978-3-319-89884-1_7.

6 Amanda Belleville, David Doty, and David Soloveichik. Hardness of computing and approximating predicates and functions with leaderless population protocols. In *Proc.* 44^{th} *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 141:1–141:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.141.

7 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: on the minimal size of population protocols. In *Proc.* 35^{th} *Symposium on Theoretical Aspects of*

Computer Science (STACS), pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

8 Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. Towards efficient verification of population protocols. In *Proc.* 36^{th} *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 423–430, 2017. doi:10.1145/3087801.3087816.

9 Michael Blondin, Javier Esparza, and Antonín Kučera. Automatic analysis of expected termination time for population protocols. *ArXiv e-prints*, 2018. arXiv:1807.00331.

10 Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. Automated recurrence analysis for almost-linear expected-runtime bounds. In *Proc.* 29^{th} *International Conference on Computer Aided Verification (CAV)*, pages 118–139, 2017. doi:10.1007/978-3-319-63387-9_6.

11 Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. *Distributed Computing*, 30(5):373–390, 2017. doi:10.1007/s00446-015-0255-6.

12 Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. Guidelines for the verification of population protocols. In *Proc.* 31^{st} *International Conference on Distributed Computing Systems (ICDCS)*, pages 215–224, 2011. doi:10.1109/ICDCS.2011.36.

13 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proc.* 14^{th} *International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340, 2008. doi:10.1007/978-3-540-78800-3_24.

14 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *Proc.* 29^{th} *International Symposium on Distributed Computing (DISC)*, pages 602–616, 2015. doi:10.1007/978-3-662-48653-5_40.

15 Moez Draief and Milan Vojnović. Convergence speed of binary interval consensus. In *Proc.* 29^{th} *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1792–1800, 2010. doi:10.1109/INFCOM.2010.5461999.

16 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.

17 Philippe Flajolet, Bruno Salvy, and Paul Zimmermann. Automatic average-case analysis of algorithm. *Theoretical Computer Science*, 79(1):37–109, 1991. doi:10.1016/0304-3975(91)90145-R.

18 Stefan Jaax. Personal communication, April 2018.

19 Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected run-times of probabilistic programs. In *Proc.* 25^{th} *European Symposium on Programming (ESOP)*, pages 364–389. Springer, 2016. doi:10.1007/978-3-662-49498-1_15.

20 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72, 2018. doi:10.1145/3156693.

21 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Communications of the ACM*, 58(1):94–102, 2015. doi:10.1145/2678280.

22 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: resource analysis for probabilistic programs. In *Proc.* 39^{th} *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 496–512, 2018. doi:10.1145/3192366.3192394.

23 Etienne Perron, Dinkar Vasudevan, and Milan Vojnović. Using three states for binary consensus on complete graphs. In *Proc.* 28^{th} *IEEE International Conference on Computer Communications (INFOCOM)*, pages 2527–2535, 2009. doi:10.1109/INFCOM.2009.5062181.

On Runtime Enforcement via Suppressions

Luca Aceto
Gran Sasso Science Institute, L'Aquila, Italy; and
Reykjavik University, Reykjavik, Iceland
luca.aceto@gssi.it

Ian Cassar
Reykjavik University, Reykjavik Iceland; and
University of Malta, Msida, Malta
ianc@ru.is

Adrian Francalanza
University of Malta, Msida, Malta
adrian.francalanza@um.edu.mt

Anna Ingólfsdóttir
Reykjavik University, Reykjavik, Iceland
annai@ru.is

Abstract

Runtime enforcement is a dynamic analysis technique that uses monitors to enforce the behaviour specified by some correctness property on an executing system. The enforceability of a logic captures the extent to which the properties expressible via the logic can be enforced at runtime. We study the enforceability of Hennessy-Milner Logic with Recursion (μHML) with respect to suppression enforcement. We develop an operational framework for enforcement which we then use to formalise when a monitor enforces a μHML property. We also show that the safety syntactic fragment of the logic, sHML, is enforceable by providing an automated synthesis function that generates correct suppression monitors from sHML formulas.

2012 ACM Subject Classification Theory of computation → Logic and verification, Software and its engineering → Software verification, Software and its engineering → Dynamic analysis

Keywords and phrases Enforceability, Suppression Enforcement, Monitor Synthesis, Logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.34

Related Version `https://arxiv.org/abs/1807.01004`

Acknowledgements The research work disclosed in this publication is partially supported by the projects "Developing Theoretical Foundations for Runtime Enforcement" (184776-051) and "TheoFoMon: Theoretical Foundations for Monitorability" (163406-051) of the Icelandic Research Fund, and by the Endeavour Scholarship Scheme (Malta), part-financed by the European Social Fund (ESF) – Operational Programme II – Cohesion Policy 2014-2020.

1 Introduction

Runtime monitoring [22, 24] is a dynamic analysis technique that is becoming increasingly popular in the turbid world of software development. It uses code units called *monitors* to aggregate system information, compare system execution against correctness specifications, or steer the execution of the observed system. The technique has been used effectively to offload certain verification tasks to a post-deployment phase, thus complementing other

(static) analysis techniques in multi-pronged verification strategies – see e.g., [6, 12, 27, 18, 28]. *Runtime enforcement* (RE) [33, 34, 21] is a specialized monitoring technique, used to ensure that the behaviour of a system-under-scrutiny (SuS) is *always* in agreement with some correctness specification. It employs a specific kind of monitor (referred to as a *transducer* [9, 42, 4] or an *edit-automaton* [33, 34]) to anticipate incorrect behaviour and counter it. Such a monitor thus acts as a proxy between the SuS and the surrounding environment interacting with it, encapsulating the system to form a composite (monitored) system: at runtime, the monitor *transforms* any incorrect executions exhibited by the SuS into correct ones by either *suppressing*, *inserting* or *replacing* events on behalf of the system.

We extend a recent line of research [25, 24, 2, 1] and study RE approaches that adopt a *separation of concerns* between the correctness specification, describing *what* properties the SuS should satisfy, and the monitor, describing *how* to enforce these properties on the SuS. Our work considers system properties expressed in terms of the process logic μHML [30, 32], and explores what properties can be operationally enforced by monitors that can suppress system behaviour. A central element for the realisation of such an approach is the *synthesis* function: it automates the translation from the *declarative* μHML specifications to *algorithmic* descriptions formulated as executable monitors. Since analysis tools ought to form part of the trusted computing base, enforcement monitoring should be, in and of itself, correct. However, it is unclear what is to be expected of the synthesised monitor to adequately enforce a μHML formula. Nor is it clear for which type of specifications should this approach be expected to work effectively – it has been well established that a number of properties are *not* monitorable [15, 40, 16, 25, 2] and it is therefore reasonable to expect similar limits in the case of enforceability [19]. We therefore study the relationship between μHML specifications and suppression monitors for enforcement, which allows us to address the above-mentioned concerns and make the following contributions:

Modelling: We develop a general framework for enforcement instrumentation that is parametrisable by any system behaviour that is expressed via labelled transitions, and can express suppression, insertion and replacement enforcement, Figure 2.

Correctness: We give formal definitions for asserting when a monitor correctly enforces a formula defined over labelled transition systems, Definitions 3 and 8. These definitions are parametrisable with respect to an instrumentation relation, an instance of which is our enforcement framework of Figure 2.

Expressiveness: We provide enforceability results, Theorems 14 and 18 (but also Proposition 24), by identifying a subset of μHML formulas that can be (correctly) enforced by suppression monitors.

As a by-product of this study, we also develop a formally-proven correct synthesis function, Definition 12, that then can be used for tool construction, along the lines of [8, 7].

The setup selected for our study serves a number of purposes. For starters, the chosen logic, μHML, is a branching-time logic that allows us to investigate enforceability for properties describing computation graphs. Second, the use of a highly expressive logic allows us to achieve a good degree of generality for our results, and so, by working in relation to logics like μHML (a reformulation of the μ-calculus), our work would also apply to other widely used logics (such as LTL and CTL [17]) that are embedded within this logic. Third, since the logic is verification-technique agnostic, it fits better with the realities of software verification in the present world, where a *variety* of techniques (e.g., model-checking and testing) straddling both pre- and post-deployment phases are used. In such cases, knowing which properties can be verified statically and which ones can be monitored for and enforced at runtime is crucial for devising effective multi-pronged verification strategies. Equipped

Syntax

$$\begin{array}{llllll} \varphi,\psi \in \mu\text{HML} ::= \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} & \mid \bigvee_{i\in I}\varphi_i & \text{(disjunction)} \\ \mid \bigwedge_{i\in I}\varphi_i & \text{(conjunction)} & \mid \langle\{p,c\}\rangle\varphi & \text{(possibility)} & \mid [\{p,c\}]\varphi & \text{(necessity)} \\ \mid \mathsf{min}\, X.\varphi & \text{(least fp.)} & \mid \mathsf{max}\, X.\varphi & \text{(greatest fp.)} & \mid X & \text{(fp. variable)} \end{array}$$

Semantics

$$\begin{array}{lll} [\![\mathsf{tt},\rho]\!] \stackrel{\text{def}}{=} \textsc{Sys} & [\![\mathsf{ff},\rho]\!] \stackrel{\text{def}}{=} \varnothing & [\![X,\rho]\!] \stackrel{\text{def}}{=} \rho(X) \\ [\![\bigwedge_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \bigcap_{i\in I}[\![\varphi_i,\rho]\!] & [\![\mathsf{max}\, X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcup \{S \mid S \subseteq [\![\varphi,\rho[X\mapsto S]]\!]\} & \\ [\![\bigvee_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \bigcup_{i\in I}[\![\varphi_i,\rho]\!] & [\![\mathsf{min}\, X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcap \{S \mid [\![\varphi,\rho[X\mapsto S]]\!] \subseteq S\} & \end{array}$$

$$[\![[\{p,c\}]\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid (\forall \alpha, r \cdot s \stackrel{\alpha}{\Rightarrow} r \text{ and } (\exists \sigma \cdot \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true})) \text{ implies } q \in [\![\varphi\sigma,\rho]\!]\}$$

$$[\![\langle\{p,c\}\rangle\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid \exists \alpha, r, \sigma \cdot (s \stackrel{\alpha}{\Rightarrow} r \text{ and } \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true} \text{ and } q \in [\![\varphi\sigma,\rho]\!])\}$$

Figure 1 μHML Syntax and Semantics.

with such knowledge, one could also employ standard techniques [36, 5, 31] to decompose a non-enforceable property into a collection of smaller properties, a subset of which can then be enforced at runtime.

Structure of the paper. Section 2 revisits labelled transition systems and our touchstone logic, μHML. The operational model for enforcement monitors and instrumentation is given in Section 3. In Section 4 we formalise the interdependent notions of correct enforcement and enforceability. These act as a foundation for the development of a synthesis function in Section 5, that produces *correct-by-construction* monitors. In Section 6 we consider alternative definitions for enforceability for logics with a specific additional interpretation, and show that our proposed synthesis function is still correct with respect to the new definition. Section 7 concludes and discusses related work.

2 Preliminaries

The Model. We assume systems described as *labelled transition systems* (LTSs), triples $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow\rangle$ consisting of a set of *system states*, $s,r,q \in \textsc{Sys}$, a set of *observable actions*, $\alpha,\beta \in \textsc{Act}$, and a distinguished silent action $\tau \notin \textsc{Act}$ (where $\mu \in \textsc{Act} \cup \{\tau\}$), and a *transition* relation, $\longrightarrow \subseteq (\textsc{Sys} \times \textsc{Act} \cup \{\tau\} \times \textsc{Sys})$. We write $s \xrightarrow{\mu} r$ in lieu of $(s,\mu,r) \in \rightarrow$, and use $s \stackrel{\mu}{\Longrightarrow} s'$ to denote weak transitions representing $s(\xrightarrow{\tau})^* \cdot \xrightarrow{\mu} \cdot (\xrightarrow{\tau})^* s'$. We refer to s' as a μ-derivative of s. Traces, $t,u \in \textsc{Act}^*$ range over (finite) sequences of observable actions, and we write $s \stackrel{t}{\Longrightarrow} r$ to denote a sequence of weak transitions $s \stackrel{\alpha_1}{\Longrightarrow} \ldots \stackrel{\alpha_n}{\Longrightarrow} r$ for $t = \alpha_1,\ldots,\alpha_n$. We also assume the classic notion of *strong bisimilarity* [39, 43] for our model, $s \sim r$, using it as our touchstone system equivalence. The syntax of the regular fragment of CCS [39] is occasionally used to concisely describe LTSs in our examples.

The Logic. We consider a slightly generalised version of μHML [32, 3] that uses *symbolic actions* of the form $\{p,c\}$. *Patterns*, p, abstract over actions using *data variables* $d,e,f \in \textsc{Var}$; in a pattern, they may either occur free, d, or as binders, (d) where a *closed pattern* is one without any free variables. We assume a (partial) *matching function* for *closed* patterns $\mathsf{mtch}(p,\alpha)$ that returns a substitution σ (when successful) mapping variables in p to the corresponding values in α, i.e., if we instantiate every bound variable d in p with

$\sigma(d)$ we obtain α. The *filtering condition*, c, contains variables found in p and evaluates wrt. the substitutions returned by successful matches. Put differently, a *closed* symbolic action $\{p, c\}$ is one where p is closed and $\mathbf{fv}(c) \subseteq \mathbf{bv}(p)$; it denotes the *set* of actions $[\![\{p, c\}]\!] \stackrel{\text{def}}{=} \{\ \alpha \mid \exists \sigma \cdot \mathsf{mtch}(p, \alpha) = \sigma \text{ and } c\sigma \Downarrow \mathsf{true}\ \}$ and allows more adequate reasoning about LTSs with infinite actions (e.g., actions carrying data from infinite domains).

The logic syntax is given in Figure 1 and assumes a countable set of logical variables $X, Y \in \text{LVar}$. Apart from standard logical constructs such as conjunctions and disjunctions ($\bigwedge_{i\in I} \varphi_i$ describes a *compound* conjunction, $\varphi_1 \wedge \ldots \wedge \varphi_n$, where $I = \{1, .., n\}$ is a finite set of indices, and similarly for disjunctions), and the characteristic greatest and least fixpoints ($\mathsf{max}\, X.\varphi$ and $\mathsf{min}\, X.\varphi$ bind free occurrences of X in φ), the logic uses necessity and possibility modal operators with symbolic actions, $[\{p, c\}]\varphi$ and $\langle\{p, c\}\rangle\varphi$, where $\mathbf{bv}(p)$ bind free data variables in c and φ. Formulas in μHML are interpreted over the system powerset domain where $S \in \mathcal{P}(\text{Sys})$. The semantic definition of Figure 1, $[\![\varphi, \rho]\!]$, is given for *both* open and closed formulas. It employs a valuation from logical variables to sets of states, $\rho \in (\text{LVar} \to \mathcal{P}(\text{Sys}))$, which permits an inductive definition on the structure of the formulas; $\rho' = \rho[X \mapsto S]$ denotes a valuation where $\rho'(X) = S$ and $\rho'(Y) = \rho(Y)$ for all other $Y \neq X$. The only non-standard cases are those for the modal formulas, due to the use of symbolic actions. Note that we recover the standard logic for symbolic actions $\{p, c\}$ whose pattern p does not contain variables ($p{=}\alpha$ for some α) and whose condition holds trivially ($c{=}\mathsf{true}$); in such cases we write $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ for short. We generally assume *closed* formulas, i.e., without free logical and data variables, and write $[\![\varphi]\!]$ in lieu of $[\![\varphi, \rho]\!]$ since the interpretation of a closed φ is independent of ρ. A system s *satisfies* formula φ whenever $s \in [\![\varphi]\!]$ whereas a formula φ is *satisfiable*, $\varphi \in \text{Sat}$, whenever there exists a system r such that $r \in [\![\varphi]\!]$.

▶ **Example 1.** Consider two systems (a good system, $s_{\mathbf{g}}$, and a bad one, $s_{\mathbf{b}}$) implementing a server that interacts on port i, repeatedly accepting *requests* that are *answered* by outputting on the same port, and terminating the service once a *close* request is accepted (on the same port). Whereas $s_{\mathbf{g}}$ outputs an answer ($i!\mathsf{ans}$) for every request ($i?\mathsf{req}$), $s_{\mathbf{b}}$ occasionally refuses to answer a given request (see the underlined branch). Both systems terminate with $i?\mathsf{cls}$.

$$s_{\mathbf{g}} = \mathsf{rec}\, x.\big(i?\mathsf{req}.i!\mathsf{ans}.x + i?\mathsf{cls}.\mathsf{nil}\big) \qquad s_{\mathbf{b}} = \mathsf{rec}\, x.\big(i?\mathsf{req}.i!\mathsf{ans}.x + \underline{i?\mathsf{req}.x} + i?\mathsf{cls}.\mathsf{nil}\big)$$

We can specify that two consecutive requests on port i indicate invalid behaviour via the μHML formula $\varphi_0 \stackrel{\text{def}}{=} \mathsf{max}\, X.[i?\mathsf{req}]\,([i!\mathsf{ans}]X \wedge [i?\mathsf{req}]\mathsf{ff})$; it defines an invariant property ($\mathsf{max}\, X.(\cdots)$) requiring that whenever a system interacting on i inputs a request, it cannot input a subsequent request, i.e., $[i?\mathsf{req}]\mathsf{ff}$, unless it outputs an answer beforehand, in which case the formula recurses, i.e., $[i!\mathsf{ans}]X$. Using symbolic actions, we can generalise φ_0 by requiring the property to hold for *any* interaction happening on *any* port number *except* j.

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\, X.[\{(d)?\mathsf{req}, d{\neq}j\}]([\{d!\mathsf{ans}, \mathsf{true}\}]X \wedge [\{d?\mathsf{req}, \mathsf{true}\}]\mathsf{ff})$$

In φ_1, $(d)?\mathsf{req}$ binds the free occurrences of d found in $d{\neq}j$ and $[\{d!\mathsf{ans}, \mathsf{true}\}]X \wedge [\{d?\mathsf{req}, \mathsf{true}\}]\mathsf{ff}$. Using Figure 1, one can check that $s_{\mathbf{g}} \in [\![\varphi_1]\!]$, whereas $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{b}} \xrightarrow{i?\mathsf{req}} \cdot \xrightarrow{i?\mathsf{req}} \ldots$

3 An Operational Model for Enforcement

Our operational mechanism for enforcing properties over systems uses the (symbolic) transducers $m, n \in \text{Trn}$ defined in Figure 2. The transition rules in Figure 2 assume closed terms, i.e., for every *symbolic-prefix transducer*, $\{p, c, p'\}.m$, p is closed and $\big(\mathbf{fv}(c) \cup \mathbf{fv}(p') \cup \mathbf{fv}(m)\big) \subseteq \mathbf{bv}(p)$, and yield an LTS with labels of the form $\gamma_{\blacktriangleright}\mu$, where $\gamma \in (\text{Act} \cup \{\bullet\})$. Our syntax

Syntax

$$m, n \in \textsc{Trn} ::= \quad \mathsf{id} \quad | \quad \{p, c, p'\}.m \quad | \quad \textstyle\sum_{i\in I} m_i \quad | \quad \mathsf{rec}\, x.m \quad | \quad x$$

Dynamics

$$\textsc{eId}\ \frac{}{\mathsf{id} \xrightarrow{\mu\blacktriangleright\mu} \mathsf{id}} \qquad \textsc{eSel}\ \frac{m_j \xrightarrow{\gamma\blacktriangleright\mu} n_j}{\sum_{i\in I} m_i \xrightarrow{\gamma\blacktriangleright\mu} n_j}\ j\in I \qquad \textsc{eRec}\ \frac{m\{\mathsf{rec}\, x.m/x\} \xrightarrow{\gamma\blacktriangleright\mu} n}{\mathsf{rec}\, x.m \xrightarrow{\gamma\blacktriangleright\mu} n}$$

$$\textsc{eTrn}\ \frac{\mathsf{mtch}(p,\gamma) = \sigma \qquad c\sigma \Downarrow \mathsf{true} \qquad \mu = p'\sigma}{\{p, c, p'\}.m \xrightarrow{\gamma\blacktriangleright\mu} m\sigma}$$

Instrumentation

$$\textsc{iTrn}\ \frac{s \xrightarrow{\alpha} s' \quad m \xrightarrow{\alpha\blacktriangleright\mu} n}{m[s] \xrightarrow{\mu} n[s']} \qquad \textsc{iAsy}\ \frac{s \xrightarrow{\tau} s'}{m[s] \xrightarrow{\tau} m[s']} \qquad \textsc{iIns}\ \frac{m \xrightarrow{\bullet\blacktriangleright\mu} n}{m[s] \xrightarrow{\mu} n[s]} \qquad \textsc{iTer}\ \frac{s \xrightarrow{\alpha} s' \quad m \overset{\alpha}{\not\longrightarrow} \quad m \overset{\bullet}{\not\longrightarrow}}{m[s] \xrightarrow{\alpha} \mathsf{id}[s']}$$

Figure 2 A model for transducers (I is a finite index set and $m \overset{\gamma}{\not\longrightarrow}$ means $\nexists \mu, n \cdot m \xrightarrow{\gamma\blacktriangleright\mu} n$).

assumes a well-formedness constraint where for every $\{p, c, p'\}.m$, $\mathbf{bv}(c) \cup \mathbf{bv}(p') = \varnothing$. Intuitively, a transition $m \xrightarrow{\alpha\blacktriangleright\mu} n$ denotes the fact that the transducer in state m *transforms* the visible action α (produced by the system) into the action μ (which can possibly become silent) and transitions into state n. In this sense, the transducer action $\alpha\blacktriangleright\tau$ represents the *suppression* of action α, action $\alpha\blacktriangleright\beta$ represents the *replacing* of α by β, and $\alpha\blacktriangleright\alpha$ denotes the *identity* transformation. The special case $\bullet\blacktriangleright\alpha$ encodes the *insertion* of α, where $\bullet$ represents that the transition is not induced by any system action.

The key transition rule in Figure 2 is eTrn. It states that the symbolic-prefix transducer $\{p, c, p'\}.m$ can transform an (extended) action γ into the concrete action μ, as long as the action matches with pattern p with substitution σ, $\mathsf{mtch}(p, \gamma) = \sigma$, and the condition is satisfied by σ, $c\sigma \Downarrow \mathsf{true}$ (the matching function is lifted to extended actions and patterns in the obvious way, where $\mathsf{mtch}(\bullet, \bullet) = \varnothing$). In such a case, the transformed action is $\mu = p'\sigma$, i.e., the action μ resulting from the instantiation of the free data variables in pattern p' with the corresponding values mapped by σ, and the transducer state reached is $m\sigma$. By contrast, in rule eId, the transducer id acts as the identity and leaves actions unchanged. The remaining rules are fairly standard and unremarkable.

Figure 2 also describes an *instrumentation* relation which relates the behaviour of the SuS s with the transformations of a transducer monitor m that *agrees* with the (observable) actions Act of s. The term $m[s]$ thus denotes the resulting *monitored system* whose behaviour is defined in terms of $\textsc{Act} \cup \{\tau\}$ from the system's LTS. Concretely, rule iTrn states that when a system s transitions with an observable action α to s' and the transducer m can *transform* this action into μ and transition to n, the instrumented system $m[s]$ transitions with action μ to $n[s']$. However, when s transitions with a silent action, rules iAsy allows it to do so independently of the transducer. Dually, rule iIns allows the transducer to *insert* an action μ independently of s's behaviour. Rule iTer is analogous to standard monitor instrumentation rules for premature termination of the transducer [22, 25, 23, 1], and accounts for underspecification of transformations. Thus, if a system s transitions with an observable action α to s', and the transducer m does not specify how to transform it ($m \overset{\alpha}{\not\longrightarrow}$), nor can it transition to a new transducer state by inserting an action ($m \overset{\bullet}{\not\longrightarrow}$), the system is still allowed to transition while the transducer's transformation activity is ceased, i.e., it acts like the identity id from that point onwards.

▶ **Example 2.** Consider the insertion transducer $m_{\mathbf{i}}$ and the replacement transducer $m_{\mathbf{r}}$ below:

$$m_{\mathbf{i}} \stackrel{\text{def}}{=} \{\bullet, \mathsf{true}, i?\mathsf{req}\}.\{\bullet, \mathsf{true}, i!\mathsf{ans}\}.\mathsf{id}$$
$$m_{\mathbf{r}} \stackrel{\text{def}}{=} \mathsf{rec}\, x.\big(\{(d)?\mathsf{req}, \mathsf{true}, j?\mathsf{req}\}.x + \{(d)!\mathsf{ans}, \mathsf{true}, j!\mathsf{ans}\}.x + \{(d)?\mathsf{cls}, \mathsf{true}, j?\mathsf{cls}\}.x\big)$$

When instrumented with a system, $m_{\mathbf{i}}$ inserts the two successive actions $i?\mathsf{req}$ and $i!\mathsf{ans}$ before behaving as the identity. Concretely in the case of $s_{\mathbf{b}}$ we can only start the computation as:

$$m_{\mathbf{i}}[s_{\mathbf{b}}] \xrightarrow{i?\mathsf{req}} \{\bullet, \mathsf{true}, i!\mathsf{ans}\}.\mathsf{id}[s_{\mathbf{b}}] \xrightarrow{i!\mathsf{ans}} \mathsf{id}[s_{\mathbf{b}}] \xrightarrow{\alpha} \ldots \qquad (\text{where } s_{\mathbf{b}} \xrightarrow{\alpha})$$

By contrast, $m_{\mathbf{r}}$ transforms input actions with either payload req or cls and output actions with payload ans on any port name, into the respective actions on port j. For instance:

$$m_{\mathbf{r}}[s_{\mathbf{b}}] \xrightarrow{j?\mathsf{req}} m_{\mathbf{r}}[i!\mathsf{ans}.s_{\mathbf{b}}] \xrightarrow{j!\mathsf{ans}} m_{\mathbf{r}}[s_{\mathbf{b}}] \xrightarrow{j?\mathsf{cls}} m_{\mathbf{r}}[\mathsf{nil}]$$

Consider now the two suppression transducers $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$ for actions on ports other than j:

$$m_{\mathbf{s}} \stackrel{\text{def}}{=} \mathsf{rec}\, x.\big(\{(d)?\mathsf{req}, d \neq j, \tau\}.x + \{(d)!\mathsf{ans}, \mathsf{true}, d!\mathsf{ans}\}.x\big)$$
$$m_{\mathbf{t}} \stackrel{\text{def}}{=} \mathsf{rec}\, x.\big(\{(d)?\mathsf{req}, d \neq j, d?\mathsf{req}\}.\mathsf{rec}\, y.\big(\{d!\mathsf{ans}, \mathsf{true}, d!\mathsf{ans}\}.x + \{d?\mathsf{req}, \mathsf{true}, \tau\}.y\big)\big)$$

Monitor $m_{\mathbf{s}}$ suppresses any requests on ports other than j, and continues to do so after any answers on such ports. When instrumented with $s_{\mathbf{b}}$, we can observe the following behaviour:

$$m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[i!\mathsf{ans}.s_{\mathbf{b}}] \xrightarrow{i!\mathsf{ans}} m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[i!\mathsf{ans}.s_{\mathbf{b}}] \xrightarrow{i!\mathsf{ans}} m_{\mathbf{s}}[s_{\mathbf{b}}] \ldots$$

Note that $m_{\mathbf{s}}$ does not specify a transformation behaviour for when the monitored system produces inputs with payload other than req. The instrumentation handles this underspecification by ceasing suppression activity; in the case of $s_{\mathbf{b}}$ we get $m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{i?\mathsf{cls}} \mathsf{id}[\mathsf{nil}]$. The transducer $m_{\mathbf{t}}$ performs slightly more elaborate transformations. For interactions on ports other than j, it suppresses consecutive input requests following any serviced request (i.e., an input on req followed by an output on ans) sequence. For $s_{\mathbf{b}}$ we can observe the following:

$$\begin{aligned} m_{\mathbf{t}}[s_{\mathbf{b}}] &\xrightarrow{i?\mathsf{req}} \mathsf{rec}\, y.\big(\{i!\mathsf{ans}, \mathsf{true}, i!\mathsf{ans}\}.m_{\mathbf{t}} + \{i?\mathsf{req}, \mathsf{true}, \tau\}.y\big)[s_{\mathbf{b}}] \\ &\xrightarrow{\tau} \mathsf{rec}\, y.\big(\{i!\mathsf{ans}, \mathsf{true}, i!\mathsf{ans}\}.m_{\mathbf{t}} + \{i?\mathsf{req}, \mathsf{true}, \tau\}.y\big)[i!\mathsf{ans}.s_{\mathbf{b}}] \xrightarrow{i!\mathsf{ans}} m_{\mathbf{t}}[s_{\mathbf{b}}] \end{aligned}$$

In the sequel, we find it convenient to refer to $\underline{p}$ as the transformed pattern p where all the binding occurrences (d) are converted to free occurrences d. As shorthand notation, we elide the second pattern p' in a transducer $\{p, c, p'\}.m$ whenever $p'{=}\underline{p}$ and simply write $\{p, c\}.m$; note that if $\mathbf{bv}(p) = \varnothing$, then $\underline{p}{=}p$. Similarly, we elide c whenever $c{=}\mathsf{true}$. This allows us to express $m_{\mathbf{t}}$ from Example 2 as $\mathsf{rec}\, x.\big(\{(d)?\mathsf{req}, d{\neq}j\}.\mathsf{rec}\, y.\big(\{d!\mathsf{ans}\}.x + \{d?\mathsf{req}, \tau\}.y\big)\big)$.

4 Enforceability

The *enforceability* of a logic rests on the relationship between the semantic behaviour specified by the logic on the one hand, and the ability of the operational mechanism (the transducers and instrumentation of Section 3 in our case) to enforce the specified behaviour on the other.

▶ **Definition 3** (Enforceability)**.** A logic $\mathcal{L}$ is enforceable iff *every* formula $\varphi{\in}\mathcal{L}$ is *enforceable*. A formula φ is *enforceable* iff there *exists* a transducer m such that m *enforces* φ.

Definition 3 depends on what is considered to be an adequate definition for "m *enforces* φ". It is reasonable to expect that the latter definition should concern *any* system that the transducer m– hereafter referred to as the *enforcer* – is instrumented with. In particular, for *any* system s, the resulting composite system obtained from instrumenting the enforcer m with it should satisfy the property of interest, φ, whenever this property *is satisfiable*.

▶ **Definition 4** (Sound Enforcement)**.** Enforcer m *soundly enforces* a formula φ, denoted as $\mathsf{senf}(m, \varphi)$, iff for *all* $s \in \text{SYS}$, $\varphi \in Sat$ implies $m[s] \in [\![\varphi]\!]$ holds.

▶ **Example 5.** Recall φ_1, $s_{\mathbf{g}}$ and $s_{\mathbf{b}}$ from Example 1 where $s_{\mathbf{g}} \in [\![\varphi_1]\!]$ (hence $\varphi_1 \in \text{SAT}$) and $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$. For the enforcers $m_{\mathbf{i}}$, $m_{\mathbf{r}}$, $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$ presented in Example 2, we have:

- $m_{\mathbf{i}}[s_{\mathbf{b}}] \notin [\![\varphi_1]\!]$, since $m_{\mathbf{i}}[s_{\mathbf{b}}] \xrightarrow{i?\mathsf{req}} \cdot \xrightarrow{i!\mathsf{ans}} \mathsf{id}[s_{\mathbf{b}}] \xrightarrow{i?\mathsf{req}} \mathsf{id}[s_{\mathbf{b}}] \xrightarrow{i?\mathsf{req}} \mathsf{id}[s_{\mathbf{b}}]$. This counter example implies that $\neg\mathsf{senf}(m_{\mathbf{i}}, \varphi_1)$.
- $m_{\mathbf{r}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{r}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$. Intuitively, this is because the ensuing instrumented systems only generate (replaced) actions that are not of concern to φ_1. Since this behaviour applies to any system $m_{\mathbf{r}}$ is composed with, we can conclude that $\mathsf{senf}(m_{\mathbf{r}}, \varphi_1)$.
- $m_{\mathbf{s}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{s}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$ because the resulting instrumented systems never produce inputs with req on a port number other than j. We can thus conclude that $\mathsf{senf}(m_{\mathbf{s}}, \varphi_1)$.
- $m_{\mathbf{t}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{t}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$. Since the resulting instrumentation suppresses consecutive input requests (if any) after any number of serviced requests on any port other than j, we can conclude that $\mathsf{senf}(m_{\mathbf{t}}, \varphi_1)$.

By some measures, sound enforcement is a relatively weak requirement for adequate enforcement as it does not regulate the *extent* of the induced enforcement. More concretely, consider the case of enforcer $m_{\mathbf{s}}$ from Example 2. Although $m_{\mathbf{s}}$ manages to suppress the violating executions of system $s_{\mathbf{b}}$, thereby bringing it in line with property φ_1, it needlessly modifies the behaviour of $s_{\mathbf{g}}$ (namely it prohibits it from producing any inputs with req on port numbers that are not j), even though it satisfies φ_1. Thus, in addition to sound enforcement we require a *transparency* condition for adequate enforcement. The requirement dictates that whenever a system s already satisfies the property φ, the assigned enforcer m should not alter the behaviour of s. Put differently, the behaviour of the enforced system should be behaviourally equivalent to the original system.

▶ **Definition 6** (Transparent Enforcement)**.** An enforcer m is *transparent* when enforcing a formula φ, denoted as $\mathsf{tenf}(m, \varphi)$, iff for *all* $s \in \text{SYS}$, $s \in [\![\varphi]\!]$ implies $m[s] \sim s$.

▶ **Example 7.** We have already argued – via the counter example $s_{\mathbf{g}}$– why $m_{\mathbf{s}}$ does *not* transparently enforce φ_1. We can also argue easily why $\neg\mathsf{tenf}(m_{\mathbf{r}}, \varphi_1)$ either: the simple system $i?\mathsf{req}.\mathsf{nil}$ trivially satisfies φ_1 but, clearly, we have the inequality $m_{\mathbf{r}}[i?\mathsf{req}.\mathsf{nil}] \not\sim i?\mathsf{req}.\mathsf{nil}$ since $m_{\mathbf{r}}[i?\mathsf{req}.\mathsf{nil}] \xrightarrow{j?\mathsf{req}} m_{\mathbf{r}}[\mathsf{nil}]$ and $i?\mathsf{req}.\mathsf{nil} \not\xrightarrow{j?\mathsf{req}}$.

It turns out that enforcer $\mathsf{tenf}(m_{\mathbf{t}}, \varphi_1)$, however. Although this property is not as easy to show – due to the universal quantification over all systems – we can get a fairly good intuition for why this is the case via the example $s_{\mathbf{g}}$: it satisfies φ_1 and $m_{\mathbf{t}}[s_{\mathbf{g}}] \sim s_{\mathbf{g}}$ holds.

▶ **Definition 8** (Enforcement)**.** A monitor m enforces property φ whenever it does so (i) soundly, Definition 4 and (ii) transparently, Definition 6.

For any reasonably expressive logic (such as μHML), it is usually the case that *not* every formula can be enforced, as the following example informally illustrates.

$$\varphi, \psi \in \text{sHML} \ ::= \ \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \bigwedge_{i \in I} \varphi_i \quad | \quad [\{p, c\}]\varphi \quad | \quad X \quad | \quad \max X.\varphi$$

Figure 3 The syntax for the safety μHML fragment, sHML.

▶ **Example 9.** Consider the μHML property φ_{ns}, together with the two systems s_{ra} and s_{r}:

$$\varphi_{\mathsf{ns}} \stackrel{\text{def}}{=} [i?\mathsf{req}]\mathsf{ff} \vee [i!\mathsf{ans}]\mathsf{ff} \qquad s_{\mathsf{ra}} \stackrel{\text{def}}{=} i?\mathsf{req}.\mathsf{nil} + i!\mathsf{ans}.\mathsf{nil} \qquad s_{\mathsf{r}} \stackrel{\text{def}}{=} i?\mathsf{req}.\mathsf{nil}$$

A system satisfies φ_{ns} if *either* it cannot produce action $i?\mathsf{req}$ *or* it cannot produce action $i!\mathsf{ans}$. Clearly, s_{ra} violates this property as it can produce both. This system can only be enforced via action suppressions or replacements because insertions would immediately break transparency. Without loss of generality, assume that our monitors employ suppressions (the same argument applies for action replacement). The monitor $m_{\mathsf{r}} \stackrel{\text{def}}{=} \mathsf{rec}\, y.\big(\{i?\mathsf{req}, \tau\}.y + \{i!\mathsf{ans}, \tau\}.y\big)$ would in fact be able to suppress the offending actions produced by s_{ra}, thus obtaining $m_{\mathsf{r}}[s_{\mathsf{ra}}] \in [\![\varphi_{\mathsf{ns}}]\!]$. However, it would also suppress the sole action $i?\mathsf{req}$ produced by the system s_{r}, even though this system satisfies φ_{ns}. This would, in turn, violate the transparency criterion of Definition 6 since it needlessly suppresses s_{r}'s actions, i.e., although $s_{\mathsf{r}} \in [\![\varphi_{\mathsf{ns}}]\!]$ we have $m_{\mathsf{r}}[s_{\mathsf{r}}] \not\sim s_{\mathsf{r}}$. The intuitive reason for this problem is that a monitor cannot, in principle, look into the computation graph of a system, but is limited to the behaviour the system exhibits at runtime.

5 Synthesising Suppression Enforcers

Despite their merits, Definitions 3 and 8 are not easy to work with. The universal quantifications over all systems in Definitions 4 and 6 make it hard to establish that a monitor correctly enforces a property. Moreover, according to Definition 3, in order to determine whether a particular property is enforceable or not, one would need to show the existence of a monitor that correctly enforces it; put differently, showing that a property is *not* enforceable entails another universal quantification, this time showing that no monitor can possibly enforce the property. Lifting the question of enforceability to the level of a (sub)logic entails a further universal quantification, this time on all the logical formulas of the logic; this is often an infinite set.We address these problems in two ways. First, we identify a non-trivial syntactic subset of μHML that is *guaranteed to be enforceable*; in a multi-pronged approach to system verification, this could act as a guide for whether the property should be considered at a pre-deployment or post-deployment phase. Second, for *every* formula φ in this enforceable subset, we provide an *automated procedure* to *synthesise* a monitor m from it that correctly enforces φ when instrumented over arbitrary systems, according to Definition 8. This procedure can then be used as a basis for constructing tools that automate property enforcement.

In this paper, we limit our enforceability study to suppression monitors, transducers that are only allowed to intervene by dropping (observable) actions. Despite being more constrained, suppression monitors side-step problems associated with what data to use in a payload-carrying action generated by the enforcer, as in the case of insertion and replacement monitors: the notion of a default value for certain data domains is not always immediate. Moreover, suppression monitors are particularly useful for enforcing *safety* properties, as shown in [33, 10, 20]. Intuitively, a suppression monitor would suppress actions as soon as it becomes apparent that a violation is about to be committed by the SuS. Such an intervention intrinsically relies on the *detection* of a violation. To this effect, we use a prior result from

[25], which identified a maximally-expressive logical fragment of μHML that can be handled by violation-detecting (recogniser) monitors. We thus limit our enforceability study to this maximal safety fragment, called sHML, since a *transparent* suppression monitor cannot judiciously suppress actions without first detecting a (potential) violation. Figure 3 recalls the syntax for sHML. The logic is restricted to *truth* and *falsehood* (tt and ff), conjunctions ($\bigwedge_{i\in I}\varphi$), and necessity modalities ($[\{p,c\}]\varphi$), while recursion may only be expressed through greatest fixpoints ($\mathsf{max}\, X.\varphi$); the semantics follows that of Figure 1.

A standard way how to achieve our aims would be to (i) define a (total) synthesis function $(\!|-|\!) :: \text{sHML} \mapsto \text{Trn}$ from sHML formulas to suppression monitors and (ii) then show that for *any* $\varphi \in \text{sHML}$, the synthesised monitor $(\!|\varphi|\!)$ enforces φ. Moreover, we would also require the synthesis function to be compositional, whereby the definition of the enforcer for a composite formula is defined in terms of the enforcers obtained for the constituent subformulas. There are a number of reasons for this requirement. For one, it would simplify our analysis of the produced monitors and allow us to use standard inductive proof techniques to prove properties about the synthesis function, such as the aforementioned criteria (ii). However, a naive approach to such a scheme is bound to fail, as discussed in the next example.

▶ **Example 10.** Consider a semantically equivalent reformulation of φ_1 from Example 1.

$$\varphi_2 \stackrel{\text{def}}{=} \mathsf{max}\, X.([\{(d)?\mathsf{req}, d{\neq}j\}][\{d!\mathsf{ans}, \mathsf{true}\}]X) \wedge ([\{(d)?\mathsf{req}, d{\neq}j\}][\{d?\mathsf{req}, \mathsf{true}\}]\mathsf{ff})$$

At an intuitive level, the suppression monitor that one would expect to obtain for the subformula $\varphi_2' \stackrel{\text{def}}{=} [\{(d)?\mathsf{req}, d{\neq}j\}][\{d?\mathsf{req}, \mathsf{true}\}]\mathsf{ff}$ is $\{(d)?\mathsf{req}, d \neq j\}.\mathsf{rec}\, y.\{d?\mathsf{req}, \tau\}.y$ (i.e., an enforcer that repeatedly drops any req inputs following a req input on the same port), whereas the monitor obtained for the subformula $\varphi_2'' \stackrel{\text{def}}{=} [\{(d)?\mathsf{req}, d{\neq}j\}][\{d!\mathsf{ans}, \mathsf{true}\}]X$ is $\{(d)?\mathsf{req}, d \neq j\}.\{d!\mathsf{ans}\}.x$ (assuming some variable mapping from X to x). These monitors would then be combined in the synthesis for $\mathsf{max}\, X.\varphi_2'' \wedge \varphi_2'$ as

$$m_{\mathbf{b}} \stackrel{\text{def}}{=} \mathsf{rec}\, x.\big(\{(d)?\mathsf{req}, d \neq j\}.\{d!\mathsf{ans}\}.x\big) \;+\; \big(\{(d)?\mathsf{req}, d \neq j\}.\mathsf{rec}\, y.\{d?\mathsf{req}, \tau\}.y\big)$$

One can easily see that $m_{\mathbf{b}}$ does *not* behave deterministically, *nor* does it soundly enforce φ_2. For instance, for the violating system $i?\mathsf{req}.i?\mathsf{req}.\mathsf{nil} \notin [\![\varphi_2]\!] (= [\![\varphi_1]\!])$ we can observe the transition sequence $m_{\mathbf{b}}[i?\mathsf{req}.i?\mathsf{req}.\mathsf{nil}] \xrightarrow{i?\mathsf{req}} \{i!\mathsf{ans}\}.m_{\mathbf{b}}[i?\mathsf{req}.\mathsf{nil}] \xrightarrow{i?\mathsf{req}} \mathsf{id}[\mathsf{nil}]$.

Instead of complicating our synthesis function to cater for anomalies such as those presented in Example 10 – also making it *less* compositional in the process – we opted for a two stage synthesis procedure. First, we consider a *normalised* subset for sHML formulas which is amenable to a (straightforward) synthesis function definition that is compositional. This also facilitates the proofs for the conditions required by Definition 8 for any synthesised enforcer. Second, we show that every sHML formula can be reformulated in this normalised form without affecting its semantic meaning. We can then show that our two-stage approach is expressive enough to show the enforceability for all of sHML.

▶ **Definition 11** (sHML normal form)**.** The set of normalised sHML formulas is defined as:

$$\varphi, \psi \in \text{sHML}_{\mathbf{nf}} ::= \mathsf{tt} \;\mid\; \mathsf{ff} \;\mid\; \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \;\mid\; X \;\mid\; \mathsf{max}\, X.\varphi\ .$$

The above grammar combines necessity operators with conjunctions into one construct $\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$. Normalised sHML formulas are required to satisfy two further conditions:

1. For every $\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, for all $j, h \in I$ where $j{\neq}h$ we have $[\![\{p_j, c_j\}]\!] \cap [\![\{p_h, c_h\}]\!] = \varnothing$.
2. For every $\mathsf{max}\, X.\varphi$ we have $X \in \mathbf{fv}(\varphi)$.

In a (closed) normalised sHML formula, the basic terms tt and ff can never appear unguarded unless they are at the top level (e.g., we can never have $\varphi \wedge \mathsf{ff}$ or $\mathsf{max}\, X_0. \ldots \mathsf{max}\, X_n.\mathsf{ff}$). Moreover, in any conjunction of necessity subformulas, $\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, the necessity guards are *disjoint* and *at most one* necessity guard can satisfy any particular action.

▶ **Definition 12.** The synthesis function $(\!|-|\!) : \text{sHML}_{\mathbf{nf}} \mapsto \text{TRN}$ is defined inductively as:

$$(\!| X |\!) \stackrel{\text{def}}{=} x \qquad (\!| \mathsf{tt} |\!) \stackrel{\text{def}}{=} (\!| \mathsf{ff} |\!) \stackrel{\text{def}}{=} \mathsf{id} \qquad (\!| \mathsf{max}\, X.\varphi |\!) \stackrel{\text{def}}{=} \mathsf{rec}\, x.(\!| \varphi |\!)$$

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i |\!) \stackrel{\text{def}}{=} \mathsf{rec}\, y.\sum_{i\in I} \begin{cases} \{p_i, c_i, \tau\}.y & \text{if } \varphi_i{=}\mathsf{ff} \\ \{p_i, c_i, \underline{p_i}\}.(\!| \varphi_i |\!) & \text{otherwise} \end{cases}$$

The synthesis function is compositional. It assumes a bijective mapping between formula variables and monitor recursion variables and converts logical variables X accordingly, whereas maximal fixpoints, $\mathsf{max}\, X.\varphi$, are converted into the corresponding recursive enforcer. The synthesis also converts truth and falsehood formulas, tt and ff, into the identity enforcer id. Normalized conjunctions, $\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, are synthesised into a *recursive summation* of enforcers, i.e., $\mathsf{rec}\, y.m_i$, where y is fresh, and every branch m_i can be either of the following:

(i) when m_i is derived from a branch of the form $[\{p_i, c_i\}]\varphi_i$ where $\varphi_i{\neq}\mathsf{ff}$, the synthesis produces an enforcer with the *identity transformation* prefix, $\{p_i, c_i, \underline{p_i}\}$, followed by the enforcer synthesised from the continuation φ_i, i.e., $[\{p_i, c_i\}]\varphi_i$ is synthesised as $\{p_i, c_i, \underline{p_i}\}.(\!| \varphi_i |\!)$;

(ii) when m_i is derived from a branch of the form $[\{p_i, c_i\}]\mathsf{ff}$, the synthesis produces a *suppression transformation*, $\{p_i, c_i, \tau\}$, that drops every concrete action matching the symbolic action $\{p_i, c_i\}$, followed by the recursive variable of the branch y, i.e., a branch of the form $[\{p_i, c_i\}]\mathsf{ff}$ is translated into $\{p_i, c_i, \tau\}.y$.

▶ **Example 13.** Recall formula φ_1 from Example 1, recast in term of $\text{sHML}_{\mathbf{nf}}$'s grammar:

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\, X.\bigwedge \big([\{(d)?\mathsf{req}, d{\neq}j\}] \ ([\{d!\mathsf{ans}, \mathsf{true}\}]X \wedge [\{d?\mathsf{req}, \mathsf{true}\}]\mathsf{ff})\big)$$

Using the synthesis function defined in Definition 12, we can generate the enforcer

$$(\!| \varphi_1 |\!) = \mathsf{rec}\, x.\mathsf{rec}\, z.\sum\big(\{(d)?\mathsf{req}, d{\neq}j\}.\mathsf{rec}\, y.(\{d!\mathsf{ans}, \mathsf{true}\}.x \ + \ \{d?\mathsf{req}, \mathsf{true}, \tau\}.y)\big)$$

which can be optimized by removing redundant recursive constructs (e.g., $\mathsf{rec}\, z.\_$), obtaining:

$$= \mathsf{rec}\, x.\{(d)?\mathsf{req}, d{\neq}j\}.\mathsf{rec}\, y.(\{d!\mathsf{ans}, \mathsf{true}\}.x \ + \ \{d?\mathsf{req}, \mathsf{true}, \tau\}.y) \ = \ m_{\mathbf{t}}$$

We now present the first main result to the paper.

▶ **Theorem 14** (Enforcement). *The (sub)logic* $\text{sHML}_{\boldsymbol{nf}}$ *is enforceable.*

Proof. By Definition 3, the result follows if we show that for all $\varphi \in \text{sHML}_{\mathbf{nf}}$, $(\!| \varphi |\!)$ *enforces* φ. By Definition 8, this is a corollary following from Propositions 15 and 16 stated below. ◀

▶ **Proposition 15** (Enforcement Soundness). *For every system* $s \in \text{Sys}$ *and* $\varphi \in \text{sHML}_{\boldsymbol{nf}}$ *then* $\varphi \in \text{Sat}$ implies $(\!| \varphi |\!)[s] \in [\![\varphi]\!]$.

▶ **Proposition 16** (Enforcement Transparency). *For every system* $s \in \text{Sys}$ *and* $\varphi \in \text{sHML}_{\boldsymbol{nf}}$ *then* $s \in [\![\varphi]\!]$ implies $(\!| \varphi |\!)[s] \sim s$.

Following Theorem 14, to show that sHML is an enforceable logic, we only need to show that for every $\varphi \in \text{sHML}$ there exists a corresponding $\psi \in \text{sHML}_{\mathbf{nf}}$ with the same semantic meaning, i.e., $[\![\varphi]\!] = [\![\psi]\!]$. In fact, we go a step further and provide a constructive proof using a transformation $\langle\!\langle - \rangle\!\rangle : \text{sHML} \mapsto \text{sHML}_{\mathbf{nf}}$ that derives a semantically equivalent $\text{sHML}_{\mathbf{nf}}$ formula from a standard sHML formula. As a result, from an arbitrary sHML formula φ we can then automatically synthesise a correct enforcer using $(\!|\langle\!\langle \varphi \rangle\!\rangle|\!)$ which is useful for tool construction.

Our transformation $\langle\!\langle \varphi \rangle\!\rangle$ relies on a number of steps; here we provide an outline of these steps. First, we assume sHML formulas that only use symbolic actions with *normalised* patterns p, i.e., patterns that do not use any data or free data variables (but they may use bound data variables). In fact, any symbolic action $\{p, c\}$ can be easily converted into a corresponding one using normalised patterns as shown in the next example.

▶ **Example 17.** Consider the symbolic action $\{d!\mathsf{ans}, d \neq j\}$. It may be converted to a corresponding normalised symbolic action by replacing every occurrence of a data or free data variable in the pattern by a fresh bound variable, and then add an equality constraint between the fresh variable and the data or data variable it replaces in the pattern condition. In our case, we would obtain $\{(e)!(f), d{\neq}j \wedge e{=}d \wedge f{=}\mathsf{ans}\}$.

Our algorithm for converting sHML formulas (with normalised patterns) to $\text{sHML}_{\mathbf{nf}}$ formulas, $\langle\!\langle - \rangle\!\rangle$, is based on Rabinovich's work [41] for determinising systems of equations which, in turn relies on the standard powerset construction for converting NFAs into DFAs. It consists in the following six stages that we outline below:

1. We unfold each recursive construct in the formula, to push recursive definitions inside the formula body. E.g., the formula $\mathsf{max}\, X.\big([\{p_1, c_1\}]X \wedge [\{p_2, c_2\}]\mathsf{ff}\big)$ is expanded to the formula $[\{p_1, c_1\}]\big(\mathsf{max}\, X.[\{p_1, c_1\}]X \wedge [\{p_2, c_2\}]\mathsf{ff}\big) \wedge [\{p_2, c_2\}]\mathsf{ff}$.
2. The formula is converted into a system of equations. E.g., the expanded formula from the previous stage is converted into the set $\{X_0 = [\{p_1, c_1\}]X_0 \wedge [\{p_2, c_2\}]X_1, X_1 = \mathsf{ff}\}$.
3. For every equation, the symbolic actions in the right hand side that are of the same kind are alpha-converted so that their bound variables match. E.g., Consider $X_0 = [\{p_1, c_1\}]X_0 \wedge [\{p_2, c_2\}]X_1$ from the previous stage where, for the sake of the example, $p_1 = (d_1)?(d_2)$ and $p_2 = (d_3)?(d_4)$. The patterns in the symbolic actions are made syntactically equivalent by renaming d_3 and d_4 in $\{p_2, c_2\}$ into d_1 and d_2 respectively.
4. For equations with matching patterns in the symbolic actions, we create a variant that symbolically covers all the (satisfiable) permutations on the symbolic action conditions. E.g., Consider $X_0 = [\{p_1, c_1\}]X_0 \wedge [\{p_1, c_3\}]X_1$ from the previous stage. We expand this to $X_0 = [\{p_1, c_1 \wedge c_3\}]X_0 \wedge [\{p_1, c_1 \wedge c_3\}]X_1 \wedge [\{p_1, c_1 \wedge \neg(c_3)\}]X_0 \wedge [\{p_1, \neg(c_1) \wedge c_3\}]X_1$.
5. For equations with branches having *syntactically equivalent* symbolic actions, we carry out a unification procedure akin to standard powerset constructions. E.g., we convert the equation from the previous step to $X_{\{0\}} = [\{p_1, c_1 \wedge c_3\}]X_{\{0,1\}} \wedge [\{p_1, c_1 \wedge \neg(c_3)\}]X_{\{0\}} \wedge [\{p_1, \neg(c_1) \wedge c_3\}]X_{\{1\}}$ using the (unified) fresh variables $X_{\{0\}}, X_{\{1\}}$ and $X_{\{0,1\}}$.
6. From the unified set of equations we generate again the sHML formula starting from $X_{\{0\}}$. This procedure may generate redundant recursion binders, i.e., $\mathsf{max}\, X.\varphi$ where $X \notin \mathbf{fv}(\varphi)$, and we filter these out in a subsequent pass.

We now state the second main result of the paper.

▶ **Theorem 18** (Normalisation). *For any* $\varphi{\in}\text{sHML}$ *there exists* $\psi{\in}\text{sHML}_{\boldsymbol{nf}}$ *s.t.* $[\![\varphi]\!]{=}[\![\psi]\!]$.

Proof. The witness formula in normal form is $\langle\!\langle \varphi \rangle\!\rangle$, where we show that each and every stage in the translation procedure preserves semantic equivalence. ◀

6 Alternative Transparency Enforcement

Transparency for a property φ, Definition 6, only restricts enforcers from modifying the behaviour of satisfying systems, i.e., when $s \in [\![\varphi]\!]$, but fails to specify any enforcement behaviour for the cases when the SuS violates the property $s \notin [\![\varphi]\!]$. In this section, we consider an alternative transparency requirement for a property φ that incorporates the expected enforcement behaviour for *both* satisfying and violating systems. More concretely, in the case of safety languages such as sHML, a system typically violates a property along a specific set of execution traces; in the case of a satisfying system this set of "violating traces" is *empty*. However, not every behaviour of a violating system would be part of this set of violating traces and, in such cases, the respective enforcer should be required to leave the generated behaviour unaffected.

▶ **Definition 19** (Violating-Trace Semantics)**.** A logic $\mathcal{L}$ with an interpretation over systems $[\![-]\!] : \mathcal{L} \mapsto \mathcal{P}(\text{Sys})$ has a violating-trace semantics whenever it has a secondary interpretation $[\![-]\!]_v : \mathcal{L} \mapsto \mathcal{P}(\text{Sys} \times \text{Act}^*)$ satisfying the following conditions for all $\varphi \in \mathcal{L}$:

1. $(s,t) \in [\![\varphi]\!]_v$ implies $s \notin [\![\varphi]\!]$ and $s \stackrel{t}{\Longrightarrow}$,
2. $s \notin [\![\varphi]\!]$ implies $\exists t \cdot (s,t) \in [\![\varphi]\!]_v$.

We adapt the work in [26] to give sHML a violating-trace semantics. Intuitively, the judgement $(s,t) \in [\![\varphi]\!]_v$ according to Definition 20 below, denotes the fact that s violates the sHML property φ along trace t.

▶ **Definition 20** (Alternative Semantics for sHML [26])**.** The forcing relation $\vdash_v \subseteq (\text{Sys} \times \text{Act}^* \times \text{sHML})$ is the least relation satisfying the following rules:

$$
\begin{array}{rl}
(s, \epsilon, \mathsf{ff}) \in \mathcal{R} & \text{always} \\
(s, t, \bigwedge_{i \in I} \varphi_i) \in \mathcal{R} & \text{if } \exists j \in I \text{ such that } (s, t, \varphi_j) \in \mathcal{R} \\
(s, \alpha t, [\{p, c\}]\varphi) \in \mathcal{R} & \text{if } \mathsf{mtch}(p, \alpha) = \sigma, c\sigma \Downarrow \mathsf{true} \text{ and } s \stackrel{\alpha}{\Longrightarrow} s' \text{ and } (s', t, \varphi\sigma) \in \mathcal{R} \\
(s, t, \mathsf{max}\, X.\varphi) \in \mathcal{R} & \text{if } (s, t, \varphi\{\mathsf{max}\, X.\varphi / X\}) \in \mathcal{R} \ .
\end{array}
$$

We write $s, t \vdash_v \varphi$ (or $(s,t) \in [\![\varphi]\!]_v$) in lieu of $(s,t,\varphi) \in \vdash_v$. We say that trace t is a *violating trace* for s with respect to φ whenever $s, t \vdash_v \varphi$. Dually, t is a *non-violating trace* for φ whenever there does *not* exist a system s such that $s, t \vdash_v \varphi$.

▶ **Example 21.** Recall $\varphi_1, s_{\mathbf{b}}$ from Example 1 where $\varphi_1 \in$ sHML, and also $m_{\mathbf{t}}$ from Example 5 where we argued in Example 13 that $(\!|\, \varphi_1 \,|\!) = m_{\mathbf{t}}$ (modulo cosmetic optimisations). Even though $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$, not all of its exhibited behaviours constitute violating traces: for instance, $s_{\mathbf{b}} \stackrel{i?\mathsf{req} \cdot i!\mathsf{ans}}{\Longrightarrow} s_{\mathbf{b}}$ is not a violating trace according to Definition 20. Correspondingly, we also have $m_{\mathbf{t}}[s_{\mathbf{b}}] \stackrel{i?\mathsf{req} \cdot i!\mathsf{ans}}{\Longrightarrow} m_{\mathbf{t}}[s_{\mathbf{b}}]$.

▶ **Theorem 22** (Adapted and extended from [26])**.** *The alternative interpretation $[\![-]\!]_v$ of Definition 20 is a violating-trace semantics for* sHML *(with $[\![-]\!]$ from Figure 1) in the sense of Definition 19.*

Equipped with Definition 20 we can define an alternative definition for transparency that concerns itself with preserving exhibited traces that are non-violating. We can then show that the monitor synthesis for sHML of Definition 12 observes non-violating trace transparency.

▶ **Definition 23** (Non-Violating Trace Transparency). An enforcer m is *transparent* with respect to the non-violating traces of a formula φ, denoted as $\mathsf{nvtenf}(m, \varphi)$, iff for *all* $s \in \textsc{Sys}$ and $t \in \textsc{Act}^*$, when $s, t \not\models_v \varphi$ then

- $s \overset{t}{\Longrightarrow} s'$ *implies* $m[s] \overset{t}{\Longrightarrow} m'[s']$ for some m', and
- $m[s] \overset{t}{\Longrightarrow} m'[s']$ *implies* $s \overset{t}{\Longrightarrow} s'$.

▶ **Proposition 24** (Non-Violating Trace Transparency). *For all* $\varphi \in \text{sHML}$, $s \in \textsc{Sys}$ *and* $t \in \textsc{Act}^*$, *when* $s, t \not\models_v \varphi$ *then*

- $s \overset{t}{\Longrightarrow} s'$ implies $(\!|\varphi|\!)[s] \overset{t}{\Longrightarrow} m'[s']$, *and*
- $(\!|\varphi|\!)[s] \overset{t}{\Longrightarrow} m'[s']$ implies $s \overset{t}{\Longrightarrow} s'$.

We can thus obtain a new definition for "m *enforces* φ" instead of Definition 8 by requiring sound enforcement, Definition 6, and non-violating trace transparency, Definition 23 (instead of the transparent enforcement of Definition 6). This in turn gives us a new definition for enforceability for a logic, akin to Definition 3. Using Propositions 15 and 24, one can show that sHML is also enforceable with respect to the new definition as well.

7 Conclusion

This paper presents a preliminary investigation of the enforceability of properties expressed in a process logic. We have focussed on a highly expressive and standard logic, μHML, and studied the ability to enforce μHML properties via a specific kind of monitor that performs suppression-based enforcement. We concluded that sHML, identified in earlier work as a maximally expressive safety fragment of μHML, is also an enforceable logic. To show this, we first defined enforceability for logics and system descriptions interpreted over labelled transition systems. Although enforceability builds upon soundness and transparency requirements that have been considered in other work, our branching-time framework allowed us to consider novel definitions for these requirements. We also contend that the definitions that we develop for the enforcement framework are fairly modular: e.g., the instrumentation relation is independent of the specific language constructs defining our transducer monitors and it functions as expected as long as the transition semantics of the transducer and the system are in agreement. Based on this notion of enforcement, we devise a two-phase procedure to synthesise correct enforcement monitors. We first identify a syntactic subset of our target logic sHML that affords certain structural properties and permits a compositional definition of the synthesis function. We then show that, by augmenting existing rewriting techniques to our setting, we can convert any sHML formula into this syntactic subset.

Related Work

In his seminal work [44], Schneider regards a property (in a linear-time setting) to be enforceable if its *violation* can be *detected* by a *truncation automaton*, and prevents its occurrence via system termination; by preventing misbehaviour, these enforcers can only enforce safety properties. Ligatti et al. in [33] extended this work via *edit automata* – an enforcement mechanism capable of *suppressing* and *inserting* system actions. A property is thus enforceable if it can be expressed as an edit automaton that *transforms* invalid executions into valid ones via suppressions and insertions. Edit automata are capable of enforcing instances of safety and liveness properties, along with other properties such as infinite renewal properties [33, 10]. As a means to assess the correctness of these automata, the authors introduced *soundness* and *transparency*. In both of these settings, there is no

clear separation between the specification and the enforcement mechanism, and properties are encoded in terms of the languages accepted by the enforcement model itself, i.e., as edit/truncation automata. By contrast, we keep the specification and verification aspects of the logic separate.

Bielova et al. [10, 11] remark that soundness and transparency do not specify to what extent a transducer should modify an invalid execution. They thus introduce a *predictability* criterion to prevent transducers from transforming invalid executions arbitrarily. More concretely, a transducer is *predictable* if one can predict the number of transformations that it will apply in order to transform an invalid execution into a valid one, thereby preventing enforcers from applying unnecessary transformations over an invalid execution. Using this notion, Bielova et al. thus devise a more stringent notion of enforceability. Although we do not explore this avenue, Definition 23 may be viewed as an attempt to constrain transformations of violating systems in a branching-time setup, and should be complementary to these predictability requirements.

Könighofer et al. in [29] present a synthesis algorithm that produces action replacement transducers called *shields* from safety properties encoded as automata-based specifications. Shields analyse the inputs and outputs of a reactive systems and enforce properties by modifying the least amount of output actions whenever the system deviates from the specified behaviour. By definition, shields should adhere to two desired properties, namely correctness and minimum deviation which are, in some sense, analogous to soundness and transparency respectively. Falcone et al. in [19, 21, 20], also propose synthesis procedures to translate properties − expressed as Streett automata − into the resp., enforcers. The authors show that most of the property classes defined within the *Safety-Progress hierarchy* [35] are enforceable, as they can be encoded as Streett automata and subsequently converted into enforcement automata. As opposed to Ligatti et al., both Könighofer et al. and Falcone et al. separate the specification of the property from the enforcement mechanism, but unlike our work they do not study the enforceability of a branching time logic.

To the best of our knowledge, the only other work that tackles enforceability for the modal μ-calculus [30] (a reformulation of μHML) is that of Martinelli et al. in [37, 38]. Their approach is, however, different from ours. In addition to the μ-calculus formula to enforce, their synthesis function also takes a "witness" system satisfying the formula as a parameter. This witness system is then used as the behaviour that is mimicked by the instrumentation via suppression, insertion or replacement mechanisms. Although the authors do not explore automated correctness criteria such as the ones we study in this work, it would be interesting to explore the applicability of our methods to their setting.

Bocchi et al. [12] adopt *multi-party session types* to project the global protocol specifications of distributed networks to *local types* defining a local protocol for every process in the network that are then either verified statically via typechecking or enforced dynamically via suppression monitors. To implement this enforcement strategy, the authors define a dynamic monitoring semantics for the local types that suppress process interactions so as to conform to the assigned local specification. They prove local soundness and transparency for monitored processes that, in turn, imply global soundness and transparency by construction. Their local enforcement is closely related to the suppression enforcement studied in our work with the following key differences: (*i*) well-formed branches in a session type are, by construction, *explicitly disjoint* via the use of distinct choice labels (i.e., similar to our normalised subset $\text{sHML}_{\mathbf{nf}}$), whereas we can synthesise enforcers for *every* sHML formula using a normalisation procedure; (*ii*) they give an LTS semantics to their local specifications (which are session types) which allows them to state that a process satisfies a specification when its behaviour is bisimilar to the operational semantics of the local specification – we do

not change the semantics of our formulas, which is left in its original denotational form; (*iii*) they do not provide transparency guarantees for processes that violate a specification, along the lines of Definition 23; (*iv*) Our monitor descriptions sit at a lower level of abstraction than theirs using a dedicated language, whereas theirs have a session-type syntax with an LTS semantics (e.g., repeated suppressions have to be encoded in our case using the recursion construct while this is handled by their high-level instrumentation semantics).

In [14], Castellani et al. adopt session types to define reading and writing privileges amongst processes in a network as global types for information flow purposes. These global types are projected into local monitors capable of preventing read and write violations by adapting certain aspects of the network. Although their work is pitched towards adaptation [24, 13], rather than enforcement, in certain instances they adapt the network by suppressing messages or by replacing messages with messages carrying a default nonce value. It would be worthwhile investigating whether our monitor correctness criteria could be adapted or extended to this information-flow setting.

Future Work

We plan to extend this work along two different avenues. On the one hand, we will attempt to extend the enforceable fragment of μHML. For a start, we intend to investigate maximality results for suppression monitors, along the lines of [25, 2]. We also plan to consider more expressive enforcement mechanisms such as insertion and replacement actions. Finally, we will also investigate more elaborate instrumentation setups, such as the ones explored in [1], that can reveal refusals in addition to the actions performed by the system.

On the other hand, we also plan to study the implementability and feasibility of our framework. We will consider target languages for our monitor descriptions that are closer to an actual implementation (e.g., an actor-based language along the lines of [26]). We could then employ refinement analysis techniques and use our existing monitor descriptions as the abstract specifications that are refined by the concrete monitor descriptions. The more concrete synthesis can then be used for the construction of tools that are more amenable towards showing correctness guarantees.

References

1 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A framework for parameterized monitorability. In *Foundations of Software Science and Computation Structures*, pages 203–220, Cham, 2018. Springer International Publishing.

2 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In Satya Lokam and R. Ramanujam, editors, *FSTTCS 2017: Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *LIPIcs*, pages 7:1–7:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007.

4 Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 599–610. ACM, 2011.

5 Henrik Reif Andersen. Partial model checking. In *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE, 1995.

6 Cyrille Artho, Howard Barringer, Allen Goldberg, Klaus Havelund, Sarfraz Khurshid, Michael R. Lowry, Corina S. Pasareanu, Grigore Rosu, Koushik Sen, Willem Visser, and

Richard Washington. Combining test case generation and runtime verification. *Theoretical Computer Science*, 336(2-3):209–234, 2005.

7 Duncan Paul Attard, Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. *A Runtime Monitoring Tool for Actor-Based Systems.*, chapter 3, pages 49–74. River Publishers, 2017.

8 Duncan Paul Attard and Adrian Francalanza. A monitoring tool for a branching-time logic. In *Runtime Verification*, pages 473–481, Cham, 2016. Springer International Publishing.

9 Jean Berstel and Luc Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

10 Nataliia Bielova. *A theory of constructive and predictable runtime enforcement mechanisms*. PhD thesis, University of Trento, 2011.

11 Nataliia Bielova and Fabio Massacci. Predictability of enforcement. In *International Symposium on Engineering Secure Software and Systems*, pages 73–86. Springer, 2011.

12 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theoretical Computer Science*, 669:33–58, 2017.

13 Ian Cassar and Adrian Francalanza. On implementing a monitor-oriented programming framework for actor systems. In *International Conference on Integrated Formal Methods*, pages 176–192. Springer, 2016.

14 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Aspects of Computing*, 28(4):669–696, July 2016.

15 Edward Chang, Zohar Manna, and Amir Pnueli. The safety-progress classification. In *Logic and Algebra of Specification*, pages 143–202. Springer, 1993.

16 Clare Cini and Adrian Francalanza. An LTL proof system for runtime verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 581–595. Springer, 2015.

17 Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *25 Years of Model Checking*, pages 196–215. Springer, 2008.

18 Ankush Desai, Tommaso Dreossi, and Sanjit A. Seshia. Combining model checking and runtime verification for safe robotics. In *Runtime Verfication (RV)*, LNCS, pages 172–189, Cham, 2017. Springer International Publishing.

19 Yliès Falcone. You should better enforce than verify. In *Runtime Verification*, pages 89–105. Springer Berlin Heidelberg, 2010.

20 Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349, jun 2012.

21 Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, jun 2011.

22 Adrian Francalanza. A Theory of Monitors. In *International Conference on Foundations of Software Science and Computation Structures*, pages 145–161. Springer, 2016.

23 Adrian Francalanza. Consistently-Detecting Monitors. In *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

24 Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In *Runtime Verification*, pages 8–29, Cham, 2017. Springer International Publishing.

25 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods in System Design*, 51(1):87–116, 2017.

26 Adrian Francalanza and Aldrin Seychell. Synthesising correct concurrent runtime monitors. *Formal Methods in System Design*, 46(3):226–261, 2015.

27 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*, pages 582–594, New York, NY, USA, 2016. ACM.

28 Katarína Kejstová, Petr Ročkai, and Jiří Barnat. From Model Checking to Runtime Verification and Back. In *RV*. Springer, 2017.

29 Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods in System Design*, 51(2):332–361, Nov 2017.

30 Dexter C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

31 Frédéric Lang and Radu Mateescu. Partial model checking using networks of labelled transition systems and boolean equation systems. In Cormac Flanagan and Barbara König, editors, *TACAS*, pages 141–156, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

32 Kim G Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990.

33 Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1):2–16, Feb 2005.

34 Jay Ligatti and Srikar Reddy. A theory of runtime enforcement, with results. In *CESORICS*, pages 87–100, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

35 Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In Cynthia Dwork, editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 377–410. ACM, 1990. doi:10.1145/93385.93442.

36 Fabio Martinelli and Ilaria Matteucci. Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties. In *Foundations of Computer Security*, pages 133–144. Citeseer, 2005.

37 Fabio Martinelli and Ilaria Matteucci. Through modeling to synthesis of security automata. *Electronic Notes in Theoretical Computer Science*, 179:31–46, 2006.

38 Fabio Martinelli and Ilaria Matteucci. An approach for the specification, verification and synthesis of secure systems. *Electronic Notes in Theoretical Computer Science*, 168:29–43, 2007.

39 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Information and computation*, 100(1):1–40, 1992.

40 Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *International Symposium on Formal Methods*, pages 573–586. Springer Berlin Heidelberg, 2006.

41 Alexander Moshe Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 530–543, London, UK, UK, 1994. Springer-Verlag.

42 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, New York, NY, USA, 2009.

43 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 2011.

44 Fred B Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

Regular Separability of Well-Structured Transition Systems

Wojciech Czerwiński[1]
University of Warsaw, Poland
wczerwin@mimuw.edu.pl
https://orcid.org/0000-0002-6169-868X

Sławomir Lasota[2]
University of Warsaw, Poland
sl@mimuw.edu.pl
https://orcid.org/0000-0001-8674-4470

Roland Meyer
TU Braunschweig, Germany
roland.meyer@tu-bs.de
https://orcid.org/0000-0001-8495-671X

Sebastian Muskalla
TU Braunschweig, Germany
s.muskalla@tu-bs.de
https://orcid.org/0000-0001-9195-7323

K. Narayan Kumar[3]
Chennai Mathematical Institute and UMI RELAX, India
kumar@cmi.ac.in

Prakash Saivasan
TU Braunschweig, Germany
p.saivasan@tu-bs.de

Abstract

We investigate the languages recognized by well-structured transition systems (WSTS) with upward and downward compatibility. Our first result shows that, under very mild assumptions, every two disjoint WSTS languages are regular separable: There is a regular language containing one of them and being disjoint from the other. As a consequence, if a language as well as its complement are both recognized by WSTS, then they are necessarily regular. In particular, no subclass of WSTS languages beyond the regular languages is closed under complement. Our second result shows that for Petri nets, the complexity of the backwards coverability algorithm yields a bound on the size of the regular separator. We complement it by a lower bound construction.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Formal languages and automata theory, Theory of computation → Regular languages, Theory of computation → Parallel computing models

Keywords and phrases regular separability, wsts, coverability languages, Petri nets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.35

[1] Supported by the Polish National Science Centre under grant 2016/21/D/ST6/01376.
[2] Partially supported by the European Research Council (ERC) project Lipa under the EU Horizon 2020 research and innovation programme (grant agreement No. 683080).
[3] Partially supported by the Indo-French project AVeCSo, the Infosys Foundation, and DST-VR Project P-02/2014.

Related Version The full version is available as technical report on arXiv [18], `https://arxiv.org/abs/1702.05334`.

Acknowledgements We thank an anonymous referee for pointing out Part (2) of Corollary 8. We thank Sylvain Schmitz for helpful discussions.

1 Introduction

We study the languages recognized by well-structured transition systems (WSTS) [24, 25, 5, 1, 28]. WSTS form a framework subsuming several widely-studied models, like Petri nets [23] and their extensions with transfer [22], data [54], and time [4], graph rewriting systems [36], depth-bounded systems [47, 57, 21], ad-hoc networks [3], process algebras [13], lossy channel systems (LCS) [5], and programs running under weak memory models [6, 7]. Besides their applicability, the importance of WSTS stems from numerous decidability results. Finkel showed the decidability of termination and boundedness [24, 25]. Abdulla came up with a backward algorithm for coverability [5], for which a matching forward procedure was found only much later [31]. Several simulation and equivalence problems are also decidable for WSTS [28]. The work on WSTS even influenced algorithms for regular languages [58] and recently led to the study of new complexity classes [55].

Technically, a WSTS is a transition system equipped with a quasi order on the configurations that satisfies two properties. It is a well quasi order and it is (upward or downward) compatible with the transition relation in the sense that it forms a simulation relation. For our language-theoretic study, we assume the transitions to be labeled and the WSTS to be equipped with sets of initial and final configurations. The set of final configurations is supposed to be upward or downward closed wrt. the quasi order of the WSTS. When specialized to VAS, this yields the so-called *covering languages*.

For WSTS languages, we study the problem of regular separability. Given two languages $\mathcal{L}$ and $\mathcal{K}$ over the same alphabet, a *separator* is a language $\mathcal{R}$ that contains one of the languages and is disjoint from the other, $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{R} \cap \mathcal{K} = \emptyset$. The separator is regular if it is a regular language. Separability has recently attracted considerable attention. We discuss the related work in a moment.

Disjointness is clearly necessary for regular separability. We show that for most WSTS, disjointness is also sufficient. Our main result is the following:

Any two disjoint WSTS languages are regular separable.

The only assumption we need is that, in the case of upward-compatible WSTS resp. downward-compatible WSTS, one of the WSTS is finitely branching resp. deterministic.

The proof proceeds in two steps. In the first step, we link inductive invariants from verification [43] to separability in formal languages. More precisely, we show that any inductive invariant (of the product of the given systems) gives rise to a regular separator – provided it can be finitely represented. We do not even need WSTS here, but only upward compatibility. An inductive invariant is a set of configurations that contains the initial ones, is closed under the transition relation, and is disjoint from the final configurations.

In a second step, we show that finitely-represented invariants always exist. To this end, we use ideal completions from lattice theory [37, 9, 27]. The insight is that, in a WSTS, any inductive invariant can be finitely represented by its ideal decomposition. This ideal decomposition yields states in the ideal completion of the WSTS, and the first step applies.

The result has theoretical as well as practical applications. On the theoretical side, recall the following about Petri nets from [49, 48]: Every two Petri net covering languages that are

complements of each other are necessarily regular. The result not only follows from ours, but the same applies to other classes of WSTS, for instance to the languages of LCS, and actually to *all* WSTS languages fulfilling the above-mentioned assumptions. For instance, if the covering language of a Petri net is the complement of the language of an LCS, they are necessarily regular; and if the languages are just disjoint, they are regular separable.

The result is also important in verification. In 2016 and 2017, the Software Verification Competition was won by so-called language-theoretic algorithms [33]. These algorithms replace the classical state-space search by proofs of language disjointness (between a refinement of the control-flow language and the language of undesirable behavior). Regular separators are precisely what is needed to prove disjointness. In this setting, regular separators seem to play the role that inductive invariants play for safety verification [43]. Indeed, our results establishes a first link between the two.

We accompany our main result by two more findings. The first ones are determinization results that broaden the applicability of our results. For upward compatibility, we show that every finitely branching WSTS can be determinized. For downward compatibility, we show that every WSTS can be determinized if the quasi order is an ω^2-wqo. In fact all examples from the literature are ω^2-WSTS, hence they determinize, and in consequence satisfy the assumptions of our results.

Our second accompanying result is on the size of regular separators for Petri nets. We show how to construct a regular separator in the form of a non-deterministic automaton of size triply exponential in size of the given nets. With the main result at hand, the result amounts to giving a bound on the size of a finite representation of an inductive invariant. As inductive invariant, we use the complement of the configurations backward reachable from the final ones. The estimation starts from a result on the size of a basis for the backward reachable configurations [40] and reasons about the complementation. There is a matching lower bound for deterministic automata.

Outline. Section 2 recalls the basics on WSTS. The determinization results can be found in Section 3. They prepare the main result in Section 4. The state complexity of separators for Petri nets is in Section 5. Section 6 concludes the paper.

Related Work. Separability is a widely-studied problem in Theoretical Computer Science. A classical result says that every two co-recursively enumerable languages are recursively separable, i.e. separable by a recursive language [30]. In the area of formal languages, separability *of* regular languages by subclasses thereof was investigated most extensively as a decision problem: Given two regular languages, decide whether they are separable by a language from a fixed subclass. For the following subclasses, among others, the separability problem of regular languages is decidable: The piecewise-testable languages, shown independently in [19] and [51], the locally testable and locally threshold-testable languages [50], the languages definable in first-order logic [53], and the languages of certain higher levels of the first-order hierarchy [52].

Regular separability of classes larger than the regular languages attracted little attention until recently. As a remarkable example, already in the 70s, the undecidability of regular separability of context-free languages has been shown [56] (see also a later proof [34]); then the undecidability has been strengthened to visibly pushdown languages [38] and to languages of one-counter automata [17].

An intriguing problem, to the best of our knowledge still open, is the decidability of regular separability of Petri net languages, under the proviso that acceptance is by *reaching* a

distinguished final configuration. As for now, positive answers are known only for subclasses of VAS languages: PSPACE-completeness for one-counter nets (i.e. one-dimensional vector addition systems with states) [17], and elementary complexity for languages recognizable by Parikh automata (or, equivalently, by integer vector addition systems) [14]. Finally, regular separability of *commutative closures of* VAS languages has been shown to be decidable in [15]. As a consequence of this paper, regular separability of two VAS languages reduces to disjointness of the same two VAS languages (and is thus trivially decidable), given that acceptance is by *covering* a distinguished final configuration.

Languages of upward-compatible WSTS were investigated e.g. in [32], where interesting closure properties have been shown, including a natural pumping lemma. Various subclasses of languages of WSTS have been considered, e.g. in [20, 2, 45].

2 Well structured transition systems

Well Quasi Orders. A quasi order $(X, \preceq)$, i.e. a set X equipped with a reflexive and transitive binary relation $\preceq$, is called *well quasi order* (wqo) if for every infinite sequence $x_1, x_2, \ldots \in X$ there are indices $i < j$ such that $x_i \preceq x_j$. It is folklore that $(X, \preceq)$ is wqo iff it admits neither an infinite descending sequence (i.e. it is well-founded) nor an infinite antichain (i.e. it has the finite antichain property).

We will be working either with wqos, or with ω^2-wqos, a strengthening of wqos. We prefer not to provide the technical definition of ω^2-wqo (which can be found, e.g in [44]), as it would not serve our aims. Instead, we take the characterization provided by Lemma 2 below as a working definition. The class of ω^2-wqos provides a framework underlying the forward WSTS analysis developed in [26, 27, 31]. Both classes, namely wqos and ω^2-wqos, are stable under various operations like taking the Cartesian product, the lifting to finite multisets (multiset embedding), and the lifting to finite sequences (Higman ordering).

A subset $U \subseteq X$ is *upward closed* with respect to $\preceq$ if $u \in U$ and $u' \succeq u$ implies $u' \in U$. Similarly, one defines *downward closed* sets. Clearly, U is upward closed iff $X \setminus U$ is downward closed. The *upward* and *downward closure* of a set $U \subseteq X$ are defined as:

$$\uparrow U = \{x \in X \mid \exists u \in U,\, x \succeq u\} \quad \text{and} \quad \downarrow U = \{x \in X \mid \exists u \in U,\, x \preceq u\} \ .$$

The family of all upward-closed resp. downward-closed subsets of X we denote by $\mathcal{P}^{\uparrow}(X)$ resp. $\mathcal{P}^{\downarrow}(X)$. If $(X, \preceq)$ is a wqo then every upward closed set is the upward closure of a finite set, namely of the set of its minimal elements. This is not the case for downward closed set; we thus distinguish a subfamily $\mathcal{P}^{\downarrow}_{\text{fin}}(X) \subseteq \mathcal{P}^{\downarrow}(X)$ of *finitary* downward closed subsets of X, i.e. downward closures of finite sets. In general, these are not necessarily finite sets (e.g. consider the set $\mathbb{N} \cup \{\omega\}$ with ω bigger than all natural numbers, and the downward closure of $\{\omega\}$). The set $\mathcal{P}^{\downarrow}_{\text{fin}}(X)$, ordered by inclusion, is a wqo whenever $(X, \preceq)$ is:

▶ **Lemma 1.** $\left(\mathcal{P}^{\downarrow}_{\text{fin}}(X), \subseteq\right)$ *is a wqo iff* $(X, \preceq)$ *is a wqo.*

This property does not necessarily extend to the whole set $\mathcal{P}^{\downarrow}(X)$ of all downward closed subsets of X. As shown in [35]:

▶ **Lemma 2.** $\left(\mathcal{P}^{\downarrow}(X), \subseteq\right)$ *is a wqo iff* $(X, \preceq)$ *is an* ω^2*-wqo.*

As a matter of fact, [35] considers the reverse inclusion order on upward closed sets, which is clearly isomorphic to the inclusion order on downward closed sets.

Labeled Transition Systems. In the sequel we always fix a finite alphabet Σ. A labeled transition system (LTS) $\mathcal{W} = (S, T, I, F)$ over Σ consists of a set of *configurations* S, a set of *transitions* $T \subseteq S \times \Sigma \times S$, and subsets $I, F \subseteq S$ of *initial* and *final* configurations. We write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in T$. A path from configuration s to configuration s' over a word $w = a_0 \cdots a_{k-1}$ is a sequence of configurations $s = s_0, s_1, \ldots, s_{k-1}, s_k = s'$ such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \in \{0, \ldots, k-1\}$. We write $s \xrightarrow{w} s'$. For a subset $X \subseteq S$ of configurations and a word $w \in \Sigma^*$ we write

$$\textsc{Reach}_{\mathcal{W}}(X, w) = \{s \in S \mid \exists x \in X \colon x \xrightarrow{w} s\} \ ,$$
$$\textsc{Reach}^{-1}_{\mathcal{W}}(X, w) = \{s \in S \mid \exists x \in X \colon s \xrightarrow{w} x\}$$

for the set of all configurations reachable (resp. reversely reachable) from X along w. Note that we have $\textsc{Reach}_{\mathcal{W}}(X, \varepsilon) = X = \textsc{Reach}^{-1}_{\mathcal{W}}(X, \varepsilon)$. Important special cases will be the set of all a-successors (resp. a-predecessors) for $a \in \Sigma$, i.e. configurations reachable along a one-letter word a, and the configurations reachable from the initial configurations I (resp. reversely reachable from the final configurations F):

$$\textsc{Succ}_{\mathcal{W}}(X, a) = \textsc{Reach}_{\mathcal{W}}(X, a) \qquad \textsc{Reach}_{\mathcal{W}}(w) = \textsc{Reach}_{\mathcal{W}}(I, w)$$
$$\textsc{Pred}_{\mathcal{W}}(X, a) = \textsc{Reach}^{-1}_{\mathcal{W}}(X, a) \qquad \textsc{Reach}^{-1}_{\mathcal{W}}(w) = \textsc{Reach}^{-1}_{\mathcal{W}}(F, w)$$

satisfying the following equalities for all $w \in \Sigma^*$ and $a \in \Sigma$:

$$\textsc{Reach}_{\mathcal{W}}(w.a) = \textsc{Succ}_{\mathcal{W}}(\textsc{Reach}_{\mathcal{W}}(w), a) \tag{1}$$
$$\textsc{Reach}^{-1}_{\mathcal{W}}(a.w) = \textsc{Pred}_{\mathcal{W}}(\textsc{Reach}^{-1}_{\mathcal{W}}(w), a) \ . \tag{2}$$

We also establish the notation for the whole set of (reversely) reachable configurations:

$$\textsc{Reach}_{\mathcal{W}} = \bigcup_{w \in \Sigma^*} \textsc{Reach}_{\mathcal{W}}(w) \qquad \textsc{Reach}^{-1}_{\mathcal{W}} = \bigcup_{w \in \Sigma^*} \textsc{Reach}^{-1}_{\mathcal{W}}(w) \ .$$

An LTS $\mathcal{W} = (S, T, I, F)$ is *finitely branching* if I is finite and for every configuration $s \in S$ and each $a \in \Sigma$ there are only finitely many configurations $s' \in S$ such that $s \xrightarrow{a} s'$. Furthermore, $\mathcal{W}$ is *deterministic* if it has exactly one initial configuration and for every $s \in S$ and each $a \in \Sigma$ there is exactly one $s' \in S$ such that $s \xrightarrow{a} s'$. If $\mathcal{W}$ is deterministic, we write $s' = \textsc{Succ}_{\mathcal{W}}(s, a)$ (resp. $s' = \textsc{Reach}_{\mathcal{W}}(w)$) instead of $\{s'\} = \textsc{Succ}_{\mathcal{W}}(s, a)$ (resp. $\{s'\} = \textsc{Reach}_{\mathcal{W}}(w)$).

The language recognized by $\mathcal{W}$, denoted $\mathcal{L}(\mathcal{W})$, is the set of words which occur on some path starting in an initial configuration and ending in a final one, i.e.

$$\mathcal{L}(\mathcal{W}) = \{w \in \Sigma^* \mid \exists i \in I, f \in F \colon i \xrightarrow{w} f\} \ .$$

We call two LTS $\mathcal{W}, \mathcal{W}'$ *equivalent* if their languages are the same.

Note that we did not allow for ε-steps in transition systems. Even if ε-steps can be eliminated by pre-composing and post-composing every transition $s \xrightarrow{a} s'$ with the reflexive-transitive closure of $\xrightarrow{\varepsilon}$, this transformation does not necessarily preserve finite branching.

Synchronized Products. Consider LTS $\mathcal{W} = (S, T, I, F)$ and $\mathcal{W}' = (S', T', I', F')$. Their *synchronized product* is the LTS $\mathcal{W} \times \mathcal{W}' = (S_\times, T_\times, I_\times, F_\times)$ defined as follows: The configurations are tuples of configurations, $S_\times = S \times S'$, and the initial and final configurations are $I_\times = I \times I'$ and $F_\times = F \times F'$, respectively. The transition relation is defined by

$$(s, s') \xrightarrow{a} (r, r') \text{ in } \mathcal{W} \times \mathcal{W}' \quad \text{if} \quad s \xrightarrow{a} r \text{ in } \mathcal{W} \quad \text{and} \quad s' \xrightarrow{a} r' \text{ in } \mathcal{W}' \ .$$

It is immediate from the definition that the language of the product is the intersection of the languages, i.e. $\mathcal{L}(\mathcal{W} \times \mathcal{W}') = \mathcal{L}(\mathcal{W}) \cap \mathcal{L}(\mathcal{W}')$. If $\mathcal{W}$ and $\mathcal{W}'$ both are finitely branching, then so is their product.

Upward-Compatible Well-Structured Transition Systems. Now we define a labeled version of *well-structured transition systems* as described in [28], here called upward-compatible well-structured transition system (UWSTS). We start by defining the more general notions of quasi ordered LTS and ULTS.

By a *quasi-ordered LTS* $\mathcal{W} = (S, T, \preceq, I, F)$ we mean an LTS (S, T, I, F) extended with a quasi order $\preceq$ on configurations.

An upward-compatible LTS (ULTS) is a quasi-ordered LTS such that the set F of final configurations F is upward closed [4] with respect to $\preceq$, and the following upward compatibility [5] is satisfied: whenever $s \preceq s'$ and $s \xrightarrow{a} r$, then $s' \xrightarrow{a} r'$ for some $r' \in S$ such that $r \preceq r'$. In other words, $\preceq$ is a simulation relation. Upward compatibility extends to words:

▶ **Lemma 3.** *For $w \in \Sigma^*$, $s \preceq s'$ with $s \xrightarrow{w} r$, we have $s' \xrightarrow{w} r'$ for some $r' \in S$ with $r \preceq r'$.*

If the order $(S, \preceq)$ in a ULTS $\mathcal{W} = (S, T, \preceq, I, F)$ is a wqo, we call $\mathcal{W}$ a UWSTS.

As F is upward closed, $\mathcal{W}$ is equivalent to its downward closure $\downarrow\mathcal{W}$, obtained from $\mathcal{W}$ by replacing the set I by its (not necessarily finite) downward closure $\downarrow I$ with respect to $\preceq$, and by extending the transition relation as follows: $s \xrightarrow{a} r$ in $\downarrow\mathcal{W}$ if $s \xrightarrow{a} r'$ in $\mathcal{W}$ for some $r' \succeq r$. Note that with respect to the extended transition relation, $\textsc{Succ}_{\downarrow\mathcal{W}}(X, a)$ is downward closed for every $X \subseteq S$. One easily checks that $\downarrow\mathcal{W}$ still satisfies upward compatibility, and every word accepted by $\mathcal{W}$ is also accepted by $\downarrow\mathcal{W}$. The converse implication follows by the following simulation of $\downarrow\mathcal{W}$ by $\mathcal{W}$:

▶ **Lemma 4.** *Let $w \in \Sigma^*$. Whenever $s \preceq s'$ and $s \xrightarrow{w} r$ in $\downarrow\mathcal{W}$, then $s' \xrightarrow{w} r'$ in $\mathcal{W}$ for some $r' \in S$ such that $r \preceq r'$.*

The synchronized product of two ULTS $(S, T, \preceq, I, F)$ and $(S', T', \preceq', I', F')$ is still a ULTS with respect to the product order $\preceq_\times$ defined by $(x, x') \preceq_\times (y, y')$ iff $x \preceq y$ and $x' \preceq' y'$. Indeed, $F \times F'$ is upward closed wrt. $\preceq_\times$ and the transition relation satisfies upward compatibility. Since the product order of two wqos is again a wqo, the synchronized product of two UWSTS is a UWSTS.

When $\preceq$ is a ω^2-wqo, the UWSTS $\mathcal{W}$ is called *ω^2-UWSTS*. When the LTS (S, T, I, F) is finitely branching (resp. deterministic), the UWSTS $\mathcal{W}$ is called *finitely-branching UWSTS* (resp. *deterministic UWSTS*). In the sequel we speak shortly of UWSTS-languages (resp. ω^2-UWSTS-languages, finitely-branching UWSTS-languages, etc.).

Downward-Compatible Well-Structured Transition Systems. A downward-compatible well-structured transition system (DWSTS) is defined like its upward-compatible counterpart, with two modifications. First, we assume the set of final configurations F to be downward closed, instead of being upward closed. Second, instead of upward compatibility, we require its symmetric variant, namely *downward compatibility*: Whenever $s' \preceq s$ and $s \xrightarrow{a} r$, then $s' \xrightarrow{a} r'$ for some $r' \in S$ such that $r' \preceq r$. In other words, the inverse of $\preceq$ is a simulation relation. Downward compatibility extends to words, which can been shown similar to Lemma 3. Symmetrically to the downward closure of a UWSTS, we may define the upward closure $\uparrow\mathcal{W}$ of a DWSTS $\mathcal{W}$ that recognizes the same language.

[4] Languages defined by upward-closed sets of final configurations are usually called *coverability languages*.
[5] In the terminology of [28], this is strong compatibility.

As above, we also speak of finitely-branching DWSTS, or ω^2-DWSTS. We jointly call UWSTS and DWSTS just WSTS.

Examples of WSTS. Various well known and intensively investigated models of computation happen to be either an UWSTS or DWSTS. The list of natural classes of systems which are UWSTS contains, among the others: vector addition systems (VAS) resp. Petri nets and their extensions (e.g. with reset arcs or transfer arcs); lossy counter machines [10]; string rewriting systems based on context-free grammars; lossy communicating finite state machines (aka lossy channel systems, LCS) [12]; and many others. In the first two models listed above the configurations are ordered by the multiset embedding, while in the remaining two ones the configurations are ordered by Higman's subsequence ordering. The natural examples of UWSTS, including all models listed above, are ω^2-UWSTS and, when considered without ε-transitions, finitely-branching.

DWSTS are less common. A natural source of examples is *gainy* models, like gainy counter system machines or gainy communicating finite state machines. For an overview, see e.g. page 31 of [28].

3 Expressibility

Our proof of regular separability assumes one of the WSTS to be deterministic. In this section, we show that this is no strong restriction. We compare the languages recognized by different classes of WSTS, in particular deterministic ones. The findings are summarized in Theorem 5, where we use $\subseteq$ to say that every language of a WSTS from one class is also the language of a WSTS from another class; and we use $\subseteq_{\text{rev}}$ to say that every language of a WSTS from one class is the reverse of the language of a WSTS from another class.

▶ **Theorem 5.** *The following relations hold between the WSTS language classes:*

$$\omega^2\text{-}UWSTS \subseteq \textit{deterministic } UWSTS = \textit{finitely-branching } UWSTS \subseteq \textit{all } UWSTS\,,$$
$$\omega^2\text{-}DWSTS \subseteq \textit{deterministic } DWSTS \subseteq \textit{finitely-branching } DWSTS = \textit{all } DWSTS\,,$$
$$\omega^2\text{-}UWSTS \subseteq_{rev} \textit{deterministic } DWSTS\,,$$
$$\omega^2\text{-}DWSTS \subseteq_{rev} \textit{deterministic } UWSTS\,.$$

In short, ω^2-UWSTS and ω^2-DWSTS determinize and reverse-determinize; finitely-branching UWSTS determinize too; and (unrestricted) DWSTS are equivalent to finitely-branching DWSTS.

4 Regular Separability

We now show our first main results: Under mild assumptions, disjoint DWSTS resp. disjoint UWSTS are regular separable. Both theorems follow from a technical result that establishes a surprising link between verification and formal language theory: Every inductive invariant (of a suitable product WSTS) that has a finite representation can be turned into a regular separator. With this, the proofs of regular separability are invariant constructions.

Main Results. We say that two languages $\mathcal{L}$ and $\mathcal{K}$ over the same alphabet are *regular separable* if there is a regular language $\mathcal{R}$ that satisfies $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{R} \cap \mathcal{K} = \emptyset$. For two WSTS $\mathcal{W}$ and $\mathcal{W}'$, we say that they are regular separable if so are their languages. Disjointness is clearly necessary for regular separability. Our first main results show that for most WSTS disjointness is also sufficient:

▶ **Theorem 6.** *Every two disjoint DWSTS, one deterministic, are regular separable.*

▶ **Theorem 7.** *Every two disjoint UWSTS, one finitely branching, are regular separable.*

The results imply that the complement of a non-regular WSTS language cannot be a WSTS language. They also show that there is no subclass of WSTS languages beyond the regular languages that is closed under complement. More formally, for a class of languages $\mathcal{C}$, we call a language *doubly* $\mathcal{C}$, if the language as well as its complement are in $\mathcal{C}$. We obtain the following corollary, generalizing earlier results for Petri net coverability languages [49, 48].

▶ **Corollary 8.** *(1) Every doubly deterministic DWSTS language resp. every doubly finitely-branching UWSTS language is regular. (2) No subclass of finitely-branching UWSTS languages resp. deterministic DWSTS languages beyond REG is closed under complement.*

The rest of the section is devoted to the proofs. We will use that the product of two disjoint WSTS is again a WSTS with the empty language. Whenever the language of a WSTS is empty, we can find an inductive invariant, a downward-closed set of configurations separating the reachability set from the final configurations. Given a finite representation for such an invariant, we show how to turn it into a regular separator, provided one of the WSTS is deterministic. This is our key technical insight, formulated as Theorem 11 below.

The proof of Theorem 6 follows directly from this result. For Theorem 7, we consider the ideal completion of an UWSTS, an extended system in which every downward-closed set has a finite representation. This in particular applies to inductive invariants, as we show in the form of Proposition 21: Any inductive invariant in the original UWSTS induces an inductive invariant in the ideal completion that has a finite representation. Combining this result with Theorem 11 yields the desired proof.

Turning Inductive Invariants into Regular Separators. Inductive invariants are a standard tool in the safety verification of programs [43]. Technically, an inductive invariant (of a program for a safety property) is a set of program configurations that includes the initial ones, is closed under the transition relation, and is disjoint from the set of undesirable states. The following definition lifts the notion to WSTS (actually to the more general ULTS), where it is natural to require inductive invariants to be downward-closed.

▶ **Definition 9.** An *inductive invariant* for a ULTS $\mathcal{W}$ with configurations S is a downward-closed set $X \subseteq S$ with the following three properties:

$$I \subseteq X \ , \tag{3}$$

$$F \cap X = \emptyset \ , \tag{4}$$

$$\textsc{Succ}_{\mathcal{W}}(X, a) \subseteq X \text{ for all } a \in \Sigma \ . \tag{5}$$

An inductive invariant X is *finitely-represented* if $X = {\downarrow}Q$ for a finite set $Q \subseteq S$.

By (3) and (5), the invariant has to contain the whole reachability set. By (4) and (5), it has to be disjoint from the predecessors of the final configurations:

$$\textsc{Reach}_{\mathcal{W}} \subseteq X \ , \qquad\qquad \textsc{Reach}^{-1}_{\mathcal{W}} \cap X = \emptyset \ .$$

This means every inductive invariant shows language emptiness. Even more, inductive invariants are complete for proving emptiness, like inductive invariants for programs are (relatively) complete for proving safety [16].

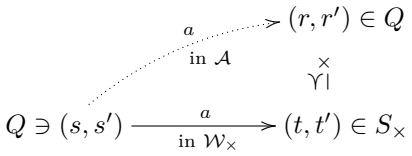


Figure 1 The transition relation of $\mathcal{A}$.

▶ **Lemma 10.** *Consider ULTS $\mathcal{W}$. Then $\mathcal{L}(\mathcal{W}) = \emptyset$ iff there is an inductive invariant for $\mathcal{W}$.*

For completeness, observe that $X = {\downarrow}\textsc{Reach}_{\mathcal{W}}$ is an inductive invariant. It is the least one wrt. inclusion. There is also a greatest inductive invariant, namely the complement of $\textsc{Reach}^{-1}_{\mathcal{W}}$. Note that, due to upward compatibility, $\textsc{Reach}^{-1}_{\mathcal{W}}$ is always upward-closed.

Other invariants may have the advantage of being easier to represent. We will be particularly interested in invariants that are finitely-represented in the sense that they form the downward closure of a finite set.

Here is the core result. Consider two disjoint ULTS. Any finitely-represented inductive invariant for the product can be turned into a regular separator. We will comment on the assumed determinism in a moment.

▶ **Theorem 11.** *Let $\mathcal{W}$ and $\mathcal{W}'$ be disjoint ULTS, one of them deterministic, such that $\mathcal{W} \times \mathcal{W}'$ admits a finitely-represented inductive invariant ${\downarrow}Q$. Then $\mathcal{W}$ and $\mathcal{W}'$ are regular separable by the language of a finite automaton with states Q.*

For the definition of the separator, let $\mathcal{W} = (S, T, \preceq, I, F)$ be an arbitrary ULTS and let $\mathcal{W}' = (S', T', \preceq', I', F')$ be a deterministic one such that their languages are disjoint. Let

$$\mathcal{W}_\times = \mathcal{W} \times \mathcal{W}' = (S_\times, T_\times, \preceq_\times, I_\times, F_\times)$$

be their synchronized product. By the disjointness of $\mathcal{W}$ and $\mathcal{W}'$ we know that $\mathcal{L}(\mathcal{W}_\times) = \emptyset$. Let $Q \subseteq S_\times$ be a finite set such that ${\downarrow}Q$ is an inductive invariant.

We define a finite automaton $\mathcal{A}$ with states Q whose language will contain $L(\mathcal{W})$ while being disjoint from $L(\mathcal{W}')$. The idea is to over-approximate the configurations of $\mathcal{W}_\times$ by the elements available in Q. The fact that $\textsc{Reach}_{\mathcal{W}_\times} \subseteq {\downarrow}Q$ guarantees that every configuration $(s, s') \in S_\times$ has such a representation. Since we seek to approximate the language of $\mathcal{W}$, the final states only refer to the $\mathcal{W}$-component. Transitions are approximated existentially.

▶ **Definition 12.** We define the *separating automaton induced by* Q to be $\mathcal{A} = (Q, \to, Q_I, Q_F)$. A state is initial if it dominates some initial configuration of $\mathcal{W}_\times$, $Q_I = \{(s, s') \in Q \mid (i, i') \preceq_\times (s, s') \text{ for some } (i, i') \in I_\times\}$. As final states we take pairs whose $\mathcal{W}$-component is final, $Q_F = \{(s, s') \in Q \mid s \in F\}$. Finally, the transition relation in $\mathcal{A}$ is an over-approximation of the transition relation in $\mathcal{W}_\times$:

$$(s, s') \xrightarrow{a} (r, r') \text{ in } \mathcal{A} \quad \text{if } (s, s') \xrightarrow{a} (t, t') \text{ in } \mathcal{W}_\times \text{ for some } (t, t') \preceq_\times (r, r') \ .$$

Figure 1 illustrates the construction.

To show separation, we need to prove $\mathcal{L}(\mathcal{W}) \subseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{W}') = \emptyset$. We begin with the former. As $\mathcal{W}'$ is deterministic, $\mathcal{W}_\times$ contains all computations of $\mathcal{W}$. Due to upward compatibility, $\mathcal{A}$ over-approximates the computations in $\mathcal{W}_\times$. Combining these two insights, which are summarized in the next lemma, yields the result.

▶ **Lemma 13.** *(1) For every* $s \in \text{REACH}_{\mathcal{W}}(w)$ *there is some* $(s, s') \in \text{REACH}_{\mathcal{W}_\times}(w)$*. (2) For every* $(s, s') \in \text{REACH}_{\mathcal{W}_\times}(w)$ *there is some* $(r, r') \in \text{REACH}_{\mathcal{A}}(w)$ *with* $(s, s') \preceq_\times (r, r')$*.*

▶ **Proposition 14.** $\mathcal{L}(\mathcal{W}) \subseteq \mathcal{L}(\mathcal{A})$.

It remains to prove disjointness of $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{W}')$. The key observation is that, due to determinism, $\mathcal{W}'$ simulates the computations of $\mathcal{A}$ – in the following sense: If upon reading a word $\mathcal{A}$ reaches a state (s, s'), then the unique computation of $\mathcal{W}'$ will reach a configuration dominated by s'.

▶ **Lemma 15.** *For every* $w \in \Sigma^*$ *and every* $(s, s') \in \text{REACH}_{\mathcal{A}}(w)$ *we have* $\text{REACH}_{\mathcal{W}'}(w) \preceq' s'$*.*

With this lemma we can show disjointness. Towards a contradiction, suppose some word w satisfies $w \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{W}')$. As $w \in \mathcal{L}(\mathcal{A})$, there is a configuration $(s, s') \in \text{REACH}_{\mathcal{A}}(w)$ with $s \in F$. As $w \in \mathcal{L}(\mathcal{W}')$, the unique configuration $\text{REACH}_{\mathcal{W}'}(w)$ belongs to F'. With the previous lemma and the fact that F' is upward-closed, we conclude $s' \in F'$. Together, $(s, s') \in F_\times$, which contradicts the fact that $\downarrow Q$ is an inductive invariant, Property (4).

▶ **Proposition 16.** $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{W}') = \emptyset$.

Together, Proposition 14 and 16 show Theorem 11. With Theorem 11 at hand, the proof of regular separability for DWSTS follows easily.

Proof of Theorem 6. Consider an arbitrary DWSTS $\mathcal{W} = (S, T, \preceq, I, F)$ and a deterministic one $\mathcal{W}' = (S', T', \preceq', I', F')$. We start with the observation that the inversed versions of $\mathcal{W}$ and $\mathcal{W}'$, namely with the orders $\preceq^{-1}$ and $(\preceq')^{-1}$ and denoted by $\mathcal{W}^{-1}$ and $(\mathcal{W}')^{-1}$, are ULTS. We claim that these ULTS satisfy the assumptions of Theorem 11. The language of $\mathcal{W}_\times^{-1} = \mathcal{W}^{-1} \times (\mathcal{W}')^{-1}$ is empty since the language of $\mathcal{W}_\times = \mathcal{W} \times \mathcal{W}'$ is empty and inversion does not change the language, $\mathcal{L}(\mathcal{W}) = \mathcal{L}(\mathcal{W}^{-1})$ and similar for $\mathcal{W}'$. Inversion also does not influence determinism.

It remains to find an inductive invariant of $\mathcal{W}_\times^{-1}$ that is finitely represented. We claim that $X = \downarrow_{-1}\text{REACH}_{\mathcal{W}_\times^{-1}}$ is a suitable choice. The subscript indicates that the downward closure is computed relative to the quasi order of $\mathcal{W}_\times^{-1}$. As the language of $\mathcal{W}_\times^{-1}$ is empty, X is an inductive invariant by Lemma 10. For the finite representation, note that inversion does not change the transition relation. Hence, $\mathcal{W}_\times$ and $\mathcal{W}_\times^{-1}$ reach the same configurations, $\text{REACH}_{\mathcal{W}_\times^{-1}} = \text{REACH}_{\mathcal{W}_\times} = Z$. With the definition of inversion, $X = \downarrow_{-1} Z = \uparrow Z$ holds. Moreover, $\uparrow Z = \uparrow \min(Z)$, with minimum and upward closure computed relative to $\mathcal{W}_\times$. Since the configurations of $\mathcal{W}_\times$ are well quasi ordered, $\min(Z)$ is finite. Another application of inversion yields $X = \uparrow \min(Z) = \downarrow_{-1} \min(Z)$. Hence, X is a finitely-represented downward-closed subset of $\mathcal{W}_\times^{-1}$.

By Theorem 11, the languages of $\mathcal{W}^{-1}$ and $(\mathcal{W}')^{-1}$ are regular separable and so are the languages of $\mathcal{W}$ and $\mathcal{W}'$. ◀

Ideal Completions of UWSTS. The proof of regular separability for UWSTS is more involved. Here, we need the notion of ideal completions [9, 27]. We show that any invariant for a WSTS yields a finitely-represented invariant for the corresponding ideal completion. Theorem 7 follows from this.

An *ideal* in a wqo $(X, \preceq)$ is a non-empty downward-closed subset $Z \subseteq X$ which is directed: For every $z, z' \in Z$ there is a $z'' \in Z$ with $z \preceq z''$ and $z' \preceq z''$. Every downward-closed set decomposes into finitely many ideals. In fact, the finite antichain property is sufficient and necessary for this.

▶ **Lemma 17** ([37, 27, 41]). *In a wqo, every downward-closed set is a finite union of ideals.*

We use $\textsc{Id-dec}_X(Z)$ to denote the set of inclusion-maximal ideals in Z. By the above lemma, $\textsc{Id-dec}_X(Z)$ is always finite and

$$Z = \bigcup \textsc{Id-dec}_X(Z) \ . \tag{6}$$

We will also make use of the fact that ideals are irreducible in the following sense.

▶ **Lemma 18** ([37, 27, 41]). *Let $(X, \preceq)$ be a wqo. If $Z \subseteq X$ is downward-closed and $I \subseteq Z$ is an ideal, then $I \subseteq J$ for some $J \in \textsc{Id-dec}_X(Z)$.*

The *ideal completion* $(\overline{X}, \subseteq)$ of $(X, \preceq)$ has as elements all ideals in X. The order is inclusion. The ideal completion $\overline{X}$ can be seen as extension of X; indeed, every element $x \in X$ is represented by $\downarrow\{x\} \in \overline{X}$, and inclusion among such representations coincides with the original quasi order $\preceq$. Later, we will also need general ideals that may not be the downward closure of a single element.

In [27, 9], the notion has been lifted to WSTS $\mathcal{W} = (S, T, \preceq, I, F)$. The ideal completion of $\mathcal{W}$ is the ULTS $\overline{\mathcal{W}}$, where the given wqo is replaced by its ideal completion. The initial configurations are the ideals in the decomposition of $\downarrow I$. The transition relation is defined similarly, by decomposing $\downarrow\textsc{Succ}_{\mathcal{W}}(X, a)$, with X an ideal. The final configurations are the ideals that intersect F.

▶ **Definition 19** ([27, 9]). For an UWSTS $\mathcal{W} = (S, T, \preceq, I, F)$, we define its *ideal completion* $\overline{\mathcal{W}} = (\overline{S}, \overline{T}, \subseteq, \overline{I}, \overline{F})$, where $(\overline{S}, \subseteq)$ is the ideal completion of $(S, \preceq)$, the transition relation is defined by $\textsc{Succ}_{\overline{\mathcal{W}}}(X, a) = \textsc{Id-dec}_S(\downarrow\textsc{Succ}_{\mathcal{W}}(X, a))$, $\overline{I} = \textsc{Id-dec}_S(\downarrow I)$, and $\overline{F} = \{X \in \overline{S} \mid X \cap F \neq \emptyset\}$.

Using upward compatibility in $\mathcal{W}$, language equivalence holds and determinism is preserved.

▶ **Lemma 20.** *The ideal completion $\overline{\mathcal{W}}$ of an UWSTS $\mathcal{W}$ is a ULTS. We have $\mathcal{L}(\overline{\mathcal{W}}) = \mathcal{L}(\mathcal{W})$. If $\mathcal{W}$ is deterministic, then so is $\overline{\mathcal{W}}$.*

As a matter of fact, $\overline{\mathcal{W}}$ is even finitely branching, but we do not need this property.

The purpose of using ideal completions is to make it easier to find inductive invariants that are finitely represented. Assume the given UWSTS $\mathcal{W}$ has an inductive invariant X, not necessarily finitely represented. By definition, X is downward-closed. Thus, by Lemma 17, X is a *finite* union of ideals. These ideals are configurations of the ideal completion $\overline{\mathcal{W}}$. To turn $\textsc{Id-dec}_S(X)$ into an inductive invariant of $\overline{\mathcal{W}}$, it remains to take the downward closure of the set. As the order among ideals is inclusion, this does not add configurations. In short, an inductive invariant for $\mathcal{W}$ induces a finitely-represented inductive invariant for $\overline{\mathcal{W}}$.

▶ **Lemma 21.** *If $X \subseteq S$ is an inductive invariant of $\mathcal{W}$, $\downarrow\textsc{Id-dec}_S(X)$ is a finitely-represented inductive invariant of $\overline{\mathcal{W}}$.*

Proof. Define $Q = \textsc{Id-dec}_S(X)$. Since Q contains all ideals $Y \subseteq X$ that are maximal wrt. inclusion, $\downarrow Q$ contains all ideals $Y \subseteq X$. We observe that $X \overset{(6)}{=} \bigcup Q = \bigcup \downarrow Q$. By Lemma 17, Q is finite and thus $\downarrow Q$ is finitely-represented. It remains to check that $\downarrow Q$ satisfies the Properties (3), (4), and (5).

We have $I \subseteq X$ by Property (3), and since X is downward-closed, we obtain $\downarrow I \subseteq X$. Consequently, any ideal that is a subset of $\downarrow I$ is also a subset of X, and $\downarrow Q$ contains all such ideals. For Property (4), assume towards a contradiction that $\downarrow Q$ contains an ideal Y that is final in $\overline{\mathcal{W}}$. This means Y contains a final configuration. Since $Y \subseteq X$, we obtain a

contradiction to $X \cap F = \emptyset$, Property (4). To check the inclusion $\textsc{Succ}_{\overline{\mathcal{W}}}(\downarrow Q, a) \subseteq \downarrow Q$, we pick an ideal $Y \in \downarrow Q$ and show $\textsc{Succ}_{\overline{\mathcal{W}}}(Y, a) \subseteq \downarrow Q$. Recall the definition $\textsc{Succ}_{\overline{\mathcal{W}}}(Y, a) = \textsc{Id-dec}_S(\downarrow \textsc{Succ}_{\mathcal{W}}(Y, a))$. Thus, any element of $\textsc{Succ}_{\overline{\mathcal{W}}}(Y, a)$ is an ideal that is a subset of $\downarrow \textsc{Succ}_{\mathcal{W}}(Y, a)$. We have $\textsc{Succ}_{\mathcal{W}}(X, a) \subseteq X$ by Property (5). This implies $\textsc{Succ}_{\mathcal{W}}(Y, a) \subseteq X$ as $Y \subseteq X$, and even $\downarrow \textsc{Succ}_{\mathcal{W}}(Y, a) \subseteq X$ as X is downward-closed. Hence, any ideal that is a subset of $\downarrow \textsc{Succ}_{\mathcal{W}}(Y, a)$ is also subset of X, and thus an element of $\downarrow Q$. ◀

Theorem 11 expects invariants for UWSTS of a particular shape, namely products $\mathcal{W} \times \mathcal{W}'$. We now show that the operation of ideal completion commutes with taking products of UWSTS, a fact that will be key to the proof of Theorem 7. We start by recalling that the ideals in a product wqo $X \times Y$ are precisely the products of the ideals in X and in Y.

▶ **Lemma 22** ([37, 27, 41]). *A set $Z \subseteq X \times Y$ is an ideal iff $Z = I \times J$, where $I \subseteq X$ and $J \subseteq Y$ are ideals.*

Lemma 22 yields the mentioned commutativity.

▶ **Lemma 23.** *For two UWSTSes $\mathcal{W}$ and $\mathcal{W}'$, $\overline{\mathcal{W}} \times \overline{\mathcal{W}'}$ and $\overline{\mathcal{W} \times \mathcal{W}'}$ are isomorphic.*

We are now prepared to apply Theorem 11 once more to establish our second main result.

Proof of Theorem 7. Let $\mathcal{W} = (S, T, \preceq, I, F)$ and $\mathcal{W}' = (S', T', \preceq', I', F')$ be disjoint UWSTS and $\mathcal{W}'$ finitely branching. By Theorem 5 we can assume $\mathcal{W}'$ is deterministic.

We would like to construct a finitely-represented inductive invariant in the synchronized product of the ideal completions $\overline{\mathcal{W}} \times \overline{\mathcal{W}'}$ and then apply Theorem 11. Indeed, by Lemma 20 we know that the ideal completions are disjoint ULTS, and that the latter one is still deterministic, so they satisfy the assumptions.

Relying on Lemma 23 we prefer to show the existence of a finitely-represented inductive invariant in $\overline{\mathcal{W} \times \mathcal{W}'}$. Using Proposition 21, it is sufficient to find any inductive invariant in $\mathcal{W} \times \mathcal{W}'$, it does not have to be finitely-represented. We know that such an inductive invariant exists by Lemma 10, since we assume $\mathcal{L}(\mathcal{W} \times \mathcal{W}') = \mathcal{L}(\mathcal{W}) \cap \mathcal{L}(\mathcal{W}') = \emptyset$. ◀

Effective Representation. The states of the separating automaton in the proof of Theorem 7 are ideals in the product systems. With Lemma 22, these are tuples of ideals in the original systems. For most types of UWSTS, it is known how ideals can be effectively represented, i.e. how to obtain finite representations on which the successors can be computed. We briefly mention such a construction for Petri nets in Lemma 28, see e.g. [9] for more examples. In general, one may exploit the fact that ideals are downward-closed sets, which in turn are complements of upward-closed sets that can be represented by finitely many minimal elements – an idea first proposed in [29]. Note that in the proof of Theorem 7, we invoke Theorem 5 to determinize the given finitely-branching UWSTS. The states of the resulting UWSTS are finitary downward-closed sets of states of the original one. For most types of UWSTS, this construction can be avoided. We demonstrate this for the case of Petri nets in the proof of Proposition 30.

5 Separator Size: The Case of Petri Nets

The UWSTS associated to Petri nets are finitely branching. Hence, Theorem 7 applies: Whenever the coverability languages of two Petri nets are disjoint, they are regular separable. We now show how to construct a triply-exponential non-deterministic finite automaton (NFA) separating two such languages, provided they are disjoint. Moreover, for deterministic finite automata (DFA), we show that this size cannot be avoided.

▶ **Theorem 24.** *Let $\mathcal{L}(N_1)$, $\mathcal{L}(N_2)$ be disjoint Petri net coverability languages. There is an NFA $\mathcal{A}$ of size triply exponential in $|N_1| + |N_2|$ such that $\mathcal{L}(A)$ separates $\mathcal{L}(N_1)$ and $\mathcal{L}(N_2)$.*

▶ **Theorem 25.** *In general, Petri net coverability languages cannot be separated by DFA of less than triply-exponential size.*

Instead of invoking Theorem 7, which uses Theorem 5 to determinize, we directly show how to construct an equivalent instance of the separability problem in which one of the nets is deterministic. In this setting, we prove an upper bound that combines Theorem 11 with a size estimation for an ideal decomposition. We then show how to handle non-determinism. The lower bound combines a classical result from automata theory, showing that minimal DFA may have exponentially many states [39], with a Petri net construction due to Lipton [42].

Petri Nets. A *Petri net* over the alphabet Σ is a tuple $N = (P, T, F, \lambda, M_0, M_f)$ where P is a finite set of places, T is a finite set of transitions with $P \cap T = \emptyset$, $F \colon (P \cup T) \times (P \cup T) \to \mathbb{N}$ is a flow function, and $\lambda \colon T \to \Sigma$ is a labeling of the transitions. The runtime behavior of Petri nets is defined in terms of so-called *markings* from $M \in \mathbb{N}^d$ with $d = |P|$. If $M(p) = k > 0$, we say place p carries k tokens. We assume to be given an initial and a final marking, $M_0, M_f \in \mathbb{N}^d$. Markings are changed by firing transitions: A transition $t \in T$ is *enabled* in marking $M \in \mathbb{N}^d$, if $M(p) \geq F(p,t)$ for all places p. An enabled transition can be *fired* leading to the marking M' with $M'(p) = M(p) - F(p,t) + F(t,p)$, denoted $M[t\rangle M'$. Note that enabledness and firing are upward compatible with the componentwise ordering $\leq$ on markings, in the following sense. If $M_1 \leq M_2$ and $M_1[t\rangle M_1'$, then $M_2[t\rangle M_2'$ with $M_1' \leq M_2'$.

Relying on this compatibility, we can define the *UWSTS induced by* N to be $\mathcal{W}_N = (\mathbb{N}^P, T', \leq, \{M_0\}, \uparrow M_f)$. The transition relation is defined by $(M, a, M') \in T'$ if there is a transition $t \in T$ such that $M[t\rangle M'$ and $\lambda(t) = a$. The language of $\mathcal{W}_N$ is also called the *(coverability) language of* N[6], and denoted by $\mathcal{L}(N)$. We call N *deterministic* if $\mathcal{W}_N$ is.

We use a *product operation* on Petri nets $N_i = (P_i, T_i, F_i, \lambda_i, M_{0,i}, M_{f,i})$, $i = 1, 2$. The product Petri net is obtained by putting the places of N_1 and N_2 side by side and creating a new transition for all pairs of transitions in $T_1 \times T_2$ that carry the same label. Formally, $N_1 \times N_2 = (P, T, F, \lambda, M_0, M_f)$ with $P = P_1 \uplus P_2$, $T = \{(t_1, t_2) \in T_1 \times T_2 \mid \lambda(t_1) = \lambda(t_2)\}$. We have $\lambda(t_1, t_2) = \lambda(t_1) = \lambda(t_2)$. The flow function is defined by the flow functions of the component Petri nets, $F(p, (t_1, t_2)) = F_x(p, t_x)$ and $F((t_1, t_2), p) = F_x(t_x, p)$, where $x = i$ if $p \in P_i$. We have $M_0(p) = M_{0,i}(p)$ for $p \in P_i$, and similar for M_f. The product operation on Petri nets coincides with the product on UWSTS.

▶ **Lemma 26.** *$\mathcal{W}_{N_1 \times N_2}$ is isomorphic to $\mathcal{W}_{N_1} \times \mathcal{W}_{N_2}$.*

We will need the size of a Petri net. It is defined using a binary encoding of the values in the range of the flow function and in the markings. Define the *infinity norm* of a vector $M \in \mathbb{N}^d$ to be $\|M\|_\infty = \max_{p \in P} M(p)$. We extend this notion to matrices, sets of vectors, and functions by taking the maximum over all entries, elements, and elements in the range, respectively. The *size* of the Petri net N is now $|N| = |P| \, |T| \, (1 + \lceil \log_2(1 + \|F\|_\infty) \rceil) + |M_0| + |M_f|$. The size of a marking M is $|M| = |P|(1 + \lceil \log_2(1 + \|M\|_\infty) \rceil)$.

An Upper Bound Assuming Determinism. Theorem 11 assumes that one of the UWSTS is deterministic. We now show that for Petri nets, in this case, the regular separator is (an NFA of size) at most doubly exponential in the size of the input Petri nets.

[6] We consider covering the final marking as acceptance condition, i.e. a sequence of transitions is accepting if it reaches some marking M' with $M'(p) \geq M_f(p)$ for all $p \in P$.

To prove the result, we show how a size estimation for the basis of $\textsc{Reach}^{-1}_{\mathcal{W}}$ with $\mathcal{W} = \mathcal{W}_{N_1 \times N_2}$ can be turned into a size estimation for the ideal decomposition of the complement. The size estimation of the basis is the following result. It is obtained by inspecting Abdulla's backward search [1].

▶ **Theorem 27** (Bozzelli & Ganty [11]). *Consider a Petri net N with final marking M_f. Then $\textsc{Reach}^{-1}_{\mathcal{W}_N} = \uparrow\{v_1, \ldots, v_k\}$, where k as well as $\|\{v_1, \ldots, v_k\}\|_\infty$ are bounded from above by*

$$g = (|T| \cdot (\|F\|_\infty + \|M_0\|_\infty + \|M_f\|_\infty + 2))^{2^{\mathcal{O}(|P| \cdot \log |P|)}} .$$

By Lemma 10, $\mathbb{N}^d \setminus \textsc{Reach}^{-1}_{\mathcal{W}}$ is an inductive invariant of $\mathcal{W}$ (provided the language is empty). We can now apply Lemma 17 to finitely represent this set by its ideal decomposition. To represent this ideal decomposition in turn, we have to explicitly represent ideals in $\mathbb{N}^d$. The following lemma gives such a representation.

Let $\mathbb{N}_\omega$ denote $\mathbb{N}$ extended by a new top element ω. Every ideal in $\mathbb{N}^d$ is the downward closure $\downarrow u$ of a single vector $u \in \mathbb{N}^d_\omega$. The lemma moreover shows how to compute the intersection of two ideals and how to obtain the ideal decomposition of the complement $\mathbb{N}^d \setminus \uparrow v$ of the upward closure of a vector $v \in \mathbb{N}^d$.

▶ **Lemma 28** (see e.g. [40]). *(1) The ideals in $\mathbb{N}^d$ have the shape $\downarrow u$ for $u \in \mathbb{N}^d_\omega$. (2) For two ideals $\downarrow u_1, \downarrow u_2$ of $\mathbb{N}^d$, the intersection is $\downarrow u_1 \cap \downarrow u_2 = \downarrow u$ with $u(i) = \min\{u_1(i), u_2(i)\}$. (3) For $v \in \mathbb{N}^d$, we have $\textsc{Id-dec}(\mathbb{N}^d \setminus \uparrow v) = \{\downarrow u_{<v(j)} \mid j \in [1..d]\}$, where $u_{<v(j)}(j) = v(j) - 1$ and $u_{<v(j)}(i) = \omega$ for $i \neq j$.*

We can now combine Theorem 27 and Lemma 28 to obtained our upper bound.

▶ **Proposition 29.** *Let N_1 be an arbitrary Petri net and let N_2 be deterministic. If N_1 and N_2 are disjoint, they can be separated by an NFA of size doubly exponential in $|N_1| + |N_2|$.*

A General Upper Bound. The previous result yields a doubly-exponential separator in the case where N_2 is deterministic. We now show how to get rid of this assumption and construct a separator in the general case.

▶ **Proposition 30.** *Let N_1 and N_2 be disjoint Petri nets. Then they are separable by an NFA of size triply exponential in $|N_1| + |N_2|$.*

The proof transforms N_1 and N_2 into $N_{-\lambda}$ and N_{det} so that N_{det} is deterministic, invokes Proposition 29, and then turns the resulting separator for $N_{-\lambda}$ and N_{det} into a separator for N_1 and N_2. The approach is inspired by [14].

Let N_2 be non-deterministic with labeling function $\lambda\colon T_2 \to \Sigma$. We define N_{det} to be a variant of N_2 that is labeled by the identity function, i.e. N_{det} is a Petri net over the alphabet T_2. We have $\mathcal{L}(N_2) = \lambda(\mathcal{L}(N_{det}))$, where we see λ as a homomorphism on words. We furthermore define $N_{-\lambda}$ to be the T_2-labeled Petri net obtained from N_1 as follows. For each a-labeled transition t_1 of N_1 and each a-labeled transition t of N_2, $N_{-\lambda}$ contains a t-labeled copy t_1^t of t_1 with the same input-output behavior. Transition t_1 itself is removed.

▶ **Lemma 31.** $\mathcal{L}(N_1 \times N_2) = \lambda(\mathcal{L}(N_{-\lambda} \times N_{det}))$.

With this lemma, and since N_1 and N_2 are disjoint, $N_{-\lambda}$ and N_{det} have to be disjoint. As N_{det} is deterministic, we can apply Proposition 29 and obtain a separator for $N_{-\lambda}$ and N_{det}. Let $\mathcal{A}$ be the doubly-exponential NFA over the alphabet T_2 with $\mathcal{L}(N_{-\lambda}) \subseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{L}(N_{det}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$. We show how to turn $\mathcal{A}$ into a separator for N_1 and N_2. The first

step is to determine the complement automaton $\mathcal{A}^{\mathcal{C}}$, which satisfies $\mathcal{L}(N_{det}) \subseteq \mathcal{L}(\mathcal{A}^{\mathcal{C}})$ and $\mathcal{L}(N_{-\lambda}) \cap \mathcal{L}(\mathcal{A}^{\mathcal{C}}) = \emptyset$. The second step is to apply λ to $\mathcal{A}^{\mathcal{C}}$. Let $\mathcal{B} = \lambda(\mathcal{A}^{\mathcal{C}})$ be the automaton obtained from $\mathcal{A}^{\mathcal{C}}$ by relabeling each t-labeled transition to $\lambda(t)$. The following lemma shows that $\mathcal{B}$ is a separator for the original nets. The observation that the size of $\mathcal{A}^{\mathcal{C}}$ and hence the size of $\mathcal{B}$ is at most exponential in the size of $\mathcal{A}$ concludes the proof of Proposition 30

▶ **Lemma 32.** *$\mathcal{L}(N_2) \subseteq \mathcal{L}(\mathcal{B})$ and $\mathcal{L}(N_1) \cap \mathcal{L}(\mathcal{B}) = \emptyset$.*

Note that $\lambda(\mathcal{A})$ is not necessarily a separator: There might be $u \in \mathcal{L}(A)$, $u \notin \mathcal{L}(N_{det})$ such that there is $u' \in \mathcal{L}(N_{det})$ with $\lambda(u) = \lambda(u')$. Thus, $\lambda(u) \in \lambda(\mathcal{L}(\mathcal{A})) \cap \mathcal{L}(N_2)$.

A Lower Bound. We now consider separation by *deterministic* finite automata (DFA). In this case, we can show a triply-exponential lower bound on the size of the separator.

▶ **Proposition 33.** *For all $n \in \mathbb{N}$, there are disjoint Petri nets $N_0(n)$ and $N_1(n)$ of size polynomial in n such that any separating DFA has size at least triply exponential in n.*

Our proof relies on the classical result that for each $x \in \{0,1\}$ and each $k \in \mathbb{N}$, the minimal DFA for the language $\mathcal{L}_{x@k} = \{w \in \{0,1\}^{\geq k} \mid \text{ the } k\text{-last letter in } w \text{ is } x\}$ needs at least 2^k states [39]. To obtain the desired lower bound, we will show how to generate $\mathcal{L}_{x@k}$ for a doubly-exponential number k by a polynomially-sized Petri net. To this end, we make use of Lipton's proof of EXPSPACE-hardness for coverability [42].

6 Conclusion

We have shown that, under mild assumptions, disjointness of WSTS languages implies their regular separability. In particular, we have shown that if one of two disjoint upward-compatible WSTS is finitely branching, they are regular separable. Using our expressibility results, it is also sufficient if the underlying order for one of the two is an ω^2-wqo. A similar result holds for downward-compatible WSTS assuming that one of them is deterministic or the underlying order is an ω^2-wqo. As WSTS are typically ω^2-WSTS, our result already implies the decidability of regular separability for almost all WSTS of practical relevance.

Our work brings together research on inductive invariants and regular separability. We show that a finite representation of an inductive invariant for the product system can be transformed into a regular separator. For Petri nets, one may use any representation of the coverability set. As we show, it is beneficial in terms of the worst-case size, to use an inductive invariant obtained from the backward coverability algorithm [1]. For lossy channel systems, the coverability set is not computable [46], but one can obtain a finitely-represented inductive invariant e.g. from the EEC-algorithm [31].

We leave some questions without answer. It is not clear whether the assumptions of Theorems 7 and 6 are necessary; we were neither able to drop the assumptions, nor to provide a counterexample. Similarly, we do not know whether the inclusions in Theorem 5 are strict. Finally, in the case of Petri nets, closing the gap between the triply-exponential size of the NFA separator and the triply-exponential lower bound for DFA remains an open problem.

As future work, one could consider the *well-behaved transition systems (WBTS)* of [8], a generalization of WSTS where only the finite-antichain property is required.

References

1 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.

2 P. A. Abdulla, G. Delzanno, and L. Van Begin. Comparing the expressive power of well-structured transition systems. In *CSL*, pages 99–114, 2007.

3 P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *FORMATS*, volume 6919 of *LNCS*, pages 256–270. Springer, 2011.

4 P. A. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *LICS*, pages 345–354. IEEE, 2004.

5 P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *LICS*, pages 160–170. IEEE, 1993.

6 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18. ACM, 2010.

7 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. What's decidable about weak memory models? In *ESOP*, volume 7211 of *LNCS*, pages 26–46. Springer, 2012.

8 M. Blondin, A. Finkel, and P. McKenzie. Well behaved transition systems. *Logical Methods in Computer Science*, 13(3), 2017.

9 M. Blondin, A. Finkel, and P McKenzie. Handling infinitely branching well-structured transition systems. *Inf. Comput.*, 258:28–49, 2018.

10 A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *STACS*, pages 323–333, 1999.

11 L. Bozzelli and P. Ganty. Complexity analysis of the backward coverability algorithm for VASS. In *RP*, pages 96–109, 2011.

12 D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

13 N. Busi, M. Gabbrielli, and G. Zavattaro. Comparing recursion, replication, and iteration in process calculi. In *ICALP*, volume 3142 of *LNCS*, pages 307–319. Springer, 2004.

14 L. Clemente, W. Czerwiński, S. Lasota, and C. Paperman. Regular separability of Parikh automata. In *ICALP*, pages 117:1–117:13, 2017.

15 L. Clemente, W. Czerwiński, S. Lasota, and C. Paperman. Separability of reachability sets of vector addition systems. In *STACS 2017*, pages 24:1–24:14, 2017.

16 S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978.

17 W. Czerwiński and S. Lasota. Regular separability of one counter automata. In *LICS*, pages 1–12, 2017.

18 W. Czerwiński, S. Lasota, R. Meyer, S. Muskalla, K. Narayan Kumar, and P. Saivasan. Regular separability of well structured transition systems. *CoRR*, abs/1702.05334, 2018. `arXiv:1702.05334`.

19 W. Czerwiński, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *ICALP*, pages 150–161, 2013.

20 G. Delzanno and F. Rosa-Velardo. On the coverability and reachability languages of monotonic extensions of Petri nets. *Theoretical Computer Science*, 467:12–29, 2013.

21 E. D'Osualdo. *Verification of Message Passing Concurrent Systems*. PhD thesis, University of Oxford, 2015.

22 C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.

23 J. Esparza. Decidability and complexity of Petri net problems - an introduction. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, pages 374–428. Springer, 1998.

24 A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *ICALP*, pages 499–508, 1987.

25 A. Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2):144–179, 1990.

26 A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In *STACS*, pages 433–444, 2009.

27 A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part II: complete WSTS. *Logical Methods in Computer Science*, 8(3), 2012.

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

29 P. Ganty, J.-F. Raskin, and L. Van Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *VMCAI*, pages 49–64, 2006.

30 W. Gasarch. A survey of recursive combinatorics. In *Handbook of recursive mathematics, Vol. 2*, volume 139 of *Stud. Logic Found. Math.*, page 1041–1176. Amsterdam: North-Holland, 1998.

31 G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *Journal of Computer and System Sciences*, 72(1):180–203, 2006.

32 G. Geeraerts, J.-F. Raskin, and L. Van Begin. Well-Structured Languages. *Acta Informatica*, 44(3-4):249–288, 2007.

33 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

34 H. B. Hunt III. On the decidability of grammar problems. *Journal of the ACM*, 29(2):429–447, 1982.

35 P. Jancar. A note on well quasi-orderings for powersets. *Inf. Process. Lett.*, 72(5-6):155–160, 1999.

36 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV*, volume 5123 of *LNCS*, pages 214–226. Springer, 2008.

37 M. Kabil and M. Pouzet. Une extension d'un théorème de P. Jullien sur les âges de mots. *ITA*, 26:449–484, 1992.

38 E. Kopczynski. Invisible pushdown languages. In *LICS*, pages 867–872, 2016.

39 D. Kozen. *Automata and computability*. Undergraduate texts in Computer Science. Springer, 1997.

40 R. Lazic and S. Schmitz. The ideal view on rackoff's coverability technique. In *RP*, pages 76–88, 2015.

41 J. Leroux and S. Schmitz. Demystifying reachability in vector addition systems. In *LICS*, pages 56–67, 2015.

42 R. J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Department of Computer Science, 1976.

43 Z. Manna and A. Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

44 A. Marcone. Foundations of bqo theory. *Transactions of the American Mathematical Society*, 345(2):641–660, 1994.

45 M. Martos-Salgado and F. Rosa-Velardo. Dynamic networks of timed Petri nets. In *Petri nets*, pages 294–313, 2014.

46 R. Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 1-3(297):337–354, 2003.

47 R. Meyer. On boundedness in depth in the pi-calculus. In *TCS*, volume 273 of *IFIP*, pages 477–489. Springer, 2008.

48 M. Mukund, K. N. Kumar, J. Radhakrishnan, and M. A. Sohoni. Robust asynchronous protocols are finite-state. In *ICALP*, pages 188–199, 1998.

49 M. Mukund, K. N. Kumar, J. Radhakrishnan, and M. A. Sohoni. Towards a characterisation of finite-state message-passing systems. In *ASIAN*, pages 282–299, 1998.

50 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *FSTTCS*, pages 363–375, 2013.

51 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *MFCS*, pages 729–740, 2013.

52 T. Place and M. Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *ICALP*, pages 342–353, 2014.

53 T. Place and M. Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science*, 12(1), 2016.

54 F. Rosa-Velardo and M. Martos-Salgado. Multiset rewriting for the verification of depth-bounded processes with name binding. *Inf. Comput.*, 215:68–87, 2012.

55 S. Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *ICALP*, volume 6756 of *LNCS*, pages 441–452. Springer, 2011.

56 T. G. Szymanski and J. H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250, 1976.

57 T. Wies, D. Zufferey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *FOSSACS*, volume 6014 of *LNCS*, pages 94–108. Springer, 2010.

58 M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.

Separable GPL: Decidable Model Checking with More Non-Determinism

Andrey Gorlin
Stony Brook University
Department of Computer Science, Stony Brook, N.Y. 11794, USA
agorlin@cs.stonybrook.edu

C. R. Ramakrishnan
Stony Brook University
Department of Computer Science, Stony Brook, N.Y. 11794, USA
cram@cs.stonybrook.edu

Abstract

Generalized Probabilistic Logic (GPL) is a temporal logic, based on the modal mu-calculus, for specifying properties of branching probabilistic systems. We consider GPL over branching systems that also exhibit internal non-determinism under linear-time semantics (which is resolved by schedulers), and focus on the problem of finding the capacity (supremum probability over all schedulers) of a fuzzy formula. Model checking GPL is undecidable, in general, over such systems, and existing GPL model checking algorithms are limited to systems without internal non-determinism, or to checking non-recursive formulae. We define a subclass, called separable GPL, which includes recursive formulae and for which model checking is decidable. A large class of interesting and decidable problems, such as termination of 1-exit Recursive MDPs, reachability of Branching MDPs, and LTL model checking of MDPs, whose decidability has been studied independently, can be reduced to model checking separable GPL. Thus, GPL is widely applicable and, with a suitable extension of its semantics, yields a uniform framework for studying problems involving systems with non-deterministic and probabilistic behaviors.

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

Keywords and phrases Modal mu-calculus, probabilistic logics, probabilistic systems, branching systems, model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.36

Related Version `https://arxiv.org/abs/1604.06118`

Funding This work was partially supported by NSF grants IIS-1447549 and CNS-1405641.

Acknowledgements We are grateful to the reviewers for their detailed and insightful comments.

1 Introduction

For finite-state systems, model checking a temporal property can be cast in terms of model checking in the modal μ-calculus, the so-called "assembly language" of temporal logics. A number of temporal logics have been proposed and used for specifying properties of finite-state *probabilistic* systems.

GPL [5] is defined over branching probabilistic systems. In these systems, each state has a set of labeled outgoing transitions; each transition, in turn, specifies a (probabilistic) distribution of target states. Semantically, GPL treats probabilistic choices in the system as

linear-time. GPL is expressive enough to serve as an "assembly language" of a large number of probabilistic temporal logics.

In this paper, we consider GPL over branching systems that can also exhibit internal non-determinism under linear-time semantics. In these systems, the internal non-determinism is resolved by schedulers. We limit our attention to finding the capacity (supremum probability over all schedulers) of fuzzy formulae, as this problem is already sufficiently interesting and challenging; indeed, it is undecidable, in general. Existing GPL model checking algorithms, therefore, were limited either to systems without internal non-determinism [5], or to checking only non-recursive formulae [26].

Contributions and Significance. GPL is expressive enough that a variety of independently-studied verification problems can be cast as model checking branching systems with GPL. In fact, undecidable problems such as termination of multi-exit *Recursive* Markov Decision Processes (Recursive MDPs or RMDPs) can be reduced in linear time to model checking with GPL. We introduce a syntactically-defined subclass, called *separable GPL*, for which model checking is decidable.

We illustrate the expressiveness of separable GPL by considering independently-studied decidable verification problems involving systems that have probabilistic and non-deterministic choice. Examples of such problems include LTL model checking of MDPs [2], reachability in branching MDPs [10], and termination of 1-exit RMDPs [12]. These problems can all be reduced, in linear time, to model checking separable GPL formulae (see Sect. 3).

We describe a procedure for model checking GPL, which either successfully returns the model checking result, or terminates with failure. We also show that the procedure always terminates successfully for separable GPL (see Sect. 4).

Termination of multi-exit RMDPs, when cast as a model checking problem over GPL along the same lines as our treatment of 1-exit RMDPs, yields an *entangled* GPL formula, which is outside of separable GPL. Thus, separability can be seen as a characteristic of the verification problems that are known to be decidable, when cast in terms of model checking in GPL. Consequently, GPL and its sublogic are useful formalisms to study the relationships between verification problems over branching systems with both probabilistic and internal non-deterministic choice. We discuss these issues in greater detail in Sect. 5.

2 GPL and Branching Systems

In this section, we formally define probabilistic branching systems and give the syntax and semantics of GPL fuzzy formulae.

2.1 Probabilistic Branching Systems

We define a probabilistic branching system (PBS).

▶ **Definition 1** (PBS). With respect to fixed sets Act and $Prop$ of *actions* and *propositions*, respectively, a PBS L is a quadruple (S, δ, P, I), where

- S is a countable set of states;
- $\delta \subseteq S \times Act \times S$ is the transition relation;
- $P : \delta \times \mathbb{N} \to [0, 1]$ is the transition probability distribution satisfying:
  - $\forall s \in S. \forall a \in Act. \forall c \in \mathbb{N}. \sum_{s' : (s,a,s') \in \delta} P(s, a, s', c) \in \{0, 1\}$, and
  - $\forall s \in S. \forall a \in Act. \forall s' \in S. (s, a, s') \in \delta \implies (\exists c \in \mathbb{N}. P(s, a, s', c) > 0)$;
- $I : S \to 2^{Prop}$ is the *interpretation*, recording the set of propositions true at a state.

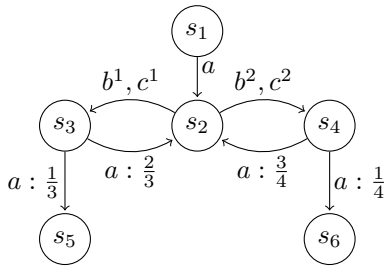


Figure 1 An example PBS.

This definition is in line with that of probabilistic automata [21, 24], in which, given an action, a probabilistic distribution is chosen non-deterministically (we assume there are finitely many distributions for any state-action pair). Other equally expressive models include alternating automata, in which labeled non-deterministic choices are followed by silent probabilistic choices. The difference between such models has been analyzed with respect to bisimulation [25]. However, a PBS is *explicitly* a branching system, as we show next.

Given $L = (S, \delta, P, I)$, a *partial computation* is a sequence $\sigma = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$, where for all $0 \leq i < n$, $(s_i, a_{i+1}, s_{i+1}) \in \delta$. Also, $\mathsf{fst}(\sigma) = s_0$ and $\mathsf{last}(\sigma) = s_n$ denote, respectively, the first and last states in σ. Each transition of a partial computation is labeled with an action $a_i \in Act$. The set of all partial computations of L is denoted by $\mathcal{C}_L$, and $\mathcal{C}_L(s) = \{\sigma \in \mathcal{C}_L \mid \mathsf{fst}(\sigma) = s\}$. *Composition* of partial computations, $\sigma \xrightarrow{a} \sigma'$, where $\sigma' = s'_0 \xrightarrow{b_1} \cdots \xrightarrow{b_m} s'_m$, represents $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_0 \xrightarrow{b_1} \cdots \xrightarrow{b_m} s'_m$ if $(s_n, a, s'_0) \in \delta$. A partial computation σ' is a *prefix* of σ if $\sigma' = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} s_i$ for some $i \leq n$.

From a set of partial computations, we can build deterministic trees. $T \subseteq \mathcal{C}_L$ is *prefix-closed* if, for every $\sigma \in T$ and σ' a prefix of σ, $\sigma' \in T$. T is *deterministic* if for every $\sigma, \sigma' \in T$ with $\sigma = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s \cdots$ and $\sigma' = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a'} s' \cdots$, either $a \neq a'$ or $s = s'$, i.e., if a pair of computations share a prefix, the first difference cannot involve transitions labeled by the same action. If a tree T has a single starting state, it is denoted $\mathsf{root}(T)$; if $s = \mathsf{root}(T)$ then $T \subseteq \mathcal{C}_L(s)$. We also let $\mathsf{edges}(T) = \{(\sigma, a, \sigma') \mid \sigma, \sigma' \in T \wedge \exists s \in S. \sigma' = \sigma \xrightarrow{a} s\}$.

▶ **Definition 2** (D-trees and outcomes)**.** A d-tree is a set of partial computations with a single starting state that is prefix-closed and deterministic. A d-tree T is maximal if there exists no d-tree T' with $T \subset T'$. An *outcome* is a maximal d-tree.

$\mathcal{T}_L$ refers to all the d-trees of L, and $\mathcal{T}_L(s) = \{T \in \mathcal{T}_L \mid \mathsf{root}(T) = s\}$. T' is a *prefix* of T if $T' \subseteq T$. $T \xrightarrow{a} T'$ means $T' = \{\sigma \mid \mathsf{root}(T) \xrightarrow{a} \sigma \in T\}$. T is *finite* if $|T| < \infty$. $\mathcal{M}_L$ and $\mathcal{M}_L(s)$ are analogous to $\mathcal{T}_L$ and $\mathcal{T}_L(s)$, but for maximal d-trees.

An example PBS and two of its outcomes are shown in Figs. 1 and 2. In the PBS, transitions are usually annotated with their action label and probability (we may omit the probability when it is 1). Note that there are two transitions labeled b (and c) from state s_2, reflecting internal non-determinism, and we use superscripts to distinguish them.

D-trees are constructed from S and δ, without regard for P. A fully probabilistic branching system (fPBS) has no internal non-determinism, i.e., its transition probability distribution P is a function of δ. An fPBS was called an RPLTS in [5], which may have obscured its branching nature.

A property of a PBS will hold for some subset of its outcomes, and we need to measure this set. To resolve the internal non-deterministic choices, we require a scheduler.

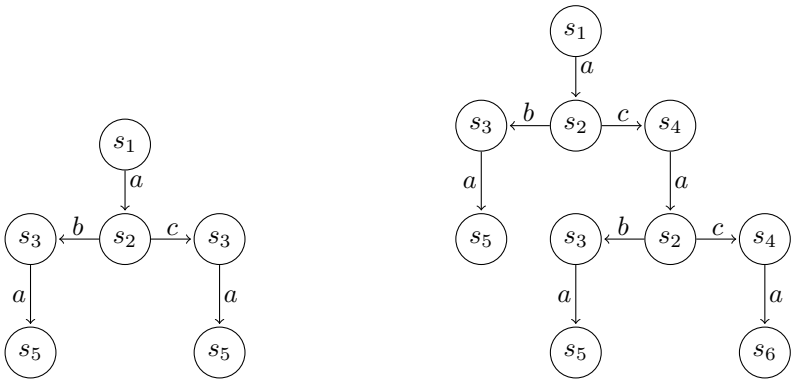


Figure 2 Selected outcomes for the PBS in Fig. 1.

▶ **Definition 3** (Scheduler)**.** A scheduler for a PBS L is a function $\gamma : \mathcal{C}_L \times Act \to \mathbb{N}$, such that if an action a is present at $s = \mathsf{last}(\sigma)$, then $\gamma(\sigma, a) = c$ implies that $\sum_{s'} P(s, a, s', c) = 1$.

Note that we have defined deterministic schedulers, which are also aware of their relevant histories. Given a scheduler γ for a PBS L, we have an fBPS $L_\gamma = (S_\gamma, \delta_\gamma, P_\gamma, I_\gamma)$, where $S_\gamma \subseteq \mathcal{C}_L$ and so $\delta_\gamma \subseteq \mathcal{C}_L \times Act \times \mathcal{C}_L$.

Thus, given a scheduler γ for a PBS L, we can follow the definitions for the measure of d-trees from [5]. A *basic cylindrical subset* of $\mathcal{M}_L(s)$ contains all trees sharing a particular prefix. Letting $s \in S$ and $T \in \mathcal{T}_L(s)$ such that T is finite, $B_T = \{T' \in \mathcal{M}_L \mid T \subseteq T'\}$. Now, we define the measure:

▶ **Definition 4.** For a PBS L with scheduler γ, the probability measure of a basic cylindrical subset B_T is defined by a partial function $\mathsf{m}^\gamma : 2^{\mathcal{M}_L} \to [0, 1]$, where:

$$\mathsf{m}^\gamma(B_T) = \prod_{(\sigma,a,\sigma') \in \mathsf{edges}(T)} P(\mathsf{last}(\sigma), a, \mathsf{last}(\sigma'), \gamma(\sigma, a)) \tag{1}$$

From here, a probability measure $\mathsf{m}^\gamma_s : \mathcal{B}_s \to [0, 1]$ on the smallest σ-field of sets $\mathcal{B}_s$ is generated from basic cylindrical subsets B_T with $\mathsf{m}^\gamma_s(B_T) = \mathsf{m}^\gamma(B_T)$ (cf. [5, Definition 8]).

▶ **Example 5.** Letting $\sigma = s_1 \xrightarrow{a} s_2$, the measure of the left outcome in Figure 2 is $\frac{1}{9}$ for any scheduler γ where $\gamma(\sigma, b) = \gamma(\sigma, c) = 1$, and 0 for all other schedulers.

2.2 GPL Syntax

GPL *fuzzy formulae* depend on *outcomes*. We give the syntax of GPL fuzzy formulae ψ, with $A \in Prop$,[1] $X \in Var$, $a \in Act$, as:

$$\psi ::= A \mid \neg A \mid X \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle a \rangle \psi \mid [a]\psi \mid \mu X.\psi \mid \nu X.\psi \ . \tag{2}$$

Operators $\mu X.\psi$ and $\nu X.\psi$ are least and greatest fixed point operators for the "equation" $X = \psi$. As in [5], fuzzy formulae are alternation-free, which allows a simpler semantics for fixed points while retaining the expressiveness required for our applications. Additionally, *diamond* implies *box*: $\langle a \rangle \psi$ means that there is an a-transition and it satisfies ψ; $[a]\psi$ means that if there is an a-transition, it satisfies ψ. We may also use a set $\alpha \subseteq Act$ for the modalities, reading $\langle \alpha \rangle \psi$ as $\bigvee_{a \in \alpha} \langle a \rangle \psi$ and $[\alpha]\psi$ as $\bigwedge_{a \in \alpha} [a]\psi$. When we write "$-$" for α, then $\alpha = Act$.

[1] The full syntax allows any state formula [5, 15] as a fuzzy formula.

Table 1 GPL semantics: fuzzy formulae.

$$
\begin{aligned}
\Theta_L(A)e &= \bigcup_{A \in I(s)} \mathcal{M}_L(s), \\
\Theta_L(X)e &= e(X), \\
\Theta_L(\langle a \rangle \psi)e &= \{T \in \mathcal{M}_L \mid \exists T' : T \xrightarrow{a} T' \wedge T' \in \Theta_L(\psi)e\}, \\
\Theta_L([a]\psi)e &= \{T \in \mathcal{M}_L \mid (T \xrightarrow{a} T') \Rightarrow T' \in \Theta_L(\psi)e\}, \\
\Theta_L(\psi_1 \wedge \psi_2)e &= \Theta_L(\psi_1)e \cap \Theta_L(\psi_2)e, \\
\Theta_L(\psi_1 \vee \psi_2)e &= \Theta_L(\psi_1)e \cup \Theta_L(\psi_2)e, \\
\Theta_L(\mu X.\psi)e &= \bigcup_{i=0}^{\infty} M_i, \text{ where } M_0 = \emptyset \text{ and } M_{i+1} = \Theta_L(\psi)e[X \mapsto M_i], \\
\Theta_L(\nu X.\psi)e &= \bigcap_{i=0}^{\infty} N_i, \text{ where } N_0 = \mathcal{M}_L \text{ and } N_{i+1} = \Theta_L(\psi)e[X \mapsto N_i].
\end{aligned}
$$

2.3 GPL Semantics

We define the semantics of fuzzy formulae with respect to a fixed PBS $L = (S, \delta, P, I)$, where Ψ is the set of all fuzzy formulae. A function $\Theta_L : \Psi \to 2^{\mathcal{M}_L}$, augmented with an extra *environment* parameter $e : Var \to 2^{\mathcal{M}_L}$, returns the set of outcomes satisfying a given fuzzy formula. For a given $s \in S$, $\Theta_{L,s}(\psi) = \Theta_L(\psi) \cap \mathcal{M}_L(s)$. For the fuzzy formulae, the semantics is as in Table 1.

We refer to the value $\sup_\gamma \mathsf{m}^\gamma_s(\Theta_{L,s}(\psi))$ as a *capacity* (following [6]), and write it as $\mathsf{Pr}_{L,s}(\psi)$. In particular, $[a]\psi$ and $\langle a \rangle \psi$ are equivalent when action a is present at a state.

As d-trees do not depend on P, several important properties for GPL over PBSs carry over naturally from [5]. First, we have distributivity on *box* and *diamond* [5, Lemma 1]:

▶ **Lemma 6** (Distributivity on modal operators). *Letting* $\oplus \in \{\wedge, \vee\}$*:*

$$
\begin{aligned}
\Theta_L([a]\psi_1 \oplus [a]\psi_2) &= \Theta_L\big([a](\psi_1 \oplus \psi_2)\big) \\
\Theta_L(\langle a \rangle \psi_1 \oplus \langle a \rangle \psi_2) &= \Theta_L\big(\langle a \rangle(\psi_1 \oplus \psi_2)\big) \\
\Theta_L([a]\psi_1 \wedge \langle a \rangle \psi_2) &= \Theta_L\big(\langle a \rangle(\psi_1 \wedge \psi_2)\big)
\end{aligned} \tag{3}
$$

Similarly, because a PBS L with a scheduler γ yields an fPBS L_γ, we can relate the probability of a conjunction with that of a disjunction and compute the effect of taking a step (where $\gamma^{s'}_a(\sigma, a') = \gamma(s \xrightarrow{a} \sigma, a')$ and $\mathsf{fst}(\sigma) = s'$) [5, Lemma 2]:

$$
\mathsf{m}^\gamma_s(\Theta_{L,s}(\psi_1 \vee \psi_2)) = \mathsf{m}^\gamma_s(\Theta_{L,s}(\psi_1)) + \mathsf{m}^\gamma_s(\Theta_{L,s}(\psi_2)) - \mathsf{m}^\gamma_s(\Theta_{L,s}(\psi_1 \wedge \psi_2)) \tag{4}
$$

$$
\mathsf{m}^\gamma_s(\Theta_{L,s}(\langle a \rangle \psi)) = \sum_{s' : (s,a,s') \in \delta} P(s, a, s', \gamma(s, a)) \cdot \mathsf{m}^{\gamma^{s'}_a}_{s'}(\Theta_{L,s'}(\psi)) \tag{5}
$$

Additionally, we can express the negation of a fuzzy formula ψ, $\mathsf{neg}(\psi)$, such that, for any PBS L and state s ([5, Lemma 3]):

$$
\Theta_{L,s}(\mathsf{neg}(\psi)) = \mathcal{M}_L(s) - \Theta_{L,s}(\psi) \tag{6}
$$

The proof involves switching all the operators to their duals.

▶ **Example 7** (Distributivity and negation). The formula $\psi = \mu X.[a][b]X \wedge [a][c]X$ is satisfied by all *finite* outcomes of the PBS in Fig. 1. By Lemma 6, it is equivalent to $\mu X.[a]([b]X \wedge [c]X)$. Also, $\mathsf{neg}(\psi) = \nu X.\langle a \rangle \langle b \rangle X \vee \langle a \rangle \langle c \rangle X$. Note that a scheduler that maximizes the measure of ψ will minimize the measure of $\mathsf{neg}(\psi)$.

Table 2 Encoding of CTL over PLTSs.

$$E_{CTL}(\psi) = \begin{cases} \psi, & \psi \in \mathit{Prop}, \\ \mathsf{neg}(E_{CTL}(\psi')), & \psi = \neg\psi', \\ E_{CTL}(\psi_1) \wedge E_{CTL}(\psi_2), & \psi = \psi_1 \wedge \psi_2, \\ \langle - \rangle E_{CTL}(\psi), & \gamma = \mathsf{EX}\psi, \\ \mu X. E_{CTL}(\psi_2) \vee (E_{CTL}(\psi_1) \wedge [-]X), & \psi = \mathsf{A}[\psi_1 \mathsf{U} \psi_2], \\ \mu X. E_{CTL}(\psi_2) \vee (E_{CTL}(\psi_1) \wedge \langle - \rangle X), & \psi = \mathsf{E}[\psi_1 \mathsf{U} \psi_2]. \end{cases}$$

3 Encoding Other Model Checking Problems

When GPL was originally introduced, its most interesting known application was to PCTL* model checking [2], as it is straightforward to encode any LTL formula as a fuzzy formula in GPL [5]. In this section, we relate GPL to two applications involving branching systems.

3.1 PTTL and Branching Processes

As GPL fuzzy formulae are interpreted over outcomes of a system, they can encode LTL [5, Sect. 3.2], in the fragment of GPL for systems with $\mathit{Act} = \{a\}$ and no terminal states; then the outcomes are paths, rather than trees. When Act is not limited to one action, so the outcomes are trees, it is correspondingly straightforward to encode CTL (Table 2). This may seem either obvious or strange, as GPL and CTL are both interpreted over trees, but CTL is typically understood as a logic over systems and not their outcomes; this encoding also reduces to the referenced LTL encoding when all the outcomes are paths.

Along similar lines, Probabilistic Tree Temporal Logic (PTTL), has been independently introduced [4]. Branching Processes (BPs) are a branching extension of Markov chains, and PTTL is a logic over BPs. PTTL's fuzzy formulae are limited to the A and E versions of the X, U, and R operators being applied to state formulae, a subset of our CTL encoding.

Additionally, BPs have been extended with non-determinism, yielding Branching MDPs (BMDPs), for which the extinction and reachability problems have been analyzed [10, 12]. When we use only the modal operators $[-]$ and $\langle - \rangle$, as with CTL and PTTL, PBSs are equivalent to BMDPs (and fPBSs to BPs).

3.2 Encoding of RMDP Termination

A Recursive MDP (RMDP) [12] is specified as a set of components. A component has an entry node and one or more exit nodes. Components may contain *boxes*, each box having a *call port* that represents a procedure call and *return ports* to represent a possible return from the called procedure. We provide the formal definition of the (more general) *Recursive Simple Stochastic Games* (RSSGs) [12].

▶ **Definition 8.** An RSSG A is a tuple $(A_1, \ldots, A_k)$, where each *component graph* A_i is a septuple $(N_i, B_i, Y_i, \mathsf{En}_i, \mathsf{Ex}_i, \mathsf{pl}_i, \delta_i)$:

- N_i is a set of nodes, containing subsets En_i and Ex_i of entry and exit nodes, respectively.
- B_i is a set of boxes, with a mapping $Y_i : B_i \to \{1, \ldots, k\}$ assigning each box to a component. Each box has a set of call and return ports, associated with the entry and

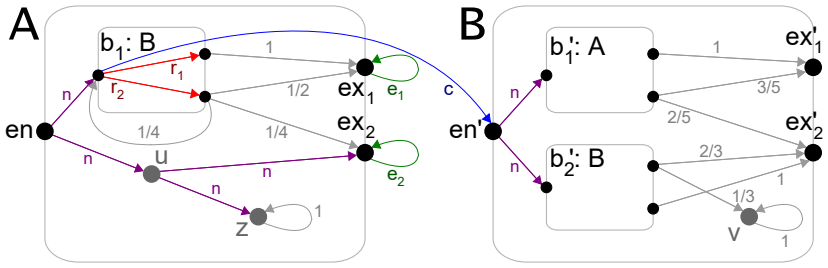


Figure 3 Example RMDP with Call, Return, and Exit edges added to Component "A".

exit nodes, respectively, in the corresponding components: $\mathsf{Call}_b = \{(b, en) \mid en \in \mathsf{En}_{Y_i(b)}\}$, $\mathsf{Return}_b = \{(b, ex) \mid ex \in \mathsf{Ex}_{Y_i(b)}\}$. Additionally, we have:

$$\mathsf{Call}^i = \bigcup_{b \in B_i} \mathsf{Call}_b, \quad \mathsf{Return}^i = \bigcup_{b \in B_i} \mathsf{Return}_b, \quad Q_i = N_i \cup \mathsf{Call}^i \cup \mathsf{Return}^i.$$

- $\mathsf{pl}_i : Q_i \to \{0, 1, 2\}$ is a mapping that specifies whether, at each state, the choice is probabilistic (i.e., *player* 0), or non-deterministic (player 1: maximizing, player 2: minimizing). As any $u \in \mathsf{Call}^i \cup \mathsf{Ex}_i$ has no outgoing transitions, let $\mathsf{pl}_i(u) = 0$ for these states.
- δ_i is the transition relation, with transitions of the form (u, p_{uv}, v), when $\mathsf{pl}_i(u) = 0$ and u is not an exit node or a call port, and v may not be an entry node or a return port. Additionally, $p_{uv} \in (0, 1]$ and, for each u, $\sum_{v':(u,\cdot,v') \in \delta_i} p_{uv'} = 1$. Meanwhile, when $\mathsf{pl}_i(u) > 0$, transitions are of the form $(u, \bot, v)$.

RMDPs only have a player 1 or player 2, depending on whether they are maximizing or minimizing, respectively. Figure 3 shows an RMDP with two components, A and B. Any call to A non-deterministically results in either a call to B (via box b_1) or a transition to u.

3.2.1 Translating RMDPs to PBSs

We can translate an RMDP into a PBS L with $Act = \{p, n, c, r_j, e_j\}$, with states of the PBS corresponding to nodes of the RMDP. We retain the RMDP's transitions, labeling them as "n" for actions from a non-deterministic choice and "p" for probabilistic choice. To this basic structure we add three new kinds of edges:

- "e_j" for the jth exit node of a component,
- "c" edges from a call port to the called component's entry node, and
- "r_j" edges from a call port to each return port in the box.

Figure 3 shows the new edges added to component A (r_1 and r_2 are *distinct* actions here). Formally, we define the translation as follows:

▶ **Definition 9** (Translated RMDP). The translated RMDP A is a PBS $L = (S, \delta, P, I)$:

- The set of states S is the set of all the nodes, as well as the call and return ports of the boxes, i.e., $S = \bigcup_i Q_i$. Additionally, we associate a consistent index with each state corresponding to an exit node or a return port.
- The transition relation δ has all the transitions of the components, labeled by action p for the probabilistic transitions and n for the non-deterministic ones. Thus, when $(u, p_{uv}, v) \in \delta_i$ for any i, then $(u, p, v) \in \delta$, and when $(u, \bot, v) \in \delta_i$, $(u, n, v) \in \delta$. Additionally, we have $((b, en), c, en) \in \delta$ and $((b, en), r_j, (b, ex_j)) \in \delta$ for every box b, and $(ex_j, e_j, ex_j) \in \delta$ for every exit node. Note the indices used, matching r_j with e_j.

- The transition probability distribution P is defined as follows:
  - $P(u, p, v, 0) = p_{uv}$.
  - Assuming a one-to-one function $num : S \to \mathbb{N}$, $P(u, n, v, num(v)) = 1$.
  - When $a \in \{c, r_j, e_j\}$ and $(u, a, v) \in \delta$, $P(u, a, v, 0) = 1$.
- $I(s) = \emptyset$ for all states $s \in S$.

While c edges denote control transfer due to a call, r edges summarize returns from the called procedure (they are similar to the *jump edges* in nested state machines [1]). Additionally, if an RMDP has no non-deterministic choices, it is called a Recursive Markov Chain (RMC), and it can be translated to an fBPS.

3.2.2 GPL Formula for RMDP Termination

We define a formula for an m-exit RMDP, which we will show models termination. In this section, we also write a mu-calculus formula via equations, where, e.g., if $\psi_1 =_\mu f_1(\psi_1, \psi_2)$ and $\psi_2 =_\mu f_2(\psi_1, \psi_2)$, then ψ_1 is $\mu X_1.f_1(X_1, \mu X_2.f_2(X_1, X_2))$.

▶ **Definition 10** (RMDP termination formula)**.** For an m-exit RMDP, the termination formula may be given as m equations, for $1 \leq i \leq m$:

$$\psi_i =_\mu \langle e_i\rangle \mathsf{tt} \vee \langle p\rangle\psi_i \vee \langle n\rangle\psi_i \vee \left(\bigvee_{j=1}^{m} \langle c\rangle\psi_j \wedge \langle r_j\rangle\psi_i\right) \tag{7}$$

We show what the formula looks like for 1 and 2 exits ((8) and (9), respectively):

$$\psi_1 = \mu X.\langle e_1\rangle \mathsf{tt} \vee \langle p\rangle X \vee \langle n\rangle X \vee (\langle c\rangle X \wedge \langle r_1\rangle X) \tag{8}$$

$$\begin{aligned} \psi_1 &=_\mu \langle e_1\rangle \mathsf{tt} \vee \langle p\rangle\psi_1 \vee \langle n\rangle\psi_1 \vee (\langle c\rangle\psi_1 \wedge \langle r_1\rangle\psi_1) \vee (\langle c\rangle\psi_2 \wedge \langle r_2\rangle\psi_1) \\ \psi_2 &=_\mu \langle e_2\rangle \mathsf{tt} \vee \langle p\rangle\psi_2 \vee \langle n\rangle\psi_2 \vee (\langle c\rangle\psi_1 \wedge \langle r_1\rangle\psi_2) \vee (\langle c\rangle\psi_2 \wedge \langle r_2\rangle\psi_2) \end{aligned} \tag{9}$$

The termination problem for 1-exit RMDPs is equivalent to that of BMDP extinction. We can see in (8) that the branching occurs at the call port, with the c and r actions.

Termination of multi-exit RMDPs is undecidable, in general [12].

▶ **Theorem 11** (RMDP translation)**.** *Given an m-exit RMDP A, its translated BPS L, and ψ_i as in (7), there is a one-to-one correspondence between the set of paths $\{P_i\}$ through A terminating at exit i and the set of minimal prefixes $\{T_i\}$ of outcomes of L satisfying ψ_i.*

Proof sketch. First, there is indeed a minimal prefix T_i for any outcome satisfying ψ_i, and we can create it by truncating the tree after any e action and pruning the "incorrect" r_j branches (as $\psi_1, \psi_2, \ldots, \psi_m$ are all mutually exclusive, $\langle c\rangle\psi_j$ is satisfied for exactly one j at each branching node of the outcome). Also, probabilistic and (internal) non-deterministic choices are made identically. We induct on the recursion depth of P_i.

If the depth of P_i is 0, then it never reaches a call port, and T_i is a path taking p and n transitions until exit i, with a concluding e_i transition.

Now, suppose that, for $1 \leq i \leq m$, for all paths P_i with depth $k < d$, we have a matching T_i. We show that for any path P_i with depth d, we also have a matching T_i.

In T_i, a node of the form (b, en) has a c transition to en and an r_j transition to (b, ex_j). The segment of P_i from en to ex_j is a path P'_j. P'_j must have depth $k < d$; so, by the induction assumption, T_i has the matching T'_j as the subtree rooted at en (after the c transition).

Then, P_i continues from (b, ex_j), which T_i matches by taking the r_j transition. Thus, T_i matches recursive calls with c transitions and, for the top level, has a partial computation taking p, n, and r_j transitions until exit i, with a concluding e_i transition. ◀

▶ **Corollary 12** (Undecidability). *GPL model checking over PBSs is undecidable.*

Proof. We can encode an undecidable problem, termination of multi-exit RMDPs, in GPL. It remains to show that the BPS scheduler has sufficient information. This follows because each partial computation in the minimal prefix includes the entire path through the RMDP to reach a particular state, *except* for the calls that already returned, which are represented by r_j transitions instead. So, the scheduler knows the current state and all of the open calls. ◀

4 Decidable Model Checking

The *modchk-fuzzy* procedure from [5, Sect. 4.1] finds the probability of a GPL fuzzy formula over a fixed fPBS. In that case, we can eliminate top-level disjunctions, via (4). In general, though, we would want the relation in (10).

$$\mathsf{Pr}_{L,s}(\psi_1 \vee \psi_2) \stackrel{?}{=} \mathsf{Pr}_{L,s}(\psi_1) + \mathsf{Pr}_{L,s}(\psi_2) - \mathsf{Pr}_{L,s}(\psi_1 \wedge \psi_2) \tag{10}$$

This requires that the same scheduler maximize ψ_1, ψ_2, $\psi_1 \wedge \psi_2$, and $\psi_1 \vee \psi_2$; these may all be distinct. Later, *modchk-fuzzy* was also adapted for systems with internal non-determinism, but limited to non-recursive fuzzy formulae [26]. This is inadequate if we want to model check LTL over MDPs or find the maximum probability of termination in a 1-exit RMDP.

4.1 Graph Construction

Disjunctions in themselves are not the problem, of course. We can still attempt a standard graph construction and see whether we can evaluate the parts of the formula dependent on the current state and cleanly separate the parts dependent on the (branching) future. In this section, we describe this construction.

First, we define a few things needed for the construction. Let $\psi[\psi'/X]$ represent the formula ψ with all free instances of X replaced with ψ'.

▶ **Definition 13** (Fischer-Ladner closure). Given a (closed) formula ψ, its *Fischer-Ladner closure*, $Cl(\psi)$, is the smallest set such that the following hold:

- $\psi \in Cl(\psi)$.
- If $\psi' \in Cl(\psi)$, then:
  - if $\psi' = \psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$, then $\psi_1, \psi_2 \in Cl(\psi)$;
  - if $\psi' = \langle a \rangle \psi''$ or $[a]\psi''$ for some $a \in Act$, then $\psi'' \in Cl(\psi)$;
  - if $\psi' = \sigma X.\psi''$, then $\psi''[\sigma X.\psi''/X] \in Cl(\psi)$, with σ either μ or ν.

It will be useful to view a fuzzy formula as an *and-or tree*.

▶ **Definition 14** (And-or tree). The and-or tree of a fuzzy formula ψ, $AO(\psi)$ is a node labeled by $\oplus$, where $\oplus \in \{\wedge, \vee\}$, with children $AO(\psi_1)$ and $AO(\psi_2)$ when $\psi = \psi_1 \oplus \psi_2$, and a leaf ψ otherwise.

A formula of the form $\langle a \rangle \psi$ or $[a]\psi$ is called *modal*. We say that ψ' is an *unguarded subformula* of ψ if it is a leaf in $AO(\psi)$. Also, $\mathcal{AO}(S)$ represents the set of and-or trees with elements of a set S as leaves. We want to separate a formula by actions, which we attempt by finding its *factored* form.

▶ **Definition 15** (Formula Transformations)**.**

- The *fixed-point expansion* of ψ, denoted by $FPE(\psi)$, is a formula ψ' obtained by expanding any unguarded subformula of the form $\sigma X.\psi_X$ to $\psi_X[\sigma X.\psi_X/X]$ where $\sigma \in \{\mu, \nu\}$.
- We say that a formula is non-probabilistic if it depends only on the current state, i.e., $A \in Prop$ or of the form $\langle a\rangle\phi$ and $[a]\phi$ for $a \in Act$ and $\phi \in \{\mathsf{tt}, \mathsf{ff}\}$. The *partial evaluation* of a fuzzy formula ψ at state s, denoted by $PE(s, \psi)$, is a formula obtained by evaluating unguarded non-probabilistic subformulae and simplifying the result (i.e., $\psi' \wedge \phi$, where ϕ is non-probabilistic, becomes ψ' if ϕ is true and ff otherwise).
- A *grouping* of a formula ψ, denoted by $GRP(\psi)$, groups modalities in a formula using distributivity. Formally, GRP maps ψ to a ψ' that is equivalent to ψ based on the equivalences in Lemma 6, applied left-to-right as much as possible on the top level (i.e., $\langle a\rangle\psi_1 \wedge \langle a\rangle\psi_2$ becomes $\langle a\rangle(\psi_1 \wedge \psi_2)$; we may also rearrange the tree via commutativity and associativity, e.g., from $\langle a\rangle\psi_1 \wedge (\langle a\rangle\psi_2 \wedge \langle b\rangle\psi_3)$ to $(\langle a\rangle\psi_1 \wedge \langle a\rangle\psi_2) \wedge \langle b\rangle\psi_3)$.

▶ **Definition 16** (Factored form and entanglement)**.** A formula ψ is in factored form if $\psi \in \{\mathsf{tt}, \mathsf{ff}\}$, or if ψ satisfies $\psi = GRP(\psi)$ and all leaves of $AO(\psi)$ are modal.
Also, ψ is *entangled* if it is in factored form and any action guards multiple leaves of $AO(\psi)$.

Given a state s, a formula ψ' can be transformed into a semantically equivalent one ψ'' that is in factored form, $\psi'' = FAC(s, \psi') = GRP\left(PE\big(s, FPE(\psi')\big)\right)$.[2]

▶ **Example 17.** For ψ_1 in (9) and s with outgoing c and r_i actions,
$FAC(s, \psi_1) = (\langle c\rangle\psi_1 \wedge \langle r_1\rangle\psi_1) \vee (\langle c\rangle\psi_2 \wedge \langle r_2\rangle\psi_1)$, which is entangled on c, with leaves $\{\langle c\rangle\psi_1, \langle c\rangle\psi_2, \langle r_1\rangle\psi_1, \langle r_2\rangle\psi_1\}$.

▶ **Definition 18** (Dependency graph)**.** The dependency graph for model checking a formula ψ with respect to a state s in PBS L, denoted by $\mathsf{Dg}(s, \psi)$, is a directed graph (N, E), where *node set* $N \subseteq S \times \mathcal{AO}(Cl(\psi))$, and *edge set* $E \subseteq N \times (Act \cup \{\varepsilon, \varepsilon^\wedge, \varepsilon^\vee\}) \times N$; i.e., the edges are labeled from $Act \cup \{\varepsilon, \varepsilon^\wedge, \varepsilon^\vee\}$. If graph construction succeeds, the sets N and E are the smallest such that:

- $(s, \psi) \in N$.
- If $(s', \psi') \in N$ and $\psi' = FAC(s', \psi')$, then ψ' is not entangled (if ψ' is entangled, graph construction terminates with failure).
- If $(s', \psi') \in N$, $\psi'' = FAC(s', \psi')$, and $\psi' \neq \psi''$: $(s', \psi'') \in N$ and $((s', \psi'), \varepsilon, (s', \psi'')) \in E$.
- If $(s', \psi'_1 \oplus \psi'_2) \in N$ (an *and*-node or an *or*-node), then $(s', \psi'_i) \in N$ for $i = 1, 2$. Moreover, $((s', \psi'_1 \oplus \psi'_2), \varepsilon^\oplus, (s', \psi'_i)) \in E$ for $i = 1, 2$, and $\oplus \in \{\wedge, \vee\}$.
- If $(s', \langle a\rangle\psi') \in N$ (action node), then $(s'', \psi') \in N$ for each s'' such that $(s', a, s'') \in \delta$. Moreover, $((s', \langle a\rangle\psi'), a, (s'', \psi')) \in E$.

When we transform ψ' to the factored form ψ'', the semantics does not change, i.e., $\Theta_{L,s'}(\psi') = \Theta_{L,s'}(\psi'')$. For the formulae in factored form, standard GPL semantics applies (Table 1). Recall that when a is present at state s', then $\langle a\rangle\psi'$ and $[a]\psi'$ are equivalent; thus, we can assume all action nodes to be of the form $(s', \langle a\rangle\psi')$. From these semantics, we also get the relationships for the capacities. Here, $\prod$ is the standard product operator, while $\coprod_{i\in I} x_i = 1 - \prod_{i\in I}(1 - x_i)$.

[2] After applying GRP, we may have a modal leaf $\langle a\rangle\psi'_a \notin \mathcal{AO}(Cl(\psi))$. Then, we may view a as a prefix label on the subtree $AO(\psi'_a) \in \mathcal{AO}(Cl(\psi))$.

▶ **Lemma 19** (Capacities). *Fix* $\mathsf{Dg}(s_0, \psi) = (N, E)$. *The capacity* $\mathsf{Pr}_{L,s}(\psi')$ *for a node* (s, ψ') *is as follows:*

- $\mathsf{Pr}_{L,s}(\mathsf{ff}) = 0$ *and* $\mathsf{Pr}_{L,s}(\mathsf{tt}) = 1$.
- *If* (s, ψ') *is an* and*-node, then:* $\mathsf{Pr}_{L,s}(\psi') = \prod_{((s,\psi'),\varepsilon^\wedge,(s,\psi'_i))\in E} \mathsf{Pr}_{L,s}(\psi'_i)$.
- *If* (s, ψ') *is an* or*-node, then:* $\mathsf{Pr}_{L,s}(\psi') = \coprod_{((s,\psi'),\varepsilon^\vee,(s,\psi'_i))\in E} \mathsf{Pr}_{L,s}(\psi'_i)$.
- *If* (s, ψ') *is an action node, i.e.,* $\psi' = \langle a\rangle\psi'_a$, *then:*

$$\mathsf{Pr}_{L,s}(\psi') = \max_{c\in\mathbb{N}} \sum_{((s,\psi'),a,(s',\psi'_a))\in E} P(s, a, s', c) \cdot \mathsf{Pr}_{L,s'}(\psi'_a) \tag{11}$$

- *The remaining nodes* (s, ψ') *have a unique successor* (s, ψ'') *with* $\mathsf{Pr}_{L,s}(\psi') = \mathsf{Pr}_{L,s}(\psi'')$.

Proof. Most of the cases are straightforward and similar to the GPL model checking algorithm [5, Lemma 8] and a result for two-player stochastic parity games [21, Theorem 4.22]. The *and*-node and *or*-node cases have the product and coproduct, respectively, due to independence. We explain the *action node* case in more detail.

The sum over the probabilistic distribution follows from (5); we explain the internal non-deterministic choice. A PBS scheduler makes a choice for an action given the partial computation σ. Here, this choice is made based on a formula, ψ'_a, to be satisfied. When the graph construction succeeds, this is well defined: given L, s, and ψ, the scheduler can deduce ψ'_a from σ, a la traversal of the dependency graph. ◀

We note that, although a particular choice may maximize $\mathsf{Pr}_{L,s}(\psi')$, this does not mean that the corresponding capacity can be reached by a *memoryless*[3] scheduler, as a scheduler that makes this choice *every time* is not necessarily optimal. Indeed, no optimal scheduler may exist, in which case we would only have ϵ-optimal schedulers for any $\epsilon > 0$ [10, 21]. The capacity may be predicated on *eventually* making a different choice. The formulation in Lemma 19 is consistent with this possibility, and the existence of (ϵ-)optimal schedulers may be justified through a method called *strategy improvement* or *strategy stealing* [12, 21]. The intuition is that, in case of a loop, we can add a choice to succeed or fail immediately, succeeding with probability equal to the corresponding capacity. This does not increase the capacity, and the maximizing scheduler can otherwise be the same, if this choice does not arise.

▶ **Theorem 20** (Model checking termination). *The graph construction of* $\mathsf{Dg}(s, \psi)$ *terminates for any fuzzy formula* ψ *and (finite) PBS* L.

Proof. Letting $c = |Cl(\psi)|$, the number of distinct formulae not in factored form is bounded by 2^{2^c} (since we may check DNF versions for equivalence). The number of actions in L and ψ is finite, which bounds the number of action nodes. Thus, construction cannot continue indefinitely. ◀

4.2 Separability of Fuzzy Formulae

Now, we describe a class of fuzzy formulae that cannot get entangled, regardless of the PBS, which we denote as *separability*.

[3] We consider a scheduler memoryless if it makes a choice based on a formula to be satisfied, but does not know the history.

Recall that we used fixed-point expansion, partial evaluation, and grouping to get a factored form. As partial evaluation replaced non-probabilistic formulae with either tt or ff, we define a "worst-case" scenario.

▶ **Definition 21** (Purely probabilistic abstraction)**.** The *purely probabilistic abstraction* of a fuzzy formula ψ, denoted by $PPA(\psi)$, is a formula obtained by removing unguarded non-probabilistic subformulae (i.e., $\psi' \wedge \phi$, where ϕ is non-probabilistic, becomes ψ').

▶ **Definition 22** (Separability)**.** The set of all separable formulae is the largest set $\mathcal{S}$ such that $\forall \psi \in \mathcal{S}$, if $\psi' = GRP(PPA(FPE(\psi)))$, then

1. ψ' is not entangled, and
2. for each leaf $\langle a \rangle \psi_a$ of $AO(\psi')$, $\psi_a \in S$.

A formula ψ is *separable* if $\psi \in \mathcal{S}$.

As separable formulae preclude entanglement, graph construction is guaranteed to succeed, and we have the following result.

▶ **Corollary 23** (Model checking separable formulae)**.** *The graph construction of* $\mathsf{Dg}(s, \psi)$ *terminates* without failure *for any separable fuzzy formula* ψ *and (finite) PBS* L*.*

The decidable problems in Sect. 3 involve separable fuzzy formulae. In the case of LTL, there is only one action, in which case any formula in factored form is a single leaf. Though a CTL formula may be entangled, all PTTL fuzzy formulae are separable, as there is no explicit conjunction or disjunction; thus, we can model check PTTL over BMDPs. Finally, for 1-exit RMDP termination, the same separable formula, (8), is used for any 1-exit RMDP.

4.3 Solving the Polynomial System

For model checking separable formulae, we show how, given the graph, to compare the probabilistic value of ψ at a state s against a threshold p. We do this by first constructing a *system of polynomial max fixed point equations* from the graph. Each node i in the dependency graph is associated with a real-valued variable x_i. Given a set of variables V, each equation in the system is of the form $x_i = e$ where e is

- a polynomial over V such that the sum of coefficients is ≤ 1; or
- of the form $\max(V')$ where $V' \subseteq V$.

Furthermore, the equations form a stratified system, where each variable x_i can be assigned a stratum $j = \mathit{stratum}(x_i)$ such that x_i is defined in terms of only variables of the form x_k such that $\mathit{stratum}(x_k) \leq \mathit{stratum}(x_i)$ (cf. [19, Def. 9]); and variables in the same stratum j fall under the same fixed point. Let $\bowtie \in \{>, \geq, <, \leq\}$.

▶ **Theorem 24.** *Given a real value* p*, a system of polynomial max fixed point equations and a distinguished variable* x *defined in the system, whether or not* $x \bowtie p$ *in its solution is decidable.*

Proof. We write the polynomial system, $\mathbf{x} = P(\mathbf{x})$, as a sentence in the first-order theory of real closed fields, similar to [19]. The additional comparison will be $x_0 \bowtie p$. Along with the equation system, we need to encode fixed points and max.

We can encode $x_i = \max(x_j, x_k)$ as (12) (cf. [12, Section 5]):

$$x_i \geq x_j \wedge x_i \geq x_k \wedge (x_i \leq x_j \vee x_i \leq x_k) \ . \tag{12}$$

Meanwhile, letting V be the set of all variables and I a subset belonging to some stratum with least fixed point, we can encode the fixed point itself as (13):

$$\forall \mathbf{x}'_I. \left(\bigwedge_{i \in I} x'_i = P_i(\mathbf{x}'_I, \mathbf{x}_{V \setminus I}) \implies \bigwedge_{i \in I} x_i \leq x'_i \right) . \tag{13}$$

The stratification of fixed points in the equation system precludes a cyclical dependency between a least and a greatest fixed point; a greatest fixed point can be encoded similarly.

The original fixed point equation system, along with the query $x \bowtie p$, (12)-(13), and the counterpart encoding the greatest fixed point, are sentences in a first order theory of real closed fields, which is decidable [27]. Hence the decidability of $x \bowtie p$ in the solution to the fixed point equations follows. ◀

We use the above result to determine whether or not $\mathsf{Pr}_{L,s}(\psi) \bowtie p$ for a separable formula ψ. The polynomial fixed point system is derived similarly to [5, Section 4.1.2], with a variable $x_{(s,\psi)}$ for each node (s, ψ) in the dependency graph $\mathsf{Dg}(s, \psi)$, and equations based on Lemma 19.

- If ψ is not in factored form, then (s, ψ) has a unique edge labeled by ε to a node (s, ψ'), and $x_{(s,\psi)} = x_{(s,\psi')}$.
- $x_{(s,\mathsf{ff})} = 0$ and $x_{(s,\mathsf{tt})} = 1$.
- If (s, ψ) is an *and*-node, then $x_{(s,\psi)} = \prod_{((s,\psi),\varepsilon^\wedge,(s,\psi_i)) \in E} x_{(s,\psi_i)}$.
- If (s, ψ) is an *or*-node, then $x_{(s,\psi)} = \coprod_{((s,\psi),\varepsilon^\vee,(s,\psi_i)) \in E} x_{(s,\psi_i)}$.
- If (s, ψ) is an action node and $\psi = \langle a \rangle \psi_a$, then

$$x_{(s,\psi)} = \max_{c \in \mathbb{N}} \sum_{((s,\psi),a,(s',\psi_a)) \in E} P(s, a, s', c) \cdot x_{(s',\psi_a)}. \tag{14}$$

▶ **Theorem 25** (Correctness). *The construction of the dependency graph* $\mathsf{Dg}(s, \psi)$*, when* ψ *is separable, yields a polynomial max fixed point equation system, such that the value of* $x_{(s,\psi)}$ *in its solution is* $\mathsf{Pr}_{L,s}(\psi)$*.*

Proof. The correctness result follows from Lemma 19 and the semantics of fixed points given by (13) (and its counterpart). ◀

4.4 Model Checking Example

▶ **Example 26** (Model Checking). For the PBS L in Fig. 1 and fuzzy formula $\psi = \mu X.[a][b]X \wedge [a][c]X$, we have symmetric non-deterministic choices on b and c from state s_2, and the formula is satisfied by all finite d-trees (since both s_3 and s_4 have a probability greater than $\frac{1}{2}$ of returning to s_2, the infinite d-trees have positive measure on any scheduler). Letting $\psi_{bc} = [b]\psi \wedge [c]\psi$, we get the dependency graph shown in Fig. 4.

We find $\mathsf{Pr}_{L,s_1}(\psi)$ as the value of x_1^a in the least fixed point from the following equations:

$$\begin{array}{llll} x_1^a = x_2^{bc} & x_2^b = \max(x_3^a, x_4^a) & x_3^a = \frac{1}{3}x_5^{bc} + \frac{2}{3}x_2^{bc} & x_5^{bc} = 1 \\ x_2^{bc} = x_2^b \cdot x_2^c & x_2^c = \max(x_3^a, x_4^a) & x_4^a = \frac{1}{4}x_6^{bc} + \frac{3}{4}x_2^{bc} & x_6^{bc} = 1 \end{array} \tag{15}$$

Solving the equations, we get $\mathsf{Pr}_{L,s_1}(\psi) = x_1^a = \frac{1}{4}$. (Also, $\mathsf{Pr}_{L,s_1}(\mathsf{neg}(\psi)) = \frac{8}{9}$.)

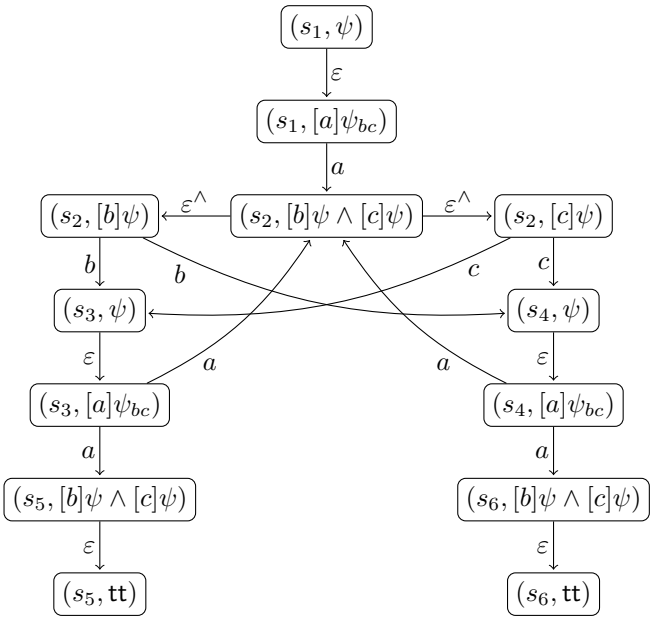


Figure 4 GPL Model Checking Example: Dependency Graph.

5 Discussion and Future Work

Many interesting questions relevant to GPL have been raised in other contexts. Also, while the syntax and semantics of GPL fuzzy formulae remains identical to [5] when over BPSs, the state formulae require a change in syntax; with the resulting logic called XPL in our related extended version [15] ([26] called its extension EGPL).

A probabilistic extension of μ-calculus closely related to separable GPL is Mio's pLμ [20, 21]. In contrast to GPL, the most expressive version of pLμ, denoted pL$\mu^{\odot}_{\oplus}$ [20, 21], defines *three* conjunction operators and their duals such that their probability values can be computed from the probabilities of the conjuncts. The logic pL$\mu^{\odot}$ is able to produce branching systems and supports an intuitive game semantics [21]. Along the same lines as our GPL encoding, we can encode termination of 1-exit RMDPs as model checking in pL$\mu^{\odot}$, and RMC termination in pL$\mu^{\odot}_{\oplus}$. However, pLμ cannot easily encode LTL and has no entanglement, so that attempting to encode multi-exit RMDP termination in pL$\mu^{\odot}_{\oplus}$ similarly to multi-exit RMC termination would lead to an incorrect, rather than undecidable, encoding. Other recent extensions of μ-calculus include the Lukasiewicz μ-calculus [22], μ^p-calculus [3], PμTL [17], and the quantitative μ-calculi, such as qMμ [18] and Qμ [13]. PμTL's expressiveness is orthogonal even with PCTL; the others are more similar to pLμ than GPL.

Although closely related, algorithms to check properties of RMCs (and pPDSs [7]) were developed independently [11]. These were related to algorithms for computing properties of systems such as branching process (BP) extinction and the language probability of Stochastic Context Free Grammars. The relationship between GPL and these systems was mentioned briefly in [16], but has remained largely unexplored. The results of this paper together with [16] also suggest that a separable GPL model checker can be encoded as a probabilistic logic program, even over systems with internal non-determinism.

There has been significant interest in the study of branching systems that also have non-deterministic choices, such as RMDPs and Branching MDPs (BMDPs) [12]. At the same time, the understanding of the polynomial systems has expanded. In [8], the class of

Probabilistic Polynomial Systems (PPS) is introduced, which characterizes when efficient solutions to polynomial equation systems are possible even in the worst case [9]. While [11] did not distinguish the systems arising from 1-exit RMCs from those from multi-exit RMCs, the PPS class is limited to 1-exit RMCs. It was also extended for 1-exit RMDP termination and, later, BMDP reachability, both having polynomial-time complexity for min/maxPPSs [8, 10].

We note that 1-exit RMDP termination, which always has optimal memoryless schedulers, can be matched with the class of least fixed point GPL formulae without disjunctions; meanwhile, BMDP reachability, for which only ϵ-optimal maximizing schedulers may exist, corresponds to least fixed point formulae without conjunctions. Additionally, these systems have been further extended into *stochastic games*, such as Recursive and Branching Simple Stochastic Games (RSSGs and BSSGs, respectively) [12], and we should be able to model these via separable GPL, as well, with a suitably extended BPS. Then, graph construction should be the same, but both min and max operators could appear in the equation system.

Polynomial systems equivalent to those arising from separable GPL over fPBSs have recently been considered in a more general setting in the context of *game automata* [19], followed by an undecidability result for more general properties on the automata [23]. Characterizing equation systems that arise from separable formulae and investigating their efficient solution is an interesting open problem. Another recent result suggests that the alternation-free restriction may be lifted [14]. Finally, this paper addressed the decidability of model checking; determining the complexity of model checking is a topic of future research.

References

1. Rajeev Alur, Swarat Chaudhuri, and P. Madhusudan. Software model checking using languages of nested trees. *ACM Trans. Program. Lang. Syst.*, 33(5):15:1–15:45, November 2011.
2. Christel Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.
3. Pablo Castro, Cecilia Kilmurray, and Nir Piterman. Tractable probabilistic mu-calculus that expresses probabilistic temporal logics. In *STACS*, volume 30 of *LIPIcs*, pages 211–223. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.
4. Taolue Chen, Klaus Dräger, and Stefan Kiefer. Model checking stochastic branching processes. In *MFCS*, pages 271–282, Berlin, Heidelberg, 2012. Springer.
5. Rance Cleaveland, S. Purushothaman Iyer, and Murali Narasimha. Probabilistic temporal logics via the modal mu-calculus. *Theoretical Computer Science*, 342(2-3):316–350, 2005.
6. Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Weak bisimulation is sound and complete for PCTL*. In *CONCUR*, volume 2421 of *LNCS*, pages 355–370. Springer Berlin Heidelberg, 2002.
7. Javier Esparza, Antonín Kucera, and Richard Mayr. Model checking probabilistic pushdown automata. In *LICS*, pages 12–21, 2004.
8. Kousha Etessami, Alistair Stewart, and Mihalis Yannakakis. Polynomial time algorithms for branching Markov decision processes and probabilistic min(max) polynomial Bellman equations. In *ICALP, Part I*, pages 314–326, Berlin, Heidelberg, 2012. Springer.
9. Kousha Etessami, Alistair Stewart, and Mihalis Yannakakis. Polynomial time algorithms for multi-type branching processes and stochastic context-free grammars. In *STOC*, pages 579–588. ACM, 2012.
10. Kousha Etessami, Alistair Stewart, and Mihalis Yannakakis. Greatest fixed points of probabilistic min/max polynomial equations, and reachability for branching Markov decision processes. In *ICALP, Part II*, pages 184–196, Berlin, Heidelberg, 2015. Springer.

11 Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, February 2009.

12 Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11:1–11:69, May 2015.

13 Diana Fischer, Erich Grädel, and Łukasz Kaiser. Model checking games for the quantitative μ-calculus. *Theory of Computing Systems*, 47(3):696–719, 2010.

14 Tomasz Gogacz, Henryk Michalewski, Matteo Mio, and Michał Skrzypczak. Measure properties of regular sets of trees. *Information and Computation*, 256:108 – 130, 2017. URL: `http://www.sciencedirect.com/science/article/pii/S0890540117300627`, `doi:https://doi.org/10.1016/j.ic.2017.04.012`.

15 Andrey Gorlin and C. R. Ramakrishnan. XPL: an extended probabilistic logic for probabilistic transition systems. *CoRR*, abs/1604.06118, 2016.

16 Andrey Gorlin, C. R. Ramakrishnan, and Scott A. Smolka. Model checking with probabilistic tabled logic programming. *TPLP*, 12(4-5):681–700, 2012.

17 Wanwei Liu, Lei Song, Ji Wang, and Lijun Zhang. A simple probabilistic extension of modal mu-calculus. In *IJCAI*, 2015.

18 Annabelle McIver and Carroll Morgan. Results on the quantitative μ-calculus qMμ. *ACM Trans. Comput. Logic*, 8(1), January 2007.

19 Henryk Michalewski and Matteo Mio. On the problem of computing the probability of regular sets of trees. In *FSTTCS*, volume 45 of *LIPIcs*, pages 489–502. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

20 Matteo Mio. Game semantics for probabilistic modal mu-calculi. PhD thesis, The University of Edinburgh, 2012.

21 Matteo Mio. Probabilistic modal μ-calculus with independent product. *Logical Methods in Computer Science*, Volume 8, Issue 4, November 2012. URL: `https://lmcs.episciences.org/789`, `doi:10.2168/LMCS-8(4:18)2012`.

22 Matteo Mio and Alex Simpson. Łukasiewicz mu-calculus. In *FICS*, volume 126 of *EPTCS*, pages 87–104, 2013.

23 Marcin Przybylko and Michal Skrzypczak. On the complexity of branching games with regular conditions. In *LIPIcs*, volume 58. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

24 Roberto Segala. A compositional trace-based semantics for probabilistic automata. In *CONCUR*, pages 234–248, 1995.

25 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.

26 Arvind Soni. Probabilistic and nondeterministic systems. Masters thesis, North Carolina State University, 2004.

27 Alfred Tarski. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society*, 59, 1951.

(Metric) Bisimulation Games and Real-Valued Modal Logics for Coalgebras

Barbara König
Universität Duisburg-Essen, Germany
barbara_koenig@uni-due.de

Christina Mika-Michalski
Universität Duisburg-Essen, Germany
christina.mika-michalski@uni-due.de

Abstract

Behavioural equivalences can be characterized via bisimulations, modal logics and spoiler-defender games. In this paper we review these three perspectives in a coalgebraic setting, which allows us to generalize from the particular branching type of a transition system. We are interested in qualitative notions (classical bisimulation) as well as quantitative notions (bisimulation metrics).

Our first contribution is to introduce a spoiler-defender bisimulation game for coalgebras in the classical case. Second, we introduce such games for the metric case and furthermore define a real-valued modal coalgebraic logic, from which we can derive the strategy of the spoiler. For this logic we show a quantitative version of the Hennessy-Milner theorem.

2012 ACM Subject Classification Theory of computation → Concurrency, Theory of computation → Logic and verification

Keywords and phrases coalgebra, bisimulation games, spoiler-defender games, behavioural metrics, modal logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.37

Related Version A full version of the paper is available at [23], `https://arxiv.org/abs/1705.10165`.

Funding Research partially supported by DFG Project BEMEGA.

Acknowledgements We thank Paul Wild, Lutz Schröder and Dirk Pattinson for inspiring discussions on fuzzy modal logic, and in particular on preservation of total boundedness.

1 Introduction

In the characterization of behavioural equivalences one encounters the following triad: First, such equivalences can be described via bisimulation relations, where the largest bisimulation (or bisimilarity) can be characterized as a greatest fixpoint. Second, a modal logic provides us with bisimulation-invariant formulas and the aim is to prove a Hennessy-Milner theorem which says that two states are behaviourally equivalent if and only if they satisfy the same formulas [21]. A third, complementary view is given by spoiler-defender games [32]. Such games are useful both for theoretical reasons, see for instance the role of games in the Van Benthem/Rosen theorem [26], or for didactical purposes, in particular for showing that two states are not behaviourally equivalent. The game starts with two tokens on two states and the spoiler tries to make a move that cannot be imitated by the defender. If the defender

is always able to match the move of the spoiler we can infer that the two initial states are behaviourally equivalent. If the states are not equivalent, a strategy for the spoiler can be derived from a distinguishing modal logic formula.

Such games are common for standard labelled transition systems, but have been studied for other types of transition systems only to a lesser extent. For probabilistic transition systems there are game characterizations in [15, 17], where the players can make moves to sets of states, rather than take a transition to a single state. Furthermore, in [11] a general theory of games is introduced in order to characterize process equivalences of the linear/branching time spectrum.

Our aim is to extend this triad of bisimulation, logics and games in two orthogonal dimensions. First, we work in the general framework of coalgebras [28], which allows to specify and uniformly reason about systems of different branching types (e.g. non-deterministic, probabilistic or weighted), parameterized over a functor. While behavioural equivalences [31] and modal logics [29, 27] have been extensively studied in this setting, there are almost no contributions when it comes to games. We are mainly aware of the work by Baltag [7], which describes a coalgebraic game based on the bisimulation relation, which differs from the games studied in this paper and is associated with another variant of logic, namely Moss' coalgebraic logics [25]. A variant of Baltag's game was used in [24] for terminal sequence induction via games. (There are more contributions on evaluation games which describe the evaluation of a modal formula on a transition system, see for instance [18].) Our contribution generalizes the games of [15] and allows us, given a new type of system characterized by a functor on the category **Set**, satisfying some mild conditions, to automatically derive the corresponding game. The second dimension in which we generalize is to move from a qualitative to a quantitative notion of behavioural equivalence. That is, we refrain from classifying systems as either equivalent or non-equivalent, which is often too strict, but rather measure their behavioural distance. This makes sense in probabilistic systems, systems with time or real-valued output. For instance, we might obtain the result, that the running times of two systems differ by 10 seconds, which might be acceptable in some scenarios (departure of a train), but inacceptable in others (delay of a vending machine). On the other hand, two states are behaviourally equivalent in the classical sense if and only if they have distance 0. Such notions are for instance useful in the area of conformance testing [22] and differential privacy [9].

Behavioural metrics have been studied in different variants, for instance in probabilistic settings [13, 14, 8] as well as in the setting of metric transition systems [12, 16], which are non-deterministic transition systems with quantitative information. The groundwork for the treatment of coalgebras in metric spaces was laid by Turi and Rutten [33]. We showed how to characterize behavioural metrics in coalgebras by studying various possibilities to lift functors from **Set** to the category of (pseudo-)metric spaces [5, 6]. Different from [33, 35] we do not assume that the coalgebra is given a priori in the category of pseudometric spaces, that is we have to first choose a lifting of the behaviour functor in order to specify the behavioural metric. Such liftings are not unique[1] and in particular we introduced in [5, 6] the Kantorovich and the Wasserstein liftings, which generalize well-known liftings for the probabilistic case and also capture the Hausdorff metric. Here we use the Kantorovich lifting, since this lifting integrates better with coalgebraic logic. Our results are parameterized over the lifting, in particular the behavioural metrics, the game and the logics are dependent on a set Γ of evaluation functions.

[1] In fact, consider the product bifunctor $F(X, Y) = X \times Y$, for which there are several liftings: we can e.g. use the maximum or the sum metric. While the maximum metric is canonically induced by the categorical product, the sum metric is also fairly natural.

In the metric setting it is natural to generalize from classical two-valued logics to real-valued modal logics and to state a corresponding Hennessy-Milner theorem that compares the behavioural distance of two states with the logical distance, i.e., the supremum of the differences of values, obtained by the evaluation of all formulas. Such a Hennessy-Milner theorem for probabilistic transition systems was shown in [14] and also studied in a coalgebraic setting [35, 34]. Similar results were obtained in [37] for fuzzy logics, on the way to proving a van Benthem theorem. Fuzzy logics were also studied in [30] in a general coalgebraic setting, but without stating a Hennessy-Milner theorem.

Here we present a real-valued coalgebraic modal logic and give a Hennessy-Milner theorem for the general coalgebraic setting as a new contribution. Our proof strategy follows the one for the probabilistic case in [35]. We need several concepts from real analysis, such as non-expansiveness and total boundedness in order to show that the behavioural distance (characterized via a fixpoint) and the logical distance coincide.

Furthermore we give a game characterization of this behavioural metric in a game where we aim to show that $d(x, y) \leq \varepsilon$, i.e., the behavioural distance of two states x, y is bounded by ε. Furthermore, we work out the strategies for the defender and spoiler: While the strategy of the defender is based on the knowledge of the behavioural metric, the strategy of the spoiler can be derived from a logical formula that distinguishes both states.

Again, work on games is scarce: [15] presents a game which characterizes behavioural distances, but pairs it with a classical logic.

The paper is organized as follows: we will first treat the classical case in Section 2, followed by the metric case in Section 3. The development in the metric case is more complex, but in several respects mimics the classical case. Hence, in order to emphasize the similarities, we will use the same structure within both sections: we start with foundations, followed by the introduction of modal logics and the proof of the Hennessy-Milner theorem. Then we will introduce the game with a proof of its soundness and completeness. Finally we will show how the strategy for the spoiler can be derived from a logical formula. In the end we wrap everything up in the conclusion (Section 4). The proofs can be found in the full version of this paper [23].

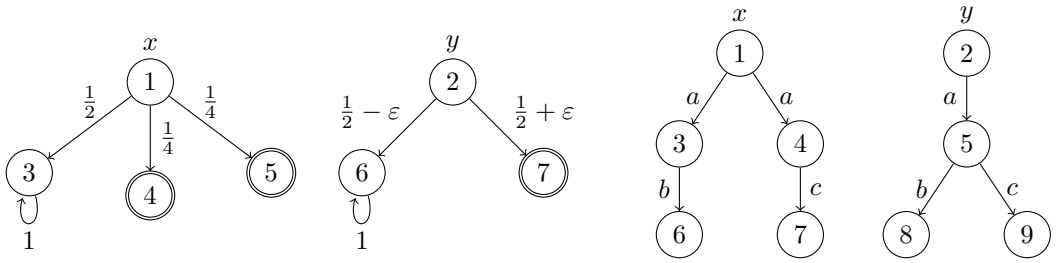
2 Logics and Games for the Classical Case

2.1 Foundations for the Classical Case

We fix an endofunctor $F\colon \mathbf{Set} \to \mathbf{Set}$, intuitively describing the branching type of the transition system under consideration. A *coalgebra*, describing a transition system of this branching type is given by a function $\alpha\colon X \to FX$ [28]. Two states $x, y \in X$ are considered to be *behaviourally equivalent* ($x \sim y$) if there exists a coalgebra homomorphism f from α to some coalgebra $\beta\colon Y \to FY$ (i.e., a function $f\colon X \to Y$ with $\beta \circ f = Ff \circ \alpha$) such that $f(x) = f(y)$.

▶ **Example 1.** We consider the (finitely or countably supported) probability distribution functor $\mathcal{D}$ with $\mathcal{D}X = \{p\colon X \to [0, 1] \mid \sum_{x \in X} p(x) = 1\}$ (where the p are either finitely or countable supported). Furthermore let $1 = \{\bullet\}$ be a singleton set.

Now coalgebras of the form $\alpha\colon X \to \mathcal{D}X + 1$ can be seen as probabilistic transition systems where each state x is either terminating ($\alpha(x) = \bullet$) or is associated with a probability distribution on its successor states. Note that one could easily integrate labels as well, but we will work with this version for simplicity.



(a) Probabilistic transition system for the functor $FX = \mathcal{D}X + 1$.

(b) Non-deterministic transition system for the functor $FX = \mathcal{P}(A \times X)$.

Figure 1

Figure 1a displays an example coalgebra (where $0 \leq \varepsilon \leq \frac{1}{2}$). Note that whenever $\varepsilon = 0$, we have $x \sim y$, since there is a coalgebra homomorphism from the entire state space into the right-hand side component of the transition system. If $\varepsilon > 0$ we have $1 \not\sim 2$.

Furthermore we need the lifting of a preorder under a functor F. For this we use the lifting introduced in [4] which guarantees that the lifted relation is again a preorder whenever F preserves weak pullbacks: Let $\leq$ be a preorder on Y, i.e. $\leq \subseteq Y \times Y$. We define the preorder $\leq^F \subseteq FY \times FY$ with $t_1, t_2 \in FY$ as follows: $t_1 \leq^F t_2$ whenever some $t \in F(\leq)$ exists such that $F\pi_i(t) = t_i$, where $\pi_i \colon \leq \to Y$ with $i \in \{1, 2\}$ are the usual projections.

▶ **Lemma 2.** *Let $(Y, \leq)$ be an ordered set and let $p_1, p_2 \colon X \to Y$ be two functions. Then $p_1 \leq p_2$ implies $Fp_1 \leq^F Fp_2$. (Both inequalities are to be read as pointwise ordering.)*

▶ **Example 3.** We are in particular interested in lifting the order $0 \leq 1$ on $2 = \{0, 1\}$. In the case of the distribution functor $\mathcal{D}$ we have $\mathcal{D}2 \cong [0, 1]$ and $\leq^{\mathcal{D}}$ corresponds to the order on the reals. For the powerset functor $\mathcal{P}$ we obtain the order $\{0\} \leq^{\mathcal{P}} \{0, 1\} \leq^{\mathcal{P}} \{1\}$ where $\emptyset$ is only related to itself.

2.2 Modal Logics for the Classical Case

We will first review coalgebraic modal logics, following mainly [27, 29], using slightly different, but equivalent notions. In particular we will introduce a logic where a predicate lifting is given by an evaluation map of the form $\lambda \colon F2 \to 2$, rather than by a natural transformation. In particular, each predicate $p \colon X \to 2$ is lifted to a predicate $\lambda \circ Fp \colon FX \to 2$. We do this to obtain a uniform presentation of the material. Of course, both views are equivalent, as spelled out in [29].

Given a cardinal κ and a set Λ of evaluation maps $\lambda \colon F2 \to 2$, we define a coalgebraic modal logic $\mathcal{L}^\kappa(\Lambda)$ via the grammar:

$$\varphi ::= \bigwedge \Phi \mid \neg\varphi \mid [\lambda]\varphi \quad \text{where } \Phi \subseteq \mathcal{L}^\kappa(\Lambda) \text{ with } \mathit{card}(\Phi) < \kappa \text{ and } \lambda \in \Lambda.$$

The last case describes the prefixing of a formula φ with a modality $[\lambda]$.

Given a coalgebra $\alpha \colon X \to FX$ and a formula φ, the semantics of such a formula is given by a map $[\![\varphi]\!]_\alpha \colon X \to 2$, where conjunction and negation are interpreted as usual and $[\![[\lambda]\varphi]\!]_\alpha = \lambda \circ F[\![\varphi]\!]_\alpha \circ \alpha$. For simplicity we will often write $[\![\varphi]\!]$ instead of $[\![\varphi]\!]_\alpha$. Furthermore for $x \in X$ we write $x \models \varphi$ whenever $[\![\varphi]\!]_\alpha(x) = 1$.

In [27] Pattinson has isolated the property of a separating set of predicate liftings to ensure that logical and behavioural equivalence coincide, i.e., the Hennessy-Milner property holds. It means that every $t \in FX$ is uniquely determined by the set $\{(\lambda, p) \mid \lambda \in \Lambda, p \colon X \to 2, \lambda(Fp(t)) = 1\}$.

▶ **Definition 4.** A set Λ of evaluation maps is separating for a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ whenever for all sets X and $t_1, t_2 \in FX$ with $t_1 \neq t_2$ there exists $\lambda \in \Lambda$ and $p\colon X \to 2$ such that $\lambda(Fp(t_1)) \neq \lambda(Fp(t_2))$.

The Hennessy-Milner theorem for coalgebraic modal logics can be stated as follows. The result has already been presented in [27, 29, 20]

▶ **Proposition 5** ([29]). *The logic $\mathcal{L}^\kappa(\Lambda)$ is sound, that is given a coalgebra $\alpha\colon X \to FX$ and $x, y \in X$, $x \sim y$ implies that $[\![\varphi]\!]_\alpha(x) = [\![\varphi]\!]_\alpha(y)$ for all formulas φ.*

Whenever F is κ-accessible[2] and Λ is separating for F, the logic is also expressive: whenever $[\![\varphi]\!]_\alpha(x) = [\![\varphi]\!]_\alpha(y)$ for all formulas φ we have that $x \sim y$.

In [29] it has been shown that a functor F has a separating set of predicate liftings iff $(Fp\colon FX \to F2)_{p\colon X \to 2}$ is jointly injective. We extend this characterization to monotone predicate liftings, respectively evaluation maps, i.e., order-preserving maps $\lambda\colon (F2, \leq^F) \to (2, \leq)$ where $\leq$ is the order $0 \leq 1$. This result will play a role in Section 2.3.

▶ **Proposition 6.** *F has a separating set of monotone evaluation maps iff $\leq^F$ is anti-symmetric and $(Fp\colon FX \to F2)_{p\colon X \to 2}$ is jointly injective.*

Note that an evaluation map is monotone if and only if its induced predicate lifting is monotone (see [23]).

2.3 Games for the Classical Case

We will now present the rules for the behavioural equivalence game. At the beginning of a game, there are two states x, y available for selection. The aim of the spoiler (S) is to prove that $x \not\sim y$, the defender (D) attempts to show the opposite.

- **Initial situation:** Given a coalgebra $\alpha\colon X \to FX$, we start with $x, y \in X$.
- **Step 1:** S chooses $s \in \{x, y\}$ and a predicate $p_1\colon X \to 2$.
- **Step 2:** D takes $t \in \{x, y\} \backslash \{s\}$ and has to answer with a predicate $p_2\colon X \to 2$, which satisfies $Fp_1(\alpha(s)) \leq^F Fp_2(\alpha(t))$.
- **Step 3:** S chooses p_i with $i \in \{1, 2\}$ and some state $x' \in X$ with $p_i(x') = 1$.
- **Step 4:** D chooses some state $y' \in X$ with $p_j(y') = 1$ where $i \neq j$.

After one round the game continues in Step 1 with states x' and y'. D wins the game if the game continues forever or if S has no move at Step 3. In the other cases, i.e. D has no move at Step 2 or Step 4, S wins.

In a sense such a game seems to mimic the evaluation of a modal formula, but note that the chosen predicates do not have to be bisimulation-invariant, as opposed to modal formulas.

▶ **Theorem 7.** *Assume that F preserves weak pullbacks and has a separating set of monotone evaluation maps. Then $x \sim y$ iff D has winning strategy for the initial situation (x, y).*

Part of the proof of Theorem 7 is to establish a winning strategy for D whenever $x \sim y$. We will quickly sketch this strategy: In Step 1 S plays p_1 which represents a set of states. One good strategy for D in Step 2 is to close this set under behavioural equivalence, i.e., to

[2] A functor $F\colon \mathbf{Set} \to \mathbf{Set}$ is κ-accessible if for all sets X and all $x \in FX$ there exists $Z \subseteq X$, $|Z| < \kappa$ such that $x \in FZ \subseteq FX$ [1]. (Note that we use the fact that **Set**-functors preserve injections $f\colon A \to B$ whenever $A \neq \emptyset$.) For details and a more categorical treatment see [2].

add all states which are behaviourally equivalent to a state in p_1, thus obtaining p_2. It can be shown that $Fp_1(\alpha(s)) \leq^F Fp_2(\alpha(t))$ always holds for this choice. Now, if S chooses x', p_1 in Step 3, D simply takes x' as well. On the other hand, if S chooses x', p_2, then either x' is already present in p_1 or a state y' with $x' \sim y'$. D simply chooses y' and the game continues.

▶ **Example 8.** Now consider an example coalgebra for the functor $FX = \mathcal{P}(A \times X)$, where $\mathcal{P}$ is the powerset functor (see Figure 1b). Obviously $x \not\sim y$, so S must have a winning strategy. Somewhat different from the usual bisimulation game, here the two players play subsets of the state space, instead of single states. Otherwise the game proceeds similarly.

Assume that S chooses $s = 1$ and defines a predicate p_1, which corresponds to the set $\{3\}$. Then $Fp_1(\alpha(s))$ is $\{(a, 0), (a, 1)\}$ (one a-successor of s – 3 – satisfies the predicate, the other – 4 – does not). Now D must take $t = 2$ and has to choose a predicate p_2 where at least $p_2(5) = 1$. In this case $Fp_2(\alpha(t))$ is $\{(a, 1)\}$ and $\{1\}$ is larger than $\{0, 1\}$ in the lifted order (see Example 3). However, now S can pick 5, which leaves only 3 to D.

In the next step, S can choose $s = 5$ and a predicate p_1, which corresponds to $\{9\}$. Hence $Fp_1(\alpha(s))$ is $\{(b, 0), (c, 1)\}$, but it is impossible for D to match this, since $(c, 1)$ is never contained in $Fp_2(\alpha(t))$ for $t = 3$.

We can see from this game why it is necessary to use an inequality $\leq^F$ instead of an equality. If there were no b, c-transitions (just a-transitions), $1 \sim 2$ would hold. And then, as explained above, D cannot match the move of S exactly, but only by choosing a larger value.

This game is inspired by the game for labelled Markov processes in [15] and the connection is explained in more detail in [23].

Note that in the probabilistic version of the game, it can again be easily seen that an inequality is necessary in Step 2: if, in the system in Figure 1a (where $\varepsilon = 0$), S chooses $s = 1$ and p_1 corresponds to $\{4\}$, then D can only answer by going to 7, which results in a strictly larger value. That is, we must allow D to do "more" than S.

Game variant

By looking at the proof of Theorem 7 it can be easily seen that the game works as well if we replace the condition $Fp_1(\alpha(s)) \leq^F Fp_2(\alpha(t))$ in Step 2 by $\lambda(Fp_1(\alpha(s))) \leq \lambda(Fp_2(\alpha(t)))$ for all $\lambda \in \Lambda$, provided that Λ is a separating set of monotone evaluation maps. This variant is in some ways less desirable, since we have to find such a set Λ (instead of simply requiring that it exists), on the other hand in this case the proof does not require weak pullback preservation, since we do not any more require transitivity of $\leq^F$. This variant of the game is conceptually quite close to the Λ-bisimulations studied in [19]. In our notation, a relation $S \subseteq X \times X$ is a Λ-bisimulation, if whenever $x\, S\, y$, then for all $\lambda \in \Lambda$, $p\colon X \to 2$, $\lambda(Fp(\alpha(x))) \leq \lambda(Fq(\alpha(y)))$, where q corresponds to the image of p under S (and the same condition holds for S^{-1}). Λ-bisimulation is sound and complete for behavioural equivalence if F admits a separating set of monotone predicate liftings, which coincides with our condition.

2.4 Spoiler Strategy for the Classical Case

In bisimulation games the winning strategy for D can be derived from the bisimulation or, in our case, from the map f that witnesses the behavioural equivalence of two states x, y (see the remark after Theorem 7). Here we will show that the winning strategy for S can be derived from a modal formula φ which distinguishes x, y, i.e., $x \models \varphi$ and $y \not\models \varphi$. We assume that all modalities are monotone (cf. Proposition 6).

The spoiler strategy is defined over the structure of φ:

- $\varphi = \bigwedge \Phi$: in this case S picks a formula $\psi \in \Phi$ with $y \not\models \psi$.
- $\varphi = \neg\psi$: in this case S takes ψ and reverses the roles of x, y.
- $\varphi = [\lambda]\psi$: in this case S chooses x and $p_1 = [\![\psi]\!]$ in Step 1. After D has chosen y and some predicate p_2 in Step 2, we can observe that $p_2 \not\leq [\![\psi]\!]$ (see proof of Theorem 9 in [23]). Now in Step 3 S chooses p_2 and a state y' with $p_2(y') = 1$ and $y' \not\models \psi$. Then D must choose $[\![\psi]\!]$ and a state x' with $x' \models \psi$ in Step 4 and the game continues with x', y' and ψ.

It can be shown that this strategy is successful for the spoiler.

▶ **Theorem 9.** *Assume that $\alpha\colon X \to FX$ is a coalgebra and F satisfies the requirements of Theorem 7. Let φ be a formula that contains only monotone modalities and let $x \models \varphi$ and $y \not\models \varphi$. Then the spoiler strategy described above is winning for S.*

3 Logics and Games for the Metric Case

We will now consider the metric version of behavioural equivalence, including logics and games. Analogous to Section 2 we will first introduce behavioural distance, which will in this case be defined via the Kantorovich lifting and is parameterized over a set Γ of evaluation maps. Then we introduce a modal logic inspired by [34] and show a quantitative coalgebraic analogue of the Hennessy-Milner theorem. This leads us to the definition of a game for the metric case, where we prove that the distance induced by the game coincides with the behavioural distance. We will conclude by explaining how the strategy for the spoiler can be derived from a logical formula distinguishing two states.

3.1 Foundations for the Metric Case

Note that this subsection contains several results which are new with respect to [6], in particular the extension of the Kantorovich lifting to several evaluation maps and Propositions 18, 19, 20, 21 and 24.

In the following we assume that $\top$ is an element of $\mathbb{R}_0$, it denotes the upper bound of our distances.

We first define the standard notions of pseudometric space and non-expansive functions.

▶ **Definition 10** (Pseudometric, pseudometric space)**.** Let X be a set and $d\colon X \times X \to [0, \top]$ a real-valued function, we call d a pseudometric if it satisfies

1. $d(x, x) = 0$ (d is a metric if in addition $d(x, y) = 0$ implies $x = y$.)
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$

for all $x, y, z \in X$. If d satisfies only Condition 1 and 3, it is a directed pseudometric.

A (directed) pseudometric space is a pair (X, d) where X is a set and d is a (directed) pseudometric on X.

▶ **Example 11.** We will consider the following (directed) metrics on $[0, \top]$: the Euclidean distance $d_e : [0, \top] \times [0, \top] \to [0, \top]$ with $d_e(a, b) = |a - b|$ and truncated subtraction with $d_\ominus(a, b) = a \ominus b = \max\{a - b, 0\}$. Note that $d_e(a, b) = \max\{d_\ominus(a, b), d_\ominus(b, a)\}$.

Maps between pseudometric spaces are given by non-expansive functions, which guarantee that mapping two elements either preserves or decreases their distance.

▶ **Definition 12** (Non-expansive function)**.** Let (X, d_X), (Y, d_Y) be pseudometric spaces. A function $f\colon X \to Y$ is called non-expansive if $d_X(x, y) \geq d_Y(f(x), f(y))$ for all $x, y \in X$. In this case we write $f\colon (X, d_X) \xrightarrow{1} (Y, d_Y)$. By **PMet** (**DPMet**) we denote the category of (directed) pseudometric spaces and non-expansive functions.

On some occasions we need to transform an arbitrary function into a non-expansive function, which can be done as follows.

▶ **Lemma 13.** *Let d be a pseudometric on X and let $f\colon X \to [0, \top]$ be any function. Then the function $h\colon (X, d) \to ([0, \top], d_e)$ defined via $h(z) = \sup\{f(u) - d(u, z) \mid u \in X\}$ is non-expansive and satisfies $f \leq h$.*

Analogously the function $g\colon (X, d) \to ([0, \top], d_e)$ defined via $g(z) = \inf\{f(u) + d(u, z) \mid u \in X\}$ is non-expansive and satisfies $g \leq f$.

We will now define the Kantorovich lifting for **Set**-functors, introduced in [5]. Given a functor F we lift it to a functor $\bar{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ such that $UF = \bar{F}U$, where U is the forgetful functor, discarding the pseudometric. The Kantorovich lifting is parameterized over a set Γ of evaluation maps $\gamma\colon F[0, \top] \to [0, \top]$, the analogue to the evaluation maps for modalities in the classical case. This is an extension of the lifting in [5] where we considered only a single evaluation map. The new version allows to capture additional examples, without going via the somewhat cumbersome multifunctor lifting described in [5].

▶ **Definition 14** (Kantorovich lifting)**.** Let F be an endofunctor on **Set** and let Γ be a set of evaluation maps $\gamma\colon F[0, \top] \to [0, \top]$. For every pseudometric space (X, d) the Kantorovich pseudometric on FX is the function $d^{\uparrow\Gamma}\colon FX \times FX \to [0, \top]$, where for $t_1, t_2 \in FX$:

$$d^{\uparrow\Gamma}(t_1, t_2) := \sup\{d_e(\gamma(Ff(t_1)), \gamma(Ff(t_2))) \mid f\colon (X, d) \xrightarrow{1} ([0, \top], d_e), \gamma \in \Gamma\}$$

We define $\bar{F}_\Gamma(X, d) = (FX, d^{\uparrow\Gamma})$ on objects, while $\bar{F}_\Gamma$ is the identity on arrows.

We will abbreviate $\tilde{F}_\gamma f = \gamma \circ Ff$. Note that $\tilde{F}_\gamma$ is a functor on the slice category $\mathbf{Set}/[0, \top]$, which lifts real-valued predicates $p\colon X \to [0, \top]$ to real-valued predicates $\tilde{F}_\gamma p\colon FX \to [0, \top]$.

It still has to be shown that $\bar{F}$ is well-defined. The proofs are a straighforward adaptation of the proofs in [5].

▶ **Lemma 15.** *The Kantorovich lifting for pseudometrics (Definition 14) is well-defined, in particular it preserves pseudometrics and maps non-expansive functions to non-expansive functions.*

As a sanity check we observe that all evaluation maps $\gamma \in \Gamma$ are non-expansive for the Kantorovich lifting of d_e. In fact, the Kantorovich lifting is the least lifting that makes the evaluation maps non-expansive. This also means that a non-expansive function $f\colon (X, d) \xrightarrow{1} ([0, 1], d_e)$ is always mapped to a non-expansive $\tilde{F}_\gamma f\colon (FX, d^{\uparrow\Gamma}) \xrightarrow{1} ([0, 1], d_e)$.

For the following definitions we need the supremum metric on functions.

▶ **Definition 16** (Supremum metric)**.** Let (Y, d) be a pseudometric space. Then the set of all functions $f\colon X \to Y$ is equipped with a pseudometric d^∞, the supremum metric, defined as $d^\infty(f, g) = \sup_{x \in X} d(f(x), g(x))$ for $f, g\colon X \to Y$.

We consider the following restrictions on evaluation maps respectively predicate liftings, which are needed in order to prove the results.

▶ **Definition 17** (Properties of evaluation maps)**.** Let $\gamma\colon F[0, \top] \to [0, \top]$ be an evaluation map for a functor $F\colon \mathit{Set} \to \mathit{Set}$.

- The predicate lifting $\tilde{F}_\gamma$ induced by γ is non-expansive wrt. the supremum metric whenever $d_e^\infty(\tilde{F}_\gamma f, \tilde{F}_\gamma g) \leq d_e^\infty(f,g)$ for all $f,g\colon X \to [0,\top]$ and the same holds if we replace d_e by $d_\ominus$.
- The predicate lifting $\tilde{F}_\gamma$ is contractive wrt. the supremum metric whenever $d_e^\infty(\tilde{F}_\gamma f, \tilde{F}_\gamma g) \leq c \cdot d_e^\infty(f,g)$ for some c with $0 < c < 1$.
- The predicate lifting $\tilde{F}_\gamma$ is ω-continuous, whenever for an ascending chain of functions f_i (with $f_i \leq f_{i+1}$) we have that $\tilde{F}_\gamma(\sup_{i<\omega} f_i) = \sup_{i<\omega}(\tilde{F}_\gamma f_i)$.

It can be shown that the first property is equivalent to a property of the lifted functor, called local non-expansiveness, studied in [33].

▶ **Proposition 18** (Local non-expansiveness). *Let Γ be a set of evaluation maps and let $\bar{F}$ be the Kantorovich lifting of a functor F via Γ. It holds that*

$$(d_Y^F)^\infty(\bar{F}f, \bar{F}g) \leq (d_Y)^\infty(f,g)$$

for all non-expansive functions $f,g\colon (X,d_X) \to (Y,d_Y)$ (where $\bar{F}(Y,d_Y) = (FY, d_Y^F)$) if and only if

$$d_e^\infty(\tilde{F}_\gamma f, \tilde{F}_\gamma g) \leq d_e^\infty(f,g)$$

for all non-expansive functions $f,g\colon (X,d_X) \to ([0,\top],d_e)$ and all $\gamma \in \Gamma$.

Assumption. *In the following we will always assume the first property in Definition 17 for every evaluation map γ, i.e., the predicate lifting $\tilde{F}_\gamma$ is non-expansive wrt. the supremum metric.*

Under this assumption it can be shown that the Kantorovich lifting itself is non-expansive (respectively contractive).

▶ **Proposition 19.** *Let Γ be a set of evaluation maps and let $d_1, d_2\colon X \times X \to [0,\top]$ be two pseudometrics. Then $d_e^\infty(d_1^{\uparrow\Gamma}, d_2^{\uparrow\Gamma}) \leq d_e^\infty(d_1,d_2)$, that is, the Kantorovich lifting of metrics is non-expansive for the supremum metric.*

If, in addition, every predicate lifting $\tilde{F}_\gamma$ for $\gamma \in \Gamma$ is contractive (cf. Definition 17), we have that $d_e^\infty(d_1^{\uparrow\Gamma}, d_2^{\uparrow\Gamma}) \leq c \cdot d_e^\infty(d_1,d_2)$ for some c with $0 < c < 1$, that is, the Kantorovich lifting of metrics is contractive.

We will now see that for the functors studied in this paper, we have evaluation maps that satisfy the required conditions.

▶ **Proposition 20.** *The following evaluation maps induce predicate liftings which are non-expansive wrt. the supremum metric and ω-continuous.*

- *The evaluation map $\gamma_\mathcal{P}$ for the (finite or general) powerset functor $\mathcal{P}$ with $\gamma\colon \mathcal{P}[0,\top] \to [0,\top]$ where $\gamma_\mathcal{P}(R) = \sup R$.*
- *The evaluation map $\gamma_\mathcal{D}$ for the (finitely or countably supported) probability distribution functor $\mathcal{D}$ (for its definition see Example 1) with $\gamma_\mathcal{D}\colon \mathcal{D}[0,1] \to [0,1]$ where $\gamma_\mathcal{D}(p) = \sum_{r\in[0,1]} r \cdot p(r)$. Note that $\gamma_\mathcal{D}$ corresponds to the expectation of the identity random variable.*
- *The evaluation map $\gamma_\mathcal{M}$ for the constant functor $\mathcal{M}X = [0,\top]$ with $\gamma_\mathcal{M}\colon [0,\top] \to [0,\top]$ and $\gamma_\mathcal{M}(r) = r$.*
- *The evaluation map $\gamma_\mathcal{S}$ for the constant functor $\mathcal{S}X = 1 = \{\bullet\}$ with $\gamma_\mathcal{S}\colon 1 \to [0,\top]$ and $\gamma_\mathcal{S}(r) = \top$.*

As shown in [5] the evaluation map $\gamma_{\mathcal{P}}$ induces the Hausdorff lifting[3] on metrics and the evaluation map $\gamma_{\mathcal{D}}$ the classical Kantorovich lifting[4] for probability distributions [36].

Contractivity can be typically obtained by using a predicate lifting which is non-expansive and multiplying with a discount factor $0 < c < 1$, for instance by using $\gamma_{\mathcal{P}}(R) = c \cdot \sup R$ in the first item of Proposition 20 above.

It can be shown that the properties of evaluation maps are preserved under various forms of composition.

▶ **Proposition 21** (Composition of evaluation maps). *The following constructions on evaluation maps preserve non-expansiveness for the supremum metric and ω-continuity for the induced predicate liftings. Let $\gamma_F \colon F[0,\top] \to [0,\top]$, $\gamma_G \colon G[0,\top] \to [0,\top]$ be evaluation maps for functors F, G.*

- *$\gamma \colon F[0,\top] \times G[0,\top] \to [0,\top]$ with $\gamma = \gamma_F \circ \pi_1$, as an evaluation map for $F \times G$.*
- *$\gamma \colon F[0,\top] + G[0,\top] \to [0,\top]$ with $\gamma(t) = \gamma_F(t)$ if $t \in F[0,\top]$ and $\gamma(t) = 0$ otherwise, as an evaluation map for $F + G$.*
- *$\gamma \colon FG[0,\top] \to [0,\top]$ with $\gamma = \gamma_F \circ F\gamma_G$, as an evaluation map for FG.*

Now we can define behavioural distance on a coalgebra, using the Kantorovich lifting. Note that the behavioural distance is parameterized over Γ, since, if we are given a coalgebra in **Set**, the notion of behaviour in the metric case is dependent on the chosen functor lifting.

▶ **Definition 22** (Behavioural distance). Let the coalgebra $\alpha \colon X \to FX$ and a set of evaluation maps Γ for F be given. We define the pseudometric $d_\alpha \colon X \times X \to [0,\top]$ as the smallest fixpoint of $d_\alpha = d_\alpha^{\uparrow\Gamma} \circ (\alpha \times \alpha)$.

Note that every lifting of metrics is necessarily monotone (since it turns the identity into a non-expansive function, cf. [5]). Since in addition the space of pseudometrics forms a complete lattice (where sup is taken pointwise), the smallest fixpoint exists by Knaster-Tarski.

It has been shown in [6] that whenever the Kantorovich lifting preserves metrics (which is the case for our examples) and the final chain converges, then d_α characterizes behavioural equivalence, i.e., $d_\alpha(x,y) = 0$ iff $x \sim y$.

▶ **Example 23.** Using the building blocks introduced above we consider the following coalgebras with their corresponding behavioural metrics, generalizing notions from the literature. In both cases we are interested in the smallest fixpoint.

- *Metric transition systems [12]:* $FX = [0,\top] \times \mathcal{P}X$ with two evaluation maps $\gamma_i \colon [0,\top] \times \mathcal{P}[0,\top] \to [0,\top]$, $i \in \{1,2\}$ with $\gamma_1(r,R) = r$, $\gamma_2(r,R) = \sup R$.
  This gives us the following fixpoint equation, where d^H is the Hausdorff lifting of a metric d. Let $\alpha(x) = (r_x, S_x)$, $\alpha(y) = (r_y, S_y)$, then

  $$d(x,y) = \max\{|r_x - r_y|, d^H(S_x, S_y)\}$$

- *Probabilistic transition systems:* $GX = \mathcal{D}X + 1$ with two evaluation maps $\bar{\gamma}_{\mathcal{D}}, \gamma_\bullet \colon \mathcal{D}[0,1] + 1 \to [0,1]$, $i \in \{1,2\}$ with $\bar{\gamma}_{\mathcal{D}}(p) = \gamma_{\mathcal{D}}(p)$, $\gamma_\bullet(p) = 0$ where $p \in \mathcal{D}[0,1]$, $\bar{\gamma}_{\mathcal{D}}(\bullet) = 0$, $\gamma_\bullet(\bullet) = 1$.

---

[3] Given a metric d on X, the Hausdorff lifting of d is the metric d^H with $d^H(X_1, X_2) = \max\{\sup_{x_1 \in X_1} \inf_{x_2 \in X_2} d(x_1,x_2), \sup_{x_2 \in X_2} \inf_{x_1 \in X_1} d(x_1,x_2)\}$ for $X_1, X_2 \subseteq X$.

[4] Given a metric d on X, the (probabilistic) Kantorovich lifting of d is the metric d^K with $d^K(p_1,p_2) = \sup\{|\sum_{x \in X} f(x) \cdot (p_1(x) - p_2(x))| \mid f \colon (X,d) \xrightarrow{1} ([0,1], d_e)\}$ where $p_1, p_2 \colon X \to [0,1]$ are probability distributions.

Table 1 Overview of the modal logic formulas, their semantics $[\![\varphi]\!]_\alpha$ and modal depths $md(\varphi)$.

φ:	$\top$	$[\gamma]\psi$	$\min(\psi, \psi')$	$\neg\psi$	$\psi \ominus q$
$[\![\varphi]\!]_\alpha$:	$\top$	$\gamma \circ F[\![\psi]\!]_\alpha \circ \alpha$	$\min\{[\![\psi]\!]_\alpha, [\![\psi']\!]_\alpha\}$	$\top - [\![\psi]\!]_\alpha$	$[\![\psi]\!]_\alpha \ominus q$
$md(\varphi)$:	0	$md(\psi)+1$	$\max\{md(\psi), md(\psi')\}$	$md(\psi)$	$md(\psi)$

This gives us the following fixpoint equation, where d^K is the (probabilistic) Kantorovich lifting of a metric d. Let $T = \{x \mid \alpha(x) = \bullet\}$ and let $p_x = \alpha(x) \neq \bullet$.

$$d(x,y) = \begin{cases} 1 & \text{if } x \in T, y \notin T \text{ or } x \notin T, y \in T \\ 0 & \text{if } x, y \in T \\ d^K(p_x, p_y) & \text{otherwise} \end{cases}$$

Some of the results on (real-valued) modal logics in Section 3.2 will require that the fixpoint iteration terminates in ω steps. This is related to the fact that the original Hennessy-Milner theorem requires finite branching.

▶ **Proposition 24.** *Let Γ be a set of evaluation maps and let $\alpha\colon X \to FX$ be a coalgebra. We define an ascending sequence of metrics $d_i\colon X \times X \to [0, \top]$ as follows: d_0 is the constant 0-function and $d_{i+1} = d_i^{\uparrow\Gamma} \circ \alpha \times \alpha$. Furthermore $d_\omega = \sup_{i<\omega} d_i$.*

- *If for all evaluation maps $\gamma \in \Gamma$ the induced predicate liftings are ω-continuous (see Definition 17) and F is ω-accessible, the fixpoint d_α equals d_ω.*
- *If for all evaluation maps $\gamma \in \Gamma$ the induced predicate liftings are contractive wrt. the supremum metric (see Definition 17), the fixpoint d_α equals d_ω.*

Hence, if we are working with the finite powerset functor or the finitely supported distribution functor, the first case applies, whereas in the case of contractiveness, these restrictions are unnecessary (compare this with the result of [33] which guarantees the existence of a final coalgebra for a class of locally contractive functors).

3.2 Modal Logics for the Metric Case

We now define a coalgebraic modal logic $\mathcal{M}(\Gamma)$, which is inspired by [35]. Assume also that Γ is a set of evaluation maps.

The formulas of the logic are defined together with their semantics $[\![\varphi]\!]_\alpha$ and their modal depth $md(\varphi)$ in Table 1. Given a coalgebra $\alpha\colon X \to FX$ and a formula φ, the semantics of such a formula is given by a real-valued predicate $[\![\varphi]\!]_\alpha\colon X \to [0, \top]$, defined inductively, where $\gamma \in \Gamma$, $q \in \mathbb{Q} \cap [0, \top]$. Again we will occasionally omit the subscript α.

Note that, given a state x and a logical formula φ, we do not just obtain a true value (true, false) dependent on whether x satisfies the formula or not. Instead we obtain a value in the interval $[0, 1]$ that measures the degree or weight according to which x satisfies φ.

▶ **Definition 25** (Logical distance)**.** Let $\alpha\colon X \to FX$ be a coalgebra and let $x, y \in X$. We define the logical distance of x, y as

$$d_\alpha^L(x,y) = \sup\{d_e([\![\varphi]\!]_\alpha(x), [\![\varphi]\!]_\alpha(y)) \mid \varphi \in \mathcal{M}(\Gamma)\}.$$

We also define the logical distance up to modal depth i.

$$d_i^L(x,y) = \sup\{d_e([\![\varphi]\!]_\alpha(x), [\![\varphi]\!]_\alpha(y)) \mid \varphi \in \mathcal{M}(\Gamma), md(\varphi) \leq i\}.$$

▶ **Example 26.** We are considering probabilistic transition systems with evaluation maps as defined in Example 23.

The formula $\varphi = [\bar{\gamma}_{\mathcal{D}}][\gamma_\bullet]\top$ distinguishes the states x, y in Figure 1a. The formula $\psi = [\gamma_\bullet]\top$ evaluates to a predicate $[\![\psi]\!]$ that assigns 1 to terminating states and 0 to non-terminating states. Now x makes a transition to a terminating state with probability $\frac{1}{2}$, which means that $[\![\varphi]\!](x) = \bar{\gamma}_{\mathcal{D}}(\mathcal{D}[\![\psi]\!](\alpha(x))) = \frac{1}{2}$. Similarly $[\![\varphi]\!](y) = \frac{1}{2} + \varepsilon$. Hence $d^L_\alpha(x, y) \geq d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) = \varepsilon$. (In fact, the logical distance equals ε.)

We will now show that the logical distance d^L_α and the behavioural distance d_α coincide, i.e. a quantitative version of the Hennessy-Milner theorem, by generalizing the proof from [35]. Note that in some respects we simplify wrt. [35] by not working in **Meas**, the category of measurable spaces, but in a discrete setting. On the other hand, we generalize by considering arbitrary **Set**-endofunctors.

▶ **Theorem 27.** *Let d_i be the sequence of pseudometrics from Proposition 24. Then:*
1. *For every $i \in \mathbb{N}_0$ $d^L_i \leq d_i$.*
2. *For every φ with $md(\varphi) \leq i$ we have non-expansiveness: $[\![\varphi]\!] \colon (X, d_i) \to ([0, \top], d_e)$.*
3. *$d^L_\alpha \leq d_\alpha$.*

Note that from Theorem 27 it also follows that for every formula φ the function $[\![\varphi]\!]$ is non-expansive. Non-expansiveness is analogous to bisimulation-invariance that holds for formulas in a classical logic. In particular, in the classical case if $x \sim y$, then $[\![\varphi]\!](x) = [\![\varphi]\!](y)$ for every φ, in other words $[\![\varphi]\!]$ is non-expansive for the discrete metric d.

The other inequality ($d^L_\alpha \geq d_\alpha$) is more difficult to prove: we will first show that each d_i is totally bounded and then show that each non-expansive function can be approximated at each pair of points by a modal formula. Since modal formulas are closed under min and max, this enables us to use a variant of a lemma from [3] to prove that the formulas form a dense subset of all non-expansive functions. In order to achieve the approximation, we need all operators of the logic.

We first have to recall some definitions from real-valued analysis.

▶ **Definition 28** (Total boundedness)**.** A pseudometric space (X, d) is totally bounded iff for every $\varepsilon > 0$ there exist finitely many elements $x_1, \ldots, x_n \in X$ such that $X = \bigcup_{i=1}^n \mathcal{B}_\varepsilon(x_i)$ where $\mathcal{B}_\varepsilon(x_i) = \{z \in X \mid d(z, x_i) \leq \varepsilon\}$ denotes the ε-ball around x_i.

Our first result is to show that the lifting preserves total boundedness, by adapting a proof from [37] from a specific functor to arbitrary functors.

▶ **Proposition 29.** *Let (X, d) be a totally bounded pseudometric space, then $(FX, d^{\uparrow\Gamma})$ is totally bounded as well.*

Using this result it can be shown that every pseudometric in the ascending chain from Proposition 24 (apart from d_ω) is totally bounded.

▶ **Proposition 30.** *Let d_i be the sequence of pseudometrics from Proposition 24. Then every (X, d_i) is a totally bounded pseudometric space.*

Since total boundedness is not preserved by taking a supremum, d_ω is not necessarily totally bounded and we can not iterate the argument. This is one of the reasons for requiring that the fixpoint is reached in ω steps in Theorem 32 below.

In the next step we show that the formulas are dense in the non-expansive functions.

▶ **Proposition 31.** *$\{[\![\varphi]\!] \colon X \to [0, 1] \mid md(\varphi) \leq i\}$ is dense (wrt. the supremum metric) in $\{f \colon (X, d^L_i) \xrightarrow{1} ([0, \top], d_e)\}$.*

Finally we can show under which conditions the inequality $d_\alpha \leq d_\alpha^L$ holds.

▶ **Theorem 32.** *If the fixpoint d_α is reached in ω steps, it holds that $d_\alpha \leq d_\alpha^L$.*

3.3 Games for the Metric Case

We will now present the two-player game characterizing the behavioural distance between two states. The roles of S and D are similar to those in the first game, where D wants to defend the statement that the distance of two states $x, y \in X$ in a coalgebra α is bounded by $\varepsilon \in [0, \top]$, i.e., $d_\alpha(x, y) \leq \varepsilon$. S wants to disprove this claim.

- **Initial situation:** Given a coalgebra $\alpha\colon X \to FX$, we start with (x, y, ε) where $x, y \in X$ and $\varepsilon \in [0, \top]$.
- **Step 1:** S chooses $s \in \{x, y\}$ and a real-valued predicate $p_1\colon X \to [0, \top]$.
- **Step 2:** D takes $t \in \{x, y\}\backslash\{s\}$ and has to answer with a predicate $p_2\colon X \to [0, \top]$, which satisfies $d_\ominus(\tilde{F}_\gamma p_1(\alpha(s)), \tilde{F}_\gamma p_2(\alpha(t))) \leq \varepsilon$ for all $\gamma \in \Gamma$.
- **Step 3:** S chooses p_i with $i \in \{1, 2\}$ and some state $x' \in X$.
- **Step 4:** D chooses some state $y' \in X$ with $p_i(x') \leq p_j(y')$ where $j \neq i$
- **Next round:** (x', y', ε') with $\varepsilon' = p_j(y') - p_i(x')$.

After one round the game continues with the initial step, but now D tries to show that $d_\alpha(x', y') \leq \varepsilon'$. D wins if the game continues forever. In the other case, e.g., D has no move at Step 2 or Step 4, S wins.

The game distance of two states is defined as follows.

▶ **Definition 33** (Game distance). Let $\alpha\colon X \to FX$ be a coalgebra and let $x, y \in X$. We define the game distance of x, y as

$$d_\alpha^G(x, y) = \inf\{\varepsilon \mid \text{D has a winning strategy for } (x, y, \varepsilon)\}.$$

We now prove that the behavioural distance and the game distance coincide. We first show that d_α^G is indeed a pseudometric.

▶ **Proposition 34.** *The game distance d_α^G is a pseudometric.*

Next we show that the game distance is always bounded by the behavioural distance.

▶ **Theorem 35.** *It holds that $d_\alpha^G \leq d_\alpha$.*

While the general proof of this theorem is given in [23], the strategy for D can be straightforwardly explained whenever X is finite. In particular we want to show that whenever $d_\alpha(x, y) \leq \varepsilon$, then D has a winnning strategy for (x, y, ε). Assume that S chooses $s \in \{x, y\}$ with $p_1\colon X \to [0, \top]$. In this case D chooses p_2 with $p_2(z) = \sup\{p_1(u) - d_\alpha(u, z) \mid u \in X\}$ in Step 2. From Lemma 13 we know that $p_1 \leq p_2$ and p_2 is non-expansive. It can be shown that this choice satisfies $d_\ominus(\tilde{F}_\gamma p_1(\alpha(s)), \tilde{F}_\gamma p_2(\alpha(t))) \leq \varepsilon$ for all $\gamma \in \Gamma$. Now S chooses i and $x' \in X$ in Step 3. Then either $i = 1$ and D can choose $y' = x'$ in Step 4 and the game continues with x', y' and $\varepsilon' = p_2(y') - p_1(x') \geq 0$. Or $i = 2$ and D can choose y' such that $p_1(y') - d_\alpha(y', x') = p_2(x')$ (the supremum is reached since X is finite). This means that $p_1(y') \geq p_2(x')$ and $\varepsilon' = p_1(y') - p_2(x') = d_\alpha(x', y')$. In both cases, the game can continue.

▶ **Example 36.** Imagine the initial game situation (x, y, ε) for our example in Figure 1a and S chooses x with predicate $p_1(4) = 1$ and zero for all remaining states. Now D follows the strategy above and plays a predicate p_2 with $p_2(4) = p_2(5) = p_2(7) = 1$ and zero for all other

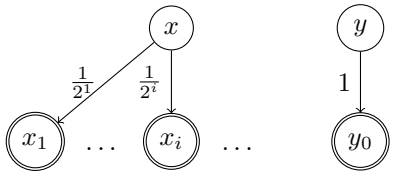


Figure 2 Probabilistic transition system for the functor $FX = \mathcal{D}X + 1$, where X is infinite.

states. Since $5, 7$ are at distance 0 to 4, they are now mapped to 1 as well. Since in particular 4 and 7 are mapped to 1, we obtain $d_{\ominus}(\tilde{\mathcal{D}}_{\bar{\gamma}_{\mathcal{D}}} p_1(\alpha(x)), \tilde{\mathcal{D}}_{\bar{\gamma}_{\mathcal{D}}} p_2(\alpha(y))) = d_{\ominus}(\frac{1}{4}, \frac{1}{2} + \varepsilon) = 0 \leq \varepsilon$ (we obtain the same value for $\gamma_\bullet$). Note again that D must be allowed to do "more" than S. Now the winning strategy for D is obvious: if S picks a terminating state x' and p_i, D can also pick a terminating state y' and p_j with $p_j(y') - p_i(x') = 0$ (similarly for non-terminating states). We then end up in $(x', y', 0)$ where x', y' are behaviourally equivalent.

If S had instead chosen y a prediate p_1 with $p_1(7) = 1$ and zero for all other states, D would choose the same predicate p_2 with $d_{\ominus}(\tilde{\mathcal{D}}_{\bar{\gamma}_{\mathcal{D}}} p_1(\alpha(y)), \tilde{\mathcal{D}}_{\bar{\gamma}_{\mathcal{D}}} p_2(\alpha(x))) = d_{\ominus}(\frac{1}{2} + \varepsilon, \frac{1}{2}) = \varepsilon$.

We now demonstrate that in the case of infinite branching, the construction of the winning strategy for the D is not as simple as described before.

▶ **Example 37.** Consider the coalgebra $\alpha\colon X \to \mathcal{D}X + 1$ in Figure 2 on the state space $X = \{y, y_0, x, x_1, x_2, \dots\}$, where the probability of going from x to x_i is $\alpha(x)(x_i) = \frac{1}{2^i}$.

For both states x, y the probability to terminate is 1 and hence $x \sim y$. Now imagine that S selects x and the real-valued predicate p_1 with $p_1(x_i) = 1 - \frac{1}{2^i}$ and $p_1(x) = 0$. If we would construct the predicate for D as above, via $p_2(z) = \sup\{p_1(u) - d_\alpha(u, z) \mid u \in X\}$, this would yield $p_2(y_0) = 1$ since the distance of all terminating states is 0.

Then S chooses $x' = y_0$ and p_2 in Step 3 and D has no available state y' with which to answer in Step 4. If $y' = x_i$, then $p_1(x_i) = 1 - \frac{1}{2^i} < 1 = p_2(x')$, otherwise $p_1(y') = 0 < 1$.

In fact, D has no winning strategy for $\varepsilon = 0$, but we can show that there is a winning strategy for every $\varepsilon > 0$ (since D can play a predicate that is below p_2, but at distance ε). Since the game distance is defined as the infinum over all such ε's it still holds that $d^G_\alpha(x, y) = 0$.

Finally, we can show the other inequality.

▶ **Theorem 38.** *It holds that* $d_\alpha \leq d^G_\alpha$.

3.4 Spoiler Strategy for the Metric Case

The strategy for S for (x, y, ε) can be derived from a modal formula φ with $d_{\ominus}([\![\varphi]\!](x), [\![\varphi]\!](y)) > \varepsilon$. If $\varepsilon < d_\alpha(x, y) = \sup\{d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) \mid \varphi\}$, such a formula must exist (since we can use negation to switch x, y if necessary). The spoiler strategy is defined over the structure of φ:

- $\varphi = \top$: this case can not occur.
- $\varphi = [\gamma]\psi$: S chooses x, $p_1 = [\![\psi]\!]$ at Step 1. After D has chosen y, p_2 at Step 2, we can observe that $p_2 \not\leq [\![\psi]\!]$ (see proof of Theorem 39 in [23]). Now in Step 3 S chooses p_2 and x' such $p_2(x') > [\![\psi]\!](x')$. Now D needs to choose y' such that $[\![\psi]\!](y') \geq p_2(x')$ in Step 4 and $\varepsilon' = [\![\psi]\!](y') - p_2(x') < [\![\psi]\!](y') - [\![\psi]\!](x') = d_e([\![\psi]\!](x'), [\![\psi]\!](y'))$ and so the game continues in the situation (x', y', ε') with the formula ψ.
- $\varphi = \min(\psi, \psi')$: In this case either $d_e([\![\psi]\!](x), [\![\psi]\!](y)) > \varepsilon$ or $d_e([\![\psi']\!](x), [\![\psi']\!](y)) > \varepsilon$ and S picks ψ or ψ' accordingly.
- $\varphi = \neg\psi$: In this case S takes ψ, since $d_e([\![\psi]\!](x), [\![\psi]\!](y)) = d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) > \varepsilon$.

- $\varphi = \psi \ominus q$: In this case $d_e([\![\psi]\!](x), [\![\psi]\!](y)) \geq d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) > \varepsilon$ and hence S takes ψ.

It can be shown that this strategy is indeed correct.

▶ **Theorem 39.** *Assume that $\alpha\colon X \to FX$ is a coalgebra. Let φ be a formula with $d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) > \varepsilon$. Then the spoiler strategy described above is winning for S in the situation (x, y, ε).*

Note that Theorem 38 is not a direct corollary of this theorem, since here we require that a formula φ with $d_e([\![\varphi]\!](x), [\![\varphi]\!](y)) > \varepsilon$ exists, which is not necessarily true in scenarios where the fixpoint iteration does not terminate in ω steps.

▶ **Example 40.** We will show how S can construct a winning strategy for $(x, y, \frac{\varepsilon}{2})$ based on the formula $\varphi = [\bar{\gamma}_{\mathcal{D}}][\gamma_\bullet]\top$ from Example 26. The transition system is shown in Figure 1a.

It holds that $d_\ominus([\![\varphi]\!](y), [\![\varphi]\!](x)) = \varepsilon > \frac{\varepsilon}{2}$. S plays y and $p_1 = [\![[\gamma_\bullet]\top]\!]$ which, due to the definition of $\gamma_\bullet$ equals 1 on terminating states and zero on non-terminating states. Now $\bar{\gamma}_{\mathcal{D}}(\mathcal{D}p_1(\alpha(y))) = \frac{1}{2} + \varepsilon$, so D must play in such a way that $\bar{\gamma}_{\mathcal{D}}(\mathcal{D}p_2(\alpha(x))) \geq \frac{1}{2} + \frac{\varepsilon}{2}$. This can only be achieved by setting $p_2(3) = \varepsilon$ (or to a larger value). Now S chooses p_2, $x' = 3$ and D can only take p_1 and either 4, 5 or 7 as y'. In each case we obtain $\varepsilon' = p_1(y') - p_2(x') = 1 - \varepsilon < 1 = d_e(0, 1) = d_e([\![[\gamma_\bullet]\top]\!](x'), [\![[\gamma_\bullet]\top]\!](y'))$.

The spoiler continues to follow his strategy and plays x', $p_1 = [\![\top]\!]$ in the next step, which is successful, since y' is a terminating state and x' is not.

4 Conclusion

Comparison to related work can be found in the introduction and throughout the text.

We will conclude by discussing some open points and questions: Section 3, which treats the metric cases, follows the outline of Section 2, which treats the classical case, with some variations. An important difference is the fact that the metric case is parameterized over a set Γ of evaluation maps. Note that we actually mimic the variant of the game discussed at the end of Section 2.3, where we fix evaluation maps, but omit the requirement of weak pullback preservation. The requirement of monotonicity is replaced by local non-expansiveness in the metric case. The fact that monotonicity for partial orders generalizes to non-expansiveness for directed metrics has already been discussed in [33]. The variant of the classical game that uses the lifted order $\leq^F$ is more reminiscent of the Wasserstein lifting for metrics, which has been introduced in [5] and compared to the Kantorovich lifting. It is future work to define a variant of the metric game that corresponds to the Wasserstein lifting (or other liftings) of metrics.

Another open question is to prove the Hennessy-Milner theorem for the real-valued logic in the case where the fixpoint is not reached in ω steps. The original variant of the Hennessy-Milner-theorem only holds for finitely-branching transition systems, but this result can be generalized if we allow infinite conjunctions (cf. the logic in Section 3.2). A natural question is whether the same solution is applicable to the metric case, by replacing the min- by an inf-operator (of restricted cardinality κ, as in Section 2.2). However, for this it seems necessary to generalize the notion of total boundedness to a new variant where we do not require that the set of "anchors" $\{x_1, \ldots, x_n\}$ of Definition 28 is finite, but bounded by κ.

A related question is the following: does the Kantorovich lifting preserve completeness of metrics? (A metric space (X, d) is complete if every Cauchy sequence converges in X.) Furthermore we would like to add ∞ as a possible distance value, as in [5]. However, this can not be integrated so easily, for instance it is unclear how to define negation.

Finally, in the quantitative case it could be interesting to know whether we can use existing efficient algorithms (for the probabilistic case), for instance in order to generate the strategy of the spoiler (see e.g. [10]).

References

1 J. Adámek, H.P. Gumm, and V. Trnková. Presentation of Set functors: A coalgebraic perspective. *Journal of Logic and Computation*, 20(5), 2010.

2 J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*, volume 189 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1994.

3 R.B. Ash. *Real Analysis and Probability*. Academic Press, 1972.

4 A. Balan and A. Kurz. Finitary functors: From **Set** to **Preord** and **Poset**. In *Proc. of CALCO '11*, pages 85–99. Springer, 2011. LNCS 6859.

5 P. Baldan, F. Bonchi, H. Kerstan, and B. König. Behavioral metrics via functor lifting. In *Proc. of FSTTCS '14*, volume 29 of *LIPIcs*, 2014.

6 P. Baldan, F. Bonchi, H. Kerstan, and B. König. Coalgebraic behavioral metrics. *Logical Methods in Computer Science*, to appear. Selected Papers of the 6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015).

7 A. Baltag. Truth-as-simulation: Towards a coalgebraic perspective on logic and games. Technical Report SEN-R9923, Centrum voor Wiskunde en Informatica (CWI), November 1999.

8 V. Castiglioni, D. Gebler, and S. Tini. Logical characterization of bisimulation metrics. In *Proc. of QAPL '16*, 2016. EPTCS 227.

9 K. Chatzikokolakis, D. Gebler, C. Palamidessi, and L. Xu. Generalized bisimulation metrics. In *Proc. of CONCUR '14*. Springer, 2014. LNCS/ARCoSS 8704.

10 D. Chen, F. van Breugel, and J. Worrell. On the complexity of computing probabilistic bisimilarity. In *Proc. of FOSSACS '12*, pages 437–451. Springer, 2012. LNCS/ARCoSS 7213.

11 X. Chen and Y. Deng. Game characterizations of process equivalences. In *Proc. of APLAS '08*, pages 107–121. Springer, 2008. LNCS 5356.

12 L. de Alfaro, M. Faella, and M. Stoelinga. Linear and branching system metrics. *IEEE Trans. Softw. Eng.*, 35(2):258–273, 2009.

13 J. Desharnais. *Labelled Markov processes*. PhD thesis, McGill University, Montreal, November 1999.

14 J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318:323–354, 2004.

15 J. Desharnais, F. Laviolette, and M. Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *Proc. of QEST '08*, pages 264–273. IEEE, 2008.

16 U. Fahrenberg, A. Legay, and C. Thrane. The quantitative linear-time–branching-time spectrum. In *Proc. of FSTTCS '11*, volume 13 of *LIPIcs*, pages 103–114, 2011.

17 N. Fijalkow, B. Klin, and P. Panangaden. Expressiveness of probabilistic modal logics. In *Proc. of ICALP '17*, volume 80 of *LIPIcs*, pages 105:1–12. Schloss Dagstuhl – Leibniz Center for Informatics, 2017.

18 G. Fontaine, R.A. Leal, and Y. Venema. Automata for coalgebras: An approach using predicate liftings. In *Proc. of ICALP '10*, pages 381–392. Springer, 2010. LNCS 6198.

19 D. Gorín and L. Schröder. Simulations and bisimulations for coalgebraic modal logics. In *Proc. of CALCO '13*, pages 253–266. Springer, 2013. LNCS 8089.

20 H. Peter Gumm. Universal coalgebras and their logics. *AJSE-Mathematics*, 34(1D):105–130, 2009.

21 M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Proc. of ICALP '80*, pages 299–309. Springer, 1980. LNCS 85.

22 N. Khakpour and M.R. Mousavi. Notions of conformance testing for cyber-physical systems: Overview and roadmap (invited paper). In *Proc. of CONCUR '15*, volume 42. LIPIcs, 2015.

23 B. König and C. Mika-Michalski. (Metric) bisimulation games and real-valued modal logics for coalgebras, 2018. arXiv:1705.10165. `arXiv:1705.10165`.

24 C. Kupke. Terminal sequence induction via games (international tbilisi symposium on language, logic and computation). In *Prof. of TbiLLC '07*, pages 257–271. Springer, 2009. LNAI 5422.

25 L.S. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96(1–3):277–317, 1999.

26 M. Otto. Elementary proof of the van Benthem-Rosen characterisation theorem. Technical Report 2342, Department of Mathematics, Technische Universität Darmstadt, 2004.

27 D. Pattinson. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1):177–193, 2003.

28 J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

29 L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoretical Computer Science*, 390(2):230–247, 2008.

30 L. Schröder and D. Pattinson. Description logics and fuzzy probability. In *Proc. of IJCAI '11*, volume 2, pages 1075–1080. AAAI Press, 2011.

31 S. Staton. Relating coalgebraic notions of bisimulation. In *Proc. of CALCO '09*, pages 191–205. Springer, 2009. LNCS 5728.

32 C. Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of the IGPL*, 7(1):103–124, 1999.

33 D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8:481–540, 1998.

34 F. van Breugel and J. Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theoretical Computer Science*, 360:373–385, 2005.

35 F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331(1):115–142, 2005.

36 C. Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.

37 P. Wild, L. Schröder, D. Pattinson, and B. König. A van Benthem theorem for fuzzy modal logic. In *Proc. of LICS '18*, 2018. to appear. `arXiv:1802.00478`.

The Complexity of Rational Synthesis for Concurrent Games

Rodica Condurache
Computer Science Department, "A.I.Cuza" University, Iaşi
700483, ROMANIA
rodica.b.condurache@gmail.com

Youssouf Oualhadj
Université Paris Est Créteil, LACL(EA 4219), UPEC
94010 Créteil Cedex, France
youssouf.oualhadj@lacl.fr

Nicolas Troquard
The KRDB Research Centre, Free University of Bozen-Bolzano
I-39100 Bozen-Bolzano BZ, Italy
nicolas.troquard@unibz.it

Abstract

In this paper, we investigate the rational synthesis problem for concurrent game structures for a variety of objectives ranging from reachability to Muller condition. We propose a new algorithm that establishes the decidability of the non cooperative rational synthesis problem that relies solely on game theoretic techniques as opposed to previous approaches that are logic based. Given an instance of the rational synthesis problem, we construct a zero-sum turn-based game that can be adapted to each one of the class of objectives. We obtain new complexity results. In particular, we show that in the cases of reachability, safety, Büchi, and co-Büchi objectives the problem is in PSpace, providing a tight upper-bound to the PSpace-hardness already established for turn-based games. In the case of Muller objective the problem is in ExpTime. We also obtain positive results when we assume a fixed number of agents, in which case the problem falls into PTime for reachability, safety, Büchi, and co-Büchi objectives.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Solution concepts in game theory

Keywords and phrases Synthesis, concurrent games, Nash equilibria

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1707.06936`.

Funding The second author is supported by the *RFSI DIM McSYS* project.

Acknowledgements The first author did part of this work when she was member of Université Paris Est Créteil, LACL, UPEC.

1 Introduction

The synthesis problem aims at automatically designing a program from a given specification. Several applications for this formal problem can be found in the design of *interactive systems* i.e., systems interacting with an environment. From a formal point of view, the synthesis

problem is traditionally modelled as a zero-sum turn-based game. The system and the environment are modeled by two players with opposite interest. The goal of the system is the desired specification. Hence, a *strategy* that allows the system to achieve its goal against any behavior of the environment is a winning strategy and is exactly the program to synthesize.

For a time, the described approach was the standard in the realm of controller synthesis. However, due to the variety of systems to model, such a pessimistic view is not always the most faithful one. For instance, consider a system that consists of a server and n clients. Assuming that all the agents have opposite interests is not a realistic assumption. Indeed, from a design perspective, the purpose of the server is to handle the incoming requests. On the other hand, each client is only concerned with its own request and wants it granted. None of the agents involved in the described interaction have antagonistic purposes. The setting of *non-zero-sum games* was proposed as model with more realistic assumptions.

In a non zero-sum game, each agent is equipped with a personal objective and the system is just a regular agent in the game. The agents interact together aiming at achieving the *best outcome*. The best outcome in this setting is often formalized by the concept of Nash equilibria. Unfortunately, a solution in this setting offers no guarantee that a specification for a given agent is achieved, and in a synthesis context one wants to enforce a specification for one or a subset of the agents.

The *rational synthesis problem* was introduced as a generalization of the synthesis problem to environment with multiple agents [4]. It aims at synthesizing a Nash equilibrium such that the induced behavior satisfies a given specification. This vision enjoys nice algorithmic properties since it matches the complexity bound of the classical synthesis problem. Later on, yet another version of the problem was proposed where the agents are rational but not cooperative [6, 7]. In the former formalization, the specification is guaranteed as long as the agents agree to behave according to the chosen equilibrium. But anything can happen if not, in particular they can play another equilibrium that does not satisfy the specification. In the Non Cooperative Rational Synthesis (NCRSP), the system has to ensure that the specification holds in any equilibrium (c.f., Section 3 for a formal definition and Figure 1a for an example). A solution for both problems was presented for specifications expressed in Linear Temporal Logic (LTL). The proposed solution relies on the fact that the problem can be expressed in a decidable fragment of a logic called *Strategy Logic*. The presented algorithm runs in 2-ExpTime. While expressing the problem in a decidable fragment of Strategy Logic gives an immediate solution, it could also hide a great deal of structural properties. Such properties could be exploited in a hope of designing faster algorithms for less expressive objectives. In particular, specifications such as reachability, liveness, fairness, *etc.*

In [3], the first author took part in a piece of work where they considered this very problem for specific objectives such as reachability, safety, Büchi, *etc* in a turn-based interaction model. They established complexity bounds for each objective.

In this paper we consider the problem of non-cooperative rational synthesis with concurrent interactions. We address this problem for a variety of objectives and give exact complexity bounds relying exclusively on techniques inspired by the theory of zero-sum games. The concurrency between agents raises a formal challenge to overcome as the techniques used in [3] do not directly extend. Intuitively, when the interaction is turn-based, one can construct a tree automaton that accepts solutions for the rational synthesis problem. The nodes of an accepted tree are exactly the vertices of the game. This helps a lot in dealing with deviations but cannot be used in concurrent games.

In Section 3, we present an alternative algorithm that solves the general problem for LTL specification. This algorithm constructs a zero-sum turn-based game. This fresh game is played between *Constructor* who tries to construct a solution and *Spoiler* who tries to falsify

the proposed solution. We then show in Section 5 how to use this algorithm to solve the NCRSP for reachability, safety, Büchi, co-Büchi, and Muller conditions. We also observe that we match the complexity results for the NCRSP in turn-based games.

2 Preliminaries

2.1 Concurrent Game Structures

▶ **Definition 1.** A game structure is defined as a tuple $\mathcal{G} = (\mathsf{St}, s_0, \mathsf{Agt}, (\mathsf{Act}_i)_{i\in\mathsf{Agt}}, \mathsf{Tab})$, where St is the set of states in the game, s_0 is the initial state, $\mathsf{Agt} = \{0, 1, ..., n\}$ is the set of agents, Act_i is the set of actions of Agent i, $\mathsf{Tab} : \mathsf{St} \times \prod_{i\in\mathsf{Agt}} \mathsf{Act}_i \to \mathsf{St}$ is the transition table.

▶ Remark. Note that we consider game structures that are complete and deterministic. That is, from each state s and any tuple of actions $\bar{a} \in \prod_{i\in\mathsf{Agt}} \mathsf{Act}_i$, there is exactly one successor state s'.

▶ **Definition 2.** A *play* in the game structure is a sequence of states and actions profile $\rho = s_0\bar{a}_0s_1\bar{a}_1s_2\bar{a}_2...$ in $(\mathsf{St}\ \prod_{i\in\mathsf{Agt}} \mathsf{Act}_i)^\omega$ such that s_0 is the initial state and for all $j \geq 0$, $s_{j+1} = \mathsf{Tab}(s_j, \bar{a}_j)$.

Throughout the paper, for every word w, over any alphabet, we denote by $w[j]$ the $j+1$-th letter, and we denote by $w[0..j]$ the prefix of w of size $j+1$.

By $\rho \restriction_{\mathsf{St}}$ we mean the projection of ρ over St, and $\mathsf{Plays}(\mathcal{G})$ is the set of all the plays in the game structure $\mathcal{G}$. We call history any finite sequence in $\mathsf{St}\ (\prod_{i\in\mathsf{Agt}} \mathsf{Act}_i\ \mathsf{St})^*$. For a history h, we denote by $h \restriction_{\mathsf{St}}$ its projection over St, and by $\mathsf{Last}_{\mathsf{St}}(h)$ the last element of $h \restriction_{\mathsf{St}}$. We denote by Hist the set of all the histories.

In this paper we allow agents to see the actions played between states. Therefore, they behave depending on the past sequence of states and tuples of actions.

▶ **Definition 3** (Strategy and strategy profile)**.** A *strategy* for Agent i is a mapping $\sigma_i : \mathsf{St}\left(\prod_{i\in\mathsf{Agt}} \mathsf{Act}_i\ \mathsf{St}\right)^* \to \mathsf{Act}_i$.

A *strategy profile* is defined as a tuple of strategies $\bar{\sigma} = \langle\sigma_0, \sigma_1, ..., \sigma_n\rangle$ and by $\bar{\sigma}[i]$ we denote the strategy of i-th position (of Agent i).

Also, $\bar{\sigma}_{-i}$ is the partial strategy profile obtained from the strategy profile $\bar{\sigma}$ from which the strategy of Agent i is ignored. The tuple of strategies $\langle\bar{\sigma}_{-i}, \sigma'_i\rangle$ is obtained from the tuple $\bar{\sigma}$ by substituting Agent i's strategy with σ'_i.

Once a strategy profile is chosen it induces a play ρ. We say that a play $\rho = s_0\bar{a}_0s_1\bar{a}_1s_2\bar{a}_2...$ in $(\mathsf{St}\ \prod_{i\in\mathsf{Agt}} \mathsf{Act}_i)^\omega$ is *compatible* with a strategy σ_i of Agent i if for every prefix of $\rho[0..2k]$ with $k \geq 0$, we have $\sigma(\rho[0..2k]) = \bar{a}_k(i)$, where $\bar{a}_k(i)$ is the action of Agent i in the vector $\bar{a}_k$.

We denote by $\mathsf{Plays}(\sigma_i)$ the set of all the plays that are compatible with the strategy σ_i for Agent i. $\mathsf{Hist}(\sigma_i)$ is the set of all the histories that are compatible with σ_i. The outcome of an interaction between agents following a certain strategy profile $\bar{\sigma}$ defines a unique play in the game structure $\mathcal{G}$ denoted $\mathsf{Out}(\bar{\sigma})$. It is the unique play in $\mathcal{G}$ compatible with all the strategies in the profile $\bar{\sigma}$ which is an infinite sequence over $(\mathsf{St}\ \prod_{i\in\mathsf{Agt}} \mathsf{Act}_i)$.

2.2 Payoff and Solution Concepts

Each Agent $i \in \mathsf{Agt}$ has an objective expressed as a set Obj_i of infinite sequences of states in $\mathcal{G}$. As defined before, a play ρ is a sequence of states and action profiles. We slightly abuse notation and also write $\rho \in \mathsf{Obj}_i$, meaning that the sequence of states in the play ρ (that

is, $\rho \restriction_{\mathsf{St}}$) is in Obj_i. We define the *payoff* function that associates with each play ρ a vector $\mathsf{Payoff}(\rho) \in \{0,1\}^{n+1}$ defined by

$$\forall i \in \mathsf{Agt}, \mathsf{Payoff}_i(\rho) = 1 \iff \rho \in \mathsf{Obj}_i \ .$$

We borrow game theoretic vocabulary and say that Agent i *wins* whenever her payoff is 1. We sometimes abuse this notation and write $\mathsf{Payoff}_i(\bar{\sigma})$, which is the payoff of Agent i associated with the *unique* play induced by $\bar{\sigma}$.

In this paper we are interested in winning objectives such as Safety, Reachability, Büchi, coBüchi, and Muller that are defined as follows. Let ρ be a play in a concurrent game structure $\mathcal{G}$. We use the following notations:

$$occ(\rho) = \{s \in \mathsf{St} \mid \exists j \geq 0 \text{ s.t. } \rho[j] = s\}$$

to denote the set of states that appear along ρ and

$$\inf(\rho) = \{s \in \mathsf{St} \mid \forall j \geq 0, \exists k \geq j \text{ s.t. } \rho[k] = s\}$$

to denote the set of states appearing infinitely often along ρ. Then,

- *Reachability:* For some $T \subseteq \mathsf{St}$, $\textsc{Reach}(T) = \{\rho \in \mathsf{St}^\omega \mid occ(\rho) \cap T \neq \emptyset\}$;
- *Safety:*For some $T \subseteq \mathsf{St}$, $\textsc{Safe}(T) = \{\rho \in \mathsf{St}^\omega \mid occ(\rho) \subseteq T\}$;
- *Büchi:* For some $T \subseteq \mathsf{St}$, $\textsc{Büchi}(T) = \{\rho \in \mathsf{St}^\omega \mid \inf(\rho) \cap T \neq \emptyset\}$;
- *coBüchi:* For some $T \subseteq \mathsf{St}$, $\textsc{coBüchi}(T) = \{\rho \in \mathsf{St}^\omega \mid \inf(\rho) \cap T = \emptyset\}$;
- *Parity:* For some priority function $p : \mathsf{St} \to \mathbb{N}$, $\textsc{Parity}(p) = \{\rho \in \mathsf{St}^\omega \mid \min\{p(s) \mid s \in \inf(\rho)\} \text{ is even }\}$;
- *Muller:* For some boolean formula μ over St, $\textsc{Muller}(\mu) = \{\rho \in \mathsf{St}^\omega \mid \inf(\rho) \models \mu\}$.

A Nash equilibrium is the formalisation of a situation where no agent can improve her payoff by unilaterally changing her behaviour. Formally:

▶ **Definition 4.** (Nash equilibrium) A strategy profile $\bar{\sigma}$ is a Nash equilibrium (NE) if for every agent i and every strategy σ' of i the following holds true:

$$\mathsf{Payoff}_i(\bar{\sigma}) \geq \mathsf{Payoff}_i(\langle \bar{\sigma}_{-i}, \sigma' \rangle) \ .$$

Throughout this paper, we will assume that Agent 0 is the agent for whom we wish to synthesize the strategy, therefore, we use the concept of 0-fixed Nash equilibria.

▶ **Definition 5** (0-fixed Nash equilibrium)**.** A profile $\langle \sigma_0, \bar{\sigma}_{-0} \rangle$ is a 0-fixed NE (0-NE), if for every strategy σ' for agent i in $\mathsf{Agt} \setminus \{0\}$ the following holds true:

$$\mathsf{Payoff}_i(\langle \sigma_0, \bar{\sigma}_{-0} \rangle) \geq \mathsf{Payoff}_i(\langle \sigma_0, (\bar{\sigma}_{-0})_{-i}, \sigma' \rangle) \ .$$

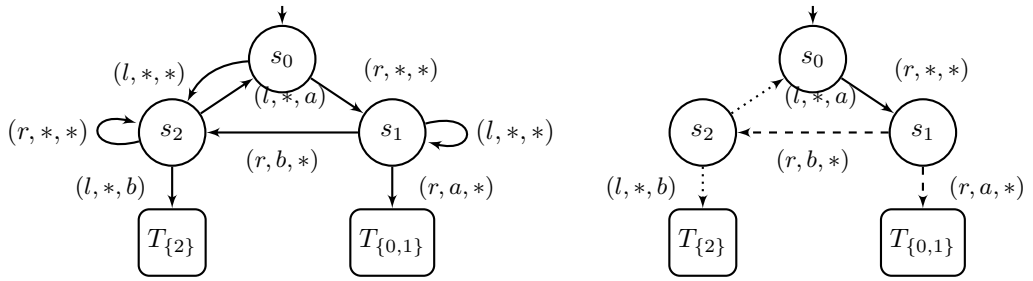
That is, fixing σ_0 for Agent 0, the other agents cannot improve their payoff by unilaterally changing their strategy.

2.3 Rational synthesis

The rational synthesis can be defined in a optimistic or pessimistic setting. The former one is the so-called Cooperative Rational Synthesis (CRSP) Formally defined as

▶ **Problem 6.** *Is there a 0-NE $\bar{\sigma}$ such that* $\mathsf{Payoff}_0(\bar{\sigma}) = 1$*?*

The latter is the so-called Non Cooperative Rational Synthesis Problem (NCRSP) and is formally defined as



(a) A concurrent game. **(b)** Subgame induced from the strategy σ_0.

Figure 1

▶ **Problem 7.** *Is there a strategy σ_0 for Agent 0 such that for every 0-NE $\bar{\sigma} = \langle \sigma_0, \bar{\sigma}_{-0} \rangle$, we have* $\mathsf{Payoff}_0(\bar{\sigma}) = 1$*?*

In this paper we study computational complexity for the rational synthesis problem in both cooperative and non-cooperative settings.

For the CRSP, the complexity results are corollaries of existing work. In particular, for Safety, Reachability, Büchi, co-Büchi, Rabin and Muller objectives, we can apply algorithms from [2] to obtain the same complexities for CRSP as for the turn-based models when the number of agents is not fixed. More precisely, in [2] the problem of finding NE in concurrent games is tackled. In this problem one asks for the existence of NE whose payoff is between two thresholds. Then, by choosing the lower thresholds to be such that only Agent 0 satisfies her objective and the upper thresholds such that all agents win, we reduce to the cooperative rational synthesis problem. Brenguier et al. [2] showed that the existence of constrained NE in concurrent games can be solved in PTime for Büchi objectives, NP for Safety, Reachability and coBüchi objectives, and PSpace for Muller objectives. All hardness results are inferred directly from the hardness results in the turn-based setting. This is a consequence of the fact that every turn-based game can be encoded as a concurrent game by allowing at each state at most one agent to have non-vacuous choices. For Streett objectives, by reducing to [2] we only obtain PSpace-easiness and the NP-hardness comes from the turn-based setting [3].

In the case of non-cooperative rational synthesis, we cannot directly apply the existing results. However, we define an algorithm inspired from the *suspect games* [2]. The suspect game was introduced to decide the existence of pure NE in concurrent games with ω-regular objectives. We inspire ourselves from that approach and design a zero-sum game that combines the behaviors of Agent 0 and an extra entity whose goal is to prove, when needed, that the current play is not the outcome of a 0-NE. We also extend the idea in [3] that consists roughly in keeping track of deviations. Recall that the non-cooperative rational synthesis problem consists in designing a strategy σ_0 for the protagonist (Agent 0 in our case) such that her objective Obj_0 is satisfied by all the plays that are outcomes of 0-NE compatible with σ_0. This is equivalent to finding a strategy σ_0 for Agent 0 such that for any play ρ compatible with it, either ρ satisfies Obj_0, or there is no strategy profile $\bar{\sigma} = \langle \sigma_0, \bar{\sigma}_{-0} \rangle$ that is a 0-NE whose outcome is ρ.

▶ **Example 8.** Consider the concurrent game with reachability objectives depicted in Figure 1a. The game starts in the state s_0. There are three agents, the controller Agent 0, Agent 1, and Agent 2. Agent 0 has two actions r for right and l for left. Agents 1 and 2 have two actions, denoted a and b. For any subset C of $\{0, 1, 2\}$, the states T_C indicate that the agents in C have reached their objectives (These states are sinks). In addition, there are three states s_0,

s_1, and s_2. The edges represent the transitions table. The labels indicate the action profiles e.g. the vector (r, a, b) means that Agent 0 took action r, Agent 1 took action a, and Agent 2 took action b. Finally action $*$ stands for the indifferent choice that is any action for a given agent. We can see that at s_0, Agent 0 is the only agent with non-vacuous choices. He can choose to go to s_1 by playing action r, or to go to s_2 by playing action l.

Now consider the strategy σ_0 for Agent 0 defined as follows: $\sigma_0(s_0) = r, \sigma_0(s_1) = r, \sigma_0(s_2) = l$ We argue that this strategy is a solution to the NCRSP. Indeed, by applying this strategy, we obtain the subgame of Figure 1b. In this game, all the plays falsifying the objective of Agent 0 are the ones where Agent 1 plays b. Notice now that these plays are not outcomes of a 0-NE since Agent 1 can deviate by playing action a.

3 Solution for Problem 6

We will now describe a general algorithm that solves the NCRSP. As a first step in our procedure, we construct a two-player turn based game.

3.1 Construction of a two-player game

Given a concurrent game $\mathcal{G} = (\mathsf{St}, s_0, \mathsf{Agt}, (\mathsf{Act}_i)_{i\in\mathsf{Agt}}, \mathsf{Tab})$ we construct a turn-based 2-player zero-sum game $\mathcal{H} = (Q, q_0, \mathsf{Act}_E, \mathsf{Act}_A, \mathsf{Tab}', \mathsf{Obj})$.

The game $\mathcal{H}$ is obtained as follows:

- $q_0 = (s_0, \emptyset, \emptyset)$
- The set Act_E is $\mathsf{Act}_E^a \cup \mathsf{Act}_E^c$ where:
  - $\mathsf{Act}_E^a = \mathsf{Act}_0 \times \prod_{i=1}^n (\mathsf{Act}_i \cup \{-\})$
  - $\mathsf{Act}_E^c = \prod_{i=1}^n (\mathsf{Act}_i \cup \{-\})$.
- The set Act_A is $\prod_{i=1}^n \mathsf{Act}_i$.
- The set Q of states is $Q_\mathsf{A} \cup Q_\mathsf{B} \cup Q_\mathsf{C} \cup Q_\mathsf{D}$ where

$$
\begin{aligned}
Q_\mathsf{A} &= \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \\
Q_\mathsf{B} &= \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times \mathsf{Act}_E^a \\
Q_\mathsf{C} &= \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times \mathsf{Act}_E^a \times \mathsf{Act}_A \\
Q_\mathsf{D} &= \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times \mathsf{Act}_E^a \times \mathsf{Act}_A \times \mathsf{Act}_E^c \ .
\end{aligned}
$$

- Player Eve plays in the states in Q_A and Q_C, while Player Adam plays in the states in Q_B and Q_D. The legal moves are given as follows:
  - From a state $(s, W, D) \in Q_\mathsf{A}$, Eve plays an action

$$\bar{a} \in \mathsf{Act}_0 \times \prod_{i=1}^n (\mathsf{Act}_i \cup \{-\}) \text{ s.t. } \forall 1 \leq i \leq n,\ \bar{a}[i] = - \iff i \notin W \ .$$

  - From a state $(s, W, D, \bar{a}) \in Q_\mathsf{B}$, Adam plays an action $\bar{b} \in \mathsf{Act}_A$.
  - From a state $(s, W, D, \bar{a}, \bar{b}) \in Q_\mathsf{C}$, Eve plays an action

$$\bar{c} \in \prod_{i=1}^n (\mathsf{Act}_i \cup \{-\}) \text{ s.t. } i \in W \cup D \implies \bar{c}[i] = - \ .$$

  - From a state $(s, W, D, \bar{a}, \bar{b}, \bar{c}) \in Q_\mathsf{D}$, Adam plays an action $\bar{d} \in \mathsf{Act}_A$.

The transition Tab' and the objective Obj of the game $\mathcal{H}$ are described next.

3.2 Transition function

The game $\mathcal{H}$ is best understood as a dialogue between Eve and Adam. In each state (s, W, D) Eve proposes an action for Agent 0 together with the actions corresponding to the winning strategies of the agents in the set W. Then, Adam responds with an action profile played by all agents in the environment. In the next step, Eve knows the entire action profile played by the agents and proposes some new deviations for the agents that do not have a deviation yet (they are neither in W nor in D). The last move is performed by Adam, it is his role to "check" that the proposed deviations and winning strategies are correct. Therefore, Adam can choose any continuation for the game and the sets W and D are updated according to the previous choices to some new values W' and D'. Each dialogue "round" is decomposed into four moves.

The transitions are given by the (partial) function $\mathsf{Tab}' : Q \times (\mathsf{Act}_E \cup \mathsf{Act}_A) \to Q$:

- When $(s, W, D) \in Q_{\mathsf{A}}$, $\mathsf{Tab}'((s, W, D), \bar{a}) = (s, W, D, \bar{a})$.
- When $(s, W, D, \bar{a}) \in Q_{\mathsf{B}}$, $\mathsf{Tab}'((s, W, D, \bar{a}), \bar{b}) = (s, W, D, \bar{a}, \bar{b})$.
- When $(s, W, D, \bar{a}, \bar{b}) \in Q_{\mathsf{C}}$, $\mathsf{Tab}'((s, W, D, \bar{a}, \bar{b}), \bar{c}) = (s, W, D, \bar{a}, \bar{b}, \bar{c})$.
- When $(s, W, D, \bar{a}, \bar{b}, \bar{c}) \in Q_{\mathsf{D}}$, $\mathsf{Tab}'((s, W, D, \bar{a}, \bar{b}, \bar{c}), \bar{d}) = (s', W', D')$, such that:
  - $s' = \mathsf{Tab}(s, (\bar{a}[0], \bar{d}))$.
  - $W' = W \cup \{i \notin W \cup D \mid (\bar{d}[i] = \bar{c}[i])$ and $(\forall j \in \mathsf{Agt} \setminus \{0, i\}, \bar{d}[j] = \bar{b}[j])\}$ $\setminus \{i \in W \mid \bar{d}[i] \neq \bar{a}[i]\}$. That is, Agent i is added to the set W' on the continuations where Agent i plays the new action proposed by Eve in $\bar{a}$ (supposedly compatible with a winning strategy) and the other agents do not change their actions with respect to $\bar{d}$. Also, any agent for whom Eve proposes an action in $\bar{c}$ is a hint to Adam that this agent can deviate from that point. It is up to Adam to agree or not. If Adam agrees, we say that he has agreed with the recommendation of Eve. In this case, Eve has to prove that she made the right choice, this will be checked by the winning condition of the game.
  - $D' = D \cup \{i \in W \mid \bar{d}[i] \neq \bar{a}[i]\}$ $\cup \{i \notin W \cup D \mid (\bar{c}[i] \neq -)$ and $(\bar{d}[i] \neq \bar{c}[i])$ and $(\forall j \in \mathsf{Agt} \setminus \{0, i\}, \bar{d}[j] = \bar{b}[j])\}\}$. This is the opposite case where Adam stood by his choices, in this case the winning condition has to check that this was a wrong decision.

3.3 Winning condition

We equip Q with the canonical projection π_i that is the projection over the i-th component. In particular, for every $(s, W, D) \in Q_{\mathsf{A}}$, we have $\pi_1((s, W, D)) = s$, $\pi_2((s, W, D)) = W$, and $\pi_3((s, W, D)) = W$. We also extend π_i over Q^+ and Q^ω as expected. Histories for Eve are finite words in $q_0(\mathsf{Act}_E Q \mathsf{Act}_A Q)^*$. Histories for Adam are finite words in $q_0(\mathsf{Act}_E Q)^*$. Plays are infinite sequence in $q_0(\mathsf{Act}_E Q \mathsf{Act}_A Q)^\omega$. Let r be a play, we denote $r \restriction_{Q_{\mathsf{A}}}$ the restriction of r over the states in Q_{A} which is an infinite sequence in ${Q_{\mathsf{A}}}^\omega$. The set $\lim \pi_2(r \restriction_{Q_{\mathsf{A}}})$ (resp. $\lim \pi_3(r \restriction_{Q_{\mathsf{A}}})$) is the set of agents in the limit of W's (resp. D's). The limit $\lim \pi_3(r \restriction_{Q_{\mathsf{A}}})$ exists because the sets D occurring in the states Q along a play are non-decreasing subsets of Agt, and Agt is finite. The limit $\lim \pi_2(r \restriction_{Q_{\mathsf{A}}})$ exists because (1) an agent is added into W only if it is not in D, and (2) when an agent leaves W, it gets into D indefinitely. This means that when an agent leaves from W, it never goes back.

We define the following sets:

$$S_0 = \{r \in Q^\omega \mid \pi_1(r \restriction_{Q_\mathsf{A}}) \in \mathsf{Obj}_0\} \ , \tag{1}$$
$$S_W = \{r \in Q^\omega \mid \forall i \in \lim \pi_2(r \restriction_{Q_\mathsf{A}}), \pi_1(r \restriction Q_\mathsf{A}) \in \mathsf{Obj}_i\} \ , \tag{2}$$
$$S_D = \{r \in Q^\omega \mid \exists i \in \lim \pi_3(r \restriction_{Q_\mathsf{A}}), \pi_1(r \restriction Q_\mathsf{A}) \notin \mathsf{Obj}_i\} \ . \tag{3}$$

$$\mathsf{Obj} = (S_0 \cup S_D) \cap S_W \ .$$

3.4 Transformations

Lifting of histories

We define a transformation over histories in $\mathcal{G}$ to create histories in $\mathcal{H}$. For every strategy σ for Eve in $\mathcal{H}$, we define the transformation $\mathcal{G}2\mathcal{H}_\sigma$.

Let h be a history in $\mathcal{G}$ and assume that $h = s_0\bar{m}_0 s_1 \bar{m}_1 ... s_k \bar{m}_k s_{k+1}$. The lifting of h is a history $\tilde{h}$ in $\mathcal{H}$ obtained by the mapping $\mathcal{G}2\mathcal{H}_\sigma$ inductively defined as follows:

$$\mathcal{G}2\mathcal{H}_\sigma(s_0) = (s_0, \emptyset, \emptyset) \ ,$$

and

$$\mathcal{G}2\mathcal{H}_\sigma(h) = \underbrace{\mathcal{G}2\mathcal{H}_\sigma(s_0\bar{m}_0 s_1 \bar{m}_1 ... s_k)}_{\tilde{h'}} \bar{a} q_b \bar{b} q_c \bar{c} q_d \bar{d} q_a \ ,$$

where

$$\bar{a} = \sigma(\tilde{h'}) \ , \qquad q_b = \mathsf{Tab}'(\mathsf{Last}(\tilde{h'}), \bar{a}) \ ,$$
$$\bar{b} = \bar{m}_{k_{-0}} \ , \qquad q_c = \mathsf{Tab}'(q_b, \bar{b}) \ ,$$
$$\bar{c} = \sigma(\tilde{h'}\bar{a}q_b\bar{b}q_c) \ , \qquad q_d = \mathsf{Tab}'(q_c, \bar{c}) \ ,$$
$$\bar{d} = \bar{b} = \bar{m}_{k_{-0}} \ , \qquad q_a = \mathsf{Tab}'(q_d, \bar{d}) \ .$$

Observe that every history $\mathcal{G}2\mathcal{H}_\sigma(h)$ ends in a state in Q_A, where Eve plays an action from Act_E^a, that always specifies an action for Agent 0. The function $\mathcal{G}2\mathcal{H}_\sigma$ is thus instrumental in obtaining a strategy σ_0 for Agent 0 in $\mathcal{G}$ from a strategy of Player Eve in $\mathcal{H}$. For every history h in $\mathcal{G}$, we define:

$$\sigma_0(h) = \sigma(\mathcal{G}2\mathcal{H}_\sigma(h))[0] \ . \tag{4}$$

For every strategy σ of Eve, we call 0-strategy the strategy obtained by Equation 4. The following claim is consequence of the same equation.

▶ **Claim 9.** *Let σ be a strategy for Eve, and let σ_0 be the 0-strategy. If a history h in $\mathcal{G}$ is compatible with σ_0 then the history $\tilde{h} = \mathcal{G}2\mathcal{H}_\sigma(h)$ in $\mathcal{H}$ is compatible with σ.*

The function $\mathcal{G}2\mathcal{H}_\sigma$ maps every history in $\mathcal{G}$ into a history in $\mathcal{H}$. We define $\mathcal{G}2\mathcal{H}_\sigma^\bullet$ as the natural extension of $\mathcal{G}2\mathcal{H}_\sigma$ over the domain of plays in $\mathcal{G}$. We extend the previous claim as expected.

▶ **Claim 10.** *Let σ be a strategy for Eve, and let σ_0 be the 0-strategy. If a run ρ in $\mathcal{G}$ is compatible with σ_0 then the run $r = \mathcal{G}2\mathcal{H}_\sigma^\bullet(\rho)$ in $\mathcal{H}$ is compatible with σ.*

▶ **Lemma 11.** *Let σ be a strategy for Eve, let ρ be a run in $\mathcal{G}$ compatible with the 0-strategy σ_0. Let h be a history in $\mathcal{G}$, assume h to be a prefix of ρ. If $\tilde{h} = \mathcal{G}2\mathcal{H}_\sigma(h)$ then $\pi_1(\tilde{h} \restriction_{Q_\mathsf{A}}) = h \restriction_{\mathsf{St}}$.*

Proof. By induction on the size of h. The base case is $h = s_0$, in which case $\mathcal{G}2\mathcal{H}_\sigma(h) = \tilde{h} = (s_0, \emptyset, \emptyset)$. We have $\pi_1(\tilde{h} \restriction_{Q_\mathsf{A}}) = s_0 = h \restriction_{\mathsf{St}}$. Now assume for induction that $\pi_1(\tilde{h} \restriction_{Q_\mathsf{A}}) = h \restriction_{\mathsf{St}}$ for every history $h = s_0\bar{m}_0 s_1 \bar{m}_1 ... s_k$ of size $1 + 2k$ and let $\mathcal{G}2\mathcal{H}_\sigma(h) = \tilde{h}$.

Now consider the history $hm_k s_{k+1}$ by definition $\mathcal{G}2\mathcal{H}_\sigma(hm_k s_{k+1}) = \tilde{h}\bar{a}q_b\bar{b}q_c\bar{c}q_d\bar{d}q_a$ where $\bar{a}, \bar{b}, \bar{c}, \bar{d}$ are obtained thanks to $\mathcal{G}2\mathcal{H}_\sigma$, by I.H. $\pi_1(\tilde{h}) = s_0 s_1 ... s_k$, it thus suffices to show that $\pi_1(q_a) = s_{k+1}$. For this, one needs to remark that $m_k = (\sigma(\tilde{h})[0], \bar{d})$, and that

$$s_{k+1} = \mathsf{Tab}(s_k, m_k) = \pi_1(\mathsf{Tab}'((s, W, D, \bar{a}, \bar{b}, \bar{c}), \bar{d})) = \pi_1(q_a)$$

where the second equality is by definition of the construction. ◀

Since the previous lemma is true for any histories that are respectively prefixes of r and ρ we obtain the following claim:

▶ **Claim 12.** *Let σ be a strategy for* Eve*, let ρ be a run in $\mathcal{G}$ compatible with the 0-strategy σ_0. If $r = \mathcal{G}2\mathcal{H}^\bullet_\sigma(\rho)$ then $\pi_1(r \restriction_{Q_\mathsf{A}}) = \rho \restriction_{\mathsf{St}}$.*

Projection of histories

We now define in some sense the reverse operation. Let us define the transformation $\mathcal{H}2\mathcal{G}$.

Let $\tilde{h}$ be a history in $\mathcal{H}$ ending in a state in Q_A.

$$\mathcal{H}2\mathcal{G}(q_0) = s_0$$

$$\mathcal{H}2\mathcal{G}(\tilde{h}\bar{a}q_b\bar{b}q_c\bar{c}q_d\bar{d}q_a) = \underbrace{\mathcal{H}2\mathcal{G}(\tilde{h})}_{\text{induction}}\underbrace{(\bar{a}[0], \bar{d}_{-0})}_{\text{action}} q_a$$

▶ **Lemma 13.** *Let $\tilde{h}$ be a run in $\mathcal{H}$, h be a history in $\mathcal{G}$. If $h = \mathcal{H}2\mathcal{G}(\tilde{h})$, then $\pi_1(\tilde{h} \restriction_{Q_\mathsf{A}}) = h \restriction_{\mathsf{St}}$*

Proof. By induction over the length of $\tilde{h}$. For $\tilde{h} = (s_0, \emptyset, \emptyset)$ the result trivially true. Assume the result holds for any history $\tilde{h}$ and let us show that it holds for $\tilde{h}\bar{a}q_b\bar{b}q_c\bar{c}q_d\bar{d}q_a$. By induction we have $\pi_1(\tilde{h} \restriction_{Q_\mathsf{A}}) = h \restriction_{\mathsf{St}}$, to conclude notice that

$$\pi_1(q_a) = \mathsf{Tab}(\mathsf{Last}(\mathcal{H}2\mathcal{G}(\tilde{h})), (\bar{a}[0], \bar{d}_{-0}))$$ ◀

The function $\mathcal{H}2\mathcal{G}$ maps every history in $\mathcal{H}$ ending in a state in Q_A into a history in $\mathcal{G}$. We define $\mathcal{H}2\mathcal{G}^\bullet$ as the natural extension of $\mathcal{H}2\mathcal{G}$ over the domain of runs in $\mathcal{H}$.

The following claim follows

▶ **Claim 14.** *Let r be a run in $\mathcal{H}$, ρ be a run in $\mathcal{G}$. If $\rho = \mathcal{H}2\mathcal{G}^\bullet(r)$, then $\pi_1(r \restriction_{Q_\mathsf{A}}) = \rho \restriction_{\mathsf{St}}$*

4 Main Theorem

▶ **Theorem 15.** *There exists a solution for the NCRSP iff* Eve *wins.*

We denote σ_i^h the strategy that mimics the strategy σ_i when the current history is h i.e.

$$\sigma_i^h(h') = \begin{cases} \sigma_i(h') \text{ if } h' \text{ is a prefix of } h \\ \sigma_i(h \cdot h') \text{ if } h \text{ is a prefix of } h' \\ \bot \text{ otherwise} \end{cases}$$

▶ **Definition 16.** Let ρ be a play and let $h = s_0\bar{a}^0 s_1 \bar{a}^1 \cdots s_k \bar{a}^k s_{k+1}$ be a prefix of ρ. We say that h is a *good deviation point* for Agent $i \in \mathsf{Agt} \setminus \{0\}$ if:

- $\rho \restriction_{\mathsf{St}} \notin \mathsf{Obj}_i$ and,
- there exists a strategy σ'_i of Agent i from $[h]$ such that for all $(\sigma_j)_{j \in \mathsf{Agt} \setminus \{0,i\}}$ we have:

$$[h] \cdot \mathsf{Out}\left(\sigma_0^{[h]}, ..., \sigma'_i, ..., \sigma_n^{[h]}\right) \in \mathsf{Obj}_i \ , \text{ where}$$

$$[h] = \rho[0..k] \cdot \langle \bar{a}^k_{-i}, \sigma'_i(\rho[0..k]) \rangle \cdot \mathsf{Tab}\left(s_k, \langle \bar{a}^k_{-i}, \sigma'_i(\rho[0..k]) \rangle\right) \ .$$

We say that ρ has a *good deviation* if some prefix h of ρ is a good deviation point.

We use the notion of deviation point in the following lemma. This lemma states that a strategy σ_0 is a solution for the NCRSP if any play ρ compatible with it, either is winning for Agent 0 or some Agent i would unilaterally deviate and win against any strategy profile of the other agents.

▶ **Lemma 17.** *A strategy σ_0 is a solution for NCRSP iff every play ρ compatible with σ_0 either $\rho \restriction_{\mathsf{St}} \in \mathsf{Obj}_0$ or, ρ has a good deviation.*

Proof. We start by establishing the if direction, let σ_0 be a solution for the NCRSP. If any outcome $\rho \in \mathsf{Plays}(\sigma_0)$ is such that $\rho \restriction_{\mathsf{St}} \in \mathsf{Obj}_0$ then there is nothing to prove. Let ρ be a play in $\mathsf{Plays}(\sigma_0)$ such that ρ is not in Obj_0. Assume toward a contradiction that ρ does not contain a good deviation point. Then by Definition 16 we know that for any prefix h of ρ, any agent $i \neq 0$ such that $\mathsf{Payoff}_i(\rho) = 0$, and any strategy τ_i of i there exists $\sigma_1, \cdots, \sigma_n$ strategies for agents 1 to n such that the following holds:

$$[h] \cdot \mathsf{Out}\left(\sigma_0^h, \sigma_1^h, \cdots, \tau_i^h, \cdots, \sigma_n^h\right) \notin \mathsf{Obj}_i \ .$$

The above equation implies that Agent i does not have a profitable deviation under the strategy σ_0, hence the profile $\langle \sigma_0, \cdots, \sigma_n \rangle$ is a 0-fixed NE contradicting the fact that σ_0 is a solution for the NCRSP.

For the only if direction, let σ_0 be a strategy for agent 0, assume that every ρ in $\mathsf{Plays}(\sigma_0)$ satisfies

1. $\rho \restriction_{\mathsf{St}} \in \mathsf{Obj}_0$ or,
2. ρ has a good deviation.

If every play ρ in $\mathsf{Plays}(\sigma_0)$ is in Obj_0 then σ_0 is a solution for NCRSP. Let ρ be a play in $\mathsf{Plays}(\sigma_0)$ such that it is not in Obj_0. By assumption, ρ has a good deviation point i.e. there exists an Agent $i \neq 0$ and a strategy τ_i for the same agent such that: *i)* $\rho \restriction_{\mathsf{St}} \notin \mathsf{Obj}_i$ and *ii)* after a finite prefix h of ρ for any tuple of strategies $(\sigma_j)_{j \in \mathsf{Agt} \setminus \{0,i\}}$ the following holds:

$$[h] \cdot \mathsf{Out}\left(\sigma_0^h, \sigma_1^h, \cdots, \tau_i^h, \cdots, \sigma_n^h\right) \in \mathsf{Obj}_i \ .$$

Hence, ρ is not the outcome of a 0-fixed NE and therefore σ_0 is a solution for the NCRSP. ◀

4.1 Correctness

▶ **Definition 18.** Eve wins if she has a strategy that ensures Obj against any strategy of Adam.

▶ **Proposition 19.** *If* Eve *wins then there exists a solution for the NCRSP.*

Proof. Let σ_E be a winning strategy for Eve in $\mathcal{H}$, and let σ_0 be the strategy for Agent 0 in $\mathcal{G}$ obtained by the construction in Sec. 3.4 Equation (4), that is, for every history h in $\mathcal{G}$, $\sigma_0(h) = \sigma_E(\mathcal{G}2\mathcal{H}_{\sigma_E}(h))[0]$. We show that σ_0 is solution to the NCRSP.

Let ρ be an arbitrary run in $\mathcal{G}$ compatible with σ_0.

According to Lemma 17 it is sufficient to show that ρ is in Obj_0 or ρ has a good deviation point. Consider the run $r = \mathcal{G}2\mathcal{H}^{\bullet}_{\sigma_E}(\rho)$ in $\mathcal{H}$. As a consequence of Claim 10, we have that r is compatible with σ_E. Since σ_E is winning, we also have $r \in \mathsf{Obj}$, i.e.,

$$r \in (S_0 \cup S_D) \cap S_W = (S_0 \cap S_W) \cup (S_D \cap S_W) \ .$$

As a first case, assume that $r \in S_0 \cap S_W$ implying $\pi_1(r \restriction_{Q_\mathsf{A}}) \in \mathsf{Obj}_0$. By Claim 12 we can write $\pi_1(r \restriction_{Q_\mathsf{A}}) = \rho \restriction_{\mathsf{St}}$, and thus $\rho \restriction_{\mathsf{St}} \in \mathsf{Obj}_0$.

As a second case, assume $r \in S_D \cap S_W$. It implies that there exists a state q_a in Q_A along r such that $q_a = (s, W, D)$ and there exists an agent i in D such that i in $\lim \pi_3(r \restriction_{Q_\mathsf{A}})$ and $\pi_1(r \restriction_{Q_\mathsf{A}}) \notin \mathsf{Obj}_i$.

We argue that Agent i has a profitable deviation from a prefix of ρ entailing that ρ contains a good deviation point.

Assume w.l.o.g. that q_a is the first state along r for which there exists an Agent i in D such that i in $\lim \pi_3(r \restriction_{Q_\mathsf{A}})$ and $\pi_1(r \restriction_{Q_\mathsf{A}}) \notin \mathsf{Obj}_i$. The run r is of the form:

$$r = \tilde{h}\bar{a}p_b\bar{b}p_c\bar{c}p_d\bar{d}q_a\tilde{t} \tag{5}$$

where $\tilde{h}$ is a finite prefix of r ending in a state in Q_A, and $\tilde{t}$ is an infinite suffix. Remember also that $r = \mathcal{G}2\mathcal{H}^{\bullet}_{\sigma_E}(\rho)$, hence there exists a history h which is a prefix of ρ such that $\tilde{h} = \mathcal{G}2\mathcal{H}(h)$. We claim that h is a good deviation point (c.f. Definition 16) for Agent i. Indeed, we use notation τ_i for the strategy defined only after h has occurred as follows: $\tau_i(h) = \bar{c}[i]$ where $\bar{c}$ is the action available for agent i in Equation (5), and for any history hh' in $\mathcal{G}$: $\tau_i(hh') = \sigma_E(\mathcal{G}2\mathcal{H}(hh'))[i]$. (Observe that by construction $\mathcal{G}2\mathcal{H}(hh')$ always ends in a state in Q_A, controlled by Eve.)

We define the set T as the set of all the plays in $\mathcal{G}$ that start with h and are compatible with τ_i. Let ρ' be a play in T, and let $r' = \mathcal{G}2\mathcal{H}(\rho')$ be a play in $\mathcal{H}$. The play r' enjoys two properties, first $i \in \lim \pi_2(r' \restriction_{Q_\mathsf{A}})$ and second it is compatible with σ_E. Hence $\pi_1(r' \restriction_{Q_\mathsf{A}}) \in \mathsf{Obj}_i$. This shows that ρ has a good deviation point after history By Lemma 17 we conclude that σ_0 is solution to the problem NCRSP. ◀

4.2 Completeness

▶ **Proposition 20.** *There exists a solution for the NCRSP then* Eve *wins.*

We first introduce some technical tools.

$$\mathsf{Deviator} : \mathsf{Hist}(\sigma_0) \times \mathsf{Agt} \to \mathsf{Act} \cup \{-\}$$

$$(h, i) \mapsto \begin{cases} a \text{ if } h \text{ is a good deviation point for Agent } i \text{ using action } a, \\ - \text{ if not.} \end{cases}$$

$$\mathsf{Root} : \mathsf{Hist} \times \mathsf{Agt} \to \mathsf{Hist} \cup \{\bot\}$$

$$(h, i) \mapsto \begin{cases} h' \text{ where } h' \text{ is the shortest prefix of } h \text{ s.t. } \mathsf{Deviator}(h', i) \in \mathsf{Act} \\ \bot \text{ if no such a prefix exists} \end{cases}$$

▶ **Claim 21.** *Let h be a history and i an agent s.t.* $\mathsf{Deviator}(h, i) \in \mathsf{Act}$*, then there exists a winning strategy τ_i from* $\mathsf{Root}(h, i)$ *for agent i.*

Indeed, assuming that $\mathsf{Deviator}(h, i) \in \mathsf{Act}$ and that there is no winning strategy from $\mathsf{Root}(h, i)$, would entail that $\mathsf{Root}(h, i)$ is not a good deviation point.

Proof of Proposition 20. Let σ_0 be a solution for the NCRSP. Given a history $\tilde{h}$ in $\mathcal{H}$ s.t. $\mathsf{Last}(\tilde{h})$ is in Q_{A}, we let $h = \mathcal{H}2\mathcal{G}(\tilde{h})$. We construct a strategy σ_E for Eve as follows: $\sigma_E(\tilde{h}) = \bar{a}$ such that $\bar{a}[0] = \sigma_0(h)$ and for every i in W, $\bar{a}[i] = \tau_i(h)$ where τ_i is the strategy described by Claim 21. Notice that this strategy is only defined for histories h that satisfy $\mathsf{Root}(h, i) \neq \bot$. This is ensured because i is in W, meaning that there exists a prefix h' of h such that h' is a good deviation point for Agent i.

We also need to define σ_E for histories ending in a Q_{C}. Consider any history of the form $\tilde{h}\bar{a}q_b\bar{b}q_c$, the strategy σ_E is defined as follows: $\sigma_E(\tilde{h}\bar{a}q_b\bar{b}q_c) = \bar{c}$ such that for every i not in $W \cup D$, $\bar{c}[i] = \mathsf{Deviator}(h, i)$ Let us show that σ_E is winning for Eve. Let r be a run compatible with σ_E. We must show that $r \in (S_0 \cup S_D) \cap S_W$. Denote $\rho = \mathcal{H}2\mathcal{G}^{\bullet}(r)$. By Claim 14 we have $\pi_1(r \restriction_{Q_{\mathsf{A}}}) = \rho \restriction_{\mathsf{St}}$.

If $\rho \restriction_{\mathsf{St}} \in \mathsf{Obj}_0$ then $r \in S_0$. If $\rho \restriction_{\mathsf{St}} \notin \mathsf{Obj}_0$, since σ_0 is a solution, it follows that along ρ some player has a good deviation point and is loosing. This entails that at some point i will be in D along r i.e. $\mathsf{Deviator}(\tilde{h}, i) \in \mathsf{Act}$ for some $\tilde{h}$ a prefix of r. Thus $r \in S_D$.

It remains to show that $r \in S_W$ this follows from the facts that 1) any player in W is due to the mapping $\mathsf{Deviator}$ that correctly guesses the correct deviations and 2) Claim 21. ◀

5 Computational Complexity

In this section, we take advantage of the construction presented in the previous section to give complexity bounds for a variety of winning conditions. In fact, we can adapt the technique used in [3] in order to establish the upper bound complexity for NCRSP.

In the case of Reachability, Safety, Büchi and coBüchi conditions, we reduce the game $\mathcal{H}$ to a finite duration game. We actually tranform the winning condition into a finite horizon condition in a finite duration game. In order to obtain this finite duration game, we simply rewrite the winning objective of $\mathcal{H}$ and obtain a new game $\mathcal{H}'$ with Parity objective. The plays in the finite duration game $\mathcal{H}^f$ are obtained by stopping the plays in the game $\mathcal{H}'$ after the first loop.

In the remainder of this section, for technical convenience, we assume that the histories in $\mathcal{H}$ are defined over the set Q^* and that the plays are defined over Q^ω. This does not affect the validity of the results since Tab' is deterministic and the actions are encoded in the states.

When the game $\mathcal{H}'$ is equipped with a parity condition, we construct a finite duration game that stops after the first loop. In particular, if $\mathsf{Pr} : \mathsf{St}' \to \mathbb{N}$ is the priority function in $\mathcal{H}'$. Then, the finite duration game $\mathcal{H}^f$ is defined over the same game structure as $\mathcal{H}'$, but each play stops after the first loop. We will consider such a play winning if the least parity in the loop is even i.e a play $r = xy_1y_2y_3 \dots y_l y_1$ where $x \in q_0Q^*$ and $y_1, y_2, \dots, y_l \in Q$ is winning for Eve if $\min\{\mathsf{Pr}(y_k) \mid 0 \leq k \leq l\}$ is even.

The following lemma establishes the relation between $\mathcal{H}'$ and $\mathcal{H}^f$. It is actually a consequence of a result that appeared in [1].

▶ **Lemma 22.** Eve *has a winning strategy in the game* $\mathcal{H}'$ *with the parity condition* Pr *if and only if she has a winning strategy in the game* $\mathcal{H}^f$*.*

The following lemma establishes the fact that inside a cycle in the game $\mathcal{H}$, the values of the sets W and D do not change.

▶ **Lemma 23.** *Let* r *be a play in* $\mathcal{H}$*, and consider a loop along* r*. Let also* q *and* q' *be two states on this loop. We have* $\pi_2(q) = \pi_2(q')$ *and* $\pi_3(q) = \pi_3(q')$*.*

Proof. Let $r = xqyqz$ be an infinite play in $\mathcal{H}$. From the definition of the transition relation in $\mathcal{H}$, $r' = x(qy)^\omega$ is also a valid play in $\mathcal{H}$. Then, assume towards contradiction that there are two states in qy having different values on states W or D. It means that there is at least one player i that is added or removed to/from W or D. Therefore, along r' we would have an infinite number of additions or removals to/from W or D. But, according to the transition relation, this is not possible because once a player is removed from W, it is added to the set D and never added to W again along r'. Also, once a player added in the set D, he is never removed. Therefore, along each path, along each loop, the values of W and D do not change. ◀

In order to check the condition of reachability, we keep along plays in the game $\mathcal{H}$ a set P of players in the environment that already have visited their target states. Then, the resulting game $\mathcal{H}'$ has states in $Q \times 2^{\mathsf{Agt}}$ where Q is the set of states in the game $\mathcal{H}$. The set P is initially equal to the set of players for which the initial state is in their target set. Let $R_i \subseteq \mathsf{St}$ be the target set of Player i. Then, $P_0 = \{i \mid s_0 \in R_i\}$ and the initial state in the resulting game $\mathcal{H}'$ is $(s_0, \emptyset, \emptyset, P_0)$.

The set P is updated as follows:

- if $(q, q') \in \mathsf{Tab}'$ in $\mathcal{H}$ and $q' = (s, W, D) \in \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$, then $((q, P), (q', P \cup \{i \mid s \in R_i\}))$ is the corresponding transition in $\mathcal{H}'$.
- if $(q, q') \in \mathsf{Tab}'$ in $\mathcal{H}$ and $q' \notin \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$, then $((q, P), (q', P))$ is the corresponding transition in $\mathcal{H}'$.

Note that the set P also eventually stabilizes since it only increases and there is a finite number of players in $\mathcal{G}$. Let $\lim \pi_4(r \restriction_{Q_\mathsf{A}})$ be the limit along the play r.

The objective of Eve in the game $\mathcal{H}$ is written in $\mathcal{H}'$ as the Büchi condition $\textsc{Büchi}(F^R)$ where:

$$F^R = \{(s, W, D, P) \mid (0 \in P \text{ or } D \setminus P \neq \emptyset) \text{ and } (W \subseteq P)\} \ .$$

Now, the Büchi objective $\textsc{Büchi}(F^R)$ can be expressed as the parity objective $\textsc{Parity}(\mathsf{Pr})$ with $\mathsf{Pr}(v) = 0$ if $v = (s, W, D, P) \in F^R$ and $\mathsf{Pr}(v) = 1$ otherwise.

We now define the finite duration game $\mathcal{H}^f$ over the same game arena as $\mathcal{H}$, but each play stops when the first state in $\mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$ is repeated. Then, each play is of the form $r = xy_1y_2y_3 \dots y_l y_1$ where $x \in q_0(Q')^*$ and $y_1, y_2, \dots, y_l \in Q'$ with $Q' = \mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$. Eve wins in the game $\mathcal{H}^f$ if $y_1 = (s, W, D, P)$ is such that $(0 \in P$ or $D \setminus P \neq \emptyset)$ and $(W \subseteq P)$. Equivalently, thanks to Lemma 23 and because the value of P does not change for the same argument that show W and D eventually stabilize. Finally Eve wins if $\min\{\mathsf{Pr}(y_k) \mid 0 \leq j < l\}$ is even.

▶ **Lemma 24.** *All plays in the game $\mathcal{H}^f$ have polynomial length in the size of the initial game.*

Proof. Since D and P are monotone, there are at most $|\mathsf{Agt}| + 1$ different values that they can take on a path of $\mathcal{H}$. Also, in the set W we can have at most one addition and one removal for each player $i \in \mathsf{Agt}$ and hence $2|\mathsf{Agt}| + 1$ different values for W. Therefore, along a play π there are at most $r = 1 + (2|\mathsf{Agt}| + 1) \cdot (|\mathsf{Agt}| + 1)^2 \cdot |\mathsf{St}|$ different states in $\mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$. Then, between two states in $\mathsf{St} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}} \times 2^{\mathsf{Agt}}$, there are three intermediate states. Therefore, since all the plays in $\mathcal{H}^f$ stop after the first cycle, the length of each play is of at most $4r + 1$ states since there is only one state that appears twice. Therefore, all plays in $\mathcal{H}^f$ have polynomial length in Agt and St of the initial play $\mathcal{G}$. ◀

▶ **Proposition 25.** *Deciding if there is a solution for the non-cooperative rational synthesis in concurrent games with Reachability objectives is in* PSpace.

Proof. Using Lemmas 22 and 24, solving the non-cooperative rational synthesis problem, reduces to solving the finite duration game $\mathcal{H}^f$ which has polynomial length plays. This can be done in PSpace using an alternating Turing machine runing in PTime. ◀

In the case of Safety, Büchi and coBüchi conditions, we essentially use similar constructions; c.f. long version for full details on the constructions. Roughly speaking, in the case of safety it sufficient to "dualize" the winning condition. In the cases of Büchi and coBüchi objectives, the idea is to transform the game $\mathcal{H}$ by possibly adding some counters such that Eve's objective can be written as a parity objective. Note that these constructions are similar to the ones in [3]. In the case of Muller conditions, we have to use Least Appearance Record (LAR) construction to get the parity game $\mathcal{H}'$ and then the finite duration game would have plays with exponential length in the size of the initial game. This approach would give ExpSpace complexity. Fortunately, the parity condition in the game $\mathcal{H}'$ that we obtain after applying the LAR construction has an exponential number of states but only a polynomial number of priorities. Then, by using the result from [5, 8], we obtain ExpTime complexity.

▶ **Theorem 26.** *Deciding if there is a solution for the non-cooperative rational synthesis problem in concurrent games is in* PSpace *for Safety, Reachability, Büchi and co-Büchi objectives and* ExpTime *for Muller objectives.*

In the case of a fixed number of agents, the game $\mathcal{H}$ that we build has polynomial size in the size of the initial game $\mathcal{G}$ (when considering that the transitions are given explicitly in the table Tab since we build nodes in $\mathcal{H}$ for each possible action profile). This lowers the complexities that we obtain for the rational synthesis problem. The theorem below holds because the game $\mathcal{H}$ has polynomial size and Eve's objective is fixed.

▶ **Theorem 27.** *Deciding if there is a solution for the non-cooperative rational synthesis in concurrent games with a fixed number of agents and Safety, Reachability, Büchi or co-Büchi objectives is in* PTime.

▶ **Theorem 28.** *Deciding if there is a solution for the non-cooperative rational synthesis in concurrent games with a fixed number of agents and Muller objectives is in* PSpace.

6 Conclusions

In some circumstances, the Non Cooperative Rational Synthesis Problem (NCRSP) introduced in [6] and defined here as Problem 6 might arguably accept undesired solutions. It asks whether there is a strategy σ_0 for Agent 0 such that *for every* 0-NE, if $\bar{\sigma} = \langle \sigma_0, \bar{\sigma}_{-0} \rangle$, then $\mathsf{Payoff}_0(\bar{\sigma}) = 1$. One sees the objective of Agent 0 as a critical property satisfied by all rational evolutions of the system. A possibly unwanted consequence is that a strategy σ_0 which does not allow any rational evolution of the system, thus forcing anarchy, would be a solution. The original definition of NCRSP can be strengthened so as to ask for a strategy σ_0 for Agent 0 such that *there is* at least one 0-NE. Another amendment can also restrict the class of game structures. For instance, one can consider pseudo turn-based games, where 0-NE are certain to exist. It suffices to add in Definition 1 the constraint that in every state, only Agent 0 and at most one other agent have non-vacuous choices. The games are still concurrent. Agent 0 can still effectively control every state, but once her strategy σ_0 is fixed, the sub-game induced by σ_0 has all the characteristics of a turn-based game, where there is always a 0-NE.

References

1 Benjamin Aminof and Sasha Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.

2 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure nash equilibria in concurrent deterministic games. *Logical Methods in Computer Science*, 11(2), 2015. doi:10.2168/LMCS-11(2:9)2015.

3 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.121.

4 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

5 Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 117–123, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109571.

6 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. In *Multi-Agent Systems - 12th European Conference, EUMAS 2014, Prague, Czech Republic, December 18-19, 2014, Revised Selected Papers*, pages 219–235, 2014. doi:10.1007/978-3-319-17130-2_15.

7 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

8 Sven Schewe. Solving parity games in big steps. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2007. doi:10.1007/978-3-540-77050-3_37.

Logics Meet 1-Clock Alternating Timed Automata

Shankara Narayanan Krishna
Department of Computer Science & Engineering IIT Bombay, India
krishnas@cse.iitb.ac.in

Khushraj Madnani
Department of Computer Science & Engineering IIT Bombay, India
khushraj@cse.iitb.ac.in

Paritosh K. Pandya
School of Technology and Computer Science, TIFR, India
pandya@tcs.tifr.res.in

Abstract

This paper investigates a decidable and highly expressive real time logic QkMSO which is obtained by extending $\mathsf{MSO}[<]$ with guarded quantification using block of less than k metric quantifiers. The resulting logic is shown to be expressively equivalent to 1-clock ATA where loops are without clock resets, as well as, RatMTL, a powerful extension of $\mathsf{MTL}[\,\mathsf{U}_I]$ with regular expressions. We also establish 4-variable property for QkMSO and characterize the expressive power of its 2-variable fragment. Thus, the paper presents progress towards expressively complete logics for 1-clock ATA.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Metric Temporal Logic, Alternating Timed Automata, MSO, Regular Expressions, Expressive Completeness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.39

Related Version A full version of the paper is available at [15], `https://arxiv.org/abs/1802.02514`.

1 Introduction

Since the inception of real-time logics and timed automata, the question of finding expressive timed logics which are also decidable has been a prominent concern. Both classical first-order/monadic second-order logics as well as temporal logics were extended with metric constraints. Expressive power of a logic is typically measured by comparing it with respect to other well established logics and automata as exemplified by the celebrated Büchi and Kamp theorems [11, 13]. In real-time scenario, Alur and Henzinger in 1990 asked whether First order logic of Distance $\mathsf{FO}[<,+]$, a hybrid logic $\mathsf{TPTL}[\mathsf{U},\mathsf{S}]$, and the metric temporal logic $\mathsf{MTL}[\mathsf{U}_\mathsf{I},\mathsf{S}_\mathsf{I}]$ all have the same expressive power [2]. It took 15 years to show that $\mathsf{MTL}[\mathsf{U}_\mathsf{I},\mathsf{S}_\mathsf{I}]$ is less expressive than TPTL over timed words [18] [3]. It is only in the last few years, that extensions of $\mathsf{MTL}[\mathsf{U}_\mathsf{I},\mathsf{S}_\mathsf{I}]$ which are expressively complete for $\mathsf{FO}[<,+]$ have been found [9, 8]. Unfortunately all these logics have undecidable satisfiability.

For establishing the decidability of satisfiability of a logic, often an effective reduction to some form of automata with decidable non-emptiness is used. Alur and Henzinger in 1991 came up with the sub-logic MITL which is $\mathsf{MTL}[\mathsf{U}_\mathsf{I},\mathsf{S}_\mathsf{I}]$ where time interval constraints I are non-singular intervals [1]. They showed the decidability of this logic by reducing its formulae essentially to language equivalent non-deterministic timed automata. However,

the logic was expressively weak as compared to timed automata . Looking for a more expressive but decidable logic, Wilke in a seminal paper introduced the *Monadic Second Order logic of relative distance*, $\mathcal{L}^{\overleftrightarrow{\mathsf{d}}}$, and showed that this had exactly the expressive power of non-deterministic timed automata [22]. Moreover, the logic had decidable satisfiability. Wilke also showed that $\mathcal{L}^{\overleftrightarrow{\mathsf{d}}}$ subsumed the expressive power of temporal logic EMITL, which was MITL extended with a finite automaton modality. Unfortunately, for any such logic, the validity and model checking (against timed automata) are necessarily undecidable as non-deterministic timed automata have undecidable universality/language inclusion. In another important paper, Henzinger, Raskin and Schobbens related *Event Clock Logic* ECL extended with *automata modality* to recursive event clock automata [6]. Attempts in these works have been to match the expressive power of timed automata and to have decidable satisfiability. The alternative is to try to match expressive power of some decidable and boolean closed class of timed automata. 1-clock Alternating Timed Automata (1-ATA) over finite words are perhaps the largest boolean closed class of timed languages for which emptiness is known to be decidable [17], [16]. Utilizing this fact, Ouaknine and Worrell showed in their seminal work that satisfiability as well as model checking of $\mathsf{MTL}[\mathsf{U}_\mathsf{I}]$ (with pointwise interpretation) over finite words is decidable. The result was proved by constructing a language equivalent 1-ATA for a formula of $\mathsf{MTL}[\mathsf{U}_\mathsf{I}]$ [17]. Unfortunately, the logic turns out to have much less expressive power than 1-ATA. In a series of papers [20], [12] we have investigated decidable extensions of $\mathsf{MTL}[\mathsf{U}_\mathsf{I}]$ with increasing expressive power culminating in RatMTL [21]. Logic RatMTL is a powerful extension of $\mathsf{MTL}[\mathsf{U}_\mathsf{I}]$ allowing counting and regularity constraints. But in retrospect it also turns out to be less expressive than 1-ATA.

The quest for expressively complete real-time logics matching the power of 1-ATA has remained open for over 13 years. In this paper, we give a partial solution to this problem. We show expressive completeness of some classical and metric temporal logics for natural subclasses of 1-ATA. We define an extension of Monadic Second Order Logic $\mathsf{MSO}[<]$ over words by adding guarded quantification with blocks of at most $k-1$ metric quantifiers to give a real time logic QkMSO. An essential syntactic restriction is that no free second order variable occurs in the scope of a metric quantifier, and a metric quantifier block results into a formula with only one free variable. In this, we have been inspired by the logic Q2MLO (over continuous time) defined by Hirshfeld and Rabinovich [7] as well as Hunter [8]. A carefully defined syntax gives us a logic which allows only future time properties to be stated. Note that punctual constraints are permitted in QkMSO unlike Q2MLO.

We investigate the decidability and expressive power of QkMSO. Firstly, we define a subclass of 1-ATA called 1-ATA with reset-free loops (1-ATA-rfl). In these automata, there is no cycle involving clock reset. Thus, on any run each transition with reset occurs at most once. As our first main result, we show that a) QkMSO, b) the 1-ATA-rfl, and, c) the metric temporal logic RatMTL, introduced earlier in [21], are all expressively equivalent (with effective reductions). In the process, we also prove that QkMSO has four variable property over timed words. This can be seen akin to the famous 3-variable property of $\mathsf{FO}[<]$ over words [10].

For our second main result, we turn to the two variable fragment Q2MSO of QkMSO. For its automaton characterization, we introduce a syntactic restriction of conjunctive-disjunctiveness (C⊕D) in 1-ATA. Here, each ATA thread is either in conjunctive mode or in disjunctive mode at a time, and it can switch modes only on a reset transition. We show that a) Q2MSO, b) the C⊕D-1-ATA-rfl, and c) the sublogic FRatMTL of RatMTL have exactly the same expressive power. Here logic FRatMTL uses only restricted version FRat of the

modality Rat of RatMTL. A similar modality was also defined earlier by Wilke [22] in logic EMITL. In summary, we show that

QkMSO	$\equiv$	1−ATA−rfl	$\equiv$	RatMTL
Q2MSO	$\equiv$	C$\oplus$D−1−ATA−rfl	$\equiv$	FRatMTL

These results make QkMSO to be amongst the highly expressive logics with decidable satisfiability and model checking problems.

2 Preliminaries

Let Σ be a finite set of propositions. A finite timed word over Σ is a tuple $\rho = (\sigma, \tau)$, where σ and τ are sequences $\sigma_1\sigma_2\ldots\sigma_n$ and $\tau_1\tau_2\ldots\tau_n$ respectively, with $\sigma_i \in \Gamma = 2^{\Sigma}\backslash\emptyset$, and $\tau_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq n$. For all $i \in dom(\rho)$, we have $\tau_i \leq \tau_{i+1}$, where $dom(\rho)$ is the set of positions $\{1, 2, \ldots, n\}$ in the timed word. For convenience, we assume $\tau_1 = 0$. The σ_i's can be thought of as labeling positions i in $dom(\rho)$. For example, given $\Sigma = \{a, b, c\}$, $\rho = (\{a, c\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$ is a timed word. ρ is strictly monotonic iff $\tau_i < \tau_{i+1}$ for all $i, i+1 \in dom(\rho)$. Otherwise, it is weakly monotonic. The set of finite timed words over Σ is denoted $T\Sigma^*$. Given $\rho = (\sigma, \tau)$ with $\sigma = \sigma_1\ldots\sigma_n \in \Gamma^+$, σ^{single} denotes the set of all words $w_1w_2\ldots w_n$ where each $w_i \in \sigma_i$. ρ^{single} consists of all timed words $(\sigma^{\mathsf{single}}, \tau)$. For the ρ as above, ρ^{single} consists of timed words $(\{a\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$ and $(\{c\}, 0)(\{a\}, 0.7)(\{b\}, 1.1)$.

2.1 Temporal Logics

In this section, we define preliminaries pertaining to the temporal logics studied in the paper. Let $I\nu$ be a set of open, half-open or closed time intervals. The end points of these intervals are in $\mathbb{N}\cup\{0, \infty\}$. Examples of such intervals are $[1, 3), [2, 2], [2, \infty)$. For a time stamp $\tau\in\mathbb{R}_{\geq 0}$ and an interval $\langle a, b\rangle$, where $\langle$ is left-open or left-closed and $\rangle$ is right-open or right-closed, $\tau + \langle a, b\rangle$ represents the interval $\langle\tau + a, \tau + b\rangle$.

Metric Temporal Logic (MTL [14]). Given a finite alphabet Σ, the formulae of logic MTL are built from Σ using boolean connectives and time constrained version of the until modality U as follows: $\varphi ::= a(\in \Sigma)\ |true\ |\varphi \wedge \varphi\ |\ \neg\varphi\ |\ \varphi\,\mathsf{U}_I\varphi$, where $I \in I\nu$. For a timed word $\rho = (\sigma, \tau) \in T\Sigma^*$, a position $i \in dom(\rho)$, and an MTL formula φ, the satisfaction of φ at a position i of ρ is denoted $\rho, i \models \varphi$, and is defined as follows: (i) $\rho, i \models a \leftrightarrow a \in \sigma_i$, (ii) $\rho, i \models \neg\varphi \leftrightarrow \rho, i \not\models \varphi$, (iii) $\rho, i \models \varphi_1 \wedge \varphi_2 \leftrightarrow \rho, i \models \varphi_1$ and $\rho, i \models \varphi_2$, (iv) $\rho, i \models \varphi_1\,\mathsf{U}_I\varphi_2 \leftrightarrow \exists j > i$, $\rho, j \models \varphi_2, \tau_j - \tau_i \in I$, and $\rho, k \models \varphi_1\ \forall\ i < k < j$. The language of a MTL formula φ is $L(\varphi) = \{\rho\ |\ \rho, 1 \models \varphi\}$. Two formulae φ and ϕ are said to be equivalent denoted as $\varphi \equiv \phi$ iff $L(\varphi) = L(\phi)$. The subclass of MTL restricting the intervals I in the until modality to non-punctual intervals is denoted MITL. Punctual intervals like [2,2] are disallowed. Note that we restrict to until-only fragment of MTL for the sake of decidability.

▶ **Theorem 1** ([17]). MTL *satisfiability is decidable over finite timed words with non-primitive recursive complexity.*

MTL with Rational Expressions (RatMTL)

We first recall an extension of MTL with rational expressions (RatMTL), introduced in [21]. The modalities in RatMTL assert the truth of a rational expression (over subformulae) within a particular time interval with respect to the present point. For example, the formula

$\mathsf{Rat}_I(\varphi_1.\varphi_2)^+$ when evaluated at a point i, asserts the existence of $2k$ points with time stamps $\tau_{j+1} < \tau_{j+2} < \cdots < \tau_{j+2k}$, $k > 0$, such that τ_{j+1} and τ_{j+2k} are the first and last time stamps in $\tau_i + I$, respectively. φ_1 evaluates to true at τ_{j+2l+1}, and φ_2 evaluates to true at τ_{j+2l+2}, for all $0 \leq l < k$.

RatMTL Syntax. Formulae of RatMTL are built from a finite alphabet Σ as:
$\varphi ::= a(\in \Sigma) \mid true \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{Rat}_I\mathsf{re}(\mathsf{S}) \mid \mathsf{FRat}_{I,\mathsf{re}(\mathsf{S})}\varphi$, where $I \in I\nu$ and S is a finite set of RatMTL subformulae, and re(S) is defined as a rational expression over S. $\mathsf{re}(\mathsf{S}) ::= \epsilon \mid \varphi(\in \mathsf{S}) \mid \mathsf{re}(\mathsf{S}).\mathsf{re}(\mathsf{S}) \mid \mathsf{re}(\mathsf{S}) + \mathsf{re}(\mathsf{S}) \mid [\mathsf{re}(\mathsf{S})]^*$. Thus, RatMTL is MTL extended with modalities URat and Rat (RatMTL=MTL+Rat+FRat). An *atomic* rational expression re is any well-formed formula $\varphi \in$ RatMTL.

RatMTL Semantics. For a timed word $\rho = (\sigma, \tau) \in T\Sigma^*$, a position $i \in dom(\rho)$, a RatMTL formula φ, and a finite set S of subformulae of φ, we define the satisfaction of φ at a position i as follows. For positions $i < j \in dom(\rho)$, let $\mathsf{Seg}(\rho, \mathsf{S}, i, j)$ denote the untimed word over 2^{S} obtained by marking the positions $k \in \{i+1, \ldots, j-1\}$ of ρ with $\psi \in \mathsf{S}$ iff $\rho, k \models \psi$. For a position $i \in dom(\rho)$ and an interval I, let $\mathsf{TSeg}(\rho, \mathsf{S}, I, i)$ denote the untimed word over 2^{S} obtained from ρ by marking all the positions k, where $\tau_k - \tau_i \in I$, with $\psi \in \mathsf{S}$ iff $\rho, k \models \psi$.

- $\rho, i \models \mathsf{FRat}_{I,\mathsf{re}(\mathsf{S})}\varphi \leftrightarrow \exists j{>}i,\ \rho, j \models \varphi, \tau_j - \tau_i \in I$ and, $[\mathsf{Seg}(\rho, \mathsf{S}, i, j)]^{\mathsf{single}} \cap L(\mathsf{re}(\mathsf{S})) \neq \emptyset$, where $L(\mathsf{re}(\mathsf{S}))$ is the language of the rational expression re formed over the set S.
- $\rho, i \models \mathsf{Rat}_I\mathsf{re} \leftrightarrow [\mathsf{TSeg}(\rho, S, I, i)]^{\mathsf{single}} \cap L(\mathsf{re}(\mathsf{S})) \neq \emptyset$.

The language accepted by a RatMTL formula φ is given by $L(\varphi) = \{\rho \mid \rho, 1 \models \varphi\}$. The subclass of RatMTL using only the FRat modality is denoted FRatMTL (FRatMTL=MTL + FRat). If we stick to non-punctual intervals, the subclass obtained is FRatMITL (FRatMITL=MITL + FRat). Some remarks are in order.

1. In [21], the URat modality was used instead of FRat; however, both have the same expressiveness. Note that $\varphi_1 \mathsf{URat}_{I,\mathsf{re}(\mathsf{S})}\varphi_2$ is equivalent to $\mathsf{FRat}_{I,\mathsf{re}'(\mathsf{S}\cup\{\varphi_1\})}\varphi_2$ where $\mathsf{re}'(\mathsf{S} \cup \{\varphi_1\}) = \mathsf{re}(\mathsf{S}) \cap \varphi_1^*$.
2. The classical $\varphi_1\, \mathsf{U}_I \varphi_2$ modality can be written in FRatMTL as $\mathsf{FRat}_{I,\varphi_1^*}\varphi_2$. Also, it can be shown (see [21]) that the URat(and thus FRat) modality can be expressed using the Rat modality.

Modal depth. The modal depth (md) of a RatMTL formula is defined as follows. Let PL_Σ be the set of propositional logic formulae over Σ (up to equivalence). An atomic RatMTL formula over Σ is an element of PL_Σ and has modal depth 0. A RatMTL formula φ over Σ having a single modality (Rat or FRat) has modal depth one and has the form $\mathsf{Rat}_I\mathsf{re}$ or $\mathsf{FRat}_{I,\mathsf{re}}\psi$ where re is a regular expression over PL_Σ and $\psi \in \mathsf{PL}_\Sigma$. Inductively, we define modal depth as follows: (i) $\mathsf{md}(\varphi \wedge \psi) = \mathsf{md}(\varphi \vee \psi) = \max[\mathsf{md}(\varphi), \mathsf{md}(\psi)]$. Similarly, $\mathsf{md}(\neg\varphi) = \mathsf{md}(\varphi)$. (ii) Let re be a regular expression over the set of subformulae $\mathsf{S} = \{\psi_1, \ldots, \psi_k\}$. $\mathsf{md}(\mathsf{FRat}_{I,\mathsf{re}}(\varphi)) = 1 + \max(\psi_1, \ldots, \psi_k, \varphi)$. Similarly $\mathsf{md}(\mathsf{Rat}_I(\mathsf{re})) = 1 + \max(\psi_1, \ldots, \psi_k)$.

▶ **Example 2.** Consider the formula $\varphi = \mathsf{Rat}_{[1,1]}(\mathsf{Rat}_{(0,1)}(aa)^*)$. Then $\varphi = \mathsf{Rat}_{[1,1]}\mathsf{re}_1$ where $\mathsf{re}_1 = \mathsf{Rat}_{(0,1)}(aa)^*$. The subformulae of interest are $\mathsf{S} = \{\mathsf{Rat}_{(0,1)}(aa)^*, a\}$. For $\rho{=}(\{a\}, 0)\ (\{a, b\}, 0.9)\ (\{a\}, 1)(\{a\}, 1.2)$, $\rho, 3 \not\models \mathsf{re}_1$, since $[\mathsf{TSeg}(\rho, \mathsf{S}, (0,1), 3)]^{\mathsf{single}}$ has only the word a and hence $[\mathsf{TSeg}(\rho, S, (0,1), 3)]^{\mathsf{single}} \cap L((aa)^*) = \emptyset$. Hence, $\rho, 1 \not\models \varphi$. On the other hand, for $\rho = (\{a\}, 0)(\{a\}, 0.3)\ (\{a\}, 1)(\{a\}, 1.1)\ (\{a\}, 1.8)$, $\rho, 1 \models \varphi$, since $aa \in [\mathsf{TSeg}(\rho, S, (0,1), 3)]^{\mathsf{single}} \cap L((aa)^*)$.

2.2 MSO with guarded metric quantifiers QkMSO

Let $\rho = (\sigma, \tau)$ be a timed word over a finite alphabet Σ, as before. We define a real-time logic QkMSO (with parameter $k \in \mathbb{N}$) which is interpreted over such words. It includes MSO[<] over words σ relativized to specify only future properties. This is extended with a notion of time constraint formula $\psi(t_i)$. All variables in our logic range over positions in the timed word and not over time stamps. There are two sorts of formulae in QkMSO which are mutually recursively defined : these are MSO^{t_0} formulae ϕ which have no real-time constraints except time constraint subformulae $\psi(t_p)$. These subformulae ($\psi(t_p)$) have only one free variable t_p, which is a first order variable. Such a time constraint formula $\psi(t_p)$ consists of a block of real-time constrained quantification applied to a QkMSO formula with no free second order variables. This form of real time constraints in first order logic was pioneered by Hirshfeld and Rabinovich [7] in their logic Q2MLO, which we refer in this paper as Q2FO, and later used by Hunter [8]. Let $t_0, t_1, \ldots$ be first order variables and $T_0, T_1, \ldots$ the monadic second-order variables. We have a two sorted logic consisting of MSO formulae ϕ and time constrained formulae ψ. Let $a \in \Sigma$, and let t_i range over first order variables, while T_i range over second order variables. Each quantified first order variable in ϕ is relativized to the future of some variable, say t_0, called anchor variable, giving formulae of MSO^{t_0}. The syntax of $\phi \in \mathsf{MSO}^{t_0}$ is given by: $t_p{=}t_q \mid t_p{<}t_q \mid Q_a(t_p) \mid T_j(t_i) \mid \phi \wedge \phi \mid \neg\phi \mid \exists t'.t'{>}t_0 \wedge \phi \mid \exists T_i \phi \mid \psi(\mathbf{t_p})$.
Here, $\psi(t_p) \in \mathsf{MSO}^{t_p}$ is a time constraint formula whose syntax and semantics are given little later. A formula in MSO^{t_0} with first order free variables $t_0, t_1, \ldots t_k$ and second-order free variables $T_1, \ldots, T_m$ and which is relativized to the future of t_0 is denoted $\phi(\downarrow t_0, \ldots t_k, T_1, \ldots, T_m)$. (The $\downarrow$ is only to indicate the anchor variable. It has no other function.) The semantics of such formulae is as usual. Given ρ, positions $a_0, \ldots a_k$ in $dom(\rho)$, and sets of positions $A_1, \ldots, A_m$ with $A_i \subseteq dom(\rho)$,we define

$$\rho, (a_0, a_1, \ldots, a_k, A_1, \ldots, A_m) {\models} \phi(\downarrow t_0, t_1, \ldots t_k, T_1, \ldots, T_m)$$

inductively, as usual.

1. $(\rho, a_0, a_1, \ldots, a_k, A_1, \ldots, A_m) {\models}\ t_i{<}t_j$ iff $a_i{<}a_j$,
2. $(\rho, a_0, a_1, \ldots, a_k, A_1, \ldots, A_m) {\models}\ Q_a(t_i)$ iff $a{\in}\sigma(a_i)$,
3. $(\rho, a_0, a_1, \ldots, a_k, A_1, \ldots, A_m) {\models}\ T_j(t_i)$ iff $a_i{\in}A_j$,
4. $(\rho, a_0, a_1, \ldots, a_k, A_1, \ldots, A_m) {\models}\ \exists t_k\ t_0{<}t_k{\wedge}\phi(\downarrow t_0, \ldots t_k, T_1, \ldots, T_m)$ iff $(\rho, a_0, \ldots, a'_k, A_1, \ldots, A_m) \models \phi(\downarrow t_0, \ldots t_k, , T_1, \ldots, T_m)$ for some $a'_k \geq a_0$.

The **time constraint** $\psi(t_0)$ has the form $\mathcal{Q}_1 t_1 \mathcal{Q}_2 t_2 \ldots \mathcal{Q}_j t_j\ \phi(\downarrow t_0, t_1, \ldots t_j)$ where $\phi \in \mathsf{MSO}^{t_0}$ and $\mathbf{j < k}$, the parameter of logic QkMSO. Each quantifier $\mathcal{Q}_i t_i$ has the form $\overline{\exists} t_i \in t_0 + I_i$ or $\overline{\forall} t_i \in t_0 + I_i$ for a time interval I_i as in MTL formulae. $\mathcal{Q}_i$ is called a metric quantifier. The semantics of such a formula is as follows. $(\rho, a_0) \models \mathcal{Q}_1 t_1 \mathcal{Q}_2 t_2 \ldots \mathcal{Q}_j t_j\ \phi(\downarrow t_0, t_1, \ldots t_j)$ iff for $1 \leq i \leq j$, there exist/for all a_i such that $a_0 \leq a_i$ and $\tau_{a_i} \in \tau_{a_0} + I_i$, we have $(\rho, a_0, a_1 \ldots a_j) \models \phi(\downarrow t_0, t_1, \ldots t_j)$. Note that each time constraint formula has exactly one free variable. Variables $t_0, t_1, \ldots, t_j$ are called time constrained in $\psi(t_0)$.

▶ **Example 3.** Let $\rho{=}(\{a\},0)\ (\{b\},2.1)\ (\{a,b\},2.75)\ (\{b\},3.1)$ be a timed word. Consider the time constraint $\psi(x) = \overline{\exists} y \in x + (2,\infty) \overline{\exists} z \in x + (3,\infty)(Q_b(y) \wedge Q_b(z))$. It can be seen that $\rho, 1{\models}Q_a(x) \wedge \psi(x)$.

Metric Depth. The *metric depth* of a formula φ denoted ($\mathsf{md}(\varphi)$) gives the nesting depth of time constraint constructs. It is defined inductively as follows: For atomic formulae φ, $\mathsf{md}(\varphi) = 0$. All the constructs of MSO^{t_i} do not increase md. For example, $\mathsf{md}[\varphi_1 \wedge \varphi_2] = max(\mathsf{md}[\varphi], \mathsf{md}[\varphi_2])$ and $\mathsf{md}[\exists t.\varphi(t)]$. However, md is incremented for each application of metric quantifier block. $\mathsf{md}[\mathcal{Q}_1 t_1 \mathcal{Q}_2 t_2 \ldots \mathcal{Q}_j t_j \phi] = \mathsf{md}[\phi] + 1$.

▶ **Example 4.** The sentence $\forall t_0\ \overline{\forall} t_1 \in t_0 + (1,2)\ \{Q_a(t_1) \rightarrow (\overline{\exists} t_0 \in t_1 + [1,1]\ Q_b(t_0))\}$ accepts all timed words such that for each a which is at distance (1,2) from some time stamp t, there is a b at distance 1 from it. This sentence has metric depth two with time constrained variables t_0, t_1.

Note that QkMSO is not closed under second order quantification: arbitrary use of second order quantification is not allowed, and its syntactic usage as explained above is restricted to prevent a second order free variable from occurring in the scope of the real-time constraint (similar to [19], [6] and [22]). For example, $\exists X.\exists t.[X(t) \wedge \overline{\exists} t' \in t+(1,2) Q_a(t')]$ is a well-formed QkMSO formula while, $\exists X.\exists t.\overline{\exists} t' \in t+(1,2)[Q_a(t') \wedge X(t)]$ is not, since X is freely used within the scope of the metric quantifier.

Special Cases of QkMSO. The case when $k = 2$ gives logic Q2MSO. The absence of second order variables and second order quantifiers gives logics QkFO and Q2FO. The formulae in example 4 is a Q2FO formulae. Note that our Q2FO is the pointwise counterpart of logic Q2MLO studied in [7] in the continuous semantics.

2.3 1-clock Alternating Timed Automata (1-ATA)

Let Σ be a finite alphabet and let $\Gamma = 2^{\Sigma} \backslash \emptyset$. A 1-ATA [17] [16] is a 5 tuple $\mathcal{A} = (\Gamma, S, s_0, F, \delta)$, where S is a finite set of locations, $s_0 \in S$ is the initial location and $F \subseteq S$ is the set of final locations. Let x denote the clock variable in the 1-ATA, and $x \in I$ denotes a clock constraint where I is an interval. Let X denote a finite set of clock constraints of the form $x \in I$. The transition function is defined as $\delta : S \times \Gamma \rightarrow \Phi(S \cup X)$ where $\Phi(S \cup X)$ is a set of formulae over $S \cup X$ defined by the grammar $\varphi ::= \top | \bot | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | s | x \in I | x.\varphi$ where $s \in S$, and $x.\varphi$ is a binding construct resetting clock x to 0. A state of 1-ATA is defined as a pair of a location and a clock valuation. In other words, a state of a 1-ATA is an element of $S \times \mathbb{R}_{\geq 0}$, where S is a set of locations. A set of states $M \subseteq S \times \mathbb{R}_{\geq 0}$ and a clock valuation $\nu \in \mathbb{R}_{\geq 0}$ defines a boolean valuation for $\Phi(S \cup X)$ as follows:

$M \models_\nu s$ iff $(s, \nu) \in M$; $M \models_\nu x \in I$ iff $\nu \in I$; $M \models_\nu x.s$ iff $(s, 0) \in M$

The conjunctions, disjunctions, $\top$ and $\bot$ are are handled in the usual way. We say that M is a minimal model of $\varphi \in \Phi(S \cup X)$ with respect to ν iff $M \models_\nu \varphi$ and no proper subset M' of M is such that $M' \models_\nu \varphi$.

A configuration of a 1-ATA is a set of states. Given a configuration $\mathcal{C} = \{(s, \nu) \mid s \in S, \nu \in \mathbb{R}_{\geq 0}\}$, we denote by $\mathcal{C}+t$ the configuration $\{(s, \nu+t) \mid (s, \nu) \in \mathcal{C}\}$, obtained after a time elapse t, i.e., when t is added to all the valuations in $\mathcal{C}$. Let $\alpha \in \Gamma$. Given any configuration $\mathcal{C}_x = \{(s_j, \nu_j) | j \in J\}$, we say that any configuration $\mathcal{C}_y$ is a 'α-discrete successor' of $\mathcal{C}_x$ if and only if $\mathcal{C}_y$ can be constructed using $\mathcal{C}_x$ by choosing a minimal model M_j of $\delta(s_j, g)$ with respect to ν_j for every $j \in J$ followed by taking their union. That is, $\mathcal{C}_y = \bigcup_{j \in J} M_j$, where M_j is a minimal model of $\delta(s_j, g)$ with respect to ν_j. Given a timed word $\rho = (\alpha_0, 0)(\alpha_1, t_1) \ldots (\alpha_m, t_m)$, a run associated with ρ over a given 1-ATA starts from the initial configuration $\mathcal{C}_0 = \{(s_0, 0)\}$ and has the form $\mathcal{C}_0 \xrightarrow{g_0} \mathcal{C}_1 \xrightarrow{t_1 - 0} \mathcal{C}_1 + t_1 \ldots \mathcal{C}_{m-1} + (t_m - t_{m-1}) \xrightarrow{g_m} \mathcal{C}_m$ and proceeds with alternating time elapse transitions and discrete transitions reading a symbol from Σ. A discrete transition $\mathcal{C}_i + (t_i - t_{i-1}) \xrightarrow{\alpha_i} \mathcal{C}_{i+1}$ is included if and only if $\mathcal{C}_{i+1}$ is a α_i-discrete successor of $\mathcal{C}_i + (t_i - t_{i-1})$. Note that there could be more than one α_i-discrete successors and hence more than one run could be associated with a timed word. Note that if at any point in the run there is no discrete successor for a discrete transition, the automaton gets stuck and that run will not be associated with the word. A configuration $\mathcal{C}$ is accepting iff for all $(s, \nu) \in \mathcal{C}$, $s \in F$. Note that the empty configuration is also an accepting configuration.

The language accepted by a 1-ATA $\mathcal{A}$, denoted $L(\mathcal{A})$ is the set of all timed words ρ such that, there exists a run associated with ρ which ends at an accepting configuration.

▶ **Example 5.** Let $\Gamma{=}2^{\{a,b\}} \setminus \emptyset$. Consider the 1-ATA $\mathcal{A}{=}(\Gamma, \{t_0,t_1,t_2\}, t_0, \{t_0,t_2\}, \delta_{\mathcal{A}})$ with transitions $\delta_{\mathcal{A}}(t_0,\{b\}){=}t_0$, $\delta_{\mathcal{A}}(t_0,\{a\}){=}(t_0 \wedge x.t_1) \vee t_2$, $\delta_{\mathcal{A}}(t_1,\{a\}){=}(t_1 \wedge x{<}1) \vee (x{>}1){=}\delta_{\mathcal{A}}(t_1,\{b\})$, and $\delta_{\mathcal{A}}(t_2,\{b\}){=}t_2, \delta_{\mathcal{A}}(t_2,\{a\}){=}\bot$, $\delta_{\mathcal{A}}(t,\{a,b\}){=}\bot$ for $t \in \{t_0,t_1,t_2\}$. The automaton accepts all strings where $\{a,b\}$ does not occur, and every non-last $\{a\}$ has no symbols at distance 1 from it, and has some symbol at distance > 1 from it.

1-ATA are closed under boolean operations using well-known constructions. One additional operation which is repeatedly used in this paper can be designated as $A_1[tr \wedge A_2]$. Here A_1, A_2 are 1-ATA over the same alphabet and $tr = \delta_1(s,a) = \phi$ is a transition of A_1. The aim is to conjunctively start the automaton A_2 on taking transition tr. Formally, transition tr is replaced by $tr' = \phi \wedge \delta_2(s^2_{init}, a)$ where s^2_{init} is the start state of A_2. Semantically, runs of A_2 must now commence with a time delay after tr. The delayed run defined below defines such an execution of A_2.

Time-Delayed Runs & Acceptance. Given a timed word $\rho = (\alpha_0,0)(\alpha_1,t_1)\ldots(\alpha_m,t_m)$, a time delayed run associated with ρ over a given 1-ATA starts from the initial configuration $\mathcal{C}'_0 = \{(s_0,0)\}$ and has the form $\mathcal{C}'_0 \xrightarrow{t_1} \mathcal{C}'_0 + t_1 - 0 \xrightarrow{\alpha_1} \mathcal{C}'_1 \xrightarrow{t_2-t_1} \mathcal{C}'_1 + (t_2-t_1) \cdots \xrightarrow{\alpha_m} \mathcal{C}'_m$, where $\mathcal{C}'_{i-1} + (t_i - t_{i-1}) \xrightarrow{\alpha_i} \mathcal{C}'_i$ is included if and only if $\mathcal{C}'_i$ is one of the α_i-discrete successors of $\mathcal{C}'_{i-1} + (t_i - t_{i-1})$. Note that the only difference between a run and a time-delayed run is that the time delayed run starts with the time elapse between the first and the second symbol of the timed word and ignores the first symbol.

We say that any timed word ρ is accepted in time-delayed semantics by an ATA $\mathcal{A} = (\Gamma, S, s_0, F, \delta)$, if and only if, there exists a time-delayed run associated with ρ which ends with an accepting configuration. The set of all timed words accepted by time -delayed semantics of $\mathcal{A}$ is $L^{\mathsf{st}}(\mathcal{A})$.

▶ **Example 6.** Let $\Gamma = 2^{\{a,b\}} \setminus \emptyset$. Let α and α_b be any symbol in Γ and $\Gamma \setminus \{\{a,\}\}$,respectively. Consider the 1-ATA $\mathcal{A}{=}(\Gamma, \{s_0,s_1,s_2,s_3\}, s_0, \{s_2\}, \delta_{\mathcal{A}})$ with transitions $\delta_{\mathcal{A}}(s_0,\alpha){=}s_1$; $\delta_{\mathcal{A}}(s_1,\alpha_b){=}[s_2 \wedge x \in (1,2)] \vee [s_3 \wedge x \in (0,1)]$; $\delta_{\mathcal{A}}(s_1,\{a\}){=}s_3$; $\delta_{\mathcal{A}}(s_2,\alpha){=}s_2$; $\delta_{\mathcal{A}}(s_3,\alpha){=}s_3$. Consider a timed word $\rho = (\{a\},0)(\{b\},0.5)(\{b\},1.2)$. The run associated with ρ is : $\{(s_0,0)\} \xrightarrow{\{a\}} \{(s_1,0)\} \xrightarrow{0.5} \{(s_1,0.5)\} \xrightarrow{\{b\}} \{(s_3,0.5)\} \xrightarrow{0.7} \{(s_3,1.2)\} \xrightarrow{b} \{(s_3,1.2)\}$ Time delayed run associated with the same word is: $\{(s_0,0)\} \xrightarrow{0.5} \{(s_0,0.5)\} \xrightarrow{\{b\}} \{(s_1,0.5)\} \xrightarrow{0.7} \{(s_2,1.2)\} \xrightarrow{b} \{(s_2,1.2)\}$. Thus ρ is not accepted by the automaton in usual semantics, but accepted in time-delayed semantics.

We will define some terms which will be used in sections 4, 5. Consider a transition $\delta(s,a) = C_1 \vee \cdots \vee C_n$ in the 1-ATA. Each C_i is a conjunction of $x \in I$, locations p and $x.p$. We say that p is *free* in C_i if there is an occurrence of p in C_i and no occurrences of $x.p$ in C_i; if C_i has an $x.p$, then we say that p is *bound* in C_i. We say that p is bound in $\delta(s,a)$ if it is bound in some C_i.

Expressive Completeness and Equivalence. Let F_i be a logic or automaton class i.e. a collection of formulae or automata describing/accepting finite timed words. For each $\phi \in F_i$ let $L(F_i)$ denote the language of F_i. We define $F_1 \subseteq_e F_2$ if for each $\phi \in F_1$ there exists $\psi \in F_2$ such that $L(\phi) = L(\psi)$. Then, we say that F_2 is expressively complete for F_1. We also say that F_1 and F_2 are expressively equivalent, denoted $F_1 \equiv_e F_2$, iff $F_1 \subseteq_e F_2$ and $F_2 \subseteq_e F_1$.

3 A Normal Form for 1-ATA

In this section, we establish a normal form for 1-ATA, which plays a crucial role in the rest of the paper. Let $\mathcal{A} = (\Gamma, S, s_0, F, \delta)$ be a 1-ATA. $\mathcal{A}$ is said to be in normal form if and only if

- The set of locations S is partitioned into two sets S_r and S_{nr}. The initial state $s_0 \in S_r$.
- The locations of S are partitioned into $P_1, \ldots, P_k$ satisfying the following: Each P_i has a unique header location $s_i^r \in S_r$. Also, $P_i - \{s_i^r\} \subseteq S_{nr}$. Moreover, for any transition of $\mathcal{A}$ of the form $\delta(s, a) = C_1 \vee C_2 \ldots \vee C_k$ with $C_i = x \in I \wedge p_1 \wedge \ldots \wedge p_m \wedge x.q_i \wedge \ldots \wedge x.q_r$ we have (a) each $q_i \in S_r$, and (b) If $s \in P_i$ then each $p_j \in P_i - \{s_i^r\}$. [1]

We refer to each partition P_j as an *island* of locations. Each island has a unique header (obtained on reset) location s_i^r. All transitions into P_j occur only to this unique header location, and only with reset of clock x. Moreover, all non-reset transitions stay in the same island until a clock is reset, at which point, the control extends to the header location of same or another island (this behaviour can be seen on each path of the run tree).

Establishing the Normal Form. The main result of this section is that every 1-ATA $\mathcal{A}$ can be normalized, obtaining a language equivalent 1-ATA, $\mathsf{Norm}(\mathcal{A})$. The key idea behind this is to duplicate locations of $\mathcal{A}$ such that the conditions of normalization are satisfied. Let the set of locations of $\mathcal{A}$ be $S = \{s_1, \ldots, s_n\}$. For each location $s_i, 1 \leq i \leq n$, create a reset copy denoted s_i^r as well as n non-reset copies $s_i^{nr,j}, 1 \leq j \leq n$. The superscript r on a location represents that all incoming transitions to it are on a clock reset, while superscripts nr, j represent that all incoming transitions to that location are on non-reset and it belongs to island P_j. If s_0 is the initial location of $\mathcal{A}$, then the initial location of $\mathsf{Norm}(\mathcal{A})$ is s_0^r. The island P_i in $\mathsf{Norm}(\mathcal{A})$ consists of locations $s_i^r, s_h^{nr,i}$ for $1 \leq h \leq n$; entry into P_i happens through the header s_i^r. A transition $\delta(s_i, a) = \varphi$ of $\mathcal{A}$ is rewritten in $\mathsf{Norm}(\mathcal{A})$ by replacing all occurrences of locations $x.s_j$ with $x.s_j^r$ (leading into P_j), while occurrences of free locations s_h are replaced with $s_h^{nr,i}$. The final locations of $\mathsf{Norm}(\mathcal{A})$ are $s_i^r, s_i^{nr,j}$ for $1 \leq j \leq n$ whenever s_i is a final location in $\mathcal{A}$. The full version gives a formal proof for the following straightforward lemma. Thanks to lemma 7, we assume without loss of generality, in the rest of the paper, that 1-ATA are in normal form.

▶ **Lemma 7.** *Given a 1-ATA $\mathcal{A}$, one can construct a 1-ATA $\mathsf{Norm}(\mathcal{A})$ in normal form such that $L(\mathcal{A}) = L(\mathsf{Norm}(\mathcal{A}))$.*

▶ **Example 8.** The 1-ATA $\mathcal{B} = (\{a, b\}, \{s_0, s_1, s_2\}, s_0, \{s_1\}, \delta)$ with transitions $\delta(s_0, b) = x.s_2, \delta(s_0, a) = (s_0 \wedge x.s_1)$, $\delta(s_1, a) = (s_1 \wedge s_0) = \delta(s_1, b)$ and $\delta(s_2, b) = x.s_0, \delta(s_2, a) = (s_2 \wedge x.s_1)$ is not in normal form. Following the normalization technique, we obtain $\mathsf{Norm}(\mathcal{B})$ with locations $S = \{s_i^r, s_j^{nr,i} \mid 0 \leq i, j \leq 2\}$ and final locations $\{s_1^r, s_1^{nr,0}, s_1^{nr,1}, s_1^{nr,2}\}$. Figure 1(right) describes $\mathsf{Norm}(\mathcal{B})$ consisting of islands P_0, P_1 and P_2.

4 1-ATA-rfl and Logics

In this section, we show the first of our expressive equivalence results connecting logics RatMTL, QkMSO and a subclass of 1-ATA called 1-ATA with reset-free loops (1-ATA-rfl).We first introduce 1-ATA-rfl. A 1-ATA $\mathcal{A}$ (in normal form) is said to be a 1-ATA-rfl if it satisfies

[1] If the transitions of the 1-ATA are presented in CNF, the equivalent restriction is that each free location in the transition should be in $P_i - \{s_i^r\}$ while each bound location should be in S_r.

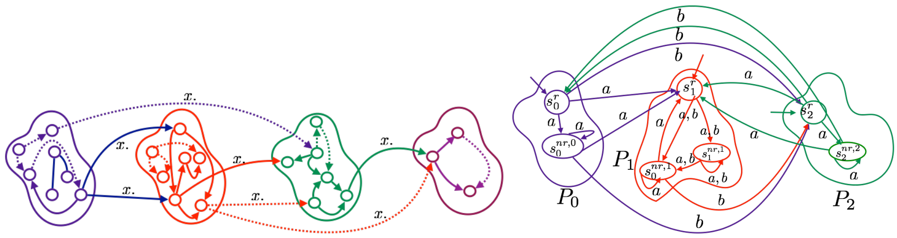


Figure 1 left: strict island hopping. right: $\mathsf{Norm}(\mathcal{B})$ corresponding to $\mathcal{B}$ in Example 8.

the following: There is a partial order $(S_r, \preceq)$ on the header locations (equivalently, islands P_i). Moreover, for any location $p \in P_i$ and a location q, if $x.q$ occurs in $\delta(p,a)$ for any a (hence $q = s_j^r$) then $s_j^r \prec s_i^r$. Thus, islands (which are only connected by reset transitions) form a DAG, and every reset transition goes to a lower level island. See Figure 1 (left), where we call this *strict island hopping*. The colored islands are all disjoint. On non-reset transitions, control stays in the same island; on resets, it may expand to another island. From this finite control, you cannot go back to the island where you started from. This provides a partial order between islands due to resets (the name rfl comes from here). Semantically, this means that on any branch of a run tree, a reset transition occurs at most once.

▶ **Example 9.** The 1-ATA with locations s, p, q and transitions $\delta(s,\alpha){=}(x.p \wedge x \leq 1) \vee (q \wedge x{=}2)$, $\delta(p,\alpha){=}x.q \wedge p$ and $\delta(q,a){=}s \wedge (0{<}x{<}1)$ is not 1-ATA-rfl, since q is bound in $\delta(p,\alpha)$ and starting from q, we can reach $x.p$ via s.

4.1 Useful Lemmas

In this section, we introduce some notations and prove lemmas (Lemma 10, Lemma 11) which will be used several times in the paper. Let $\rho = (a_0, 0)(a_1, \tau_1)(a_2, \tau_2)\ldots(a_m, \tau_m)$ be a timed word and let $i \in dom(\rho)$. Let $\mathsf{rel}(\rho, i)$ denote the timed word obtained from ρ by relativizing at position i; $\mathsf{rel}(\rho, i) = (a_i, 0)(a_{i+1}, \tau_{i+1} - \tau_i)(a_{i+2}, \tau_{i+2} - \tau_i)\ldots(a_m, \tau_m - \tau_i)$.

Region Words. Let c_{max} be any non-negative integer. Let $reg_{c_{max}} = \{0, (0,1), \ldots, c_{max}, (c_{max}, \infty)\}$ be the set of regions. Given a finite alphabet Σ, with $\Gamma = 2^{\Sigma} \backslash \emptyset$, a *region word* is a word over the alphabet $\Gamma \times reg_{c_{max}}$ called the *interval* alphabet. A region word $w = (a_1, I_1)\ (a_2, I_2)\ \ldots(a_m, I_m)$ is *good* iff $I_j \leq I_k \Leftrightarrow j < k$. Here, $I_j \leq I_k$ represents that either $I_j = I_k$ or the upper bound of I_j is at most the lower bound of I_k. The timed word $\rho = (a_0, 0)(a_1, \tau_1)(a_2, \tau_2)\ldots(a_m, \tau_m)$ is consistent with a region word $(a_1, I_1)(a_2, I_2)\ldots(a_m, I_m)$ iff $\tau_j \in I_j$ for all $0 < j \leq m$. Note that for technical reasons, which will be clearer later, the region abstraction of the word defined here is by ignoring the first point. The set of timed words ρ consistent with a good region word w is denoted $\mathcal{T}_w$. Likewise, given a timed word ρ, $reg(\rho)$ represents the good region word w such that $\rho \in \mathcal{T}_w$.

▶ **Lemma 10** (Untiming P to $\mathcal{A}(P)$)**.** *Let P be a 1-ATA over Γ having no resets. We can construct an alternating finite automaton $\mathcal{A}(P)$ over the interval alphabet $\Gamma \times reg_{c_{max}}$ such that for any good region word $w{=}(a_1, I_1)\ldots,(a_n, I_n)$, $w \in L(\mathcal{A}(P)) \Rightarrow \forall \rho \in \mathcal{T}_w, \rho \in L^{\mathsf{st}}(P)$. Conversely, $\rho \in L^{\mathsf{st}}(P) \Rightarrow reg(\rho) \in L(\mathcal{A}(P))$. Hence, $L^{\mathsf{st}}(P){=}\{\rho \mid w \in L(A(P) \wedge \rho \in \mathcal{T}_w\}$.*

Proof. Given a reset-free 1-ATA $P = (\Gamma, S, s_0, F, \delta)$ we construct an alternating finite automaton (AFA) $\mathcal{A}(P) = (\Gamma \times reg_{c_{max}}, S, s_0, F, \delta')$ where c_{max} is the maximum constant used in the clock constraints of P and δ' is defined using δ as follows. A clock constraint

$x \in I$ is called a *region constraint* if $I \in reg_{c_{max}}$. Without loss of generality, we assume that all the transitions of P have the form $C_1 \vee \cdots \vee C_m$, where each clause C_i has exactly one region constraint.

Construction of δ'. Given a transition $\delta(s,a) = C_1 \vee \cdots \vee C_n$ of P, let $C_{I_1}, \ldots, C_{I_k} \in \{C_1, \ldots, C_n\}$ be clauses containing the region constraint $x \in I$, $I \in reg_{c_{max}}$. We construct $\delta'(s,(a,I))$ as $C'_{I_1} \vee \cdots \vee C'_{I_k}$ where C'_{I_j} is obtained by removing the conjunct $x \in I$ from C_{I_j}. If there is no clause containing $x \in I$ in $\delta(s,a)$, then $\delta'(s,(a,I)) = \bot$.

1. For any timed word ρ, $\rho \in L^{\mathsf{st}}(P) \Rightarrow reg(\rho) \in L(\mathcal{A}(P))$: Let $\rho = (a_1,0)\ldots(a_m,\tau_m)$. The time-delayed accepting run from the initial configuration $\mathcal{C}_0 = \{(s_0,0)\}$ to an accepting configuration $\mathcal{C}_m$ in P is as follows: $\mathcal{C}_0 \xrightarrow{\tau_1} \mathcal{C}_0 + \tau_1 \xrightarrow{a_1} \mathcal{C}_1 \xrightarrow{\tau_2-\tau_1} \cdots \xrightarrow{\tau_m-\tau_{m-1}} \mathcal{C}_{m-1} + (\tau_m - \tau_{m-1}) \xrightarrow{a_m} \mathcal{C}_m$. As P is a reset-free 1-ATA, the valuation $\nu(x)$ at any point is equal to the total time elapse till that point. Hence for all positions $j \in dom(\rho)$, $I \in reg_{c_{max}}$, $\tau_j \in I$ iff $\nu(x) \in I$. By construction of $\mathcal{A}(P)$, for each transition $\delta(s,a_j) = (x \in I_j \wedge \psi) \vee C$ of P we have a transition $\delta'(s,(a_j,I_j)) = \psi$ (wlg we assume that C has no occurrence of $x \in I_j$). The accepting run of P translates into the run $\mathcal{D}_0 \xrightarrow{(a_1,I_1)} \mathcal{D}_1 \xrightarrow{(a_2,I_2)} \mathcal{D}_2 \cdots \xrightarrow{(a_m,I_m)} \mathcal{D}_m$ in $\mathcal{A}(P)$, where $\mathcal{D}_0 = \{s \mid (s,0) \in \mathcal{C}_0\}$, and $\mathcal{D}_j = \{s \mid (s,t) \in \mathcal{C}_j, t \in I_j\}$, $j \geq 1$. Since all locations in $\mathcal{C}_m$ are accepting, $\mathcal{D}_m$ is an accepting configuration in $\mathcal{A}(P)$ accepting $(a_1,I_1)\ldots(a_m,I_m)$.
2. For any good word w, $w \in L(\mathcal{A}(P)) \Rightarrow \rho \in L^{\mathsf{st}}(P)$ for all $\rho \in \mathcal{T}_w$. The argument is similar to the previous bullet. Full proof can be found in the full version. ◀

▶ **Lemma 11** ($\mathcal{A}(P)$ to RatMTL). *Let $\mathcal{A}(P)$ be an AFA over the interval alphabet $\Gamma \times reg_{c_{max}}$ constructed from a reset-free 1-ATA P as in lemma 10. We can construct a* RatMTL *formula φ^{st} such that for any good word w, $w \in L(\mathcal{A}(P)) \Rightarrow \rho \in L(\varphi^{\mathsf{st}})$ for all $\rho \in \mathcal{T}_w$, and, for any timed word ρ, $\rho \in L(\varphi^{\mathsf{st}}) \Rightarrow reg(\rho) \in L(\mathcal{A}(P))$. $L(\varphi^{\mathsf{st}}) = \{\mathcal{T}_w \mid w \in L(\mathcal{A}(P))\}$. Hence, by lemma 10, $L(\varphi^{\mathsf{st}}) = L^{\mathsf{st}}(P)$.*

Proof. Let $Det[\mathcal{A}(P)]$ be the deterministic automaton which is language equivalent to $\mathcal{A}(P)$. Let δ_D be the transition function of $Det[\mathcal{A}(P)]$ obtained from δ' (see lemma 10) and let $\hat{\delta}_D$ be its extension to words. Let s_0 be the initial location of $Det[\mathcal{A}(P)]$ and let F be the set of its final locations. For any pair of locations p,q of $Det[\mathcal{A}(P)]$ and $I_i \in reg_{c_{max}}$, we construct a regular expression $\mathsf{re}(p,q,I_i)$ equivalent to the language $\{w \in (\Gamma \times \{I_i\})^+ \mid \hat{\delta}_D(p,w) = q\}$.

Construction of $\mathsf{re}(p,q,I_i)$. Let $Det[\mathcal{A}(P)[p,q]]$ be the DFA obtained from $Det[\mathcal{A}(P)]$ by setting the initial location to p and set of final locations to $\{q\}$. Let $\mathcal{A}(I_i^*)$ denote the DFA accepting all words $(\Gamma \times \{I_i\})^*$. Let $Det[\mathcal{A}_i] = Det[\mathcal{A}(P)[p,q]] \cap \mathcal{A}(I_i^*)$. Then $L(\mathsf{re}(p,q,I_i)) = L(Det[\mathcal{A}_i])$.

Obtaining RatMTL formula. Consider a sequence of locations $\mathsf{sseq}{=}q_0,q_1,q_2\ldots q_k$ with $k{=}2*c_{max}+2$, $q_0{=}s_0$ and $q_k{\in}F$. Let $\mathsf{re}{=}\mathsf{re}(q_0,q_1,[0,0]).\mathsf{re}(q_1,q_2,(0,1)).\ \cdots.\mathsf{re}(q_{k-1},q_k,(c_{max},\infty))$ where $\mathsf{re}(q_{2i-1},q_{2i},(i-1,i)) \subseteq (\Gamma \times (i-1,i))^*$ and $\mathsf{re}(q_{2i},q_{2i+1},[i,i]) \subseteq (\Gamma \times [i,i])^*$, and $L(\mathsf{re}) \subseteq L(Det\mathcal{A}(P))$. Define the RatMTL formula $\phi(\mathsf{sseq}) = \mathsf{Rat}_{[0,0]}\mathsf{re}(q_0,q_1,[0,0]) \wedge \mathsf{Rat}_{(0,1)}\mathsf{re}(q_1,q_2,(0,1)) \wedge \cdots \wedge \mathsf{Rat}_{(c_{max},\infty)}[\mathsf{re}(q_{k-1},q_k,(c_{max},\infty))]$. Then $L(\phi(\mathsf{sseq}))=\{\mathcal{T}_w \mid w \in L(\mathsf{re}(q_0,q_1,[0,0]).\mathsf{re}(q_1,q_2,(0,1))\cdots\mathsf{re}(q_{k-1},q_k,(c_{max},\infty))\}$. Let $\varphi^{\mathsf{st}} = \vee_{\mathsf{sseq}}\ \phi(\mathsf{sseq})$[2]. Then clearly, $L(\varphi^{\mathsf{st}}) = \{\mathcal{T}_w \mid w \in L(\mathcal{A}(P))\}$. By lemma 10, we obtain $L(\varphi^{\mathsf{st}}) = L^{\mathsf{st}}(P)$. ◀

[2] The superscript st in φ^{st} represents that it is a strict future formulae. The truth of the formula at any point i is only dependent on points $j > i$

Remark. $\rho, i \models \varphi^{\mathsf{st}}$ iff $\mathsf{rel}(\rho, i) \in L^{\mathsf{st}}(P)$. $\rho, i \models \varphi^{\mathsf{st}}$ iff $\mathsf{rel}(\rho, i) \in L(\varphi^{\mathsf{st}})$ since the truth of φ^{st} only depends on the truth value of the propositions at the points strictly greater than i and the relative time difference with respect to i. The proof of Lemma 12 is in the full version.

▶ **Lemma 12.** *Let $P = (\Gamma, S, s_0, F, \delta)$ be any reset free 1-ATA and $\mathcal{A}(P)$ be an AFA as constructed in lemma 10.Let φ^{st} be the formula constructed from $\mathcal{A}(P)$ as in lemma 11.We can construct a formula $\varphi \in$ RatMTL such that $L(\varphi) = L(P)$.*

4.2 1-ATA-rfl meets RatMTL

▶ **Theorem 13.** *1-ATA-*rfl *are expressively equivalent to* RatMTL.

Proof.

1. 1-ATA-rfl $\subseteq_e$ RatMTL: Let $\mathcal{A}$ be a 1-ATA-rfl in normal form. For each location $s_i^r \in S_r$ (which is the header of partition P_i) let $\mathcal{A}[s_i^r]$ denote the same automaton as $\mathcal{A}$ except that the initial location is changed to s_i^r, the header location of P_i. We can also delete all islands higher than P_i as their locations are not reachable. For each such automaton $\mathcal{A}[s_i^r]$, we construct a pair of RatMTL formulae $\mathsf{mtl}(\mathcal{A}[s_i^r])$ and $\mathsf{mtl}^{\mathsf{st}}(\mathcal{A}[s_i^r])$ such that $L(\mathsf{mtl}(\mathcal{A}[s_i^r])) = L(\mathcal{A}[s_i^r])$ and $L(\mathsf{mtl}^{\mathsf{st}}(\mathcal{A}[s_i^r])) = L^{\mathsf{st}}(\mathcal{A}[s_i^r])$. Note that $(S_r, \preceq)$ is a partial order. The construction and proof of equivalence are by complete induction on the level of the header location s_i^r of island P_i in the partial order. All $x.s_j^r$ occurring in any transition of $\mathcal{A}[s_i^r]$ are of lower level in the partial order $(S_r, \preceq)$. Hence, by induction hypothesis, there is a RatMTL formula $\psi_j^{\mathsf{st}} = \mathsf{mtl}^{\mathsf{st}}(\mathcal{A}[s_j^r])$ such that $L^{\mathsf{st}}(\mathcal{A}[s_j^r]) = L(\psi_j^{\mathsf{st}})$. The behaviour of all the lower level s_j^r is independent of the label of the transition that calls them. In other words, the automata $\mathcal{A}_i[s_j^r]$ which is called at position i does not read the symbol at point i. Thus any lower level automaton starting from some s_j^r called at a point i restricts the behaviour of the points strictly in the future of i, fixing the anchor (the point with respect to which the time differences are checked) at i. Let w_j be a fresh witness variable for each $x.s_j^r$ above, which also corresponds to RatMTL formula ψ_j^{st}. Let the set of such witness variables be $\{\mathsf{w}_1, \ldots, \mathsf{w}_k\}$. We construct a modified automaton $\mathcal{A}^{\mathsf{w}}[s_i^r]$ with transition function δ' and set of locations P_i as follows. Its alphabet is $\Gamma \times \{0,1\}^k$ with the jth component giving the truth value of witness w_j. Let $\delta'(s, a, \mathsf{w}_1, \ldots, \mathsf{w}_k) = \delta(s, a)[\mathsf{w}_j / x.s_j^r]$, i.e., each occurrence of $x.s_j^r$ is replaced by the truth value of w_j for $1 \leq j \leq k$.Note that $\mathcal{A}^{\mathsf{w}}[s_i^r]$ is a reset-free 1-ATA. By lemmas 10, 11, we get a RatMTL formula $\phi^{\mathsf{st,w}}$ such that $L^{\mathsf{st}}(\mathcal{A}^{\mathsf{w}}[s_i^r]) = L(\phi^{\mathsf{st,w}})$ and using lemma 12 we get a language equivalent formula ϕ^{w} over the variables $\Sigma \cup \{\mathsf{w}_1, \ldots, \mathsf{w}_k\}$. Now we substitute each w_j by ψ_j^{st} (and hence $\neg\mathsf{w}_j$ by $\neg\psi_j^{\mathsf{st}}$) in ϕ^{w} and $\phi^{\mathsf{st,w}}$ to obtain the required formulae $\mathsf{mtl}(\mathcal{A}[s_i^r])$ and $\mathsf{mtl}^{\mathsf{st}}(\mathcal{A}[s_i^r])$, respectively. It is clear from the substitution that $L^{\mathsf{st}}(\mathcal{A}[s_i^r]) = L(\mathsf{mtl}^{\mathsf{st}}(\mathcal{A}[s_i^r]))$ and $L(\mathcal{A}[s_i^r]) = L(\mathsf{mtl}(\mathcal{A}[s_i^r]))$. An example illustrating this construction is in the full version
2. RatMTL $\subseteq_e$ 1-ATA-rfl: Consider a formula $\psi_1 = \mathsf{Rat}_I(\mathsf{re}_0)$ with $I = [l, u)$. The case of other intervals are handled similarly. For the base case, ψ_1 has modal depth 1 and has a single modality. As the formula is of modal depth 1, re_0 is an atomic regular expression over alphabet 2^Σ. Let $D = (\Gamma, Q, q_0, Q_f, \delta')$ be a DFA such that $L(D) = L(\mathsf{re}_0)$, with $\Gamma = 2^\Sigma \backslash \emptyset$. From D, we construct the 1-ATA $\mathcal{A}$=$(\Gamma, Q \cup \{q_{init}, q_{\mathsf{timecheck}}, q_f\}, q_{init}, \{q_f\}, \delta)$ where $q_{init}, q_{\mathsf{timecheck}}, q_f$ are disjoint from Q. The transitions δ are as follows. Assume $l > 0$.
   - $\delta(q_{init}, a) = x.q_{\mathsf{timecheck}}, a \in \Gamma$,
   - $\delta(q_{\mathsf{timecheck}}, a) = [(x \geq l \wedge \delta'(q_0, a) \vee (q_{\mathsf{timecheck}})] \vee [x > u \wedge q_f]$ where the latter disjunct is added only when $q_0 \in Q_f$,
   - $\delta(q, a) = (x \in [l, u)) \wedge \delta'(q, a))$, for all $q \in Q \setminus Q_f$,

- $\delta(q,a) = (x \in [l,u) \wedge \delta'(q,a)) \vee (x > u \wedge q_f)$, for all $q \in Q_f$, $\delta(q_f,a) = q_f$.

It is easy to see that $\mathcal{A}$ has the reset-free loop condition since q_f is the only location entered on resets, and control stays in q_f once it enters q_f. The correctness of $\mathcal{A}$ is easy to establish, the location $q_{\mathsf{timecheck}}$ is entered on the first symbol, resetting the clock; control stays in $q_{\mathsf{timecheck}}$ as long as $x < l$, and when $x \geq l$, the DFA is started. As long as $x \in [l,u)$, we simulate the DFA. If $x > u$ and we are in a final location of the DFA, the control switches to the final location q_f of $\mathcal{A}$. If q_0 is itself a final location of the DFA, then from $q_{\mathsf{timecheck}}$, we enter q_f when $x > u$. It is clear that $\mathcal{A}$ indeed checks that re_0 is true in the interval $[l,u)$. If $l = 0$, then the interval on which re_0 should hold good is $[0,u)$. In this case, if q_0 is non-final, we have the transition $\delta(q_{init},a) = x.\delta'(q_0,a), a \in \Gamma$ (since our timed words start at time stamp 0, the first symbol is read at time 0, so $x.\delta'(q_0,a)$ preserves the value of x after the transition $\delta'(q_0,a)$). The location $q_{\mathsf{timecheck}}$ is not used then. The case when ψ_1 has modal depth 1 but has more than one Rat modality is dealt as follows. Firstly, if $\psi_1 = \neg\mathsf{Rat}_I(\mathsf{re}_0)$, then the result follows since 1-ATA-rfl are closed under complementation (the fact that the resets are loop-free on a run does not change when one complements). For the case when we have a conjunction $\psi_1 \wedge \psi_2$ of formulae, having 1-ATA-rfl $\mathcal{A}_1 = (\Gamma, Q_1, q_1, F_1, \delta_1)$ and $\mathcal{A}_2 = (\Gamma, Q_2, q_2, F_2, \delta_2)$ such that $L(\mathcal{A}_1) = L(\psi_1)$ and $L(\mathcal{A}_2) = L(\psi_2)$, we construct $\mathcal{A} = (\Gamma, Q_1 \cup Q_2 \cup \{q_{init}\}, q_{init}, F, \delta)$ such that $\delta(q_{init},a) = x.\delta_1(q_1,a) \wedge x.\delta_2(q_2,a)$. Clearly, $\mathcal{A}$ is a 1-ATA-rfl since $\mathcal{A}_1, \mathcal{A}_2$ are. It is easy to see that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. The case when $\psi = \psi_1 \vee \psi_2$ follows from the fact that we handle negation and conjunction. The case of formulae of higher modal depth $\psi_{k+1} = \mathsf{Rat}_I(\mathsf{re}_k)$ is handled by substituting all lower depth formulae in re_k with witness variables, and using the inductive hypothesis that there exist 1-ATA-rfl equivalent to these. The main argument is then to show that on plugging-in these automata corresponding to the witnesses, we obtain a 1-ATA-rfl equivalent to ψ_{k+1}. Details in the full version. ◀

4.3 RatMTL meets QkMSO

▶ **Theorem 14.** QkMSO *is expressively equivalent to* RatMTL*.*

Proof.

1. QkMSO $\subseteq_e$ RatMTL: Proof is by Induction on the metric depth of the formula. For the base case,consider a formula $\psi(t_0) = \mathcal{Q}_1 t_1 \ldots \mathcal{Q}_{k-1} t_{k-1} \varphi(\downarrow t_0, t_1, \ldots, t_{k-1})$ of metric depth one.Let c_{max} be the maximal constant used in the metric quantifiers $\mathcal{Q}_i$. Let $R_j(t)$ for j in reg=$\{0,(0,1),1,\ldots,c_{max},(c_{max},\infty)\}$ be fresh monadic predicates. We modify $\psi(t_0)$ to obtain an untimed MSO formula $\psi_{rg}(t_0)$ over the alphabet $2^\Sigma \times \{0,1\}^{|reg|} \times \{0,1\}$ as follows. Define $CON(I_i,t_i) = \vee\{R_j(t_i) \mid j \subseteq I_i\}$. We replace every quantifier $\exists t_i \in t_0 + I_i\ \phi$ by $\exists t_i (t_0 \leq t_i) \wedge CON(I_i,t_i) \wedge \phi$. Every quantifier $\forall t_i \in t_0 + I_i\ \phi$ is replaced by $\forall t_i.(t_0 \leq t_i \wedge CON(I_i,t_i) \rightarrow \phi)$. To the resulting MSO formula we add a conjunct WELLREGION that states that (a) exactly one $R_j(t)$ holds at any t, and (b) $\forall t,t'.\ [t<t' \wedge R_j(t) \wedge R_{j'}(t')] \rightarrow j \leq j'$ (asserting region order). Note that these are natural properties of region abstraction of time. This gives us the formula $\psi_{rg}(t_0)$. It has predicates $R_j(t)$ for $j \in reg$ and free variable t_0. Being an MSO formula, we can construct a DFA $\mathcal{A}(\psi_{rg}(0))$ for it over the alphabet $2^\Sigma \times \{0,1\}^{|reg|}$. Note that we have substituted 0 for t_0. This is isomorphic to automaton over the alphabet $2^\Sigma \times reg$. From the construction, it is clear that $\rho \models \psi(0)$ iff $reg(\rho) \in L(\mathcal{A}(\psi_{rg}(0)))$. By Lemma 12, we then obtain an equivalent RatMTL formula ζ. It is easy to see that $L(\psi(0)) = L(\zeta)$. Because $\psi(0)$ and ζ are purely future time formulae, this gives $\rho, i \models \psi(t_0)$ iff $\rho, i \models \zeta$. For the induction step, consider a metric depth $n+1$ formula $\psi(t_0)$. We can replace every time constraint sub-formula

$\psi_i(t_k)$ occurring in it by a witness monadic predicate $w_i(t_k)$. This gives a metric depth 1 formula and we can obtain a RatMTL formula, say ζ, over variables $\Sigma \cup \{w_i\}$ exactly as in the base step. Notice that each $\psi_i(t_k)$ was a formula of modal depth n or less. Hence by induction hypothesis we have an equivalent RatMTL formula ζ_i. Substituting ζ_i for w_i in ζ gives us a formula language equivalent to $\psi(t_0)$.

2. RatMTL $\subseteq_e$ QkMSO : Let $\varphi \in$ RatMTL. The proof is by induction on the modal depth of φ. For the base case, let $\varphi = \mathsf{Rat}_I(\mathsf{re})$ where re is a regular expression over propositions. Let $\zeta(x, y)$ be an MSO formula with the property that $\sigma, i, j \models \zeta(x, y)$ iff $\sigma[x : y] \in L(re)$, where $\sigma[x : y]$ denotes the sub-string $\sigma(x+1) \dots \sigma(y)$. Since MSO has exactly the expressive power of regular languages, such a formula can always be constructed. Consider $\psi(t_0)$: $\overline{\exists} t_{first} \in t_0 + I.\ \overline{\exists} t_{last} \in t_0 + I. \overline{\forall} t' \in t_0 + I. [(t' = t_{first} \vee t' = t_{last} \vee t_{first} < t' < t_{last}) \wedge \zeta(t_{first}, t_{last})]$ Then, it is clear that $\rho, i \models \varphi$ iff $\rho, i \models \psi(t_0)$. Note that the time constraint formula $\psi(t_0)$ is actually a formula of QkMSO with $k = 4$ using time constrained variables $t_0, t_{first}, t_{last}, t'$ only. Atomic and boolean constructs can be straightforwardly translated. Now let $\varphi = \mathsf{Rat}_I(\mathsf{re})$ where re is over a set of subformulae S. For each $\zeta_i \in S$, substitute it by a witness proposition w_i to get a formula φ_{flat}. This is a modal depth 1 formula and we can construct a language equivalent formula of QkMSO, say $\Xi(t_0)$ over alphabet $\Sigma \cup \{w_i\}$. By induction hypothesis, for each ζ_i there exists a language equivalent time constrained QkMSO formula $\kappa_i(t_0)$. Now substitute $\kappa_i(t_j)$ for each occurrence of $w_i(t_j)$ in $\Xi(t_0)$ to get a formula $\psi(t_0)$. Then $\psi(t_0)$ is language equivalent to φ. Note that $\psi(t_0) \in$ Q4MSO as it only uses time constrained variables $t_0, t_{first}, t_{last}, t'$ which are inductively reused. ◀

5 C⊕D-1-ATA-rfl and Logics

In this section, we show our second main result connecting logics FRatMTL, Q2MSO and a subclass of 1-ATA called *conjunctive-disjunctive* (abbreviated C⊕D) 1-ATA with reset-free loops. Let $\mathcal{A} = (\Gamma, Q, q_0, F, \delta)$ be a 1-ATA. Let $Q_x = \{x.q \mid q \in Q\}$ and let $\mathcal{B}(Q_x) ::= \mathsf{true} | \mathsf{false} | \alpha | \alpha \wedge \alpha | \alpha \vee \alpha$, where $\alpha \in Q_x$. $\mathcal{A}$ is said to be a C⊕D 1-ATA if

1. Q is partitioned into $Q_\wedge$ and $Q_\vee$,
2. Let $q \in Q_\wedge$. Transitions $\delta(q, a)$ can be written as $D_1 \wedge D_2 \wedge \dots \wedge D_m$, where any D_i has one the following forms. (i) $D_i = q' \vee \mathcal{B}(Q_x)$ where $q' \in Q_\wedge$, (ii) $D_i = x \notin I \vee \mathcal{B}(Q_x)$.[3] Each D_i has at most one free location from $Q_\wedge$, or at most one clock constraint $x \notin I$.
3. Let $q \in Q_\vee$. Transitions $\delta(q, a)$ can be written as $C_1 \vee C_2 \vee \dots \vee C_m$ where any C_i has one of the following forms. (i) $C_i = q' \wedge \mathcal{B}(Q_x)$, where $q' \in Q_\vee$, (ii) $C_i = x \in I \wedge \mathcal{B}(Q_x)$. Thus, each C_i has at most one free location from $Q_\vee$, or at most one clock constraint $x \in I$.

In other words, all the non-reset transitions coming out of $q \in Q_\wedge$ will go to all the locations in $Q' \subseteq Q_\wedge$ conjunctively asserting some time constraint. Similarly, all the non-reset transitions coming out of $q \in Q_\vee$ will non-deterministically go to one of the locations in $Q_\vee$, without checking any time constraint.

Remark. If any C⊕D-1-ATA A is normalized to Norm(A), then any island of Norm(A) is exclusively made of locations from $Q_\wedge$ or $Q_\vee$ (not both). Note that if we delete all the reset transitions from any island of Norm(A), all the transitions within the island will be either conjunctive or disjunctive. Thus, the name C⊕D is based on the fact that each island of Norm$(\mathcal{A})$ is either conjunctive or disjunctive. A 1-ATA which has both reset-free loops and

[3] Note that $x \notin I$ can be re-written as $x \in [0, l\rangle \vee x \in \langle u, \infty)$, where $[0, l\rangle \cup \langle u, \infty)$ is the complement of the interval I with lower and upper bounds l, u respectively. We restrict the specification of time intervals in this form to make sure that, using conjunctions and non-punctual intervals, one cannot express punctual intervals. This is used to define non-punctual C⊕D-1-ATA.

conjunctive-disjunctiveness is denoted C⊕D-1-ATA-rfl. If all the intervals in a C⊕D-1-ATA are non-punctual, then it is referred to as a np-C⊕D-1-ATA.

▶ **Example 15.** We illustrate examples of 1-ATA violating the C⊕D condition.

(a) For $a \in \Sigma$, let S_a and $S_{\neg a}$ denote any set containing a and not containing a, respectively. Consider the automaton $\mathcal{B}$ with transitions $\delta(s_0, S_a)=s_0 \vee x > 1$, $\delta(s_0, S_{\neg a})=s_0 \wedge x \notin (1, \infty)$, where s_0 is the only location, which is non-final. The only way to accept a word is by reaching an empty configuration. The C⊕D condition is violated due to the combination of having a free location and a clock constraint simultaneously in a clause irrespective of $s_0 \in Q_\vee$ or $s_0 \in Q_\wedge$. This accepts the set of all words where the first symbol in $(1, \infty)$ has an a.

(b) Let S_a, $S_{\neg a}$ be as above. The automaton $\mathcal{B}$ with $\delta(s_0, S_a)=s_0 \vee s_1$, $\delta(s_2, \Gamma) = x \leq 1$, $\delta(s_0, S_{\neg a})=s_0 \wedge s_2$, $\delta(s_1, \Gamma)=x > 1$ with s_0 being initial and none of the locations being final satisfies rfl but violates C⊕D. The C⊕D condition is violated since a clause contains more than one free location irrespective of $s_0, s_1, s_2 \in Q_\vee$ or $s_0, s_1, s_2 \in Q_\wedge$. This accepts the language of all words where the last symbol in $(0, 1)$ has an a.

(c) The automaton $\mathcal{B}$ with $\delta(s_0, \Gamma)=s_0 \vee s_1$, $\delta(s_1, S_a) = s_2 \wedge s_3$, $\delta(s_1, S_{\neg a}) = \bot$, $\delta(s_2, \Gamma)=(x \leq 1)$, $\delta(s_3, \Gamma)=s_4$, $\delta(s_4, \Gamma)=x > 1$ with s_0 being initial and s_1 being final satisfies rfl but violates C⊕D. The C⊕D condition is violated since the automata switches between conjunctive and disjunctive locations without any reset. Note that $s_0 \in Q_\vee$ while $s_1 \in Q_\wedge$. The language accepted is all words where the second last symbol in $(0, 1)$ has an a.

5.1 C⊕D-1-ATA-rfl meets FRatMTL

▶ **Theorem 16.** C⊕D-*1-ATA*-rfl *are expressively equivalent to* FRatMTL.

Proof. We only detail the containment C⊕D-1-ATA-rfl $\subseteq_e$ FRatMTL; the converse direction is almost identical to the proof of RatMTL $\subseteq_e$1-ATA$-$rfl and is provided in the full version.

1. C⊕D-1-ATA-rfl $\subseteq_e$ FRatMTL : The first thing is to convert C⊕D 1-ATA with no resets to FRat formula of modal depth 1 as in Lemma 17.

   ▶ **Lemma 17.** *Given a* C⊕D *1-ATA* $\mathcal{A}$ *over* Σ *with no resets, we can construct a* FRat *formula* φ *such that for any timed word* $\rho = (a_1, \tau_1) \dots (a_m, \tau_m)$, $\rho, i \models \varphi^{\mathsf{st}}$ *iff* $\mathcal{A}$ *accepts* $\mathsf{rel}(\rho, i)$ *in time-delayed semantics.* $L(\varphi^{\mathsf{st}}) = L^{\mathsf{st}}(\mathcal{A})$.

   Assuming $q_0 \in Q_\vee$, the key idea is to check how an accepting configuration is reached. The reset-freeness ensures that any transition $\delta(q, a) = C_1 \vee \dots \vee C_m$ is such that C_i is either a location or a clock constraint $x \in I$. Assume time-delayed acceptance happens through an empty configuration via a clock constraint $x \in I_a$, from some location q on an a, and q is reachable from q_0. Let re_{I_a} be the regular expression whose language is the set of all such words reaching some q, from where acceptance happens via interval I_a on an a. The formula $\mathsf{FRat}_{I_a, \mathsf{re}_{I_a}} a$ sums up all such words. Disjuncting over all possible intervals and symbols, we have the result. The second case is when a final state q_f is reached from some q' reachable from q_0. If $\mathsf{re}_{q_f, a}$ is the regular expression whose language is all words reaching such a q', the formula $\mathsf{FRat}_{[0,\infty), \mathsf{re}_{q_f,a}}(a \wedge \Box\bot)$ sums up all words accepted via q', a, q_f.The $\Box\bot$ ensures that no further symbols are read, and can be written as $\neg\mathsf{FRat}_{[0,\infty),\Sigma^*} \top$. Disjuncting over all possible final states q_f and $a \in \Sigma$ gives us the formula. The case when $q_0 \in Q_\wedge$ is handled by negating the automaton, obtaining $q_0 \in Q_\vee$ and negating the resulting formula. Details in the full version Like lemma 12, given any C⊕D-1-ATA-rfl $\mathcal{A}$, we can construct $\varphi \in$ FRatMTL such that $L(\varphi) = L(\mathcal{A})$. The rest of the proof is very similar to Theorem 13(1) and omitted.

2. FRatMTL $\subseteq_e$ C$\oplus$D-1-ATA-rfl: This is almost identical to the proof of Theorem 13(2), and is provided in the full version for completeness. ◀

5.2 FRatMTL meets Q2MSO

▶ **Theorem 18.** FRatMTL *is expressively equivalent to* Q2MSO.

Proof.

1. Q2MSO $\subseteq_e$ FRatMTL : We first consider formulae of metric depth one. These have the form $\psi(t_0) = \mathcal{Q}_1 t_1 \varphi(\downarrow t_0, t_1)$ and $\varphi(\downarrow t_0, t_1)$ is an MSO formula (bound first order variables t' in φ only have the comparison $t' > t_0$, and there are no free variables other than t_0, t_1, and hence no metric comparison exists in φ). Let re_φ be the regular expression equivalent to $\varphi(\downarrow t_0, t_1)$. The presence of free variables t_0, t_1 implies that re_φ ie over the alphabet $2^\Sigma \times \{0,1\}^2$, where the last two bits are for t_0, t_1. As seen in the case of QkMSO to RatMTL, t_0 is assigned the first position of re_φ since all other variables take up a position to its right. Hence re_φ can be rewritten as $(2^\Sigma, 1, 0)\mathsf{re}'$. Since t_1 is assigned a unique position, there is exactly one occurrence of a symbol of the form $(2^\Sigma, 0, 1)$ in re'. Using (Lemma 7, page 16) [4], we can write re' as a finite union of disjoint expressions each of the form $\mathsf{re}_\ell(\alpha, 0, 1)\mathsf{re}_r$ where $\alpha \in 2^\Sigma$, and $\mathsf{re}_\ell, \mathsf{re}_r \subseteq [(2^\Sigma, 0, 0)]^*$. $\varphi(\downarrow t_0, t_1)$ is thus equivalent to having a symbol $(\alpha, 0, 1)$ at a time point $t \in t_0 + I$, and $(2^\Sigma, 0, 1)\mathsf{re}_\ell$ holds till t, and beyond t, re_r holds. This is captured by the formula $\mathsf{FRat}_{I,\mathsf{re}'}[\bigvee_{\alpha \in 2^\Sigma}(\alpha, 0, 1) \wedge \mathsf{FRat}_{(0,\infty),\mathsf{re}_r} \Box\bot]$. Here, $\mathsf{re}' = (2^\Sigma, 0, 1)\mathsf{re}_\ell$, and the $\Box\bot$ symbolizes the fact that we see re_r in the latter part after $(\alpha, 0, 1)$ and no more symbols after that. See the full version for higher depth case.
2. FRatMTL $\subseteq_e$ Q2MSO : Proof is similar to RatMTL $\subseteq_e$ QkMSO and is in the full version. ◀

6 Discussion

We have defined a new real-time logic QkMSO by extending MSO[$<$] with guarded quantification with a block of $k-1$ metric quantifiers. We have shown that it is expressively equivalent to 1-ATA where loops do not have clock-reset. We have also shown that it is expressively equivalent to a powerful extension of MTL[U_I] called RatMTL. This makes QkMSO as well as RatMTL to be amongst highly expressive, real-time logics (future only) with decidable satisfiability and model-checking. We have established a 4-variable property for QkMSO and also characterized the expressive power of its two variable fragment. The question of a logic expressively equivalent to full 1-ATA remains open. It may be noted that [5] gave an extension of hybrid logic 1-TPTL with fixed point operators to get the expressive power of full 1-ATA but our quest is for a more traditional logic. The expressive power of 3 variable fragment of QkMSO also remains unexplored.

We briefly discuss some special cases and extensions of our results. PO-1-ATA [21] and PO$-$C$\oplus$D-1-ATA are subclasses of 1-ATA and C$\oplus$D-1-ATA respectively, where the only loops in the automaton are reset-free self-loops. Likewise, SfrMTL and FSfrMTL are subclasses of RatMTL and FRatMTL respectively, where the regular expression used in the modality has an equivalent star-free expression.

1. Thanks to theorems 13, 14 and 16, we can obtain the expressive equivalence of (i) QkFO, SfrMTL and PO-1-ATA, and (ii) Q2FO, FSfrMTL and PO$-$C$\oplus$D-1-ATA, In the absence of second order quantification, the regular expressions have a star-free equivalent, and the respective automata are aperiodic.

2. The emptiness problem for non punctual C⊕D-1-ATA is elementarily decidable. The proof is via a reduction to FRatMITL with least fix points [15]. To the best of our knowledge, this is the largest known subclass of 1-ATA which is elementarily decidable.
3. Lastly, to obtain a logical equivalence with 1-ATA, the logic RatMTL with least fixpoint operators suffices [15]. Previously, Haase et al [5] showed that $1-$TPTL with fixed points had the expressive power of 1-ATA.

References

1 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 139–152, 1991. doi:10.1145/112600.112613.

2 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 390–401, 1990. doi:10.1109/LICS.1990.113764.

3 Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 432–443, 2005. doi:10.1007/11590156_35.

4 J. Engelfriet and H. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

5 C. Haase, J. Ouaknine, and J. Worrell. On process-algebraic extensions of metric temporal logic. In *Reflections on the Work of C. A. R. Hoare.*, pages 283–300. 2010.

6 Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, pages 580–591, 1998. doi:10.1007/BFb0055086.

7 Y. Hirshfeld and A. Rabinovich. An expressive temporal logic for real time. In *MFCS*, pages 492–504, 2006.

8 P. Hunter. When is metric temporal logic expressively complete? In *CSL*, pages 380–394, 2013.

9 P. Hunter, J. Ouaknine, and J. Worrell. Expressive completeness for metric temporal logic. In *LICS*, pages 349–357, 2013.

10 Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 83(2):121–139, 1989. doi:10.1016/0890-5401(89)90055-2.

11 J.R.Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the 1960 Congress on Logic, Methdology and Philosophy of Science, Stanford Univeristy Press, Stanford*, 1962.

12 S. N. Krishna K. Madnani and P. K. Pandya. Partially punctual metric temporal logic is decidable. In *TIME*, pages 174–183, 2014.

13 Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Ucla, 1968.

14 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. doi:10.1007/BF01995674.

15 Shankara Narayanan Krishna, Khushraj Madnani, and Paritosh K. Pandya. Büchi-kamp theorems for 1-clock ATA. *CoRR*, abs/1802.02514, 2018. arXiv:1802.02514.

16 S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2):10:1–10:27, 2008. doi:10.1145/1342991.1342994.

17 J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197, 2005.

18 Paritosh K. Pandya and Simoni S. Shah. On expressive powers of timed logics: Comparing boundedness, non-punctuality, and deterministic freezing. In *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, pages 60–75, 2011. doi:10.1007/978-3-642-23217-6_5.

19 Jean Francois Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, Universite de Namur, 1999.

20 S.Krishna, K. Madnani, and P. K. Pandya. Metric temporal logic with counting. In *FoSSaCS*, pages 335–352, 2016.

21 P. K. Pandya S.Krishna, K. Madnani. Making metric temporal logic rational. In *MFCS*, 2017.

22 T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRTFT*, pages 694–715, 1994.

Progress-Preserving Refinements of CTA

Massimo Bartoletti
Università degli Studi di Cagliari
Cagliari, Italy

Laura Bocchi
University of Kent
Canterbury, UK

Maurizio Murgia
University of Kent
Canterbury, UK

Abstract

We develop a theory of refinement for timed asynchronous systems, in the setting of Communicating Timed Automata (CTA). Our refinement applies point-wise to the components of a system of CTA, and only affecting their time constraints – in this way, we achieve compositionality and decidability. We then establish a decidable condition under which our refinement preserves behavioural properties of systems, such as their global and local progress. Our theory provides guidelines on how to implement timed protocols using the real-time primitives of programming languages. We validate our theory through a series of experiments, supported by an open-source tool which implements our verification techniques.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases protocol implementation, communicating timed automata, message passing

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.40

Funding This work has been partially supported by EPSRC EP/N035372/1 "Time-sensitive protocol design and implementation", and by Aut. Reg. of Sardinia project P.I.A. 2013 "NOMAD".

1 Introduction

Formal reasoning of real-time computing systems is supported by established theories and frameworks based on e.g., timed automata [4,32,44]. In the standard theory of timed automata, communication between components is *synchronous*: a component can send a message only when its counterpart is ready to receive it. However, in many concrete scenarios, such as web-based systems, communications are *asynchronous* and often implemented through middlewares supporting FIFO messaging [5,42]. These systems can be modelled as Communicating Timed Automata (CTA) [29], an extension of timed automata with asynchronous communication. Asynchrony comes at the price of an increased complexity: interesting behavioural properties, starting from reachability, become undecidable in the general case, both in the timed [1,22] and in the untimed [14] setting. Several works propose restrictions of the general model, or sound approximate techniques for the verification of CTA [11,22]. These works leave one important problem largely unexplored: *the link between asynchronous timed models and their implementations.*

Relations between models at different levels of abstraction are usually expressed as *refinements*. These have been used, e.g., to create abstract models which enhance effectiveness of verification techniques (e.g., abstraction refinement [25,43], time-wise refinement [40]), or

to concretize abstract models into implementations [21, 23]. Existing notions of refinement between timed models are based on *synchronous* communications [7, 17, 26, 33]. Asynchronous refinement has been investigated in the *untimed* setting, under the name of subtyping between session types [8, 20, 24, 34–36]. To our knowledge, no notion of refinement has been yet investigated in the *asynchronous timed* setting. The only work that studies a notion close to that of refinement is [12], which focusses on the relation between timed multiparty session types and their implementations (processes in an extended π-calculus). The work in [12] has two main limitations. First, their model is not as general as CTA: in particular, it does not allow states with both sending and receiving outgoing transitions (so-called *mixed states*). Mixed states are crucial to capture common programming patterns like *timeouts* [38] (e.g. a server waiting for a message that sends a timeout notification after a deadline). Some programming languages provide specific primitives to express timeouts, e.g. the receive/after construct of Erlang [6]. The second limitation of [12] is that its calculus is very simple (actions are statically set to happen at precise points in time), and cannot express common real-world blocking receive primitives (with or without timeout) that listen on a channel until a message is available.

To be usable in practice, a theory of refinements should support real-world programming patterns (e.g., timeouts *à la* Erlang) and primitives, and feature *decidable* notions of refinement. Further, refinement should be *compositional* (i.e. a system can be refined by refining its single components, independently), and preserve desirable properties (e.g., progress) of the system being refined. These goals contrast with the fact that, in general (e.g. when refinements may arbitrarily alter the interaction structures) establishing if an *asynchronous* FIFO-based communication model is a refinement of another is *undecidable*, even in the untimed setting [15, 30]. Therefore, when defining an asynchronous refinement, a loss of generality is necessary to preserve decidability.

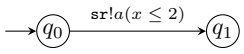
Contributions

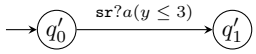
We develop a theory of asynchronous timed refinement for CTA. Our main purpose is to study preservation of behavioural properties under refinement, focussing on two aspects: *timed behaviour* and *progress*. The former kind of preservation, akin timed similarity [18], ensures that the observable behaviour of the concrete system can be simulated by the abstract system. The latter requires that refinement does not introduce deadlocks, either *globally* (i.e., the whole system gets stuck), or *locally* (i.e., a single CTA gets stuck, although the whole system may still proceed).

Refinement. We introduce a new refinement relation, which is decidable and compositional, so enabling modular development of systems of CTA. Our refinement is *structure preserving*, i.e. it may only affect time constraints: refinements can only restrict them; further, for receive actions, refinements must preserve the deadline of the original constraint (i.e., the receiving component must be ready to receive until the very last moment allowed of the original constraint). This way of refining receive actions, and structure preservation, are key to obtain decidability and other positive results. Furthermore, structure preservation reflects the common practice of implementing a model: starting from a specification (represented as a system of CTA), one derives an implementation by following the interaction structure of the CTA, and by adjusting the timings of actions as needed, depending on implementation-related time constraints, and on the programming primitives one wants to use for each action (e.g., blocking/unblocking, with/without timeout). We illustrate in Section 6 how to exploit our theory in practice, to implement progress-preserving timed protocols in Go.

Positive and negative results. Our main positive result (Theorem 26) is a decidable condition called *Locally Latest-Enabled Send Preservation* (LLESP) ensuring preservation of timed behaviour, global and local progress under our refinement. Our refinement and the LLESP condition naturally apply to most of the case studies found in literature (Section 4) In Section 6 we show how our tool and results can be used to guide the implementation of timed protocols with the Go programming language. We also considered other refinement strategies: (i) arbitrary restriction of constraints of send and receive actions (similarly to [12]), and (ii) asymmetric restriction where constraints of send actions may be restricted, and those of receive actions may be relaxed (this is the natural timed extension of the subtyping relation in [24]). Besides being relevant in literature, (i) and (ii) reflect common programming practices: (i) caters for e.g. non-blocking receive with constraint reduced to an arbitrary point in the model's guard, and (ii) caters e.g. for blocking receive without timeouts. For (i) and (ii) we only have negative results, even when LLESP holds, and if mixed states are forbidden (Fact 27). Our negative results have a practical relevance on their own: they establish that if you implement a CTA as described above, you have no guarantees of behaviour/progress preservation.

A new semantics for CTA. The original semantics for CTA [29] was introduced for studying decidability issues for timed languages. To achieve such goals, [29] adopts the usual language-based approach of computability theory: (1) it always allows time to elapse, even when this prevents the system from performing any available action, and (2) it rules out "bad" executions *a posteriori*, e.g. only keeping executions that end in final states. Consider, for example, the following two CTA:

$\mathsf{A}_{\mathtt{s}}$: 

$\mathsf{A}_{\mathtt{r}}$: 

The CTA $\mathsf{A}_{\mathtt{s}}$ models a sender $\mathtt{s}$ who wants to deliver a message a to a receiver $\mathtt{r}$. The guard $x \leq 2$ is a time constraint, stating that the message must be sent within 2 time units. The receiver wants to read the message a from $\mathtt{s}$ within 3 time units. In [29], a possible (partial) computation of the system $(\mathsf{A}_{\mathtt{s}}, \mathsf{A}_{\mathtt{r}})$ would be the following:

$$\gamma_0 = ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 0\}) \xrightarrow{5} ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 5\})$$

The tuple γ_0 at the LHS of the arrow is the initial *configuration* of the system, where both CTA are in their initial states; the pair $(\varepsilon, \varepsilon)$ means that the communication queues between $\mathtt{r}$ and $\mathtt{s}$ are empty; the last component means that the clocks x and y are set to 0. The label on the arrow represents a delay of 5 time units. This computation does *not* correspond to a reasonable behaviour of the protocol: we would expect the send action to be performed *before* the deadline expires.

To capture this intuition, we introduce a semantics of CTA, requiring that the elapsing of time does not disable the send action in $\mathsf{A}_{\mathtt{s}}$. Namely, we can procrastinate the send for 2 time units; then, time cannot delay further, and the only possible action is the send:

$$\gamma_0 \xrightarrow{2} ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 2\}) \xrightarrow{\mathtt{sr}!a} ((q_1, q'_0), (a, \varepsilon), \{x, y \mapsto 2\})$$

We prove (Theorem 7) that our semantics enjoys a form of *persistency*: if at least one receive action is guaranteed to be enabled in the future (i.e. a message is ready in its queue and its time constraint is satisfiable now or at some point in the future) then time passing preserves at least one of these guaranteed actions. Instead, time passing can disable all send actions, but *only if* it preserves at least one guaranteed receive.

```
receive {s,a1} -> Body1
        ...
        {s,ak} -> Bodyk
after   10     -> p!b
```

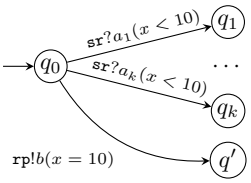
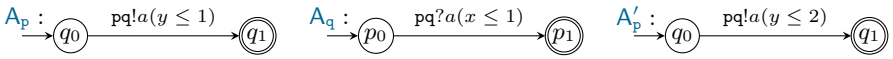


Figure 1 The `receive`/`after` pattern of Erlang (left), and the corresponding CTA (right).

It is well known that language-based approaches are not well suited to deal with concurrency issues like those addressed in this paper. To see this, consider the following CTA, where the states with a double circle are accepting:



The systems $S = (\mathsf{A_p}, \mathsf{A_q})$ and $S' = (\mathsf{A'_p}, \mathsf{A_q})$ accept the same language, namely t_0 $\mathtt{pq}!a$ t_1 $\mathtt{pq}?a$ t_2 with $t_0 + t_1 \leq 1$ and $t_2 \in \mathbb{R}_{\geq 0}$. So, the language-based approach does not capture a fundamental difference between S and S': S enjoys progress, while S' does not. Our approach to defining CTA semantics provides us with a natural way to reason on standard properties of protocols like progress, and to compare behaviours using e.g., (bi)simulation.

Our semantics allows for CTA with mixed states, by extending the one in [11] (where, instead, mixed states are forbidden). As said above, mixed states enable useful programming patterns. Consider e.g. the code snippet in Figure 1 (left), showing a typical use of the `receive`/`after` construct in Erlang. The snippet attempts to receive a message matching one of the patterns `{s,`a_1`}`,...,`{s,`a_k`}`, where `s` represents the identifier of the sender, and $\mathtt{a}_1$,...,$\mathtt{a}_k$ are the message labels. If no such message arrives within 10 ms, then the process in the `after` branch is executed, sending immediately a message `b` to process `p`. This behaviour can be modelled by the CTA in Figure 1 (right), where q_0 is mixed. Our semantics properly models the intended behaviour of timeouts.

Urgency. Another practical aspect that is not well captured by the existing semantics of CTA [11, 29] is *urgency*. Indeed, while in known semantics receive actions can be deferred, the receive primitives of mainstream programming languages unblock as soon as the expected message is available. These primitives include the non-blocking (resp. blocking) `WaitFreeReadQueue.read()` (resp. `WaitFreeReadQueue.waitForData()`) of Real-Time Java [16], and `receive...after` in Erlang, just to mention some. Analysing a system only on the basis of a non-urgent semantics may result in an inconsistence between the behaviour of the model and that of its implementation. To correctly characterise urgent behaviour, we introduce a second semantics (Definition 28), that is *urgent* in what it forces receive actions as soon as the expected message is available. Theorem 29 shows that the urgent semantics preserves the behaviour of the non-urgent. However, the urgent semantics does *not* enjoy the preservation results of Theorem 26. Still, it is possible to obtain preservation under refinement by combining Theorem 26 with Theorem 33. More specifically, the latter ensures that, if a system of CTA enjoys progress in the non-urgent semantics, then it will also enjoy progress in the urgent one, under a minor and common assumption on the syntax of time constraints. So, one can use Theorem 26 to obtain a progress-preserving refinement (in the non-urgent semantics), and then lift the preservation result to the urgent semantics through Theorem 33. Overall, our theory suggests that, despite the differences between semantics of CTA and programming languages, verification techniques based on CTA can be helpful for implementing distributed timed programs.

Artifact and experiments. We validate our approach through a suite of use cases, which we analyse through a tool we have developed to experiment with our theory (`https://github.com/cta-refinement`). The suite includes real-world use cases, like e.g. SMTP [41] and Ford Credit web portal [39]. Experimentation shows that for each use case we can find a refinement which implements the specification in a correct way. All use cases require less than twenty control states, and our tool takes a few milliseconds to perform the analysis. In the absence of larger use cases in literature, we tried the tool on a deliberately large example with thousands of states and multiple clocks: even in that case, termination time is in the order of dozens of minutes. Performance data, as well as the proofs of our statements, are available at www.cs.kent.ac.uk/people/staff/lb514/catr.html.

2 Communicating Timed Automata

We assume a finite set $\mathcal{P}$ of *participants*, ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, ..., and a finite set $\mathbb{A}$ of *messages*, ranged over by a, b, ... We define the set $\mathcal{C}$ of *channels* as $\mathcal{C} = \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$. We denote with $\mathbb{A}^*$ the set of finite words on $\mathbb{A}$ (ranged over by w, w', ...), with ww' the concatenation of w and w', and with ε the empty word.

Clocks, guards and valuations. Given a (finite) set of *clocks* X (ranged over by $x, y, \ldots$), we define the set Δ_X of *guards* over X (ranged over by $\delta, \delta', \ldots$) as follows:

$$\delta ::= \mathit{true} \mid x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2 \qquad (c \in \mathbb{Q}_{\geq 0})$$

We denote with $\mathbb{V} = X \to \mathbb{R}_{\geq 0}$ the set of *clock valuations* on X. Given $t \in \mathbb{R}_{\geq 0}$, $\lambda \subseteq X$, and a clock valuation ν, we define the clock valuations:

- **(i)** $\nu + t$ as the valuation mapping each $x \in X$ to $\nu(x) + t$;
- **(ii)** $\lambda(\nu)$ as the valuation which resets to 0 all the clocks in $\lambda \subseteq X$, and preserves to $\nu(x)$ the values of the other clocks $x \notin \lambda$.

Furthermore, given a set K of clock valuations, we define the *past* of K as the set of clock valuations $\downarrow K = \{\nu \mid \exists \delta \geq 0 : \nu + \delta \in K\}$. The *semantics of guards* is defined as function $[\![\cdot]\!] : \Delta_X \to \wp(\mathbb{V})$, where: $[\![\mathit{true}]\!] = \mathbb{V}$, $[\![x \leq c]\!] = \{\nu \mid \nu(x) \leq c\}$, $[\![\delta_1 \wedge \delta_2]\!] = [\![\delta_1]\!] \cap [\![\delta_2]\!]$, $[\![\neg\delta]\!] = \mathbb{V} \setminus [\![\delta]\!]$, and $[\![c \leq x]\!] = \{\nu \mid c \leq \nu(x)\}$.

Actions. We denote with $\mathsf{Act} = \mathcal{C} \times \{!, ?\} \times \mathbb{A}$ the set of *untimed actions*, and with $\mathsf{TAct}_X = \mathsf{Act} \times \Delta_X \times 2^X$ the set of *timed actions* (ranged over by $\ell, \ell', \ldots$). A (timed) action of the form $\mathtt{sr}!a(\delta, \lambda)$ is a *sending action*: it models a participant $\mathtt{s}$ who sends to $\mathtt{r}$ a message a, provided that the guard δ is satisfied. After the message is sent, the clocks in $\lambda \subseteq X$ are reset. An action of the form $\mathtt{sr}?a(\delta, \lambda)$ is a *receiving action*: if the guard δ is satisfied, $\mathtt{r}$ receives a message a sent by $\mathtt{s}$, and resets the clocks in $\lambda \subseteq X$ afterwards. Given $\ell = \mathtt{pr}!a(\delta, \lambda)$ or $\ell = \mathtt{qp}?a(\delta, \lambda)$, we define:

- **(i)** $\mathsf{msg}(\ell) = a$,
- **(ii)** $\mathsf{guard}(\ell) = \delta$,
- **(iii)** $\mathsf{reset}(\ell) = \lambda$,
- **(iv)** $\mathsf{subj}(\ell) = \mathtt{p}$, and
- **(v)** $\mathsf{act}(\ell)$ is $\mathtt{pr}!$ (in the first case) or $\mathtt{qp}?$ (in the second case).

We omit δ if true, and λ if empty.

CTA and systems of CTA. A *CTA* A is a tuple of the form (Q, q_0, X, E), where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, X is a set of clocks, and $E \subseteq Q \times \mathsf{TAct}_X \times Q$ is a set of *edges*, such that the set $\bigcup\{\mathsf{subj}(e) \mid e \in E\}$ is a singleton, that we denote as $\mathsf{subj}(\mathsf{A})$. We write $q \xrightarrow{\ell} q'$ when $(q, \ell, q') \in E$. We say that a state is *sending* (resp. *receiving*) if it has some outgoing sending (resp. receiving) edge. We say that A has *mixed states* if it has some state which is both sending and receiving. We say that a state q is *final* if there exist no ℓ and q' such that $(q, \ell, q') \in E$. *Systems* of CTA (ranged over by $S, S', \ldots$) are sequences $(\mathsf{A}_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$, where each $\mathsf{A}_\mathtt{p} = (Q_\mathtt{p}, q_{0\mathtt{p}}, X_\mathtt{p}, E_\mathtt{p})$ is a CTA, and

(i) for all $\mathtt{p} \in \mathcal{P}$, $\mathsf{subj}(\mathsf{A}_\mathtt{p}) = \mathtt{p}$;
(ii) for all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$, $X_\mathtt{p} \cap X_\mathtt{q} = \emptyset = Q_\mathtt{p} \cap Q_\mathtt{q}$.

Configurations. CTA in a system communicate via asynchronous message passing on FIFO queues, one for each channel. For each couple of participants $(\mathtt{p}, \mathtt{q})$ there are two channels, $\mathtt{pq}$ and $\mathtt{qp}$, with corresponding queues $w_\mathtt{pq}$ (containing the messages from $\mathtt{p}$ to $\mathtt{q}$) and $w_\mathtt{qp}$ (messages from $\mathtt{q}$ to $\mathtt{p}$). The state of a system S, or *configuration*, is a triple $\gamma = (\vec{q}, \vec{w}, \nu)$ where:

(i) $\vec{q} = (q_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ is the sequence of the current states of all the CTA in S;
(ii) $\vec{w} = (w_\mathtt{pq})_{\mathtt{pq}\in\mathcal{C}}$ with $w_\mathtt{pq} \in \mathbb{A}^*$ is a sequence of queues;
(iii) $\nu : \bigcup_{\mathtt{p}\in\mathcal{P}} X_\mathtt{p} \to \mathbb{R}_{\geq 0}$ is a *clock valuation*.

The initial configuration of S is $\gamma_0 = (\vec{q_0}, \vec{\varepsilon}, \nu_0)$ where $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $\vec{\varepsilon}$ is the sequence of empty queues, and $\nu_0(x) = 0$ for each $x \in \bigcup_{\mathtt{p}\in\mathcal{P}} X_\mathtt{p}$. We say that $(\vec{q}, \vec{w}, \nu)$ is *final* when all $q \in \vec{q}$ are final.

We introduce a new semantics of systems of CTA, that generalises Definition 9 in [11] to account for mixed states. To this aim, we first give a few auxiliary definitions. We start by defining when a guard δ' is satisfiable *later* than δ in a clock valuation.

▶ **Definition 1** (Later satisfiability)**.** For all ν, we define the relation $\leq_\nu$ as:

$$\delta \leq_\nu \delta' \iff \forall t \in \mathbb{R}_{\geq 0} : \nu + t \in [\![\delta]\!] \implies \exists t' \geq t : \nu + t' \in [\![\delta']\!]$$

The following lemma states some basic properties of later satisfiability.

▶ **Lemma 2.** *The relation $\leq_\nu$ is a total preorder, for all clock valuations ν. Further, for all guards δ, δ', for all $t \in \mathbb{R}_{\geq 0}$, and $c, d \in \mathbb{Q}_{\geq 0}$:*

(a) $(x \leq c) \leq_\nu (x \leq c + d)$;
(b) $\delta \wedge \delta' \leq_\nu \delta'$;
(c) $\delta \leq_\nu \delta' \implies \delta \leq_{\nu+t} \delta'$.

▶ **Definition 3** (FE, LE, ND)**.** In a configuration $(\vec{q}, \vec{w}, \nu)$, we say that an edge $(q, \ell, q') \in E_\mathtt{p}$ is future-enabled (FE), latest-enabled (LE), or non-deferrable (ND) iff, respectively:

- $\exists t \in \mathbb{R}_{\geq 0}.\ \nu + t \in [\![\mathsf{guard}(\ell)]\!]$ (FE)
- $\forall \ell', q'' : (q, \ell', q'') \in E_\mathtt{p} \implies \mathsf{guard}(\ell') \leq_\nu \mathsf{guard}(\ell)$, and (q, ℓ, q') is FE (LE)
- $\exists \mathtt{s}, w' : \mathsf{act}(\ell) = \mathtt{sp}?,\ w_\mathtt{sp} = \mathsf{msg}(\ell)w'$ and (q, ℓ, q') is FE (ND)

An edge is FE when its guards can be satisfied at some time in the future; it is LE when no other edge (starting from the same state) can be satisfied later than it. The type of action (send or receive) and the co-party involved are immaterial to determine FE and LE edges. A receiving edge is ND when the expected message is already at the head of the queue, and there is some time in the future when it can be read. Note that an edge $(q, \mathtt{sp}?a(\delta, \lambda), q')$ is deferrable when $w_\mathtt{sp} = bw'$ and $a \neq b$ (i.e., the first message in the queue is not the expected

one). Non-deferrability is not affected by the presence of send actions in the outgoing edges. It could happen that two receiving edges in a CTA are ND, if both expected messages are in the head of each respective queue.

The semantics of systems is given as a timed transition system (TLTS) between configurations.

▶ **Definition 4** (Semantics of systems). Given a system S, we define the TLTS $[\![S]\!]$ as $(Q, \mathcal{L}, \to)$, where

(i) Q is the set of configurations of S,
(ii) $\mathcal{L} = \mathsf{Act} \cup \mathbb{R}_{\geq 0}$,
(iii) $\gamma = (\vec{q}, \vec{w}, \nu) \xrightarrow{\alpha} (\vec{q'}, \vec{w'}, \nu') = \gamma'$ holds when one of the following rules apply:

1. $\alpha = \mathtt{sr}!a$, $(q_{\mathtt{s}}, \alpha(\delta, \lambda), q'_{\mathtt{s}}) \in E_{\mathtt{s}}$, and
   (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{s}$;
   (b) $w'_{\mathtt{sr}} = w_{\mathtt{sr}} a$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$;
   (c) $\nu' = \lambda(\nu)$ and $\nu \in [\![\delta]\!]$;
2. $\alpha = \mathtt{sr}?a$, $(q_{\mathtt{r}}, \alpha(\delta, \lambda), q'_{\mathtt{r}}) \in E_{\mathtt{r}}$, and
   (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{r}$;
   (b) $w_{\mathtt{sr}} = a w'_{\mathtt{sr}}$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$;
   (c) $\nu' = \lambda(\nu)$ and $\nu \in [\![\delta]\!]$;
3. $\alpha = t \in \mathbb{R}_{\geq 0}$, and
   (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \in \mathcal{P}$;
   (b) $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \in \mathcal{C}$;
   (c) $\nu' = \nu + t$;
   (d) for all $\mathtt{p} \in \mathcal{P}$, if some *sending* edge starting from $q_{\mathtt{p}}$ is LE in γ, then such edge is LE also in γ';
   (e) for all $\mathtt{p} \in \mathcal{P}$, if some edge starting from $q_{\mathtt{p}}$ is ND in γ, then there exists an edge starting from $q_{\mathtt{p}}$ that is ND in γ'.

We write $\gamma \to \gamma'$ when $\gamma \xrightarrow{\alpha} \gamma'$ for some label α, and $\gamma \xrightarrow{\alpha}$ if $\gamma \xrightarrow{\alpha} \gamma'$ for some configuration γ'. We denote with $\to^*$ the reflexive and transitive closure of $\to$.

Rules (1), (2) and the first three items of (3) are adapted from [11]. In particular, (1) allows a CTA $\mathtt{s}$ to send a message a on channel $\mathtt{sr}$ if the time constraints in δ are satisfied by ν; dually, (2) allows $\mathtt{r}$ to consume a message from the channel, if δ is satisfied. In both rules, the clocks in λ are reset. Rule (3) models the elapsing of time. Items 3a and 3b require that states and queues are not affected by the passing of time, which is implemented by item 3c. Items 3d and 3e put constraints on when time can pass. Condition 3d requires that time passing preserves LE sending edges: this means that if the current state of a CTA has the option to send a message (possibly in the future), time passing cannot prevent it to do so. Instead, condition 3e ensures that, if at least one of the expected messages is already at the head of a queue, time passing must still allow at least one of the messages already at the head of some queue to be received.

Our semantics (Definition 4) enjoys two classic properties [38] of timed systems, recalled below.

▶ **Definition 5.**

$$\gamma \xrightarrow{t} \gamma' \ \wedge\ \gamma \xrightarrow{t} \gamma'' \implies \gamma' = \gamma'' \qquad \text{(Time determinism)}$$

$$\gamma \xrightarrow{t+t'} \gamma' \iff \exists \tilde{\gamma} : \gamma \xrightarrow{t} \tilde{\gamma} \ \wedge\ \tilde{\gamma} \xrightarrow{t'} \gamma' \qquad \text{(Time additivity)}$$

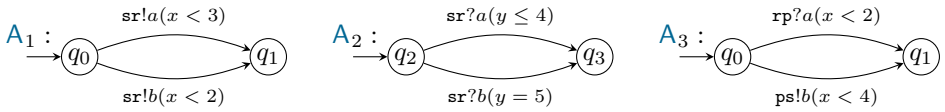


Figure 2 A collection of CTA, to illustrate the semantics of systems.

▶ **Lemma 6.** *The semantics of CTA enjoys time determinism and time additivity [38].*

Our semantics does not, instead, enjoy *persistency* [38], because the passing of time can suppress the ability to perform some actions. However, it enjoys a weaker persistency property, stated by Theorem 7. More specifically, if a receive action is ND, then time passing cannot suppress *all* receive actions: at least a ND action (not necessarily the first one) always remains FE after a delay. Instead, time passing can disable all send actions, but *only if* it preserves at least a ND receive action.

▶ **Theorem 7** (Weak persistency). *For all configurations* γ, γ'*:*

$$\gamma \xrightarrow{t'} \xrightarrow{\texttt{rp?}} \wedge\ \gamma \xrightarrow{t} \gamma' \implies \exists \gamma'', \mathtt{s}, t'' : \gamma' \xrightarrow{t''} \gamma'' \wedge \mathtt{p} \textit{ has a ND edge in } \gamma''$$
$$\gamma \xrightarrow{t'} \xrightarrow{\texttt{pr!}} \wedge\ \gamma \xrightarrow{t} \gamma' \implies \exists \gamma'', \mathtt{s}, t'' : \gamma' \xrightarrow{t''} \gamma'' \wedge \mathtt{p} \textit{ has a FE sending edge or a ND edge in } \gamma''$$

Definition 8 below will be useful to reason on executions of systems.

▶ **Definition 8** (Maximal run). A *run* of a system S starting from γ is a (possibly infinite) sequence $\rho = \gamma_1 \xrightarrow{t_1} \gamma_1' \xrightarrow{\alpha_1} \gamma_2 \xrightarrow{t_2} \cdots$ with $\gamma_1 = \gamma$ and $\alpha_i \in \mathsf{Act}$ for all i. We omit the clause "starting from s" when $\gamma = \gamma_0$. We call *trace* the sequence $t_1\, \alpha_1\, t_2 \cdots$. For all $n > 0$, we define the partial functions: $conf_n(\rho) = \gamma_n$, $delay_n(\rho) = t_n$, $act_n(\rho) = \alpha_n$. We say that a run is *maximal* when it is infinite, or given its last element γ_n it never happens that $\gamma_n \xrightarrow{t} \xrightarrow{\alpha}$, for any $t \in \mathbb{R}_{\geq 0}$ and $\alpha \in \mathsf{Act}$.

We show the peculiarities of our semantics through the CTA in Figure 2. First, consider the system composed of A_0 and A_0. A possible maximal run of $(\mathsf{A}_0, \mathsf{A}_0)$ from the initial configuration $\gamma_0 = ((q_0, q_2), \vec{\varepsilon}, \nu_0)$ is the following:

$$\gamma_0 \xrightarrow{2} \gamma_1 = ((q_0, q_2), (\varepsilon, \varepsilon), \nu_0 + 2) \qquad \xrightarrow{\texttt{sr!}a} \gamma_2 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 2)$$
$$\xrightarrow{1.5} \gamma_3 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 3.5) \qquad \xrightarrow{\texttt{sr?}a} \gamma_4 = ((q_1, q_3), (\varepsilon, \varepsilon), \nu_0 + 3.5)$$

The first delay transition is possible because there are no ND edges in A_0 (both edges are sending), and the LE edge $(q_0, \texttt{sr!}a(x < 3), q_1)$, continues to be LE in $\nu_0 + 2$; further, in A_0 there are no LE sending edges, and no ND edges (since the queue `sr` is empty). Note that condition 3d prevents γ_0 from making transitions with label $t \geq 3$, since $(q_0, \texttt{sr!}a(x < 3), q_1)$ is LE in γ_0, but it is *not* LE in $\nu_0 + t$ if $t \geq 3$. The transition from γ_1 to γ_2 corresponds to a send action. The delay transition from γ_2 to γ_3 is possible because the state of A_0 is final, while the state q_2 of A_0 has a ND edge, $(q_2, \texttt{sr?}a(y \leq 4), q_3)$, which is still ND at $\nu_0 + 3.5$. Note instead that condition 3e prevents γ_2 from making a transition with $t > 2$, because no edge is ND in $\nu_0 + 2 + t$ if $t > 2$. Indeed, the last moment when the edge $(q_2, \texttt{sr?}a(y \leq 4), q_3)$ is FE is $y = 4$. Finally, the transition from γ_3 to γ_4 corresponds to a receive action.

The CTA A_0 has mixed states, with the send action enabled for longer than the receive action. We show the behaviour of A_0 (abstracting from its co-parties that, we assume, always allow delays e.g. have all guards set to *true*). This CTA has a LE sending action $(q_0, \texttt{ps!}b(x < 4), q_1)$ in the initial configuration γ_0. Hence, condition 3d is satisfied in γ_0 iff the delay t is less than 4. Condition 3e is satisfied in γ_0, as there are no ND edges. When A_0

is at state q_0, with $w_{\mathtt{rp}} = a$ and $\nu(x) = 0$, the CTA allows a delay t iff $t < 2$: later, no edge would be ND, so 3e would be violated. If the message a is in the queue but it is too late to receive it (i.e., $\nu(x) \geq 2$), then the receive action would be deferrable, and so a delay would be allowed – if condition 3d is respected.

3 Compositional asynchronous timed refinement

In this section we introduce a decidable notion of refinement for systems of CTA. Our system refinement is defined *point-wise* on its CTA. Point-wise refinement $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ only alters the guards, in the refined CTA A', while leaving the rest unchanged. The guards of A' – both in send and receive actions – must be narrower than those of A. Further, the guards in receive actions must have *the same past* in both CTA. Formally, to define the relation $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ we use *structure-preserving* functions that map the edges of A into those of A', preserving everything but the guards.

▶ **Definition 9** (Structure-preserving). Let E, E' be sets of edges of CTA. We say that a function $f : E \to E'$ is *structure-preserving* when, for all $(q, \ell, q') \in E$, $f(q, \ell, q') = (q, \ell', q')$ with $\mathsf{act}(\ell) = \mathsf{act}(\ell')$, $\mathsf{msg}(\ell) = \mathsf{msg}(\ell')$, and $\mathsf{reset}(\ell) = \mathsf{reset}(\ell')$.

▶ **Definition 10** (Refinement). Let $\mathsf{A} = (Q, q_0, X, E)$ and $\mathsf{A}' = (Q, q_0, X, E')$ be CTA. The relation $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ holds whenever there exists a structure-preserving isomorphism $f : E \to E'$ such that, for all edges $(q, \ell, q') \in E$, if $f(q, \ell, q') = \ell'$, then:

(a) $[\![\mathsf{guard}(\ell')]\!] \subseteq [\![\mathsf{guard}(\ell)]\!]$;

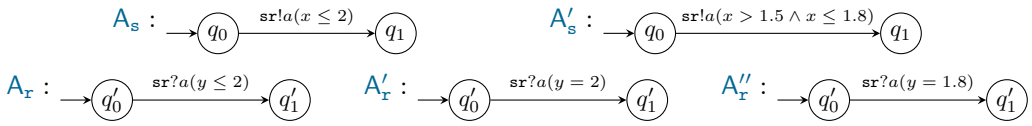
(b) if (q, ℓ, q') is a receiving edge, then $\downarrow [\![\mathsf{guard}(\ell')]\!] = \downarrow [\![\mathsf{guard}(\ell)]\!]$.

Condition a allows the guards of send/receive actions to be restricted. For receive actions, condition b requires restriction to preserve the final deadline.

System refinement reflects a modular engineering practice where parts of the system are implemented independently, without knowing how other parts are implemented.

▶ **Definition 11** (System Refinement). Let $S = (\mathsf{A}_1, \ldots, \mathsf{A}_n)$, and let $S' = (\mathsf{A}'_1, \ldots, \mathsf{A}'_n)$. We write $S \sqsubseteq S'$ iff $\mathsf{A}_i \sqsubseteq_1 \mathsf{A}'_i$ for all $i \in 1 \ldots n$.

▶ **Example 12.** With the CTA below, we have: $\mathsf{A}'_{\mathtt{s}} \sqsubseteq_1 \mathsf{A}_{\mathtt{s}}$, $\mathsf{A}'_{\mathtt{r}} \sqsubseteq_1 \mathsf{A}_{\mathtt{r}}$, and $\mathsf{A}''_{\mathtt{r}} \not\sqsubseteq_1 \mathsf{A}_{\mathtt{r}}$.



Theorem 13 establishes decidability of $\sqsubseteq_1$. This follows by the fact that CTA have a finite number of states and that:

(i) the function $\downarrow [\![\delta]\!]$ is computable, and the result can be represented as a guard [10, 27];

(ii) the relation $\subseteq$ between guards is computable.

▶ **Theorem 13.** *Establishing whether* $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ *is decidable.*

We now formalise properties of systems of CTA that one would like to be preserved upon refinement. *Behaviour preservation*, which is based on the notion of timed similarity [18], requires that an implementation (refining system) at any point of a run allows *only* actions that are allowed by its specification (refined system). Below, we use $\uplus$ to denote the disjoint union of TLTSs, i.e. $(Q_1, \Sigma_1, \to_1) \uplus (Q_2, \Sigma_2, \to_2) = (Q_1 \uplus Q_2, \Sigma_1 \cup \Sigma_2, \{((i, q), a, (i, q')) \mid (q, a, q') \in \to_i\})$, where $Q_1 \uplus Q_2 = \{(i, q) \mid q \in Q_i\}$.

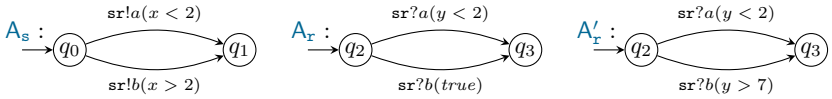
▶ **Definition 14** (Timed similarity). Let $(Q, \mathcal{L}, \to)$ be a TLTS. A timed simulation is a relation $\mathcal{R} \subseteq Q \times Q$ such that, whenever $\gamma_1 \mathcal{R} \gamma_2$:

$$\forall \alpha \in \mathcal{L} \ : \ \gamma_1 \xrightarrow{\alpha} \gamma_1' \implies \exists \gamma_2' : \gamma_2 \xrightarrow{\alpha} \gamma_2' \text{ and } \gamma_1' \mathcal{R} \gamma_2'$$

We call *timed similarity* (in symbols, $\lesssim$) the largest timed simulation relation.

▶ **Definition 15** (Behaviour preservation). Let $\mathcal{R}$ be a binary relation between systems. We say that $\mathcal{R}$ *preserves behaviour* iff, whenever $S_1 \mathcal{R} S_2$, we have $(1, \gamma_0^1) \lesssim (2, \gamma_0^2)$ in the TLTS $[\![S_1]\!] \uplus [\![S_2]\!]$, where γ_0^1 and γ_0^2 are the initial configurations of S_1 and S_2.

▶ **Example 16** (Behaviour preservation). Let $\mathcal{R}$ be the inclusion of runs, let $S_1 = (\mathsf{A_s}, \mathsf{A'_r})$ and $S_2 = (\mathsf{A_s}, \mathsf{A_r})$, where:



We have that $S_2 \mathcal{R} S_1$, while $S_1 \mathcal{R} S_2$ does not hold, since the traces with b in S_1 strictly include those of S_2. The relation $\mathcal{R}$ preserves timed behaviour in $\{S_1, S_2\}$: indeed, $(\gamma_0^2, 1) \lesssim (\gamma_0^1, 2)$ follows by trace inclusion and by the fact that S_1, S_2 have deterministic TLTS. Now, let S_3 be as S_2, but for the guard of $\mathtt{sr}?b(true)$, which is replaced by $y < 2$. We have that $S_3 \mathcal{R} S_2$, and $\mathcal{R}$ preserves timed behaviour in $\{S_2, S_3\}$. However, S_3 does *not* allow to continue with the message exchange: b is sent too late to be received by $\mathtt{r}$, who keeps waiting while b remains in the queue forever.

As shown by Example 16, behaviour preservation may allow a system (e.g., S_3) to remove "too much" from the runs of the original system (e.g., S_2): while ensuring that no new actions are introduced, it may introduce deadlocks. So, besides behaviour preservation we consider two other properties: *global* progress of the overall system, and *local* progress of each single participant.

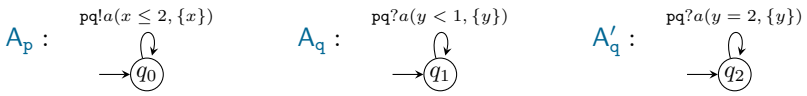
▶ **Definition 17** (Global/local progress). We say that a system S enjoys

- *global progress* when: $\forall \gamma : \gamma_0 \to^* \gamma$ not final $\implies \exists t \in \mathbb{R}_{\geq 0}, \alpha \in \mathsf{Act} : \gamma \xrightarrow{t} \xrightarrow{\alpha}$
- *local progress* when: $\forall \gamma, \mathtt{p} : \gamma_0 \to^* \gamma = (\vec{q}, \vec{w}, \nu)$ and $\vec{q} \ni q_{\mathtt{p}}$ not final $\implies$
  $\forall$ maximal runs ρ from $\gamma : \exists n : \mathsf{subj}(act_n(\rho)) = \mathtt{p}$

▶ **Lemma 18.** *If a system enjoys local progress, then it also enjoys global progress.*

The converse of Lemma 18 does not hold, as witnessed by Example 19.

▶ **Example 19** (Global vs. local progress). Consider the following CTA:



The system $(\mathsf{A_p}, \mathsf{A_q})$ enjoys global progress, since, in each reachable configuration, $\mathsf{A_p}$ can always send a message (hence the system makes an action in Act). However, if $\mathsf{A_p}$ sends a after time 1, then $\mathsf{A_q}$ cannot receive it, since its guard $y < 1$ is not satisfied. Formally, in any maximal run starting from $((q_0, q_1), (a, \varepsilon), \{x, y \mapsto 1\})$, there will be no actions with subject $\mathtt{q}$, so $(\mathsf{A_p}, \mathsf{A_q})$ does *not* enjoy local progress. The system $(\mathsf{A_p}, \mathsf{A'_q})$, instead, enjoys both global and local progress.

▶ **Definition 20** (Progress preservation). Let $\mathcal{R}$ be a binary relation between systems. We say that $\mathcal{R}$ *preserves global (resp. local) progress* iff, whenever $S_1 \mathcal{R} S_2$ and S_2 enjoys global (resp. local) progress, then S_1 enjoys global (resp. local) progress.

▶ **Example 21.** Let S_1, S_2, S_3 be as in Example 16. While S_1 and S_2 enjoy local and global progress, S_3 does not enjoy neither. Hence, $\mathcal{R} = \{(S_2, S_1), (S_3, S_1), (S_3, S_2)\}$ (i.e., trace inclusion restricted to the three given systems), does not preserve local nor global progress.

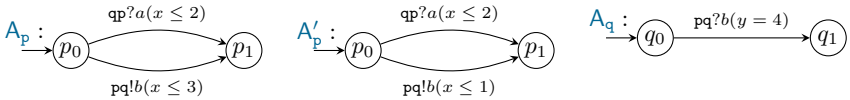
4 Verification of properties of refinements

We now study preservation of behaviour/progress upon refinements. Our first result is negative: *in general*, refinement does not preserve behaviour nor (local/global) progress, even for CTA without mixed states. This is shown by the following examples.

▶ **Example 22.** Consider A_p and A'_p below, with $\mathsf{A}'_\mathsf{p} \sqsubseteq_1 \mathsf{A}_\mathsf{p}$.

When A_p reaches p_1, the guard of the outgoing edge is satisfiable. Instead, A'_p gets stuck in p_1.

▶ **Example 23.** Let $S = (\mathsf{A}_\mathsf{p}, \mathsf{A}_\mathsf{q})$, and let $S' = (\mathsf{A}'_\mathsf{p}, \mathsf{A}_\mathsf{q})$, where:



We have that $\mathsf{A}'_\mathsf{p} \sqsubseteq_1 \mathsf{A}_\mathsf{p}$, and so $S' \sqsubseteq S$. Behaviour is not preserved as S' allows the run $\gamma_0 \xrightarrow{4}$, while S does not. This is because A_p has a LE sending edge, which prevents step $\xrightarrow{4}$ by condition 33d of Definition 4, while A'_p does not have a LE sending edge. Progress (local and global) is enjoyed by S. Instead, S' does not enjoy progress: S' allows $\gamma_0 \xrightarrow{2} \gamma = ((p_0, q_0), \vec{\varepsilon}, \nu_0 + 2)$, but there are no t and $\alpha \in \mathsf{Act}$ such that $\gamma \xrightarrow{t} \xrightarrow{\alpha}$ as the sending action is expired and all the queues are empty.

The issue in Example 23 is that a LE sending edge, which was crucial for making execution progress, is lost after the refinement. In Definition 25 we devise a decidable condition – which we call LLESP after *locally LE send preservation* – that excludes scenarios like the above. In Theorem 26 we show that, with the additional LLESP condition, $\sqsubseteq_1$ guarantees preservation of behaviour and progress. Unlike Definition 10, which is defined "edge by edge", LLESP is defined "state by state". This is because LLESP preserves the existence of LE sending edges (outgoing from the given state), and not necessarily the LE sending edge himself, making the analysis more precise.

▶ **Definition 24.** Let $\mathsf{A} = (Q, q_0, X, E)$, let $q \in Q$, and let K be a set of clock valuations. We define the following sets of clock valuations:

$$Pre_q^\mathsf{A} = \{\nu_0 \mid q_0 = q\} \cup \{\nu \mid \exists q', \ell, \nu' : (q', \ell, q) \in E,\ \nu' \in [\![\mathsf{guard}(\ell)]\!],\ \nu = \mathsf{reset}(\ell)(\nu')\}$$
$$Les_q^\mathsf{A} = \{\nu \mid q \text{ has a LE sending edge in } \nu\}$$
$$Post_q^\mathsf{A}(K) = \left\{\nu + t \,\middle|\, \nu \in K \ \wedge\ (\nu \in Les_q^\mathsf{A} \implies \nu + t \in Les_q^\mathsf{A})\right\}$$

We briefly comment the auxiliary definition above. The set Les_q^{A} is self-explanatory, and its use is auxiliary to the definition of *Post*. Let $(\vec{q}, \vec{w}, \nu)$, where q is in $\vec{q}$, that can be reached by the initial configuration of some system S containing A. The set Pre_q^{A} contains all (but not only) the clock valuations under which a configuration like the one above can be reached with a label $\alpha \in \mathsf{Act}$ fired by A. Instead, $Post_q^{\mathsf{A}}(K)$ computes a symbolic step of timed execution, in the following sense: if $\nu \in K$ and $\gamma \xrightarrow{t} (\vec{q}, \vec{w}, \nu')$, where q is in $\vec{q}$, then $\nu' \in Post_q^{\mathsf{A}}(K)$. This is obtained by defining $Post_q^{\mathsf{A}}(K)$ as the set of clock valuations that would satisfy item 3d of Definition 4 for A at runtime, when starting from a configuration whose clock valuation is in K. Since every configuration reachable with a finite run and with an action in Act as last label can also be reached by a run ending with a delay (the original run followed by a null delay), the set $Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}})$ contains the set of clock valuations ν such that $(\vec{q}, \vec{w}, \nu)$, with q is in $\vec{q}$, can be reached by the initial configuration of some system S containing A.

▶ **Definition 25** (LLESP)**.** A relation $\mathcal{R}$ is *locally LE send preserving* (in short, LLESP) iff, for all $\mathsf{A} = (Q, q_0, X, E)$ and $\mathsf{A}' = (Q, q_0, X, E')$ such that $\mathsf{A}' \ \mathcal{R} \ \mathsf{A}$, and for all $q \in Q$: $Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'}) \cap Les_q^{\mathsf{A}} \subseteq Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'}) \cap Les_q^{\mathsf{A}'}$. We define $\sqsubseteq_1^L$ as the largest LLESP relation contained in $\sqsubseteq_1$.

Basically, LLESP requires that, whenever $\mathsf{A}' \ \mathcal{R} \ \mathsf{A}$, if q has a LE sending edge in ν with respect to A, then q has a LE sending edge in ν with respect to A', where ν ranges over elements of $Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'})$.

It follows our main result: $\sqsubseteq_1^L$ preserves behaviour and progress (both global and local). Further, LLESP is decidable, so paving the way towards automatic verification.

▶ **Theorem 26** (Preservation under LLESP)**.** *$\sqsubseteq_1^L$ preserves behaviour, and global and local progress. Furthermore, establishing whether $\mathsf{A}' \sqsubseteq_1^L \mathsf{A}$ is decidable.*

Negative results on alternative refinement strategies. Besides introducing a new refinement we have investigated behavioural/progress preservation under two refinement strategies inspired from literature. They are both variants of our definition of refinement that alter conditions a and b in Definition 10. The first strategy (e.g., [12]) is a naïve variant of Definition 10 where b is dropped. The second strategy (e.g., [24]) is an asymmetric variant of Definition 10 that allows to *relax* guards of the receive actions: a is substituted by $[\![\mathsf{guard}(\ell')]\!] \supseteq [\![\mathsf{guard}(\ell)]\!]$ and b is dropped.

▶ **Fact 27.** *LLESP restrictions of "naïve" and "asymmetric" refinements do not preserve behaviour, global progress, nor local progress, not even if mixed states are ruled out.*

We refer to www.cs.kent.ac.uk/people/staff/lb514/catr.html for counter-examples of behaviour and progress preservation for LLESP restrictions of "naïve" and "asymmetric" refinements without mixed states. Exampke 23, which has mixed states, is also a counter-example for such refinements.

Experiments. We evaluate our theory against a suite of protocols from literature. To support the evaluation we built a tool that determines, given A and A', if $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ and if $\mathsf{A}' \sqsubseteq_1^L \mathsf{A}$. For each participant of each protocol we construct three refinement *strategies*. For sending edges, if the guard has an upper bound (e.g. $x \leq 10$) then we refine it with, respectively: (strategy #1) the lower bound (e.g. $x = 0$), (strategy #2) the average value (e.g. $x = 5$), and (strategy #3) the upper bound (if any) (e.g. $x = 10$). In all strategies, receiving

Table 1 Benchmarks. Participants satisfying LLESP are marked with ✓, the others with ✗. We omitted participants for which the strategy was not meaningful, or gave identical results as the other columns.

Case study	Strategy #1	Strategy #2	Strategy #3
Ford Credit web portal [39]	✗Server	✓ Server	
Scheduled Task Protocol [11]	✓User ✓Worker ✓Aggregator	✓User ✓Worker ✓Aggregator	✓User ✓Worker ✓Aggregator
OOI word counting [37]	✓Master ✓Worker ✓Aggregator	✓Master	✓Master
ATM [19]	✗Bank, ✓User ✗Machine	✓Bank ✓User ✓Machine	✓Bank ✓User
Fisher Mutual Exclusion [9]	✓Producer ✓Consumer	✓Producer	✓Producer
SMTP [41]	✓Client	✓Client	

edges are refines in the same way: if the guard has a not strict upper bound (e.g. $x \leq 10$) then we restrict the guard *as* its upper bound (e.g. $x = 10$); if the upper bound is strict (e.g. $x < 10$) we "procrastinate" the guard, but making it fully left-closed (Definition 31) (e.g. $10 - \varepsilon \leq x < 10$, where we set ε as a unit of time); if there is no upper bound (e.g. $x > 10$) the guard is left unchanged. Our tool correctly classifies the pairs of CTA defined above as refinements. In Table 1 we show the output of the tool when checking LLESP. We can see that strategies #2 and #3 never break the LLESP property. While this should always hold for strategy #3 (procrastinating sending edges guarantees that LE sending edges are preserved), the case for strategy #2 is incidental. Among the case studies, Ford Credit web portal and SMTP contain mixed states (used to implement timeouts). The fact that, for each protocol, there is always some refinement strategy that satisfies LLESP (hence a provably safe way to implement that protocol) witnesses the practicality of our theory. Surprisingly, the states that falsify LLESP are not mixed. The three models for which strategy #1 does not produce "good" refinements suffer from the same issue of Example 22: the guard of a sending edge is restricted in a way that makes it possibly unsatisfiable with respect to the guard of the previous action.

5 Preservation under an urgent semantics

The semantics in Definition 4 does not force the receive actions to happen, (unless time passing prevents the CTA from receiving in the future, by condition 33e. This behaviour, also present in [11, 29], contrasts with the actual behaviour of the `receive` primitives of mainstream programming languages which return *as soon as* a message is available. We now introduce a variant of the semantics in Definition 4 which faithfully models this behaviour. We make receive actions *urgent* [13, 38] by forbidding delays when a receiving edge is enabled and the corresponding message is at the head of the queue. Below, $\mathsf{Act}^?$ denotes the set of input labels.

▶ **Definition 28** (Urgent semantics of systems)**.** Given a system S, we define the TLTS $[\![S]\!]_u = (Q, \mathcal{L}, \to_u)$, where Q is the set of configurations of S, $\mathcal{L} = \mathsf{Act} \cup \mathbb{R}_{\geq 0}$, and:

$$\gamma \xrightarrow{\alpha}_u \gamma' \iff \begin{cases} \gamma \xrightarrow{\alpha} \gamma' & \text{if } \alpha \in \mathsf{Act} \\ \gamma \xrightarrow{t} \gamma' & \text{if } \alpha = t \text{ and } \forall t' < t, \gamma'', \alpha' \in \mathsf{Act}^? : \gamma \xrightarrow{t'} \gamma'' \implies \gamma'' \not\xrightarrow{\alpha'} \end{cases}$$

The non-urgent and the urgent semantics are very similar: they only differ in time actions. In the urgent semantics, a system can make a time action t only if no receive action is possible earlier than t (hence no message is waiting in a queue with "enabled" guard). Theorem 29 formally relates the two semantics. Since the urgent semantics restricts the behaviour of systems (by dropping some timed transitions), the urgent semantics preserves the behaviour of the non-urgent one.

▶ **Theorem 29.** *For all systems S, the relation $\{((1,\gamma),(2,\gamma)) \mid \gamma$ is a configuration of $S\}$ between states of $[\![S]\!]_u \uplus [\![S]\!]$ is a timed simulation.*

In general, however, a system that enjoys progress with the non-urgent semantics may not enjoy progress with the urgent one. This is illustrated by Example 30.

▶ **Example 30.** Consider the system $S = (\mathsf{A_s}, \mathsf{A_r})$, where:

With the non-urgent semantics, $\gamma_0 \xrightarrow{\mathtt{sr}!a} \xrightarrow{3} \gamma = ((q_1, q_0'), (a, \varepsilon), \nu_0 + 3) \xrightarrow{t} \xrightarrow{\mathtt{sr}?a}$, for all $t \in \mathbb{R}_{\geq 0}$. With the urgent semantics, $\gamma_0 \xrightarrow{\mathtt{sr}!a}_u \xrightarrow{3}_u \gamma \not\xrightarrow{\alpha}_u$, for all $\alpha \neq 0$. Hence, the non-urgent semantics leads to a final state, whereas the urgent semantics does not.

The issue highlighted by Example 30 is subtle (but known in literature [13]): if there is no precise point in time in which a guard becomes enabled (e.g. in $x > 3$), then the run may get stuck. In Definition 31 we deal with this issue through a restriction on guards, which guarantees that urgent semantics preserves progress. Our restriction, generalising the notion of *right-open time progress* [13] (to deal with non-convex guards), corresponds to forbidding guards defined as the conjunction of sub-guards of the form $x > c$ (but we allow subguards of the form $x \geq c$). To keep our results independent from the syntax of guards, our definition is based on sets of clock valuations.

▶ **Definition 31** (Fully left closed)**.** For all ν, and for all sets of clock valuations K, let $D_\nu(K) = \{t \mid \nu + t \in K\}$ and let $\inf Z$ denote the infimum of Z. We say that a guard δ is *fully left closed* iff: $\forall \nu : \forall K \subseteq [\![\delta]\!] : \big(D_\nu(K) \neq \emptyset \implies \nu + \inf D_\nu(K) \in [\![\delta]\!]\big)$. We say that a CTA is *input fully left closed* when all guards in its receiving edges are fully left closed. A system is input fully left closed when all its components are such.

Fully left closed guards ensure that there is an exact time instant in which a guard of an urgent action becomes enabled. The requirement that left closedness must hold for any subset K of the semantics of the guards is needed to cater for non-convex guards (i.e. guards with disjunctions). Consider e.g. $\delta = 1 \leq x \leq 3 \vee x > 4$. While δ is left closed, it is *not* fully left closed: indeed, for $K = [\![x > 4]\!] \subseteq [\![\delta]\!]$, it holds that $\inf D_{\nu_0}(K) = 4$, but $\nu + 4 \notin [\![\delta]\!]$.

▶ **Example 32.** The guard $x > 3$ in Example 30 is not fully left closed, as $\inf D_{\nu_0}([\![x > 3]\!]) = \inf\{t \mid t > 3\} = 3$, but $\nu_0 + 3 \notin [\![x > 3]\!]$. Instead, guard $x \geq 3$ is fully left closed. Consider now a variant of the system of Example 30 where guard $x > 3$ is replaced by $x \geq 3$. The run $\gamma_0 \xrightarrow{\mathtt{sr}!a}_u \xrightarrow{3}_u \gamma$ would not get stuck and allow $\gamma \xrightarrow{\mathtt{sr}?a}$.

The following theorem states that urgent semantics preserves progress with respect to non-urgent semantics, when considering fully left closed systems.

▶ **Theorem 33** (Preservation of progress vs. urgency)**.** *Let S be input fully left closed. If S enjoys global (resp. local) progress under the non-urgent semantics, then S enjoys global (resp. local) progress under the urgent semantics.*

6 Implementing protocols via refinement

We illustrate how to exploit our theory to implement timed protocols, by considering the real-world protocol in [37], which distributedly counts the occurrences of a word in a log. Because of space limitations, we slightly simplify and adapt the protocol in [37]. The system

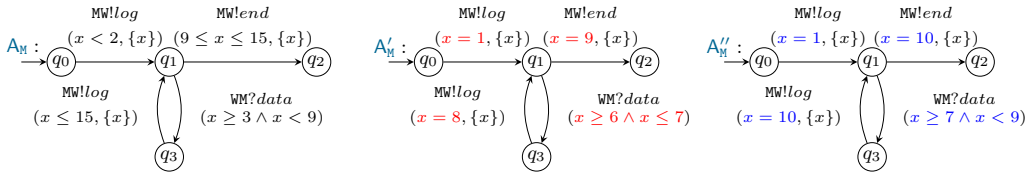


Figure 3 A_M (left); $A'_M \not\sqsubseteq_1 A_M$ (centre); $A''_M \sqsubseteq_1 A_M$ (right).

has two nodes: a master `M` and a worker `W`. We focus on `M`, modelled as A_M in Figure 3 (left). A_M repeatedly: sends a log to A_W, then either receives data from A_W (within timeout $x < 9$) or sends a notice and terminates. We implement the CTA in Go, a popular programming language with concurrency features. Here, we just sketch an implementation which intuitively follows the CTA model. A rigorous correspondence between the Go primitives and the CTA model (supporting e.g., automatic code generation) is a future work that is out of the scope of this paper. We use: (i) variables of type `time.Time` as clocks (e.g., `x`), and (ii) function `rel` below to return the value (of type `time.Duration`) of a clock (since the last reset):

```
func rel(x time.Time) time.Duration {return time.Now().Sub(x)}
```

A naïve implementation in Go. We first attempt to implement A_M following A'_M (Figure 3). A'_M is obtained from A_M by restricting guards obliviously of our results. We start from the edge from q_0 to q_1, assuming that the preparation of the log to send takes 1s (with negligible jitter). This could result in the snippet below:

```
1  x := time.Now() // initial setting of clock x
2  time.Sleep(time.Second * 1 - rel(x)) // sleep for 1s
3  x = time.Now() // reset x
4  MW <- "log" // send string "log" on FIFO channel MW
```

The statement in line 2 represents the invocation of a time-consuming function that prepares the log to be sent in line 4 (here we send the string `"log"`). In general, implementations may be informed by estimated durations of code instructions. Providing such information is made possible by orthogonal research on cost analysis, e.g. [28]. Next, we want to (i) implement the receive action from q_1 to q_3 as a *blocking* primitive with timeout, (ii) minimise the waiting time of the master listening on the channel, and restrict the interval to $x \geq 6 \wedge x \leq 7$. This could result in the following:

```
1  time.Sleep(time.Second *6 - rel(x))
2  select { case res := <- WM:
3       // here goes the implementation of edge q3 -- > q1
4         case <- time.After(time.Second * 7 + time.Nanosecond * 1 - rel(x)):
5          // here goes the implementation of edge q1 -- > q2
```

Note that without the addition of one nanosecond in line 4 above the snippet would implement a constraint $(x \geq 6 \wedge x < 7)$. To enable the program to read the message when $x = 7$, we add the smallest time unit in Go, which is negligible with respect to the protocol delays. The study of implementability of such equality constraints at this granularity of time is left as future work.

Next, we implement the edge from q_1 to q_2 by substituting line 5 above with:

```
1  time.Sleep(time.Second *9 - rel(x))
2  x = time.Now() // reset x
3  MW <- "end" // send string "end"
```

The edge from q_3 to q_1 can be implemented in a similar way, where the sleep statement represents a time-consuming log preparation of 1s, as before.

Assessing implementations via our tool. The implementation sketched in the previous paragraphs corresponds to A'_M (Figure 3). Analysis of A'_M with our tool reveals that $A'_M \not\sqsubseteq_1 A_M$: the constraints of receiving edges of A_M have been restricted not respecting the final deadlines. From Section 4 we know that A'_M may *not* preserve behaviour and progress. Suppose that the worker node is set to send the data to A_M when $x = 8.5$: according to the original specification A_M, this message is in time, hence the worker will expect a log message back from the master. However, in the implementation reflected in A'_M, the master will reply with an end message, potentially causing a deadlock. Thanks to Theorem 26 we know that we can, instead, safely restrict the constraints using $\sqsubseteq_1$: guard $x \geq 6 \wedge x \leq 7$ of A'_M can be amended as $x \geq 7 \wedge x < 9$. After this amendment, however, the tool detects a violation of LLESP: the deadlines set by guards of sending edges from q_3 and q_1 are after the deadline of the receive action. A correct refinement $A''_M \sqsubseteq_1^L A_M$ is shown in Figure 3 (right) and can be used to produce the following implementation in Go:

```
MW := make(chan string, 100)
WM := make(chan string, 100)
go func(){
  // q0 -- > q1
  x := time.Now()
  time.Sleep(time.Second *1 - rel(x))
  x = time.Now()
  MW <- "log"
  // q1 -- > q3
  time.Sleep(time.Second *7 - rel(x))
  select {
  case res := <- WM:
    // q3 -- > q2
    x = time.Now()
    time.Sleep(time.Second *10 - rel(x))
    MW <- "log"
  case <- time.After(time.Second *9 - rel(x)):
    // q1 -- > q2
    time.Sleep(time.Second *10 - rel(x))
    x = time.Now()
    MW <- "end" }}()
}
```

Practicality. In some scenarios, one may want to implement receive actions with *non-blocking* primitives (unlike above, where we have used blocking ones). Non-blocking primitives can be modelled as CTA refinements where constraints (e.g., $x \leq 9$) are restricted to a point in time (e.g., $x = 9$). Punctual guards can be attained in the real world by assuming a tolerance (e.g., around 9) that is negligible against the scale of x. In some cases, it may be desirable to *not* restrict the constraint of receive actions, to be able to receive a message as soon as possible.

CTA can capture delays of the communication medium e.g., by adding them at the receiver side. This is common when using semantics where actions are timeless and delays are modelled separately, as these semantics can be encoded into ones where actions have an associated duration.

Our theory can be applied to non real-time operating systems and languages (like, e.g., Go), as long as the time granularity of the modelled protocols is coarse enough with respect to the jitter of the operating system / language. However, negligible delays may accumulate, eventually compromising the correctness of long-lived protocols. In this case, adjustments like e.g. those suggested in [37] or based on analysis on the robustness of protocols to jitters [31], may be in order to recover correctness.

7 Conclusions

Our theory provided a formal basis to support implementation of well-behaved systems from well-behaved models. This is obtained through a decidable refinement relation, and a condition (LLESP) that guarantees behaviour and progress preservation. To overcome the undecidability results of refinement in asynchronous models [15, 30], we considered "purely timed" refinements, that only affect time constraints. While not fully general, our refinement captures the practical relations between models, and implementations obtained by following them (Section 6). Moreover, our refinement and the LLESP condition apply well to realistic protocols expressed as CTA (Section 4): for each participant of each protocol in our portfolio,

there exist one or more non-trivial (i.e. not the identity) LLESP refinements, from which one can derive behaviour- and progress-preserving implementations of that protocol. Evaluating our theory was facilitated by a tool, that can also be used to guide implementations. Being this the first work which enables refinements between CTA, there is no benchmark against which to study limitations or compare with. Other "purely timed" refinements strategies inspired by literature gave only negative results (Fact 27) when applied to the asynchronous timed setting, hence e.g., even if an implementation preserves the interactions structure of the initial CTA, and even if the timings of actions chosen for the implementation are within the range of the guards of the initial CTA, still that implementation may not preserve behaviour or progress.

Technically, we focused on interaction-based (rather than language-based) semantics, improving the state of the art in two ways: mixed choices and urgency. Mixed choices cannot be expressed in models based on session types of [11, 12]. There, interactions follow two constructs: *selection*, which corresponds to an internal choice of send actions, and *branching*, an external choice of receive actions. The behaviour of mixed states captured by our semantics falls somewhere in between internal and external choices, so it is not expressible in the setting of [11, 12]. Besides, the known semantics [11, 12, 29] do not account for urgency. Our preservation results from non-urgent to urgent semantics pave the way to *implementations* of refinements that preserve behaviour and progress (e.g. derived incrementally using the non-urgent semantics, and relying on the results in Section 4).

Other work on relating timed models with implementations is, e.g. [2, 3]. The work [2] approximates dense time models in synchronous models with fixed sampling rates, so to enable for hardware implementations. Here, instead, we considered asynchronous models, and delays at a coarser granularity, aiming at time-sensitive (not necessarily real-time) languages. The work [3] generates Erlang code from real-time Rebeca models (so, focussing on the actor model, rather than on FIFO channels). Extending our tool in this direction is an ongoing work of ours.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Shankara Narayanan Krishna. What is decidable about perfect timed channels? *CoRR*, abs/1708.05063, 2017. `arXiv:1708.05063`.

2 Parosh Aziz Abdulla, Pavel Krcál, and Wang Yi. Sampled semantics of timed automata. *Logical Methods in Computer Science*, 6(3), 2010. URL: `http://arxiv.org/abs/1007.2783`.

3 Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Arni Hermann Reynisson, Steinar Hugi Sigurdarson, and Marjan Sirjani. Modelling and simulation of asynchronous real-time systems using timed Rebeca. *Sci. Comput. Program.*, 89:41–68, 2014.

4 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

5 Advanced Message Queuing protocols (AMQP) homepage. `https://www.amqp.org/`.

6 Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007.

7 Massimo Bartoletti, Tiziana Cimoli, and Maurizio Murgia. Timed session types. *Logical Methods in Computer Science*, 13(4), 2017. `doi:10.23638/LMCS-13(4:25)2017`.

8 Massimo Bartoletti, Alceste Scalas, and Roberto Zunino. A semantic deconstruction of session types. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 402–418. Springer, 2014. `doi:10.1007/978-3-662-44584-6_28`.

9 Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on Uppaal. In *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004. doi:10.1007/b110123.

10 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi:10.1007/b98282.

11 Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting deadlines together. In *CONCUR*, volume 42 of *LIPIcs*, pages 283–296. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.283.

12 Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2014. doi:10.1007/978-3-662-44584-6_29.

13 Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. In *COMPOS*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129. Springer, 1997. doi:10.1007/3-540-49213-5_5.

14 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

15 Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Inf. Comput.*, 256:300–320, 2017.

16 Eric J. Bruno and Greg Bollella. *Real-Time Java Programming: With Java RTS*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2009.

17 Stefano Cattani and Marta Z. Kwiatkowska. A refinement-based process algebra for timed automata. *Formal Asp. Comput.*, 17(2):138–159, 2005.

18 Karlis Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer, 1992. doi:10.1007/3-540-56496-9.

19 Prakash Chandrasekaran and Madhavan Mukund. Matching scenarios with timing constraints. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2006. doi:10.1007/11867340.

20 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.

21 Chris Chilton, Marta Z. Kwiatkowska, and Xu Wang. Revisiting timed specification theories: A linear-time perspective. In *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2012. doi:10.1007/978-3-642-33365-1_7.

22 Lorenzo Clemente, Frédéric Herbreteau, Amélie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013. doi:10.1007/978-3-642-37075-5_6.

23 Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, Louis-Marie Traonouez, and Andrzej Wasowski. Real-time specifications. *STTT*, 17(1):17–45, 2015. doi:10.1007/s10009-013-0286-x.

24 Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2011. doi:10.1007/978-3-642-23217-6_19.

25 Henning Dierks, Sebastian Kupferschmid, and Kim Guldstrand Larsen. Automatic abstraction refinement for timed automata. In *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2007. doi:10.1007/978-3-540-75454-1_10.

26 Harald Fecher, Mila E. Majster-Cederbaum, and Jinzhao Wu. Refinement of actions in a real-time process algebra with a true concurrency model. *Electr. Notes Theor. Comput. Sci.*, 70(3):260–280, 2002. doi:10.1016/S1571-0661(05)80496-7.

27 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

28 Jan Hoffmann and Zhong Shao. Automatic static cost analysis for parallel programs. In *ESOP*, volume 9032 of *Lecture Notes in Computer Science*, pages 132–157. Springer, 2015. doi:10.1007/978-3-662-46669-8.

29 Pavel Krcál and Wang Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 249–262. Springer, 2006. doi:10.1007/11817963.

30 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. doi:10.1007/978-3-662-54458-7.

31 Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014. doi:10.1016/j.tcs.2013.08.015.

32 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

33 Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003.

34 Dimitris Mostrous. *Session Types in Concurrent Calculi: Higher-Order Processes and Objects*. PhD thesis, Imperial College London, November 2009.

35 Dimitris Mostrous and Nobuko Yoshida. Session-based communication optimisation for higher-order mobile processes. In *TLCA*, volume 5608 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2009. doi:10.1007/978-3-642-02273-9_16.

36 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP*, volume 5502 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2009.

37 Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Asp. Comput.*, 29(5):877–910, 2017. doi:10.1007/s00165-017-0420-8.

38 Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991.

39 Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and applications of timed service protocols. *ACM Trans. Softw. Eng. Methodol.*, 19(4):11:1–11:38, 2010.

40 Steve Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

41 The Simple Mail Transfer Protocol. http://tools.ietf.org/html/rfc5321.

42 Steve Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89, 2006.

43 Weifeng Wang and Li Jiao. Trace abstraction refinement for timed automata. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2014. doi:10.1007/978-3-319-11936-6.

44 Sergio Yovine. Kronos: A verification tool for real-time systems. (Kronos user's manual release 2.2). *STTT*, 1(1-2):123–133, 1997. doi:10.1007/s100090050009.

Automated Detection of Serializability Violations Under Weak Consistency

Kartik Nagar
Purdue University, USA
nagark@purdue.edu

Suresh Jagannathan
Purdue University, USA
suresh@cs.purdue.edu

Abstract

While a number of weak consistency mechanisms have been developed in recent years to improve performance and ensure availability in distributed, replicated systems, ensuring the correctness of transactional applications running on top of such systems remains a difficult and important problem. Serializability is a well-understood correctness criterion for transactional programs; understanding whether applications are serializable when executed in a weakly-consistent environment, however remains a challenging exercise. In this work, we combine a dependency graph-based characterization of serializability and leverage the framework of abstract executions to develop a fully-automated approach for statically finding bounded serializability violations under *any* weak consistency model. We reduce the problem of serializability to satisfiability of a formula in First-Order Logic (FOL), which allows us to harness the power of existing SMT solvers. We provide rules to automatically construct the FOL encoding from programs written in SQL (allowing loops and conditionals) and express consistency specifications as FOL formula. In addition to detecting bounded serializability violations, we also provide two orthogonal schemes to reason about unbounded executions by providing sufficient conditions (again, in the form of FOL formulae) whose satisfiability implies the absence of anomalies in any arbitrary execution. We have applied the proposed technique on TPC-C, a real-world database program with complex application logic, and were able to discover anomalies under Parallel Snapshot Isolation (PSI), and verify serializability for unbounded executions under Snapshot Isolation (SI), two consistency mechanisms substantially weaker than serializability.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Weak Consistency, Serializability, Database Applications

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.41

Related Version A full version of the paper is available at [24], `https://arxiv.org/abs/1806.08416`.

1 Introduction

We consider the problem of detecting serializability violations of transactional programs executing in a weakly-consistent replicated distributed database. An execution of such programs is said to be *serializable* if it is equivalent to some sequential execution of the transactions that comprise the program. Ensuring that all executions of such programs are serializable greatly simplifies reasoning about program correctness by reducing the complexity of understanding concurrent executions to the problem of understanding sequential ones. Unfortunately, *enforcing* serializability using runtime synchronization mechanisms

is problematic in geo-replicated distributed systems without sacrificing availability (low-latency) [18]. To reap the correctness benefits of serializability with the performance and scalability benefits of high-availability, we study the conditions under which transactional programs can be statically identified to always yield a serializable execution *without* the need for global synchronization. The challenge to realizing this goal stems from the complexity in reasoning about replicated state in which not all replicas share the same view of the data they hold.

To address this challenge, we present a fully automated static analysis that precisely encodes salient dependencies in the program as abstract executions defined in terms of an axiomatic specification of a particular weak consistency model (§4). The analysis then leverages a theorem prover to systematically search for the presence or absence of cycles in these executions consistent with these dependencies; the presence of a cycle indicates a serializability violation (§5.1). Notably, our approach can be applied to any weak consistency model whose specification can be expressed in first-order logic, a class that subsumes all realistic data stores we are aware of. More specifically, our approach constructs a dependency graph [2] from the input program containing a cycle and then asks whether there exists a valid execution under the given consistency specification that can result in this graph. To do this, we automatically extract dependency conditions from the transactional program, and relate these dependencies to artifacts in an event-based model to find whether there exists a valid abstract execution corresponding to the dependency graph. These dependencies are encoded in a first-order logic formula that is satisfiable only if there exists an execution that violates serializability.

Given a transactional program written in SQL, we discover serializability violations of bounded length under the given weak consistency model (with the bound limiting the number of concurrent transaction instances that are considered). We output the actual anomaly including the transactions involved and their inputs. This output can then be used to strengthen the consistency of the transactions involved in the anomaly (or even modifying the transactions themselves). Since the approach is parametric on a consistency policy, it can also be used to determine the weakest consistency policy for which the program is serializable. Similar to other bounded verification techniques used to detect bugs in e.g., concurrent programs [23], we posit that most serializability violations will manifest using a small number of transaction instances.

Nonetheless, we additionally provide two orthogonal schemes to reason about arbitrarily long executions with an unbounded number of transaction instances (§5.2, §5.3). The first scheme formalizes the argument that it is enough to check serializability violations in bounded executions by proving that longer violations beyond that bound would induce violations within the bound. The second scheme applies an inductive argument to check the absence of anomalies in arbitrarily long executions. Our approach is sound, but not complete - while all discovered anomalies are justified by counterexamples offered by the theorem prover, we cannot rule out the possibility of serializability violations appearing in unbounded executions that are not identified by these two schemes.

As serious case studies to assess the applicability of our approach, we have applied our technique on TPC-C, a real-world transactional program, and a course grading application [19] (§6). In both cases, we were able to detect multiple serializability violations under Eventual Consistency and a weaker variant of snapshot isolation (SI) called parallel snapshot isolation [26], and verified that these anamolies do not occur when using SI for unbounded executions. We now present an overview of our approach using a simple example.

```
withdraw (ID, Amount)
    SELECT Balance AS bal WHERE AccID=ID
    IF bal > Amount
        UPDATE SET Balance=bal-Amount WHERE AccID=ID
```

Figure 1 Example Application.

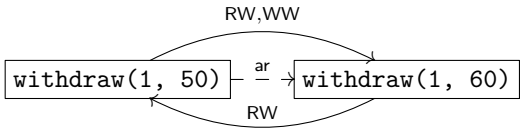


Figure 2 Abstract Execution E and its Dependency Graph.

2 Overview

In this section, we show how our approach discovers serializability violations, and how the output of our analysis can be used to repair violations using selective synchronization. Consider a simple banking application which maintains the balance of multiple accounts in a table `Account` which is indexed using the primary key `AccID` and contains the field `Balance`. Consider a `withdraw` operation (shown in Fig. 1) written in a SQL-style language, which takes `ID` and `Amount` as input, and deducts the amount from the account with account number `ID`, if the balance is sufficient. Suppose the application is deployed in a distributed, replicated environment which allows concurrent invocations of the `withdraw` operation at potentially different replicas, with the only guarantee provided being eventual consistency - eventually, all replicas will witness all updates to the `Balance` field. Under eventual consistency, the application is clearly not serializable, since concurrent withdraws operations to the same account–whose total withdrawn amount exceeds the balance of the account–could both succeed, which is not possible in a serializable execution.

A convenient way to express executions in such an environment is to use an axiomatic event-based representation. In this framework, an abstract execution [12] is expressed as the tuple $(T, \mathsf{vis}, \mathsf{ar})$, where T is the set of transaction invocations, $\mathsf{vis} \subseteq T \times T$ is a visibility relation such that if $t \xrightarrow{\mathsf{vis}} t'$ then updates of t are visible to t', and $\mathsf{ar} \subseteq T \times T$ is an arbitration relation which totally orders all writes to the same location and ensures eventual consistency [9]. For example, if $t_1 =$ `withdraw`(1,50), $t_2 =$ `withdraw`(1,60), then $E = (\{t_1, t_2\}, \{\}, \{(t_1, t_2)\})$ is an abstract execution which is not serializable, because the final value of `Balance` in the account number 1 will only reflect the `withdraw` operation t_2 (assuming an initial `Balance` of 100 in `AccID` 1), since there is no visibility constraint enforced between the two operations. This is an example of a *lost update* [5] anomaly. Our goal is to automatically construct such anomalous executions.

A useful technique to detect serializability violations is to build dependency graphs from abstract executions, and then search for cycles in the dependency graph. The nodes of the dependency graph are invocations, and edges indicate dependencies between them. There are three type of dependencies relevant to serializability detection: $t_1 \xrightarrow{\mathsf{WR}} t_2$ is a read dependency, which means that t_2 reads a value written by t_1, $t_1 \xrightarrow{\mathsf{WW}} t_2$ is a write dependency, which means that both t_1 and t_2 write to the same location, with the write of t_2 arbitrated after t_1, and $t_1 \xrightarrow{\mathsf{RW}} t_2$ is an anti-dependency, which means that t_1 does *not* read a value written by t_2 but instead reads an older version. For example, the dependency graph of the anomalous execution E described above is shown in Fig. 2.

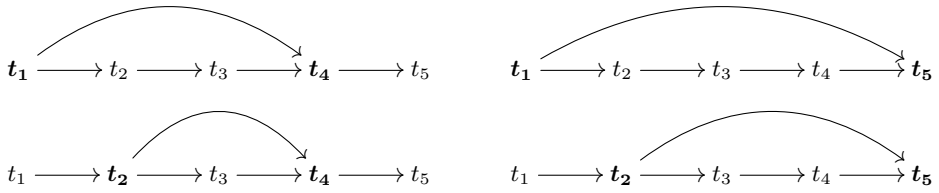


Figure 3 Different possibilities for paths of length 4 in the dependency graphs of the banking application. Note that transactions in bold perform writes.

In our approach, we start with a dependency graph containing a cycle, and then ask whether an execution corresponding to the dependency graph is possible. From the transaction code, we automatically extract the conditions under which a dependency edge can manifest between invocations of the transactions. In our running example, a dependency edge (of any type) between two `withdraw` invocations can only manifest if they are called with the same account ID. Further, we link the dependency edges with the relations vis and ar of the corresponding abstract execution. For example, $t_1 \xrightarrow{\mathsf{RW}} t_2 \Rightarrow \neg(t_2 \xrightarrow{vis} t_1)$, because otherwise, t_1 would read the value written by t_2. This is useful because different consistency schemes can be axiomatically expressed by placing constraints on the vis and ar relations.

In order to prevent the anomalous execution in our running example, we can use PSI [26] which ensures that if two invocations write to the same location, then they cannot be concurrent. While PSI is implemented using a complex, distributed protocol, in our abstract framework, it can be simply expressed using the following constraint: $\forall t, t'.\ t \xrightarrow{\mathsf{WW}} t' \implies t \xrightarrow{\mathsf{vis}} t'$. Now, the anomalous execution E is not possible, because $t_1 \xrightarrow{\mathsf{WW}} t_2 \Rightarrow t_1 \xrightarrow{\mathsf{vis}} t_2$, which contradicts $t_2 \xrightarrow{\mathsf{RW}} t_1$.

To summarize, the following is the relevant portion of formulae that we generate for the above application under PSI:

$$\forall t, t'.\ t \xrightarrow{\mathsf{RW}} t' \Rightarrow (\exists r.\ \mathtt{AccID}(r) = \mathtt{ID}(t) \wedge \mathtt{AccID}(r) = \mathtt{ID}(t') \wedge \mathtt{bal}(t') > \mathtt{Amount}(t')) \quad (1)$$

$$\forall t, t', r.\ (\mathtt{AccID}(r) = \mathtt{ID}(t) \wedge \mathtt{bal}(t) > \mathtt{Amount}(t) \wedge \mathtt{AccID}(r) = \mathtt{ID}(t') \wedge \mathtt{bal}(t') > \mathtt{Amount}(t') \wedge t \xrightarrow{\mathsf{ar}} t') \Rightarrow t \xrightarrow{\mathsf{WW}} t' \quad (2)$$

$$\forall t, t'.\ t \xrightarrow{\mathsf{RW}} t' \Rightarrow \neg(t' \xrightarrow{\mathsf{vis}} t) \quad (3)$$

$$\forall t, t'.\ t \xrightarrow{\mathsf{WW}} t' \Rightarrow t \xrightarrow{\mathsf{vis}} t' \quad (4)$$

We use t, t' to denote invocations of the transaction, and r to denote a record in the database. We define the function `AccID` to access the primary key of a record. Similarly, `ID`, `Amount`, etc. are functions which map an invocation to its parameters and local variables. The existence of a dependence between two invocations forces the existence of a record that both invocations must access, as well as conditions on the local variables required to perform the access (Eqn. 1). On the other hand, if two invocations are guaranteed to write to the same location, there must exist a WW dependency between them (Eqn. 2). Now, it is not possible to have invocations t_1 and t_2, obeying Eqns. (1)-(4) such that $t_1 \xrightarrow{\mathsf{RW}} t_2$ and $t_2 \xrightarrow{\mathsf{RW}} t_1$, the condition necessary to induce a cycle and thus manifest a serializability violation.

In fact, it is not possible to have a cycle of any arbitrary length in a dependency graph of this application under PSI. To show this, we use the following observation: any long path in a dependency graph generated by the above application will have chords in it, resulting in a shorter path. In fact, it can be shown that the shortest path between any two invocations in any dependency graph of the application (if there is a path) will always be less than or equal

to 3. This can shown by using the above constraints (1)-(4) (and adding similar constraints for WR edges), instantiating a path of length 4 such that there is no chord between any of the nodes involved in the path, and then showing the unsatisfiability of such an encoding. Since a cycle is also a path, it is now sufficient to only check for cycles of length 3, since any longer cycle will necessarily induce a cycle of length less than or equal to 3.

Intuitively, this is happening in the banking application because the presence of any dependency edge between two nodes implies that both invocations must access the same account, and at least one of them must perform a write. Further, any two writes are always related by a WW edge. Now, as shown in Figure 3, in any path of length 4 in the dependency graph, one of t_1 or t_2 and one of t_4 or t_5 must be a write, which implies a chord between the two writes. Hence, there will always be a shorter path of length less than or equal to 3 between t_1 and t_5.

3 Preliminaries

3.1 Input Language and Database Model

$$\texttt{v} \in \texttt{Variables} \quad \texttt{f} \in \texttt{Fields} \quad \mathcal{Q} \in \{\texttt{MIN}, \texttt{MAX}, \texttt{COUNT}\}$$
$$\oplus \in \{+, -, \times, /\} \quad \odot \in \{<, \leq, =, >, \geq\} \quad \circ \in \{\wedge, \vee\}$$

$$\begin{array}{lcl}
e_d & \coloneqq & \texttt{f} \mid \texttt{v} \mid e_d \oplus e_d \mid \mathbb{Z} \\
\phi_d & \coloneqq & \texttt{f} \odot e_d \mid \texttt{f} \in \texttt{v} \mid \neg\phi_d \mid \phi_d \circ \phi_d \\
e_c & \coloneqq & \texttt{v} \mid \texttt{CHOOSE v} \mid e_c \oplus e_c \mid \mathbb{Z} \\
\phi_c & \coloneqq & \texttt{v} \odot e_c \mid \texttt{v} = \texttt{NULL} \mid \texttt{v}_1 \in \texttt{v}_2 \mid \neg\phi_c \mid \phi_c \circ \phi_c \\
c & \coloneqq & \texttt{SELECT } \bar{\texttt{f}} \texttt{ AS v WHERE } \phi_\texttt{d} \mid \texttt{SELECT } \mathcal{Q}\ \texttt{f AS v WHERE } \phi_\texttt{d} \mid \texttt{UPDATE SET f} = \texttt{e}_\texttt{c} \texttt{ WHERE } \phi_\texttt{d} \mid \\
 & & \texttt{INSERT VALUES } \bar{\texttt{f}} = \bar{\texttt{e}}_\texttt{c} \mid \texttt{DELETE WHERE } \phi_\texttt{d} \mid \texttt{v} = \texttt{e}_\texttt{c} \mid \texttt{IF } \phi_\texttt{c} \texttt{ THEN c ELSE c} \mid \texttt{c ; c} \\
 & & \texttt{FOREACH } \texttt{v}_1 \texttt{ IN } \texttt{v}_2 \texttt{ DO c END} \mid \texttt{SKIP} \\
vlist & \coloneqq & \texttt{v} \mid vlist, vlist \\
\mathcal{T} & \coloneqq & \texttt{Tname}(vlist)\{\texttt{c}\}
\end{array}$$

We start with description of the language of transactional programs in our framework. We assume a database model, where data is organized in tables with multiple records, where each record has multiple fields and transactions can insert/delete records and read/modify fields in selected records. The grammar is essentially a simplified version of standard SQL, allowing SQL statements which access the database to be combined with usual program connectives such as conditionals, sequencing and loops. Every transactional program $\mathcal{T}$ has a set of parameter variables ($vlist$) which are instantiated with values on invocation, and a set of local variables which are used to store intermediate values from the database (typically as output of `SELECT` queries). For a transactional program $\mathcal{T}$, let $\texttt{Vars}(\mathcal{T})$ be the set of parameters and local variables of $\mathcal{T}$. Let $\texttt{Stmts}(\mathcal{T})$ be the set of SQL statements (i.e. `INSERT`, `DELETE`, `SELECT` or `UPDATE`) in $\mathcal{T}$.

To simplify the presentation, we will assume that there is only one table and each record is a set of values indexed by the set `Fields`. Furthermore all fields store integer values. The `FOREACH` loop iterates over a set of records in $\texttt{v}_2$, and assigns $\texttt{v}_1$ to an individual record during each iteration. We call $\texttt{v}_2$ as the loop variable. Let $\mathcal{D}(\texttt{v})$ denote the nesting depth of v, which is 0 if v is assigned a value outside any loop (or is a parameter variable), and otherwise is the number of enclosing loops. For a variable v assigned a value inside a loop, let $\texttt{LVar}(\texttt{v}, i)$ denote the loop variable at depth i, for all $1 \leq i \leq \mathcal{D}(\texttt{v})$.

SQL statements use predicates ϕ_d to select records that would be accessed/modified, where ϕ_d allows all boolean combinations of comparison predicates between fields and values. Conditionals used inside `IF` statements (ϕ_c) are only allowed to used local variables and

parameters. To check whether the output of a `SELECT` query is empty, we use the conditional expression `v = NULL`, where `v` stores the output of the query.

We assume a fixed non-empty subset of `Fields` to be the primary key `PK`. Any two records must have distinct values in at least one of their `PK` fields. Assume that there is a special field called `Alive` $\in$ `Fields` whose value is 1 if the record is in the database, 0 otherwise. Initially, all records are not `Alive`. When a record is inserted into the database, it becomes `Alive`, and when the record is deleted, it again becomes not `Alive`.

3.2 Abstract Executions

Executions of transactional programs in our framework are expressed using an event structure, which is based on the approach used in [5]. The execution of a transaction instance consists of events, which are database operations. A database operation is a read or write to a field of a record. Let $\mathcal{R} = \texttt{PK} \to \mathbb{Z}$ be the set of all possible primary keys. Then, the set of all database operations is $\mathcal{O} = \{\texttt{wri}(r, f, n) \mid r \in \mathcal{R}, f \in \texttt{Fields} \setminus \texttt{PK}, n \in \mathbb{Z}\} \cup \{\texttt{rd}(r, f, n) \mid r \in \mathcal{R}, f \in \texttt{Fields}, n \in \mathbb{Z}\}$.

To simplify the presentation, we assume that a transaction reads (writes) at most once from (to) a field of a record and does not read any record that it writes, inserts or deletes. These assumptions allow us to ignore the ordering among events of a single transaction instance. Our approach can be easily adapted if these assumptions are not satisfied.

▶ **Definition 1** (Transaction Instance)**.** A transaction instance is a tuple $\sigma = (\texttt{TID}, \varepsilon)$, where `TID` is a unique transaction instance-ID and $\varepsilon \subseteq \mathcal{O}$ is a set of events.

In this work, we assume that transactions are executed in an environment which guarantees atomicity and isolation (also called atomic visibility [12]). That is, either all events of a transaction are made visible to other transactions, or none are, and the same set of transactions are visible to all events in a transaction. Atomicity and isolation are crucial properties for transactional programs, and both can be implemented efficiently in a replicated, distributed environment [9, 3]. Note that atomicity and isolation does not guarantee serializability, as seen in example in §2, and our goal is to explore serializability in this context of weak consistency.

▶ **Definition 2** (Abstract Execution)**.** An abstract execution is a tuple $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$, where Σ is a set of transaction instances, $\mathsf{vis} \subseteq \Sigma \times \Sigma$ is an anti-symmetric, irreflexive relation, and $\mathsf{ar} \subseteq \Sigma \times \Sigma$ is a total order on Σ such that $\mathsf{vis} \subseteq \mathsf{ar}$.

Intuitively, given transaction instances σ, σ' in an abstract execution χ, if $\sigma \xrightarrow{\mathsf{vis}} \sigma'$, then all writes performed by σ are visible to σ' and hence may affect the output of the reads performed by σ'. ar is used to order all writes to the same location. We use the notation $\sigma \vdash o$ to specify that transaction instance σ performs a database operation o. The length of an abstract execution is defined to be the number of transaction instances involved in the execution (i.e. $|\Sigma|$).

Given a set of transaction instances Σ', we use the notation $[\Sigma']_{<\texttt{wri}(r,f)>} = \{\sigma \in \Sigma' \mid \sigma \vdash \texttt{wri}(r, f, n), n \in \mathbb{Z}\}$ to denote the set of transactions which are writing to field `f` of record `r`. We use the notation $\mathsf{MAX}_{\mathsf{ar}}(\Sigma')$ to denote $\sigma \in \Sigma'$ such that $\forall \sigma' \in \Sigma'.\ \sigma = \sigma' \vee \sigma' \xrightarrow{\mathsf{ar}} \sigma$. Given a transaction instance σ, we use $\mathsf{vis}^{-1}(\sigma)$ to denote the set $\{\sigma' \in \Sigma \mid \sigma' \xrightarrow{\mathsf{vis}} \sigma\}$. The last writer wins nature of the database dictates that a transaction reads the most recent value (according to ar) written by the transactions visible to it. Formally, this is specified as follows: $\sigma \vdash \texttt{rd}(r, f, n) \Rightarrow (f \notin \texttt{PK} \Rightarrow \mathsf{MAX}_{\mathsf{ar}}([\mathsf{vis}^{-1}(\sigma)]_{<\texttt{wri}(r,f)>}) \vdash \texttt{wri}(r, f, n)) \wedge (f \in \texttt{PK} \Rightarrow r(f) = n)$.

▶ **Definition 3** (Dependency Graph). Given an abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$, the dependency graph $G_\chi = (\Sigma, E)$ is a directed, edge-labeled multigraph where the edges and their labels are defined as follows :

- $\sigma \xrightarrow{\mathsf{WR}_{r,f}} \sigma'$ if $\sigma' \vdash \mathtt{rd}(r, f, n)$ and $\sigma = \mathsf{MAX}_{\mathsf{ar}}([\mathsf{vis}^{-1}(\sigma')]_{<\mathtt{wri}(r,f)>})$.
- $\sigma \xrightarrow{\mathsf{WW}_{r,f}} \sigma'$ if $\sigma \vdash \mathtt{wri}(r, f, n)$, $\sigma' \vdash \mathtt{wri}(r, f, m)$ and $\sigma \xrightarrow{\mathsf{ar}} \sigma'$.
- $\sigma \xrightarrow{\mathsf{RW}_{r,f}} \sigma'$ if $\sigma \vdash \mathtt{rd}(r, f, n)$, $\sigma' \vdash \mathtt{wri}(r, f, m)$ and there exists another transaction instance σ'' such that $\sigma'' \xrightarrow{\mathsf{WR}_{r,f}} \sigma$ and $\sigma'' \xrightarrow{\mathsf{WW}_{r,f}} \sigma'$.

Edges in the dependency graph G_χ also induce corresponding binary relations on the transaction instances (we use the same notation for these relations). Let $\mathsf{WR}, \mathsf{WW}, \mathsf{RW}$ be the union of $\mathsf{WR}_{r,f}, \mathsf{WW}_{r,f}, \mathsf{RW}_{r,f}$ for all r, f respectively. The following lemma follows directly from the definition[1]:

▶ **Lemma 4.** *Given an abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$ and its dependency graph $G_\chi = (\Sigma, E)$, the following are true:*

- *If $\sigma \xrightarrow{\mathsf{WR}_{r,f}} \sigma' \in E$, then $\sigma \xrightarrow{\mathsf{vis}} \sigma'$.*
- *If $\sigma \xrightarrow{\mathsf{WW}_{r,f}} \sigma' \in E$, then $\sigma \xrightarrow{\mathsf{ar}} \sigma'$.*
- *If $\sigma \xrightarrow{\mathsf{RW}_{r,f}} \sigma' \in E$, then $\neg(\sigma' \xrightarrow{\mathsf{vis}} \sigma)$.*

In our framework, transaction instances are generated by assigning values to all the parameter variables of a transactional program $\mathcal{T}$, written using the grammar specified in §3.1. We use the notation $\Gamma(\sigma)$ to denote the transactional program $\mathcal{T}$ associated with transaction instance σ.

Different weak consistency and weak isolation models can be expressed by placing constraints on vis and ar relations associated with an abstract execution. This gives rise to the notion of *valid* abstract executions under a specific model, which are executions satisfying the constraints associated with those models. Below, we provide examples of several known weak consistency and weak isolation models:

- Full Serializability : $\Psi_{\text{Ser}} \triangleq \mathsf{vis} = \mathsf{ar}$
- Selective Serializability for Transactional Programs $\mathcal{T}_1, \mathcal{T}_2$ [16] : $\Psi_{\text{Ser}(\mathcal{T}_1,\mathcal{T}_2)} \triangleq \forall \sigma_1, \sigma_2.$ $((\Gamma(\sigma_1) = \mathcal{T}_1 \wedge \Gamma(\sigma_2) = \mathcal{T}_2) \vee (\Gamma(\sigma_1) = \mathcal{T}_2 \wedge \Gamma(\sigma_2) = \mathcal{T}_1) \wedge \sigma_1 \xrightarrow{\mathsf{ar}} \sigma_2) \Rightarrow \sigma_1 \xrightarrow{\mathsf{vis}} \sigma_2$
- Causal Consistency (CC) [22] : $\Psi_{\text{CC}} \triangleq \forall \sigma_1, \sigma_2, \sigma_3.\ \sigma_1 \xrightarrow{\mathsf{vis}} \sigma_2 \wedge \sigma_2 \xrightarrow{\mathsf{vis}} \sigma_3 \Rightarrow \sigma_1 \xrightarrow{\mathsf{vis}} \sigma_3$
- Prefix Consistency (PC) (equivalent to *repeatable read* in centralized databases) [27, 10] : $\Psi_{\text{PC}} \triangleq \forall \sigma_1, \sigma_2, \sigma_3.\ \sigma_1 \xrightarrow{\mathsf{ar}} \sigma_2 \wedge \sigma_2 \xrightarrow{\mathsf{vis}} \sigma_3 \Rightarrow \sigma_1 \xrightarrow{\mathsf{vis}} \sigma_3$
- Parallel Snapshot Isolation (PSI) [26] : $\Psi_{\text{PSI}} \triangleq \forall \sigma_1, \sigma_2.\ \sigma_1 \xrightarrow{\mathsf{WW}} \sigma_2 \Rightarrow \sigma_1 \xrightarrow{\mathsf{vis}} \sigma_2$

Different models can be also be combined together to create a hybrid model. For example, $\Psi_{PSI} \wedge \Psi_{PC}$ is equivalent to Snapshot Isolation [4] in centralized databases. Below, we formalize the classical notion of conflict serializability [6] in our setting and then relate it to the presence of cycles in the dependency graph.

▶ **Definition 5** (Serializable Execution). An abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$ is said to be serializable if there exists another abstract execution $\chi' = (\Sigma, \mathsf{vis}', \mathsf{ar}')$ which satisfies Ψ_{Ser} such that G_χ and $G_{\chi'}$ are isomorphic.

▶ **Theorem 6.** *Given an abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$, if there is no cycle in the dependency graph G_χ, then χ is serializable.*

[1] All proofs can be found in Appendix C in the extended version of the paper[24].

3.3 Operational Semantics

We now propose an operational semantics to generate abstract executions from transactional programs under a consistency specification. The purpose of the operational semantics is to link SQL statements with abstract database operations, and to prove the soundness of our encoding in FOL. Here, we only provide an informal overview; the full operational semantics can be found in Appendix B of [24].

The semantics is a transition system $\mathcal{S}_{\mathbb{T},\Psi} = (\Delta, \rightarrow)$ parametrized over a set of transactional programs $\mathbb{T}$ and a consistency specification Ψ. The state ($\delta \in \Delta$) is stored as a tuple $(\Sigma, \mathsf{vis}, \mathsf{ar}, \mathcal{P})$ where Σ is the set of committed transaction instances, vis and ar are relations on Σ, and $\mathcal{P}$ is the running pool of transaction instances. The transitions are of two types : spawning a new instantiation of a transactional program $\mathcal{T} \in \mathbb{T}$ or executing a statement of a transaction instance in the running pool. When a new execution of a transaction instance begins, a subset of Σ is non-deterministically selected to be made visible to the new instance. A view of the database is constructed for the new instance based on the set of visible transactions and the ar relation (ensuring the last writer wins policy), and all queries of the transaction instance are answered on the basis of this view. At any point, any transaction instance from $\mathcal{P}$ can be non-deterministically selected for execution of its next statement. Any new event generated during the execution of a transaction instance is stored in the running pool. Finally, when a transaction instance wants to commit, it is checked whether the consistency specification (Ψ) is satisfied if the instance were to commit, and if yes, it is added to Σ. We can now define a valid abstract execution in terms of traces of the transition system:

▶ **Definition 7** (Valid execution of $\mathbb{T}$ under Ψ)**.** An abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$ is said to be a valid execution produced by $\mathbb{T}$ under Ψ if there exists a trace $(\{\}, \{\}, \{\}, \{\}) \rightarrow^* (\Sigma, \mathsf{vis}, \mathsf{ar}, \{\})$ of the transition system $\mathcal{S}_{\mathbb{T},\Psi}$.

4 FOL Encoding

4.1 Vocabulary

Given a set of transactional programs $\mathbb{T}$ and a consistency specification Ψ we now show how to construct a formula in FOL such that any valid abstract execution χ of $\mathbb{T}$ under Ψ and its dependency graph G_χ is a satisfying model of the formula. The encoding is parametric over $\mathbb{T}$ and Ψ. We first describe the vocabulary of the encoding. We define two uninterpreted sorts τ and R, such that members of τ are transaction instances, and members of R are records. In addition, we also define a finite sort $\mathbb{T}$ which contains the transaction types, where each type is a transactional program.

The function $\Gamma : \tau \rightarrow \mathbb{T}$ associates each transaction instance with its type. For each transactional program $\mathcal{T} \in \mathbb{T}$ and for each variable $\mathtt{v} \in \mathtt{Vars}(\mathcal{T})$, the variable projection function $\rho_\mathtt{v}$ gives the value of $\mathtt{v}$ in a transaction instance. The signature of $\rho_\mathtt{v}$ depends upon the type of the variable and whether it is assigned inside a loop. First, let us consider variables which are assigned values outside any loop. In our framework, variables are of two types : a value or a set of values. Further, the value can be either an integer (e.g. the parameter `ID` of the `withdraw` transaction) or a record. Let $\mathbb{V} = \mathbb{Z} \cup R$. If $\mathtt{v}$ is a value, the $\rho_\mathtt{v}$ has the signature $\tau \rightarrow \mathbb{V}$. If $\mathtt{v}$ is a set of values, then $\rho_\mathtt{v}$ is a predicate with signature $\mathbb{V} \times \tau \rightarrow \mathbb{B}$, such that $\rho_\mathtt{v}(r, t)$ is true if r belongs to $\mathtt{v}$ in the transaction instance t.

Consider a loop of the form : `FOREACH` $\mathtt{v}_1$ `IN` $\mathtt{v}_2$ `DO` c `END`. All local variables which are assigned values inside the loop body (including $\mathtt{v}_1$) will be indexed by values in the set $\mathtt{v}_2$. Hence, if a local variable $\mathtt{v}_3$ is assigned inside the loop, and it is a value, then $\rho_{\mathtt{v}_3}$ will have the signature $\mathbb{V} \times \tau \to \mathbb{V}$. On the other hand, if $\mathtt{v}_3$ stores a set of values, then $\rho_{\mathtt{v}_3}$ will have the signature $\mathbb{V} \times \mathbb{V} \times \tau \to \mathbb{B}$, with the interpretation that $\rho_{\mathtt{v}_3}(r_1, r_2, t)$ is true if $\mathtt{v}_3$ contains r_2 in the iteration where $\mathtt{v}_1$ is $r_1 \in \mathtt{v}_2$. Similarly, nested loops will have local variables which are indexed by records in all enclosing loops.

To summarize, the signature of $\rho_{\mathtt{v}}$ is either $\mathbb{V}^{\mathcal{D}(\mathtt{v})} \times \tau \to \mathbb{V}$ or $\mathbb{V}^{\mathcal{D}(\mathtt{v})+1} \times \tau \to \mathbb{B}$. Similar to the variable projection function, the field projection function $\rho_{\mathtt{f}} : R \to \mathbb{Z}$ is defined for each field $\mathtt{f} \in \mathtt{Fields}$, such that $\rho_{\mathtt{f}}(r)$ gives the value of $\mathtt{f}$ in a record r.

We define predicates $\mathsf{WR}, \mathsf{WW}, \mathsf{RW}$ all of type $\tau \times \tau \to \mathbb{B}$ which specify the read, write and anti-dependency relations respectively between transaction instances. We also define predicates $\mathsf{WR}^R, \mathsf{RW}^R, \mathsf{WW}^R$ all of type $R \times \mathtt{Fields} \times \tau \times \tau \to \mathbb{B}$ which provide more context by also specifying the records and fields causing the dependencies. Predicates $\mathsf{vis}, \mathsf{ar}$ of type $\tau \times \tau \to \mathbb{B}$ specify the visibility and arbitration relation between transaction instances. The predicate $\mathtt{Alive} : R \times \tau \to \mathbb{B}$ indicates whether a record is `Alive` for a transaction instance.

4.2 Relating Dependences with Abstract Executions

By Lemma 4, in any abstract execution, the presence of a dependency edge between two transaction instances enforces constraints on the vis and/or ar relations between the two instances. The following formula encodes this along with basic constraints satisfied on vis and ar:

$$\begin{aligned}\varphi_{basic} = \textsc{TotalOrder}(\mathsf{ar}) \wedge \forall(t, s : \tau).\ (\mathsf{vis}(t, s) \Rightarrow \neg\mathsf{vis}(s, t)) \wedge (\mathsf{vis}(t, s) \Rightarrow \mathsf{ar}(t, s)) \\ \wedge (\mathsf{WR}(t, s) \Rightarrow \mathsf{vis}(t, s)) \wedge (\mathsf{WW}(t, s) \Rightarrow \mathsf{ar}(t, s)) \wedge (\mathsf{RW}(t, s) \Rightarrow \neg\mathsf{vis}(s, t)) \qquad (5)\end{aligned}$$

The following formula encodes a fundamental constraint involving the dependency relations on the same field of the same record due to the last writer wins nature of the database:

$$\varphi_{dep} = \bigwedge_{\mathtt{f} \in \mathtt{Fields}} \forall(t_1, t_2, t_3 : T)(r : R).\ \mathsf{WR}^R(r, \mathtt{f}, t_2, t_1) \wedge \mathsf{RW}^R(r, \mathtt{f}, t_1, t_3) \Rightarrow \mathsf{WW}^R(r, \mathtt{f}, t_2, t_3)$$

Finally, the consistency specification Ψ can be directly encoded using the relations and functions defined in our vocabulary (we denote this formula by φ_Ψ).

4.3 Relating dependences with transactional programs

The presence of a dependency edge between two transaction instances places constraints on the type of transactional programs generating the instances and their parameters. To automatically infer these constraints, we use the following strategy : if there is a dependency edge between two instances, then there must exist SQL statements in both transactions which access a common record.

To encode this, we first extract the conditions under which a SQL statement in a transactional program can be executed. By performing a simple syntactic analysis over the code of a transaction $\mathcal{T}$, we obtain a mapping $\Lambda_{\mathcal{T}}$ from each SQL statement in $\mathtt{Stmts}(\mathcal{T})$ to a conjunction of enclosing `IF` conditionals (the complete algorithm can be found in Appendix A of [24]).

The FOL encoding of all conditionals in a program and all `WHERE` clauses in a SQL statement is constructed by replacing variables and fields with the corresponding variable projection and field projection functions respectively. A representative set of rules for this

$$
\begin{array}{lcll}
[\![\mathtt{v} = \mathtt{NULL}]\!]_t & = & (\exists(r_1,\ldots,r_{\mathcal{D}(\mathtt{v})} : R).\ \bigwedge_{i=1}^{\mathcal{D}(\mathtt{v})} \mathcal{V}([\![r_i \in \mathtt{LVar}(\mathtt{v},i)]\!]_t), & \mathsf{fresh}(r_1,\ldots,r_{\mathcal{D}(\mathtt{v})},r) \\
 & & \forall(r:R).\neg\rho_{\mathtt{v}}(r_1,\ldots,r_{\mathcal{D}(\mathtt{v})},r,t)) & \\
[\![r \in \mathtt{v}]\!]_t & = & (\exists(r_1,r_2,\ldots,r_{\mathcal{D}(\mathtt{v})} : R).\ \bigwedge_{i=1}^{\mathcal{D}(\mathtt{v})} \mathcal{V}([\![r_i \in \mathtt{LVar}(\mathtt{v},i)]\!]_t), & \mathsf{fresh}(r_1,\ldots,r_{\mathcal{D}(\mathtt{v})}) \\
 & & \rho_{\mathtt{v}}(r_1,\ldots,r_{\mathcal{D}(\mathtt{v})},r,t)) & \\
[\![\mathtt{v}_1 \in \mathtt{v}_2]\!]_t & = & (\varphi_1 \wedge \varphi_2, \psi_2) & [\![\mathtt{v}_1]\!]_t = (\varphi_1,\psi_1) \\
 & & & [\![\psi_1 \in \mathtt{v}_2]\!]_t = (\varphi_2,\psi_2) \\
[\![\mathtt{f} \odot e]\!]_{t,r} & = & \begin{cases} (\varphi, \rho_{\mathtt{f}}(r) \odot \psi) & \text{if } \mathtt{f} \cup \mathcal{F}(e) \subseteq \mathtt{PK} \\ (\mathsf{true},\mathsf{true}) & \text{otherwise} \end{cases} & \\
 & & & [\![e]\!]_{t,r} = (\varphi,\psi)
\end{array}
$$

Figure 4 Encoding conditionals and `WHERE` clauses.

encoding are shown in Fig. 4. For conditionals ϕ used in `IF` statements, we use the notation $[\![\phi]\!]_t$ to describe the FOL encoding specialized to transaction instance t. The interpretation is that $[\![\phi]\!]_t$ is satisfiable only if the conditional ϕ is true in the transaction instance t. If ϕ is inside a loop, then $[\![\phi]\!]_t$ must be satisfiable if ϕ is true in any arbitrary iteration of the enclosing loop(s) in t. For this reason, $[\![\phi]\!]_t$ is actually represented as a tuple (φ,ψ), where φ chooses any arbitrary iteration of enclosing loops, and the formula ψ is the value of the conditional in that iteration. We define an evaluation function $\mathcal{V}(\varphi,\psi) = \varphi \wedge \psi$ which gives the final FOL encoding.

The formula φ chooses an iteration by instantiating records belonging to loop variables of all enclosing loops. For example, consider the encoding of $\mathtt{v} = \mathtt{NULL}$. Here, φ instantiates a record belonging to the loop variable of every enclosing loop of $\mathtt{v}$ (encoded as $\mathcal{V}([\![r_i \in \mathtt{LVar}(\mathtt{v},i)]\!]_t)$), and ψ encodes that $\rho_{\mathtt{v}}$ in the chosen iteration is false for every record. Similarly, in the encoding of $[\![r \in \mathtt{v}]\!]_t$, $\rho_{\mathtt{v}}$ must be true for the record r. In the encoding of $[\![\mathtt{v}_1 \in \mathtt{v}_2]\!]_t$, we first obtain the value of $\mathtt{v}_1$ (the second term in the tuple $[\![\mathtt{v}_1]\!]_t$), and then check whether it is present in $\mathtt{v}_2$.

A similar procedure is used to obtain the encoding of the `WHERE` clauses used inside SQL statements. Since `WHERE` clauses are evaluated on records, the encoding is specialized on both records and transaction instances, for which we use the notation $[\![\phi]\!]_{t,r}$. The interpretation is that $[\![\phi]\!]_{t,r}$ is satisfiable only if ϕ is true for transaction instance t on record r. The encoding replaces field accesses with the corresponding field projection function applied on r. Note that the field projection function is only used for primary key fields which are accessed within `WHERE` clauses (expressed as $\mathcal{F} \subseteq \mathtt{PK}$). The complete encoding for all types of conditionals and `WHERE` clauses can be found in the Appendix A of [24].

As stated earlier, our strategy is to encode the necessary condition for a dependency edge based on the access of a common record. For each pair of transaction types $\mathcal{T}_1, \mathcal{T}_2 \in \mathbb{T}$, each dependency type $\mathcal{R} \in \{\mathsf{WR},\mathsf{RW},\mathsf{WW}\}$, and each pair of SQL statements $\mathtt{c}_1 \in \mathtt{Stmts}(\mathcal{T}_1), \mathtt{c}_2 \in \mathtt{Stmts}(\mathcal{T}_2)$, we compute a necessary condition $\eta^{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$ for dependency $\mathcal{R}$ to exist between instances t_1 and t_2 of types $\mathcal{T}_1$ and $\mathcal{T}_2$ due to statements $\mathtt{c}_1$ and $\mathtt{c}_2$ respectively. The following formula encodes the fact that a dependency between two transaction instances can be caused due to a dependency between any two SQL statements in those transactions:

$$
\varphi_{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2} \triangleq \forall(t_1,t_2:\tau).(\Gamma(t_1) = \mathcal{T}_1 \wedge \Gamma(t_2) = \mathcal{T}_2 \wedge \mathcal{R}(t_1,t_2)) \Rightarrow \bigvee_{\substack{\mathtt{c}_1 \in \mathtt{Stmts}(\mathcal{T}_1) \\ \mathtt{c}_2 \in \mathtt{Stmts}(\mathcal{T}_2)}} \eta^{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)
$$

The general format of $\eta^{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$ is a conjunction of the conditionals required to execute the statements $\mathtt{c}_1$ and $\mathtt{c}_2$ (i.e. $\Lambda_{\mathcal{T}_1}(\mathtt{c}_1)$ and $\Lambda_{\mathcal{T}_2}(\mathtt{c}_2)$) in t_1 and t_2 resp. and the `WHERE` clauses of the two statements evaluated on some record r. If they can never access the

same field of the same record, then $\eta^{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$ is simply false. While this is the general format of the clauses, in addition, we can also infer more information depending upon the type of the SQL statements. To illustrate this we present a sample rule below:

$$\frac{\begin{array}{c}\mathtt{c}_1 \equiv \texttt{SELECT MAX f AS v WHERE } \phi_1 \quad \mathtt{c}_2 \equiv \texttt{UPDATE SET f = e WHERE } \phi_2 \\ \mathtt{c}_1 \in \texttt{Stmts}(\mathcal{T}_1) \quad \mathtt{c}_2 \in \texttt{Stmts}(\mathcal{T}_2) \quad \Gamma(t_1)=\mathcal{T}_1 \quad \Gamma(t_2)=\mathcal{T}_2 \quad [\![\mathtt{v}]\!]_{t_1}=(\varphi_1,\psi_1) \quad [\![\mathtt{e}]\!]_{t_2}=(\varphi_2,\psi_2)\end{array}}{\begin{array}{c}\eta^{\mathsf{RW}\rightarrow,\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2) = (\exists r.\ \mathcal{V}([\![\Lambda_{\mathcal{T}_1}(\mathtt{c}_1)]\!]_{t_1}) \wedge \mathcal{V}([\![\phi_1]\!]_{t_1,r}) \wedge \mathcal{V}([\![\Lambda_{\mathcal{T}_2}(\mathtt{c}_2)]\!]_{t_2}) \wedge \mathcal{V}([\![\phi_2]\!]_{t_2,r}) \wedge \\ \texttt{Alive}(r,t_2) \wedge \varphi_1 \wedge \varphi_2 \wedge \psi_1 < \psi_2)\end{array}}$$

The rule encodes a necessary condition for an anti-dependency to exist from a `SELECT MAX` to a `UPDATE` statement. First, it encodes that the conflicting SQL statements actually execute in their respective transactions and there is a common record which satisfies the `WHERE` clauses of both statements. `SELECT MAX` selects the record with the maximum value in the field `f` among all records that satisfy ϕ_1, and stores the value in variable `v`. If there is an anti-dependency from `SELECT MAX` to `UPDATE`, then the updated value must be greater than the output of `SELECT MAX`, because otherwise, the update does not affect the output of `SELECT MAX`.

In addition, some transaction instances may be guaranteed to execute certain SQL statements, which forces the presence of a dependency edge between them. For example, if two transaction instances are guaranteed to update the same field of a record, then there must be a WW dependeny between them. For each pair of transaction types $\mathcal{T}_1, \mathcal{T}_2 \in \mathbb{T}$, each dependency type $\mathcal{R} \in \{\mathsf{WR}, \mathsf{RW}, \mathsf{WW}\}$, and each pair of SQL statements $\mathtt{c}_1 \in \texttt{Stmts}(\mathcal{T}_1), \mathtt{c}_2 \in \texttt{Stmts}(\mathcal{T}_2)$, we compute a condition $\eta^{\rightarrow\mathcal{R},\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$ which forces the dependency $\mathcal{R}$ to exist between instances t_1 and t_2 of types $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively due to $\mathtt{c}_1$ and $\mathtt{c}_2$. The following formula encodes this:

$$\varphi_{\rightarrow\mathcal{R},\mathcal{T}_1,\mathcal{T}_2} \triangleq \forall t_1,t_2.(\Gamma(t_1)=\mathcal{T}_1 \wedge \Gamma(t_2)=\mathcal{T}_2 \wedge \bigvee_{\substack{\mathtt{c}_1\in\texttt{Stmts}(\mathcal{T}_1)\\ \mathtt{c}_2\in\texttt{Stmts}(\mathcal{T}_2)}} \eta^{\rightarrow\mathcal{R},\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)) \Rightarrow \mathcal{R}(t_1,t_2)$$

$$\frac{\begin{array}{c}\mathtt{c}_1 \equiv \texttt{UPDATE SET f} = \mathtt{e}_1 \texttt{ WHERE } \phi_1 \quad \mathtt{c}_2 \equiv \texttt{UPDATE SET f} = \mathtt{e}_2 \texttt{ WHERE } \phi_2 \\ \mathtt{c}_1 \in \texttt{Stmts}(\mathcal{T}_1) \quad \mathtt{c}_2 \in \texttt{Stmts}(\mathcal{T}_2) \quad \Gamma(t_1)=\mathcal{T}_1 \quad \Gamma(t_2)=\mathcal{T}_2\end{array}}{\begin{array}{c}\eta^{\rightarrow\mathsf{WW},\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2) = (\exists r.\ \mathcal{V}([\![\Lambda_{\mathcal{T}_1}(\mathtt{c}_1)]\!]_{t_1}) \wedge \mathcal{V}([\![\phi_1]\!]_{t_1,r}) \wedge \mathcal{V}([\![\Lambda_{\mathcal{T}_1}(\mathtt{c}_1)]\!]_{t_2}) \wedge \mathcal{V}([\![\phi_2]\!]_{t_2,r}) \wedge \\ \texttt{Alive}(r,t_1) \wedge \texttt{Alive}(r,t_2) \wedge \mathsf{ar}(t_1,t_2))\end{array}}$$

$\eta^{\rightarrow\mathcal{R},\mathcal{T}_1,\mathcal{T}_2}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$ is computed in the same manner as $\eta^{\mathcal{R},\mathcal{T}_1,\mathcal{T}_2\rightarrow}_{\mathtt{c}_1,\mathtt{c}_2}(t_1,t_2)$. As an example consider the above rule. Two `UPDATE` statements modifying the same field are guaranteed to cause a WW dependency if both statements actually execute in their respective transactions, and there exists a common record accessed by both statements which is `Alive` to both transactions.

In addition, there are some auxiliary facts which are satisfied by all abstract executions (which we encode as the formula φ_{aux}) such as a record present in the output variable of a `SELECT` query must satisfy the `WHERE` clause of the query, the value of the iterator variable in a loop must belong to the loop variable, etc. For more details, we again refer to the Appendix. The final encoding is defined as follows:

$$\varphi_{\mathbb{T},\Psi} \triangleq \varphi_{basic} \wedge \varphi_{dep} \wedge \bigwedge_{\mathcal{R}\in\{\mathsf{WR},\mathsf{RW},\mathsf{WW}\}} \bigwedge_{\mathcal{T}_1,\mathcal{T}_2\in\mathbb{T}} (\varphi_{\mathcal{R}\rightarrow,\mathcal{T}_1,\mathcal{T}_2} \wedge \varphi_{\rightarrow\mathcal{R},\mathcal{T}_1,\mathcal{T}_2}) \wedge \varphi_\Psi \wedge \varphi_{aux} \quad (6)$$

▶ **Theorem 8.** *Given a set of transactional programs $\mathbb{T}$ and a consistency specification Ψ, for any valid abstract execution $\chi = (\Sigma, \mathsf{vis}, \mathsf{ar})$ generated by $\mathbb{T}$ under Ψ and its dependency*

graph G_χ, there exists a satisfying model of the formula $\varphi_{\mathbb{T},\Psi}$ with $\tau = \Sigma$ and the binary predicates $\mathsf{vis}, \mathsf{ar}, \mathsf{WR}, \mathsf{RW}, \mathsf{WW}$ *being equal to the corresponding relations in χ and G_χ.*

Note that $\varphi_{\mathbb{T},\Psi}$ is always satisfiable, since the empty abstract execution is a satisfying model.

5 Applications

5.1 Bounded Anomaly Detection

By Theorem 6, any execution which violates serializability must have a cycle in its dependency graph. We can directly instantiate a dependency graph which contains a cycle of bounded length and then ask for a satisfying model of the formula built in the previous section which contains the cycle. We introduce a new predicate $D : \tau \times \tau \to \mathbb{B}$ which represents the presence of any dependency edge between two transaction instances : $\varphi_D \triangleq \forall(t_1, t_2 : \tau).\mathsf{D}(t_1, t_2) \Leftrightarrow (t_1 = t_2) \vee \mathsf{WR}(t_1, t_2) \vee \mathsf{RW}(t_1, t_2) \vee \mathsf{WW}(t_1, t_2)$. A cycle of length less than or equal to k can now be directly encoded as follows: $\varphi_{\text{Cycle,k}} \triangleq \exists t_1, \ldots, t_k.\ \bigwedge_{i=1}^{k-1} \mathsf{D}(t_i, t_{i+1}) \wedge \mathsf{D}(t_k, t_1) \wedge (t_1 \neq t_k)$.

▶ **Theorem 9.** *Given a set of transactional programs $\mathbb{T}$ and a consistency specification Ψ, if $\varphi_{\mathbb{T},\Psi} \wedge \varphi_D \wedge \varphi_{\text{Cycle,k}}$ is* UNSAT*, then all valid abstract executions produced by $\mathbb{T}$ under Ψ of length less than or equal to* k *are serializable.*

5.2 Verifying Serializability: The Shortest Path Approach

We propose a condition, which can be also be encoded in FOL, and which if satisfied would imply that it is enough to check for violations of bounded length to prove the absence of violations of any arbitrary length.

The condition is based on the simple observation that any long path in the dependency graph could induce a short path due to chords among the nodes in the path (as demonstrated in the example in §2). This would imply that any long cycle would also induce a short cycle, and hence lack of short cycles would imply the lack of longer cycles. To check for this condition, we encode a shortest path of length k in the dependency graph and then ask whether there is a satisfying model:

$$\varphi_{\text{Shortest Path,k}} \triangleq \exists t_1, \ldots, t_k, t_{k+1}.\ \bigwedge_{i=1}^{k} \mathsf{D}(t_i, t_{i+1}) \wedge \bigwedge_{i=1}^{k-1} \bigwedge_{j=i+2}^{k+1} \neg\mathsf{D}(t_i, t_j) \wedge \bigwedge_{1 \leq i < j \leq k+1} t_i \neq t_j$$

The condition instantiates a path of length k in the dependency graph and also asserts the absence of any chord, which implies that the path is shortest. If there does not exist a shortest path of length k, then there also cannot exist a shortest path of greater length, because if not, such a path would necessarily contain a shortest path of length k. Now, it is enough to check for cycles of length less than or equal to k, because any longer cycle would contain a path of length at least k, which would imply the presence of a shorter path and thus a cycle of length less than or equal to k.

▶ **Theorem 10.** *Given a set of transactional programs $\mathbb{T}$ and a consistency specification Ψ, if both $\varphi_{\mathbb{T},\Psi} \wedge \varphi_D \wedge \varphi_{\text{Shortest Path,k}}$ and $\varphi_{\mathbb{T},\Psi} \wedge \varphi_D \wedge \varphi_{\text{Cycle,k}}$ are* UNSAT*, then all valid abstract executions produced by $\mathbb{T}$ under Ψ are serializable.*

5.3 Verifying Serializability: An Inductive Approach

We now present an alternative approach to verifying serializability which uses the transitivity and irreflexivity of the ar relation to show lack of cycles. In this approach, our goal is to show that if there is a path in the dependency graph from t_1 to t_2, then $t_1 \xrightarrow{\mathsf{ar}} t_2$. By the irreflexivity of ar, this would imply that there cannot be a cycle in the dependency graph. Since paths can be of arbitrary length, we will use the transitivity of ar and an inductive argument to obtain a simple condition which can be encoded in FOL.

▶ **Lemma 11.** *Given a set of transactional programs $\mathbb{T}$, a consistency specification Ψ and a subset of programs $\mathbb{T}' \subseteq \mathbb{T}$,* ***if*** *for all valid executions χ and their dependency graphs G_χ, the following conditions hold:*

1. *if $\sigma_1 \to \sigma_2$ in G_χ and $\Gamma(\sigma_1) \in \mathbb{T}'$, then $\sigma_1 \xrightarrow{\mathsf{ar}} \sigma_2$*
2. *if $\sigma_1 \to \sigma_2 \to \sigma_3$ in G_χ, then either $\sigma_1 \xrightarrow{\mathsf{ar}} \sigma_3$ or $\sigma_2 \xrightarrow{\mathsf{ar}} \sigma_3$*

then *all valid executions which contain at least one instance of a program in $\mathbb{T}'$ are serializable.*

The proof uses an inductive argument to show that if there is path from σ_1, an instance of a program in $\mathbb{T}'$ to any other instance σ_2, then $\sigma_1 \xrightarrow{\mathsf{ar}} \sigma_2$. This would imply that any instance of $\mathbb{T}'$ cannot be present in a cycle. The above conditions can be directly encoded in FOL:

$$\varphi_{\text{Inductive},\mathbb{T}'} \triangleq (\exists(t_1, t_2 : \tau).\ \Gamma(t_1) \in \mathbb{T}' \wedge \mathsf{D}(t_1, t_2) \wedge t_1 \neq t_2 \wedge \neg\mathsf{ar}(t_1, t_2)) \vee$$
$$(\exists(t_1, t_2, t_3 : \tau).\mathsf{D}(t_1, t_2) \wedge \mathsf{D}(t_2, t_3) \wedge \bigwedge_{1 \leq i < j \leq 3} t_i \neq t_j \wedge \neg\mathsf{ar}(t_1, t_3) \wedge \neg\mathsf{ar}(t_2, t_3)) \tag{7}$$

▶ **Theorem 12.** *Given a set of programs $\mathbb{T}$ and a consistency specification Ψ, if $\varphi_{\mathbb{T},\Psi} \wedge \varphi_D \wedge \varphi_{\text{Inductive},\mathbb{T}'}$ is* UNSAT*, then all valid executions of $\mathbb{T}$ under Ψ which contains at least one instance of a program in $\mathbb{T}'$ are serializable.*

If $\mathbb{T}' = \mathbb{T}$, then all valid executions of $\mathbb{T}$ are serializable, otherwise, we can focus only on programs in $\mathbb{T} \setminus \mathbb{T}'$, and re-apply the technique with $\varphi_{\mathbb{T}',\Psi} \wedge \varphi_D \wedge \varphi_{\text{Inductive},\mathbb{T}''}$ for $\mathbb{T}'' \subseteq \mathbb{T}'$. In the next section, we show how we use this technique to verify serializability of TPC-C, a real-world database benchmark.

6 Case Studies

We have developed a tool called ANODE which takes a set of programs written in the language presented in §3.1 and a consistency specification and uses the encoding rules presented in §4 to automatically generate an FOL encoding. We use the Z3 SMT solver to determine the satisfiabiliy of the generated formulae. In order to evaluate the effectiveness of our approach, we have applied the proposed technique on TPC-C [1], a well-known Online Transaction Processing (OLTP) benchmark widely used in the database community, and a Courseware application (used in [19]) which is a representative of course registration systems used in universities.

TPC-C. TPC-C has a complex database schema with 9 tables, and complex application logic in its 5 transactions. The transactions contain loops and conditionals, have multiple parameters and behave differently depending upon the values of the parameters; they also use complex queries such as `SELECT MIN` and `SELECT MAX`. To the best of our knowledge, this is the first automated static analysis for validating serializability of TPC-C under weak consistency.

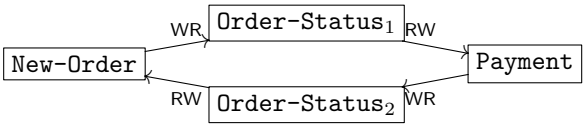


Figure 5 Long fork anomaly in TPC-C under PSI.

Under eventual consistency, TPC-C has a number of 'lost update' anomalies, similar to the anomaly in the banking application described in §2. These anomalies are small in length and were automatically detected using encoding presented in §5.1 (with $k = 2$) . To get rid of these anomalies, we upgraded the consistency specification to PSI [26]. Under PSI, we did not find any anomalies for $k = 2$ or $k = 3$, but for $k = 4$, the 'long fork' anomaly involving the New-Order, Payment and Order-Status transactions was discovered, as shown in Fig. 5.

This anomaly happens because the `New-Order` and `Payment` transactions update two different tables (`Order` and `Customer` table resp.) while the `Order-Status` transaction reads both those tables. Since there is no synchronization between `New-Order` and `Payment` transactions, it is possible for `Order-Status`$_1$ to see the update of `New-Order` but not `Payment`, and the vice versa for `Order-Status`$_2$. We also discovered a similar anomaly involving two instances of `New-Order` and two instances of `Stock-level` transactions.

To get rid of these anomalies, we further upgraded the consistency level to Snapshot Isolation (SI), after which we did not find any anomalies for $k = 4$. We then turned our attention to verifying serializability of TPC-C under SI. We first tried the Shortest Path approach (which worked well for the banking application), but we were able to discover a long path (which can be arbitrarily extended) without any chords. Next, we tried the inductive approach, which was successful in proving serializability of TPC-C. Specifically, with $\mathbb{T}' = \{\texttt{New-Order}, \texttt{Payment}\}$, the formula $\varphi_{\text{Inductive},\mathbb{T}'}$ was shown to be UNSAT, and with the remaining 3 transactions $\varphi_{\text{Inductive},\{\texttt{Delivery}\}}$ was UNSAT. The remaining two transactions do not have any dependencies between them, which implies that all executions of TPC-C under SI are serializable.

Courseware. The Courseware application maintains a database of courses and students, and provides the functionality of adding/removing students and courses, and enrolling students into courses subject to course capacities. Under EC, the following anomalies were discovered by our encoding : (1) two concurrent `Enroll` transactions may enroll students beyond the course capacity, (2) two courses with the same name or two students with the same name may be registered, (3) a student may be enrolled in a course which is being concurrently removed, or the student is being concurrently removed. Note that all these anomalies were discovered for $k = 2$.

In order to remove these violations, we upgraded the consistency model in a number of ways : the `Enroll` transaction was upgraded to PSI, while selective serializability was used for two instances of `AddCourse` and `AddStudent`, and for instances of `Enroll` and `RemCourse`, `Enroll` and `RemStudent`. While these upgrades took care of the above mentioned anomalies, we discovered a new long fork anomaly (for $k = 4$) as shown in Fig. 6. Here, two `Enroll` transactions trying to enroll a student (`s`) into a course (`c`) see conflicting views of the database, with one `Enroll` witnessing the student but not the course, and vice versa for the other. We note that while this is an actual serializability violation, it is completely harmless as both transactions which witness inconsistent database states ultimately fail, so that the final database state is the same as that which manifests at the end of an execution in which

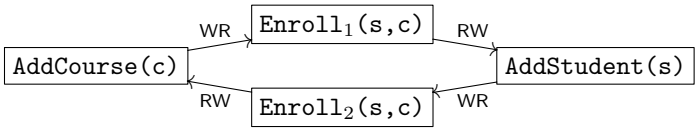


Figure 6 Long fork anomaly in the Courseware application under PSI.

the effects of neither of the two enroll transactions occur. This is a limitation of our analysis as it does not provide any way to ignore harmless serializability violations. We plan to address this issue as part of future work.

In order to remove this violation, we upgraded the consistency level of `Enroll` to SI, after which we did not find any anomalies. Next, we moved to verification, and here we were successfully able to use the Shortest Path approach and prove that there does not exist a shortest path in any dependency graph of the Courseware application of length greater than or equal to 8. Along with the fact there does not exist any cycle of length less than or equal to 8, this implies that any execution of the application is serializable. In all instances, the solver produced its output in a few (< 10) seconds.

7 Related Work and Conclusions

Serializability is a well-studied problem in the database community, but there is a lack of static automated techniques to check for serializability of database applications. Early work by Fekete *et al.* [17] and Jorwekar *et al.* [20] proposed lightweight syntactic analyses to check for serializability under SI in centralized databases, by looking for dangerous structures in the static dependency graph of an application (which is an over-approximation of all possible dynamic dependency graphs). Several recent works [5, 12, 13, 14, 30, 28] have continued along this line, by deriving different types of dangerous structures in dependency graphs that are possible under different weak consistency mechanisms, and then checking for these structures on static dependency graphs.

However, static dependency graphs are highly imprecise representations of actual executions, and any analysis reliant on these graphs is likely to yield a large number of false positives. Indeed, recent effortsin this space [5, 13, 14] recognize this and propose complex conditions to reduce false positives for specific consistency mechanisms, but these works do not provide any automated methodology to check those conditions on actual programs. Further, application logic could prevent these harmful structures from manifesting in actual executions, for example as in TPC-C, which has a harmful structure in its static dependency graph under SI, but which does not appear in any dynamic dependency graph. In our work, we precisely model the application logic and the consistency specification using FOL, so that the solver would automatically derive harmful structures which are possible under the given consistency specification and search for them in actual dependency graphs taking application logic into account.

[8] proposes a static analysis for serializability under causal consistency by constructing actual dependency graphs with cycles using a FOL encoding. While this work is similar to ours in spirit, their notion of serializability is stronger than ours, since they allow transactions to be grouped together in sessions, with the serial order forced to accommodate the chosen session order. While this eases the task of verifying serializability for unbounded executions, it also results in a large number of harmless serializability violations, for which they propose

various *ad hoc* filtering approaches. Further, their focus is on programs operating on high-level data types rather than SQL programs, and their analysis is not parametric on consistency specifications.

There are also dynamic anomaly detection techniques [29, 11, 7] which either build the dependency graphs at run-time and check for cycles, or analyze the trace of events after execution. These approaches do not provide any guarantee that all anomalies will be detected, even for bounded executions. A number of approaches have been proposed recently [25, 19, 21, 15] which attempt to verify that high-level application invariants are preserved under weak consistency. These approaches are also parametric on consistency specifications, but they are not completely automated as they require correctness conditions in the form of invariants from the user, and they do not tackle serializability.

To conclude, in this paper we take the first step towards building a precise, fully automated static analysis for veifying serializability of database applications under weak consistency. We leverage the acyclic dependency graph based characterization of serializability and the framework of abstract executions to develop a FOL based analysis which is parametric on the consistency specification. We show how our approach can be used to detect bounded anomalies, and to verify serializability under specific conditions for unbounded executions. We show the practicality of our approach by successfully applying it on several realistic database benchmarks.

References

1 Tpc-c benchmark. `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf`. Online; Accessed 20 April 2018.

2 Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 67–78, 2000. `doi:10.1109/ICDE.2000.839388`.

3 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Scalable atomic visibility with RAMP transactions. *ACM Trans. Database Syst.*, 41(3):15:1–15:45, 2016. `doi:10.1145/2909870`.

4 Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 1–10, 1995. `doi:10.1145/223784.223785`.

5 Giovanni Bernardi and Alexey Gotsman. Robustness against consistency models with atomic visibility. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 7:1–7:15, 2016. `doi:10.4230/LIPIcs.CONCUR.2016.7`.

6 Philip A. Bernstein, Vassco Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

7 Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin T. Vechev. Serializability for eventual consistency: criterion, analysis, and applications. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 458–472, 2017. URL: `http://dl.acm.org/citation.cfm?id=3009895`.

8 Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin T. Vechev. Static serializability analysis for causal consistency. In *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, 2018. doi:10.1145/3192366.3192415.

9 Sebastian Burckhardt, Daan Leijen, Manuel Fähndrich, and Mooly Sagiv. Eventually consistent transactions. In *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*, pages 67–86, 2012. doi:10.1007/978-3-642-28869-2_4.

10 Sebastian Burckhardt, Daan Leijen, Jonathan Protzenko, and Manuel Fähndrich. Global sequence protocol: A robust abstraction for replicated shared state. In *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic*, pages 568–590, 2015. doi:10.4230/LIPIcs.ECOOP.2015.568.

11 Michael J. Cahill, Uwe Röhm, and Alan David Fekete. Serializable isolation for snapshot databases. *ACM Trans. Database Syst.*, 34(4):20:1–20:42, 2009. doi:10.1145/1620585.1620587.

12 Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 58–71, 2015. doi:10.4230/LIPIcs.CONCUR.2015.58.

13 Andrea Cerone and Alexey Gotsman. Analysing snapshot isolation. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 55–64, 2016. doi:10.1145/2933057.2933096.

14 Andrea Cerone, Alexey Gotsman, and Hongseok Yang. Algebraic laws for weak consistency. In *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, pages 26:1–26:18, 2017. doi:10.4230/LIPIcs.CONCUR.2017.26.

15 Natacha Crooks, Youer Pu, Lorenzo Alvisi, and Allen Clement. Seeing is believing: A client-centric specification of database isolation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 73–82, 2017. doi:10.1145/3087801.3087802.

16 Alan Fekete. Allocating isolation levels to transactions. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 206–215, 2005. doi:10.1145/1065167.1065193.

17 Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, and Patrick E. O'Neil a fnd Dennis E. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005. doi:10.1145/1071610.1071615.

18 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. doi:10.1145/564585.564601.

19 Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 'cause i'm strong enough: reasoning about consistency choices in distributed systems. 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 371–384, 2016. doi:10.1145/2837614.2837625.

20 Sudhir Jorwekar, Alan Fekete, Krithi Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1263–1274, 2007. URL: http://www.vldb.org/conf/2007/papers/industrial/p1263-jorwekar.pdf.

21 Gowtham Kaki, Kartik Nagar, Mahsa Najafzadeh, and Suresh Jagannathan. Alone together: compositional reasoning and inference for weak isolation. *PACMPL*, 2(POPL):27:1–27:34, 2018. doi:10.1145/3158115.

22 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 *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 401–416, 2011. doi:10.1145/2043556.2043593.

23 Madan Musuvathi. Systematic concurrency testing using CHESS. In *Proceedings of the 6th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), PADTAD 2008, Seattle, Washington, USA, July 20-21, 2008*, page 10, 2008. doi:10.1145/1390841.1390851.

24 Kartik Nagar and Suresh Jagannathan. Automated Detection of Serializability Violations under Weak Consistency (Extended Version). arXiv:1806.08416.

25 K. C. Sivaramakrishnan, Gowtham Kaki, and Suresh Jagannathan. Declarative programming over eventually consistent data stores. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 413–424, 2015. doi:10.1145/2737924.2737981.

26 Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li. Transactional storage for geo-replicated systems. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 385–400, 2011. doi:10.1145/2043556.2043592.

27 Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP 13, Farmington, PA, USA, November 3-6, 2013*, pages 309–324, 2013. doi:10.1145/2517349.2522731.

28 Todd Warszawski and Peter Bailis. Acidrain: Concurrency-related attacks on database-backed web applications. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 5–20, 2017. doi:10.1145/3035918.3064037.

29 Kamal Zellag and Bettina Kemme. Consistency anomalies in multi-tier architectures: automatic detection and prevention. *VLDB J.*, 23(1):147–172, 2014. doi:10.1007/s00778-013-0318-x.

30 Yang Zhang, Russell Power, Siyuan Zhou, Yair Sovran, Marcos K. Aguilera, and Jinyang Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pages 276–291, 2013. doi:10.1145/2517349.2522729.

Effective Divergence Analysis for Linear Recurrence Sequences

Shaull Almagor
Department of Computer Science, Oxford University, UK
shaull.almagor@cs.ox.ac.uk

Brynmor Chapman
MIT CSAIL
brynmor@mit.edu

Mehran Hosseini
Department of Computer Science, Oxford University, UK
mehran.hosseini@cs.ox.ac.uk

Joël Ouaknine[1]
Max Planck Institute for Software Systems, Germany &
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

James Worrell[2]
Department of Computer Science, Oxford University, UK
jbw@cs.ox.ac.uk

Abstract

We study the growth behaviour of rational linear recurrence sequences. We show that for low-order sequences, divergence is decidable in polynomial time. We also exhibit a polynomial-time algorithm which takes as input a divergent rational linear recurrence sequence and computes effective fine-grained lower bounds on the growth rate of the sequence.

2012 ACM Subject Classification Computing methodologies → Algebraic algorithms, Theory of computation → Logic and verification

Keywords and phrases Linear recurrence sequences, Divergence, Algebraic numbers, Positivity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.42

Related Version The full version of this paper can be found at `https://arxiv.org/abs/1806.07740`.

1 Introduction

Linear recurrence sequences (LRS), such as the Fibonacci numbers, permeate a wide range of scientific fields, from economics and theoretical biology to computer science and mathematics. In computer-aided verification, for example, LRS techniques play a key rôle in the termination analysis of a large class of simple while loops – see [17] for a short survey on this topic. Likewise, the ergodic behaviour of Markov chains in probability theory [1], or the stability of supply-and-demand price equilibria in laggy markets in economics (the so-called "cobweb

[1] Supported by ERC grant AVS-ISS (648701)
[2] Supported by EPSRC Fellowship EP/N008197/1

model") [3] can be analysed through an examination of the asymptotic behaviour of certain types of LRS; in particular, instability of price equilibria corresponds precisely to *divergence* of the associated LRS.

In this paper, we undertake a systematic and fine-grained analysis of the growth behaviour of rational linear recurrence sequences from the point of view of effectiveness and complexity. In order to describe our main results, we first require some preliminary definitions. A sequence of real numbers $\mathbf{u} = \langle u \rangle_{n=1}^{\infty}$ is said to satisfy a *linear recurrence of order* k if there are real numbers $a_1, \ldots, a_{k+1}$ such that

$$u_{n+k} = a_1 u_{n+k-1} + \cdots + a_{k-1} u_{n+1} + a_k u_n + a_{k+1} \tag{1}$$

for all $n \in \mathbb{N}$. Such a recurrence is said to be *homogeneous* if $a_{k+1} = 0$ and *inhomogeneous* if $a_{k+1} \neq 0$. The *characteristic polynomial* of the recurrence is

$$p(x) := x^k - a_1 x^{k-1} - \cdots - a_{k-1} x - a_k \,.$$

The zeros of p are called the *characteristic roots*. A characteristic root of maximum modulus is said to be *dominant* and its modulus is the *dominant modulus*. The *multiplicity* of a characteristic root γ is the maximal $m \in \mathbb{N}$ such that $(x - \gamma)^m$ divides $p(x)$.

An LRS is said to be *rational* if it consists of rational numbers, *integral* if it consists of integers, and *algebraic* if it consists of algebraic numbers. An LRS is *simple* if all of its characteristic roots have multiplicity 1, and is *non-degenerate* if no ratio of two distinct characteristic roots is a root of unity.[3]

We say that an LRS $\mathbf{u}$ *diverges to* ∞ if $\lim_{n\to\infty} u_n = \infty$ (technically speaking: for all $T \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that, for all $n \geq N$, we have $u_n \geq T$). We also say that $\mathbf{u}$ is *absolutely divergent* (or *diverges in absolute value*) if $\lim_{n\to\infty} |u_n| = \infty$.

The LRS $\mathbf{u}$ is said to be *positive* if $u_n \geq 0$ for all $n \geq 1$, and *ultimately positive* if there is some $N \in \mathbb{N}$ such that $u_n \geq 0$ for all $n \geq N$.

A celebrated result from the 1930s, the Skolem-Mahler-Lech theorem (see [9]), implies that all non-degenerate integral LRS are absolutely divergent. This statement is however *non-effective* in a very basic sense: given a finite representation of a non-degenerate integral LRS $\mathbf{u}$, there is no known algorithm to compute a bound N such that $u_n \neq 0$ for $n \geq N$. It is also worth pointing out that the divergence assertion fails in general for non-integral LRS.

The question of the so-called *rate of absolute divergence* for non-degenerate integral LRS was subsequently extensively studied; see [9, Sec. 2.4] for an account of some of the key results accumulated over the last several decades. To begin with, a fairly straightforward fact is the following: if $\mathbf{u}$ is an algebraic LRS of order k with dominant modulus ρ, then there is an effectively computable constant a such that, for all $n \geq 1$, $|u_n| \leq a\rho^n n^k$. In the 1970s, a conjecture was formulated to the effect that any non-degenerate integral LRS has, essentially, the maximal possible growth rate (see the next theorem for a precise statement). The conjecture was finally settled positively independently by Evertse [10] and by van der Poorten and Schlickewei [21]:

▶ **Theorem 1.** *For any non-degenerate algebraic LRS* $\mathbf{u}$ *of dominant modulus* $\rho > 1$*, and any* $\varepsilon > 0$*, there exists a constant* N *such that, for all* $n \geq N$*, we have* $|u_n| \geq \rho^{(1-\varepsilon)n}$*.*

[3] For most practical purposes – and certainly for all of the computational tasks considered in this paper – LRS can be assumed to be non-degenerate, since any degenerate LRS can be effectively decomposed into a finite number of non-degenerate LRS; moreover this reduction can be carried out in polynomial time for rational LRS of bounded order [9, 14].

This is a highly non-trivial result making use of deep number-theoretic tools concerning bounds on the sum of S-units. Unfortunately, the proof is not effective, in the sense that given $\varepsilon > 0$, it does not provide estimates for the corresponding value of N. This effectiveness issue is described as "an important open problem" in [9], where it is furthermore suggested that any progress on the matter would likely hinge upon substantial improvements of deep number-theoretic results, such as Roth's theorem, the prospects of which currently appear to be remote.

Nevertheless – and in particular for algorithmic applications in computer science – effectiveness is of central importance. The sharpest known results in this vein are due to Mignotte [12] as well as Shorey and Stewart [19], capping a long line of work in this area:

▶ **Theorem 2.** *For any homogeneous non-degenerate integral LRS* $\mathbf{u}$ *of order at most* 3 *with dominant modulus* ρ*, there are effective constants* a *and* d *such that, for all* $n \geq 1$*, we have* $|u_n| \geq \frac{a\rho^n}{n^d}$.

For most problems in computer science and automated verification, such as the analysis of the long-run behaviour of dynamical systems or the termination of linear while loops, the primary notion of *divergence* is clearly much more relevant than that of 'divergence in absolute value'. In view of the above results, however, one might expect that little could be said about effective rates of divergence. Somewhat surprisingly, divergence does turn out to be significantly more tractable than absolute divergence. At a high level, the main results of this paper can now be summarised as follows:

> Given a rational LRS $\mathbf{u}$, homogeneous or inhomogeneous, either of order at most 5, or, if the LRS is simple, of order at most 8, we can carry out the following tasks in polynomial time:
> - decide if $\mathbf{u}$ diverges to ∞ or not; and
> - in divergent instances, provide effective fine-grained lower bounds on the rate of divergence of $\mathbf{u}$.

The precise statements can be found in Theorems 12 and 13. The most obvious contrast in comparison with Theorem 2 is the higher order of LRS that can be handled effectively (5 and 8 versus 3). It is also worth noting, however, that our results apply more generally to rational (as opposed to integral) LRS, and that we can handle inhomogeneous sequences at no cost – this is remarkable in that the folklore wisdom usually broadly equates inhomogeneous LRS of order k with homogeneous LRS of order $k + 1$ (this assertion, as well as the manner in which we circumvent it, are made precise in the main body of the paper).

Finally, let us point out that our analysis of divergence rates relies, among others, on improvements to results concerning the positivity and ultimate positivity of LRS, which were originally developed in [15, 14, 16]. As a by-product, therefore, stronger results on the Positivity and Ultimate Positivity Problems – notably dealing with inhomogeneous LRS – can be found in the present paper, in particular in the form of Theorems 19 and 20.

2 Preliminaries

2.1 Linear Recurrence Sequences

Let us start by reformulating the notion of linear recurrence more abstractly as follows. Define the *shift operator* $E : \mathbb{R}^{\mathbb{N}} \to \mathbb{R}^{\mathbb{N}}$ by $E(f)(n) = f(n + 1)$ for a sequence $f \in \mathbb{R}^{\mathbb{N}}$. The polynomial ring $\mathbb{R}[E]$ acts on the set of sequences $\mathbb{R}^{\mathbb{N}}$ on the left in a natural way, turning

$\mathbb{R}^{\mathbb{N}}$ into a left $\mathbb{R}[E]$ module. Then a sequence $\mathbf{u} = \langle u \rangle_{n=1}^{\infty}$ satisfies the recurrence equation (1) if and only if $p(E) \cdot \mathbf{u} = a_{k+1} \cdot \mathbf{1}$, where p is the characteristic polynomial of the recurrence and $\mathbf{1}$ is the all-ones sequence.

The following homogenization construction is well known.

▶ **Proposition 3.** *Let $\mathbf{u} = \langle u_n \rangle_{n=1}^{\infty}$ satisfy an inhomogeneous linear recurrence of order k. Then $\mathbf{u}$ satisfies a homogeneous recurrence of order $k+1$.*

Proof. By assumption we have that $p(E) \cdot \mathbf{u} = \mathbf{c}$ for some monic polynomial $p(x)$ of degree k and constant sequence $\mathbf{c}$. Writing $q(x) = (x-1)p(x)$, we have $q(E) \cdot \mathbf{u} = (E-1) \cdot \boldsymbol{c} = 0$. ◀

We have the following partial converse to Proposition 3.

▶ **Proposition 4.** *Let $\mathbf{u} = \langle u_n \rangle_{n=1}^{\infty}$ satisfy a homogeneous linear recurrence of order $k+1$ with a positive real characteristic root ρ. Then the sequence $\mathbf{v} = \langle v_n \rangle_{n=1}^{\infty}$ defined by $v_n = \frac{u_n}{\rho^n}$ satisfies an inhomogeneous linear recurrence of order k.*

Proof. By assumption, $\mathbf{u}$ satisfies the recurrence $f(E) \cdot \mathbf{u} = 0$ for some monic polynomial $f(x) \in \mathbb{R}[x]$ of degree $k+1$ that has a positive real root ρ. Define a sequence $\mathbf{v} = \langle v_n \rangle_{n=1}^{\infty}$ by $v_n := \frac{u_n}{\rho^n}$ for all $n \in \mathbb{N}$. Then $\mathbf{v}$ satisfies the recurrence $g(E) \cdot \mathbf{v} = 0$ where g is the monic polynomial $g(x) = \rho^{-(k+1)} f(\rho x)$.

But $g(1) = 0$ and hence we have the factorization $g(x) = (x-1)h(x)$ for some monic polynomial $h(x) \in \mathbb{R}[x]$. It follows that $(E-1)h(E) \cdot \mathbf{v} = 0$ and hence $h(E) \cdot \mathbf{v}$ is constant, i.e., $\mathbf{v}$ satisfies an inhomogeneous recurrence of order k. ◀

Let $\|\mathbf{u}\|$ denote the binary representation length[4] of $\mathbf{u}$. We remark that the transformations back and forth between homogeneous and inhomogeneous LRS can be carried out in polynomial time in $\|u\|$ if the given LRS have real algebraic coefficients. For an inhomogeneous LRS $\mathbf{u}$ of order k, we refer to the corresponding homogeneous LRS obtained as per Proposition 3 as the *homogenization* of $\mathbf{u}$, denoted $\textsc{hom}(\mathbf{u})$. The proof of Proposition 3 gives us the following useful property.

▶ **Property 5.** *The characteristic roots of $\textsc{hom}(\mathbf{v})$ are the same as those of $\mathbf{v}$, with the same multiplicities, except for the characteristic root 1, which always occurs in $\textsc{hom}(\mathbf{v})$, and whose multiplicity is $m+1$, where m is the multiplicity of 1 in $\mathbf{v}$.*

Consider an LRS $\mathbf{u}$ with integer coefficients. Then, since the characteristic polynomial p of an LRS $\mathbf{u}$ has integer coefficients, the characteristic roots of $\mathbf{u}$ comprise real-algebraic roots $\{\rho_1, \ldots, \rho_d\}$, and conjugate pairs of complex-algebraic roots $\{\gamma_1, \overline{\gamma_1}, \ldots, \gamma_m, \overline{\gamma_m}\}$. There are now univariate polynomials $A_1, \ldots, A_d \in$ with real-algebraic coefficients and $C_1, \ldots, C_m$ with complex-algebraic coefficients such that, for every $n \geq 0$,

$$u_n = \sum_{i=1}^{d} A_i(n)\rho_i^n + \sum_{j=1}^{m} (C_j(n)\gamma^n + \overline{C_j(n)}\overline{\gamma}^n)$$

This expression is referred to as the *exponential polynomial* solution of $\mathbf{u}$. The degree of each of the polynomials is strictly smaller than the multiplicity of the corresponding root. For a fixed order k, the coefficients appearing in the polynomials can be computed in time polynomial in $\|\mathbf{u}\|$.

[4] In general, we denote by $\|\cdot\|$ the binary-representation length of objects.

We now turn to present two results regarding the asymptotic analysis of LRS.

The following result due to Braverman [5] enables us to reason about the complex part of the exponential polynomial above.

▶ **Lemma 6** (Complex Units Lemma). *Let $\zeta_1, \zeta_2, \ldots, \zeta_m \in \mathbf{S}_1 \setminus \{1\}$ be distinct complex numbers (where $\mathbf{S}_1 = \{z \in \mathbb{C} : |z| = 1\}$), and let $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{C} \setminus \{0\}$. Set $z_n := \sum_{k=1}^{m} \alpha_k \zeta_k^n$. Then there exists $c < 0$ such that for infinitely many n, $Re(z_n) < c$.*

In particular, Lemma 6 immediately implies that an LRS without a real dominant characteristic root, is neither positive, ultimately positive, nor divergent.

Finally, the following proposition from [14] allows us to bound the growth rate of the low-order terms in the exponential polynomial of an LRS.

▶ **Proposition 7.** *Consider an LRS $\mathbf{u} = \langle u_n \rangle_{n=1}^{\infty}$ of bounded order, with dominant modulus ρ, and write*

$$\frac{u_n}{\rho^n} = A(n) + \sum_{i=1}^{m} \left(C_i(n)\lambda_i^n + \overline{C_i(n)}\,\overline{\lambda_i}^n \right) + r(n)$$

where A is a real polynomial, C_i are non-zero complex polynomials, $\rho\lambda_i$ and $\rho\overline{\lambda_i}$ are conjugate pairs of non-real dominant roots of $\mathbf{u}$, and r is an exponentially decaying function.

We can compute in polynomial time $\epsilon \in (0,1)$ and $N \in \mathbb{N}$ such that

$$\frac{1}{\epsilon} = 2^{\|\mathbf{u}\|^{O(1)}},$$
$$N = 2^{\|\mathbf{u}\|^{O(1)}},$$
for all $n > N, |r(n)| < (1-\epsilon)^n$.

2.2 Mathematical Tools

In this section we introduce several tools that will be used throughout the paper.

Algebraic Numbers. A complex number α is *algebraic* if it is a root of a polynomial $p \in \mathbb{Z}[x]$. The *defining polynomial* of α is the unique monic polynomial of minimal degree that has α as a root, and is denoted p_α. The *degree* and the *height* of α are the degree and the height (i.e., maximum absolute value of the coefficients) of p_α, respectively. An algebraic number α can be represented by a polynomial that has α as a root, along with an approximation of α by a complex number with rational real and imaginary parts. We denote by $\|\alpha\|$ the representation length of α. Basic arithmetic operations as well as equality testing and comparisons for algebraic numbers can be carried out in polynomial time (see [4, 7] for efficient algorithms).

The following lemma from [15] is a consequence of the celebrated lower bound for linear forms in logarithms due to Baker and Wüstholz [2].

▶ **Lemma 8.** *There exists $D \in \mathbb{N}$ such that, for all algebraic numbers $\lambda, \zeta \in \mathbb{C}$ of modulus 1, and for all $n \geq 2$, if $\lambda^n \neq \zeta$, then $|\lambda^n - \zeta| > \frac{1}{n^{(\|\lambda\|+\|\zeta\|)^D}}$.*

Multiplicative Relations. Multiplicative relations between characteristic roots of an LRS play a key role in our analysis. The following result, due to Masser [11] enables us to efficiently elicit these relationships.

▶ **Theorem 9.** *Let m be fixed, and let $\lambda_1, \ldots, \lambda_m$ be complex algebraic numbers of modulus 1. Let $L = \{(v_1, \ldots, v_m) \in \mathbb{Z}^m : \lambda_1^{v_1} \cdots \lambda_m^{v_m} = 1\}$ be the group of multiplicative relations between the λ_i. L has a basis $\{\boldsymbol{\ell_1}, \ldots, \boldsymbol{\ell_p}\} \subseteq \mathbb{Z}^m$ (with $p \leq m$), where the entries of each of the $\boldsymbol{\ell_j}$ are all polynomially bounded in $\|\lambda_1\| + \ldots + \|\lambda_m\|$. Moreover, such a basis can be computed in time polynomial in $\|\lambda_1\| + \ldots + \|\lambda_m\|$.*

The First-Order Theory of the Reals. A sentence in the first-order theory of the reals is of the form $Q_1 x_1 \cdots Q_m x_m \varphi(x_1, \ldots, x_m)$ where each Q_i is a quantifier ($\exists$ or $\forall$), each x_i is a real valued variable, and φ is a boolean combination of atomic predicates of the form $p(x_1, \ldots, x_m) \sim 0$ for some $p \in \mathbb{Z}[x_1, \ldots, x_m]$ and $\sim \in \{>, =\}$. The first-order theory of the reals admits quantifier elimination, a famous result due to Tarski [20], whose procedure unfortunately has non-elementary complexity. In this paper we consider only the case where the number of variables is uniformly bounded. Then, we can invoke the following result due to Renegar [18].

▶ **Theorem 10** (Renegar). *Let $M \in \mathbb{N}$ be fixed. Let $\tau(\mathbf{y})$ be a formula of the first order theory of the reals. Assume that the number of (free and bound) variables in $\tau(\mathbf{y})$ is bounded by M. Denote the degree of $\tau(\mathbf{y})$ by d and the number of atomic predicates in $\tau(\mathbf{y})$ by n.*

There is a polynomial time (polynomial in $\|\tau(\mathbf{y})\|$) procedure which computes an equivalent quantifier-free formula

$$\chi(\mathbf{y}) = \bigvee_{i=1}^{I} \bigwedge_{j=1}^{J_i} h_{i,j}(y) \sim_{i,j} 0$$

where each $\sim_{i,j}$ is either $>$ or $=$, with the following properties:

1. *Each of I and J_i (for $1 \leq i \leq I$) is bounded by $(n+d)^{O(1)}$.*
2. *The degree of $\chi(\mathbf{y})$ is bounded by $(n+d)^{O(1)}$.*
3. *The height of $\chi(\mathbf{y})$ is bounded by $2^{\|\tau(\mathbf{y})\|(n+d)^{O(1)}}$.*

Asymptotic Analysis. We conclude this section with the following simple lemma from [15].

▶ **Proposition 11.** *Let $a \geq 2$ and $\epsilon \in (0,1)$ be real numbers. Let $B \in \mathbb{Z}[x]$ have degree at most a^{D_1} and height at most $2^{a^{D_2}}$, and assume that $1/\epsilon \leq 2^{a^{D_3}}$ for some $D_1, D_2, D_3 \in \mathbb{N}$. Then there is $D_4 \in \mathbb{N}$ depending only on D_1, D_2, D_3 such that for all $n \geq 2^{a^{D_4}}$, $\frac{1}{B(n)} > (1-\epsilon)^n$.*

3 Divergence

Recall from Theorem 1 that an LRS $\mathbf{u}$ with dominant modulus ρ necessarily diverges in absolute value if $\rho > 1$. More precisely, if $\rho > 1$ then given $\varepsilon > 0$ there exists a threshold N such that $|u_n| > \rho^{(1-\varepsilon)n}$ for all $n > N$. However this result is *ineffective* – it is not known how to compute N given $\mathbf{u}$ and ε.

In this section we derive *effective* divergence bounds for sequences that diverge to ∞ (i.e., sequences that both diverge in absolute value and that are ultimately positive). The bounds on divergence have the following form: for a divergent sequence $\mathbf{u}$ with dominant modulus $\rho = 1$ we aim to show that for every $n > N$, $u_n > an^d$ for effective constants $a > 0, d \in \mathbb{N}$, and $N \in \mathbb{N}$. In case of a dominant modulus $\rho > 1$ we aim to show that for every $n > N$,

$u_n > \frac{a\rho^n}{n^d}$ for effective constants $a > 0, d \in \mathbb{N}$, and $N \in \mathbb{N}$. Henceforth we refer to bounds of these respective forms as *divergence bounds*.[5]

In Section 3.1, we show how to compute effective divergence bounds of LRS up to certain orders. Then, in Section 3.2, we provide hardness results for the decidability of divergence.

3.1 Effective Divergence is Solvable

In this section we prove the following theorems:

▶ **Theorem 12.** *There is a polynomial-time procedure that given a rational LRS of order at most 5 decides whether it diverges and, in case of divergence, outputs divergence bounds.*

▶ **Theorem 13.** *There is a polynomial-time procedure that, given a simple rational LRS of order at most 8, decides whether it diverges and, in case of divergence, outputs divergence bounds.*

The proofs of Theorems 12 and 13 build on techniques developed in [15, 14, 16], using a fine-grained analysis in the results thereof, along with some new ideas. To avoid unnecessary repetition, we sketch the main ideas of the proofs simultaneously.

Consider an LRS $\mathbf{u}$ of order k. For uniformity, if $\mathbf{u}$ is inhomogeneous, we homogenize it as per Proposition 3. Thus, either $k \leq 6$ or $\mathbf{u}$ is simple and $k \leq 9$, where if $k = 6$ or if $\mathbf{u}$ is simple and $k = 9$, then $\mathbf{u}$ has a special structure according to Property 5.

As mentioned in Section 1, we can assume without loss of generality that $\mathbf{u}$ is non-degenerate. Let ρ be the dominant modulus of $\mathbf{u}$, we also note that if $\rho < 1$, then $|u_n| \to 0$ as $n \to \infty$, and in particular the sequence does not diverge. Thus, we may assume $\rho \geq 1$. In addition, by Lemma 6, if $\mathbf{u}$ does not have a real positive dominant root, then $u_n \not\to \infty$. Thus, we may assume a real positive dominant characteristic root ρ. Note that all other dominant roots must be complex, and come in conjugate pairs, since if $-\rho$ were a root, then $\mathbf{u}$ would be degenerate.

Writing u_n as an exponential polynomial and dividing by ρ^n, we have

$$\frac{u_n}{\rho^n} = A(n) + \sum_{i=1}^{m} \left(C_i(n)\lambda_i^n + \overline{C_i(n)}\overline{\lambda_i}^n \right) + r(n) \tag{2}$$

where A is a real polynomial, C_i are non-zero complex polynomials, $\rho\lambda_i$ and $\rho\overline{\lambda_i}$ are conjugate pairs of non-real dominant characteristic roots of $\mathbf{u}$ (so $|\lambda_i| = 1$), and $r(n)$ is an exponentially decaying function (possibly identically zero). More precisely, the degree of each of $A(n), C_1(n), \ldots, C_m(n)$ is strictly smaller than the multiplicity of the corresponding characteristic root. We can assume that either $A(n) \not\equiv 0$ or $m \neq 0$. Indeed, otherwise we can consider the LRS $\langle \rho^n r(n) \rangle_{n=1}^{\infty}$, which is of lower order than $\mathbf{u}$.

In the following, if $A(n)$ (resp. $C_i(n)$ for some $1 \leq i \leq m$) is a constant, we denote it by A (resp. C_i).

We proceed to decide divergence by a case analysis of Equation (2).

[5] Note that not only do we seek effective divergence bounds, but also that these bounds are asymptotically tighter than the bounds from Theorem 1 since for any fixed $d > 0$, it is clear that $a\rho^n/n^d$ eventually dominates $\rho^{(1-\varepsilon)n}$ for any $\varepsilon > 0$.

Case 1: $\rho = 1$ and $A(n) = A$ is a constant

Note that in this case, $\frac{u_n}{\rho^n} = u_n$. Since A is a constant, then it does not affect the divergence of $\mathbf{u}$. We claim that $u_n \not\to \infty$. Indeed, by Lemma 6, the expression $\sum_{i=1}^{m} \left(C_i(n)\lambda_i^n + \overline{C_i(n)}\overline{\lambda_i}^n\right)$ becomes negative infinitely often (regardless of whether $C_i(n)$ are constants or polynomials), whereas the effect of $r(n)$ is exponentially decreasing. Thus, $\mathbf{u}$ does not diverge.

Case 2: $\rho = 1$, $A(n)$ is not a constant, and every C_i is a constant

In this case we can rewrite Equation (2) as

$$u_n = A(n) + \sum_{i=1}^{m} \left(C_i\lambda_i^n + \overline{C_i}\overline{\lambda_i}^n\right) + r(n) \tag{3}$$

Since $|\lambda_i| = 1$ for all i, and since $r(n)$ is exponentially decreasing, then clearly $u_n \to \infty$ iff the leading coefficient of $A(n)$ is positive.

Recall that since $\rho = 1$, then if $\mathbf{u}$ diverges, there exist $N, d \in \mathbb{N}$ and $a > 0$ such that $u_n \geq an^d$ for all $n > N$. We now show how to effectively compute N, d, and a.

From Proposition 7, we can compute in polynomial time $\epsilon \in (0,1)$ and $N_1 \in \mathbb{N}$ such that $r(n) < (1-\epsilon)^n < 1$ for all $n > N_1$. We thus have that $u_n \geq A(n) - 2\sum_{i=1}^{m} |C_i| - 1$, and we can easily compute $N_2 \in \mathbb{N}$ and $a \in \mathbb{Q}$ (depending on the coefficients of $A(n)$) such that for all $n > N_2$ we have $A(n) - 2\sum_{i=1}^{m} |C_i| - 1 \geq an^d$, where d is the degree of $A(n)$. Taking $N = \max\{N_1, N_2\}$, we conclude this case.

Case 3: $\rho = 1$, $A(n)$ is not a constant, and there exists a non-constant $C_i(n)$

We notice that if there exists a non-constant $C_i(n)$, it follows by Property 5 that $\mathbf{u}$ is not obtained by homogenizing a simple LRS. That is, we are in the case where $k \leq 6$. In the notations of Equation (2), we then have that $m = 1$, $A(n)$ is linear, $C_1(n)$ is linear, and $r(n) \equiv 0$. Indeed, this corresponds to the case where the characteristic roots of u_n are $1, \lambda, \overline{\lambda}$, each with multiplicity 2. Let $A(n) = a_1 n + b_1$ and $C_1(n) = a_2 n + b_2$, then we can write

$$u_n = a_1 n + b_1 + (a_2 n + b_2)\lambda^n + (\overline{a_2} n + \overline{b_2})\overline{\lambda}^n = n(a_1 + a_2\lambda^n + \overline{a_2}\overline{\lambda^n}) + (b_1 + b_2\lambda^n + \overline{b_2\lambda^n})$$

Since $|(b_1 + b_2\lambda^n + \overline{b_2\lambda^n})|$ is bounded, then u_n diverges iff $n(a_1 + a_2\lambda^n + \overline{a_2}\overline{\lambda^n})$ diverges. Let $\theta = \arg\lambda$ and $\varphi = \arg a_2$. We have $n(a_1 + a_2\lambda^n + \overline{a_2}\overline{\lambda^n}) = n(a_1 + 2|a_2|\cos(n\theta + \varphi))$.

Observe that since $\mathbf{u}$ is non-degenerate, then θ is not a rational multiple of π. It follows that $\{[n\theta + \varphi]_{2\pi} : n \in \mathbb{N}\}$ (where $[x]_{2\pi} = x - 2\pi j$ where j is the unique integer such that $0 \leq x - 2\pi j < 2\pi$) is dense in $[0, 2\pi)$, so $\{\cos(n\theta + \varphi) : n \in \mathbb{N}\}$ is dense in $[-1, 1]$. Again, we split into cases.

- If $a_1 > 2|a_2|$, we have that u_n diverges. Then, we can compute in polynomial time a rational $\epsilon > 0$ and $N \in \mathbb{N}$ such that $a_1 - 2|a_2| > \epsilon$ and $n(a_1 + 2|a_2|) - (b_1 - 2|b_2|) > \epsilon n$ for all $n > N$. We then have that $u_n > \epsilon n$ for all $n > N$, thus concluding effective decidability of divergence in this case.
- If $a_1 < 2|a_2|$, then u_n does not diverge, as it becomes negative infinitely often, by the density argument above.
- The remaining case is when $a_1 = 2|a_2|$, and the expression above becomes $na_1(1+\cos(n\theta+\varphi))$. We show that in this case, u_n does not diverge.

By Taylor approximation, for every $x \in (-\pi, \pi]$ it holds that $1 - \cos(x) \leq \frac{x^2}{2}$. For $n \in \mathbb{N}$, write $\Lambda(n) = n\theta + \varphi - (2j+1)\pi$, where $j \in \mathbb{Z}$ is the unique integer such that $-\pi < \Lambda(n) \leq \pi$. We now have that

$$na_1(1 + \cos(n\theta + \varphi)) = na_1(1 - \cos(n\theta + \varphi + \pi)) = na_1(1 - \cos(\Lambda(n))) < na_1 \frac{\Lambda(n)^2}{2}.$$

By Dirichlet's Approximation Theorem, we have that $|\Lambda(n)| < \frac{t}{n}$ for infinitely many values of n, where t is a constant depending on φ. Thus, we have $na_1\frac{\Lambda(n)^2}{2} < \frac{a_1 t^2}{2n}$ for infinitely many values of n. It follows that u_n is infinitely often bounded by a constant, so it does not diverge.

Case 4: $\rho > 1$ and there exists a non-constant $C_i(n)$

As in Case 3, it holds that $k \leq 6$. Moreover, since $\rho > 1$, then whether or not $\mathbf{u}$ was obtained by homogenization, the characteristic root 1 (if it exists) is captured in $r(n)$. Therefore, we have that $m = 1$, C_1 is linear, and $A(n) = A$ is constant. Let C_1 have leading coefficient $b \neq 0$. By Lemma 6, there exists $\epsilon > 0$ such that $b\lambda^n + \overline{b\lambda}^n < -\epsilon$ infinitely often. Then $C_1(n)\lambda_1^n + \overline{C_1(n)\lambda_1}^n$ (and hence u_n) is unbounded below, so u_n does not diverge.

Case 5: $\rho > 1$, $A(n)$ is not a constant, and every C_i is a constant

Since $A(n)$ is not a constant and $\rho > 1$, this case may only arise for $k \leq 6$ and $m \leq 1$. We write

$$\frac{u_n}{\rho^n} = A(n) + C_1\lambda_1^n + \overline{C_1\lambda_1}^n + r(n).$$

where if $m = 0$ then take $C_1 = 0$.

If $A(n)$ has a negative leading coefficient, then u_n is unbounded from below, and in particular u_n does not diverge.

If $A(n)$ has a positive leading coefficient, we can compute in polynomial time $N_0 \in \mathbb{N}$ and a rational $\epsilon_0 > 0$ such that $A(n) - 2|C_1| > 2\epsilon_0$ for all $n > N_0$. By Proposition 7, we can also compute in polynomial time $N_1 \in \mathbb{N}$ and $\epsilon_1 \in (0, 1)$ such that $|r(n)| < (1 - \epsilon_1)^n$ for all $n > N_1$. Taking $N_2 \geq \log_{1-\epsilon_1} \epsilon_0$, we have that for all $n > \max\{N_0, N_1, N_2\}$, $|r(n)| < \epsilon_0$, and thus

$$\frac{u_n}{\rho^n} \geq A(n) - 2|C_1| + r(n) \geq A(n) - 2|C_1| - \epsilon_0 > 2\epsilon_0 - \epsilon_0 = \epsilon_0.$$

Thus we have $u_n \geq \epsilon_0\rho^n$ for all $n > \max\{N_0, N_1, N_2\}$, which immediately yields effective divergence bounds in this case.

Case 6: $\rho > 1$, $A(n) = A$ is a constant, and every C_i is a constant

This case is the most involved, and utilizes deep mathematical results. Our proof works along the lines of [16]. For completeness, the full proof can be found in the full version.

We rewrite Equation (2) as

$$\frac{u_n}{\rho^n} = A + \sum_{i=1}^{m} \left(C_i\lambda_i^n + \overline{C_i\lambda_i}^n \right) + r(n) \tag{4}$$

Observe that $m \leq 3$. Indeed, if $k \leq 8$ this is trivial, and if $k = 9$ then by Property 5, 1 must be a non-dominant characteristic root of $\mathbf{u}$, so $r(n) \not\equiv 0$ and thus $m \leq 3$.

In the following, we handle the case $m = 3$. The cases where $m < 3$ are similar and slightly simpler.

Let $L = \left\{(v_1, \ldots, v_3) \in \mathbb{Z}^3 : \lambda^{v_1} \cdots \lambda^{v_3} = 1\right\}$, and let $\{\boldsymbol{\ell_1}, \ldots, \boldsymbol{\ell_p}\}$ be a basis for L of cardinality p. Write $\boldsymbol{\ell_q} = (\ell_{q,1}, \ldots, \ell_{q,3})$ for $1 \leq q \leq p$. From Theorem 9, such a basis can be computed in polynomial time, and moreover – each $\ell_{q,j}$ may be assumed to have magnitude polynomial in $\|u\|$.

Consider the set $\mathbb{T} = \{(z_1, z_2, z_3) \in \mathbb{C}^3 : |z_1| = |z_2| = |z_3| = 1$ and for each $1 \leq q \leq p$, $z_1^{\ell_{q,1}} z_2^{\ell_{q,2}} z_3^{\ell_{q,3}} = 1\}$.

Define $h : \mathbb{T} \to \mathbb{R}$ by setting $h(z_1, z_2, z_3) = \sum_{i=1}^{3}(C_i z_i + \overline{C_i}\overline{z_i})$, so that for every $n \in \mathbb{N}$, $\frac{u_n}{\rho^n} = A + h(\lambda_1^n, \lambda_2^n, \lambda_3^n) + r(n)$. By Kronecker's theorem on inhomogeneous Diophantine approximation [6], the set $\{\lambda_1^n, \lambda_2^n, \lambda_3^n : n \in \mathbb{N}\}$ is a dense subset of $\mathbb{T}$. Since h is continuous, it follows that $\inf\{h(\lambda_1^n, \lambda_2^n, \lambda_3^n) : n \in \mathbb{N}\} = \min h|_{\mathbb{T}} = \mu$ for some $\mu \in \mathbb{R}$.

In the full proof, we show that μ is algebraic, computable in polynomial time, with $\|\mu\| = \|\mathbf{u}\|^{O(1)}$.

We now split to cases according to the sign of $A + \mu$.

- If $A + \mu < 0$, then $\mathbf{u}$ is infinitely often negative, and does not diverge.
- If $A + \mu > 0$, then $\mathbf{u}$ diverges, and we obtain an effective bound similarly to Case 5.
- It remains to analyze the case where $A+\mu = 0$. To this end, let $\lambda_j = e^{i\theta_j}$ and $C_j = |C_j|e^{i\varphi_j}$ for $1 \leq j \leq 3$. From Equation (4) we have

$$\frac{u_n}{\rho^n} = A + \sum_{j=1}^{3} 2|C_j| \cos(n\theta_j + \varphi_j) + r(n).$$

We further assume that all the C_j are non-zero. Indeed, if this does not hold, we can recast our analysis in lower dimension.

In the full proof, we use zero-dimensionality results to show that h achieves its minimum μ over $\mathbb{T}$ only at a finite set of points $Z = \{(\zeta_1, \zeta_2, \zeta_3) \in \mathbb{T} : h(\zeta_1, \zeta_2, \zeta_3) = \mu\}$.

We concentrate on the set Z_1 of first coordinates of Z. Write

$$\tau_1(x) = \exists z_1(\mathrm{Re}(z_1) = x \wedge z_1 \in Z_1)$$
$$\tau_2(y) = \exists z_1(\mathrm{Im}(z_1) = y \wedge z_1 \in Z_1)$$

By rewriting these formulas in the first order theory of the reals, we are able to show, using Theorem 10, that any $\zeta_1 = \hat{x} + i\hat{y} \in Z_1$ is algebraic, and moreover satisfies $\|\zeta_1\| = \|\mathbf{u}\|^{O(1)}$. In addition, we show that the cardinality of Z_1 is at most polynomial in $\|\mathbf{u}\|$.

Since λ_1 is not a root of unity, for each $\zeta_1 \in Z_1$ there is at most one value of n such that $\lambda_1^n = \zeta_1$. Theorem 9 then entails that this value (if it exists) is at most $M = \|\mathbf{u}\|^{O(1)}$, which we can take to be uniform across all $\zeta_1 \in Z_1$. We can now invoke Corollary 8 to conclude that, for $n > M$, and for all $\zeta_1 \in Z_1$, we have

$$|\lambda_1^n - \zeta_1| > \frac{1}{n^{\|\mathbf{u}\|^D}} \tag{5}$$

where $D \in \mathbb{N}$ is some absolute constant.

Let $b > 0$ be minimal such that the set

$$\left\{z_1 \in \mathbb{C} : |z_1| = 1 \text{ and, for all } \zeta_1 \in Z_1, |z_1 - \zeta_1| \geq \frac{1}{b}\right\}$$

is non empty. Thanks to our bounds on the cardinality of Z_1, we can use the first-order theory of the reals, together with Theorem 10, to conclude that b is algebraic and $\|b\| = \|\mathbf{u}\|^{O(1)}$.

Define the function $g : [b, \infty) \to \mathbb{R}$ as follows:

$$g(x) = \min \left\{ h(z_1, z_2, z_3) - \mu : (z_1, z_2, z_3) \in \mathbb{T} \text{ and for all } \zeta_1 \in Z_1, |z_1 - \zeta_1| \geq \frac{1}{x} \right\}.$$

In the full proof we show that we can compute in polynomial time a polynomial $P \in \mathbb{Z}[x]$ such that, for all $x \in [b, \infty)$,

$$g(x) \geq \frac{1}{P(x)} \tag{6}$$

with $\|P\| = \|\mathbf{u}\|^{O(1)}$

By Proposition 7 we can find $\epsilon \in (0, 1)$ and $N = 2^{\|\mathbf{u}\|^{O(1)}}$ such that for all $n > N$, we have $|r(n)| < (1 - \epsilon)^n$, and moreover $1/\epsilon = 2^{\|\mathbf{u}\|^{O(1)}}$. In addition, by Proposition 11, there is $N' = 2^{\|\mathbf{u}\|^{O(1)}}$ such that for every $n \geq N'$

$$\frac{1}{2P(n^{\|\mathbf{u}\|^D})} > (1 - \epsilon)^n. \tag{7}$$

Combining Equations (4)–(7), we get

$$\begin{aligned} \frac{u_n}{\rho^n} &= A + h(\lambda_1^n, \lambda_2^n, \lambda_3^n) + r(n) \geq -\mu + h(\lambda_1^n, \lambda_2^n, \lambda_3^n) - (1 - \epsilon)^n \geq g(n^{\|\mathbf{u}\|^D}) - (1 - \epsilon)^n \\ &\geq \frac{1}{P(n^{\|\mathbf{u}\|^D})} - (1 - \epsilon)^n = \frac{1}{2P(n^{\|\mathbf{u}\|^D})} + \frac{1}{2P(n^{\|\mathbf{u}\|^D})} - (1 - \epsilon)^n \geq \frac{1}{2P(n^{\|\mathbf{u}\|^D})} \end{aligned}$$

provided $n > \max\{M, N, N'\}$. We thus have that $\frac{u_n}{\rho^n}$ is eventually lower bounded by an inverse polynomial and hence we have effective divergence bounds in this case.

Finally, Cases 1–6 allow us to conclude both Theorem 12 and Theorem 13.

3.2 Hardness of Divergence

We now turn to show lower bounds for the divergence problem. Surprisingly, our lower bounds hold already for homogeneous LRS, and for the divergence decision problem, even without requiring effectively computable bounds.

In [14], it is shown that the Ultimate Positivity problem for homogeneous LRS of order at least 6 is hard, in the sense that if Ultimate Positivity is decidable for such LRS, then certain hard open problems in Diophantine approximation become solvable. We show hardness of divergence for homogeneous LRS of order at least 6 by reducing from Ultimate Positivity.

▶ **Theorem 14.** *Ultimate Positivity is reducible to Divergence.*

Proof. We show a reduction from the Ultimate Positivity problem for non-degenerate LRS of order 6, shown to be hard in [14]. The key ingredient in the reduction is Theorem 1.

Consider a non-degenerate homogeneous LRS $\langle u_n \rangle$ of order 6 with dominant modulus ρ, and let $\mu = \max\left\{2, \frac{2}{\rho}\right\}$, then the sequence $v_n = \mu^n u_n$ is a non-degenerate homogeneous LRS of order 6 with dominant modulus $\mu\rho \geq 2$. By Theorem 1, taking $\epsilon = \frac{1}{2}$, it follows that there exists $N \in \mathbb{N}$ such $|v_n| \geq 2^{n/2}$ for every $n > N$. It immediately follows that v_n is ultimately positive iff $v_n \to \infty$. Clearly, however, v_n and u_n have the same sign, and therefore u_n is ultimately positive iff v_n diverges, and we are done. ◀

4 Positivity and Ultimate Positivity

In this section we study the Positivity and Ultimate Positivity problems for inhomogeneous LRS. These problems were studied in [14, 15, 16] for homogeneous LRS. Using Proposition 3 and some careful analysis, we extend the decidability results to the inhomogeneous case.

We start by citing some results from [14, 15, 16], split to upper and lower bounds.

▶ **Theorem 15** (Upper Bounds from [14, 15, 16])**.**

1. *Positivity and Ultimate Positivity are decidable for homogeneous LRS of order 5 or less with complexities in* $\mathbf{coNP}^{\mathbf{PosSLP}}$ *and* **PTIME***, respectively.*
2. *Positivity is decidable for simple homogeneous LRS or order 9 or less with complexity in* $\mathbf{coNP}^{\mathbf{PosSLP}}$*.*
3. *Ultimate Positivity is decidable for simple homogeneous LRS of any order with complexity in* **PTIME***.*
4. *Effective Ultimate Positivity is solvable for simple homogeneous LRS of order 9 or less with complexity in* **PTIME***.*

The following notion of hardness will be made precise in Section 4.2.

▶ **Theorem 16** (Lower Bounds from [14, 15, 16])**.** *Positivity and Ultimate Positivity for LRS of order at least 6 are hard with respect to certain hard open problems in Diophantine approximation.*

4.1 Upper Bounds

We proceed to prove analogous results to Theorem 15 for inhomogeneous LRS.

Theorem 15(1.) along with Proposition 3 readily give us the following:

▶ **Theorem 17.** *Positivity and Ultimate Positivity are decidable for inhomogeneous LRS of order* 4 *or less, with complexity in* $\mathbf{coNP}^{\mathbf{PosSLP}}$*.*

For simple LRS, things become more involved, as Proposition 3 does not preserve simplicity. However, Property 5 shows that simplicity is almost preserved, up to the multiplicity of the characteristic root 1. As we now show, this is sufficient to obtain upper bounds for inhomogeneous simple LRS.

We start by addressing effective Ultimate Positivity, which we then use for addressing Positivity.

▶ **Theorem 18.** *Effective Ultimate Positivity is solvable in polynomial time for simple inhomogeneous LRS of order* 8 *or less.*

Proof. Let $\mathbf{v}$ be a simple, non-degenerate, inhomogeneous LRS or order 8 or less, and consider the homogeneous LRS $\mathbf{u} = \textsc{hom}(\mathbf{v})$. By Proposition 3, $\mathbf{u}$ is of order at most 9. If $\mathbf{u}$ is a simple LRS, then by [15] we can effectively decide its Ultimate Positivity. We hence assume that $\mathbf{u}$ is not simple.

By Property 5, it follows that the characteristic roots of $\mathbf{u}$ all have multiplicity 1, apart from the characteristic root 1 which has multiplicity 2. Consider the dominant modulus ρ of $\mathbf{u}$. If $\rho > 1$, then by writing the exponential polynomial of $\mathbf{u}$, we have

$$\frac{u_n}{\rho^n} = a + \sum_{i=1}^{m} (c_i \lambda_i^n + \overline{c_i}\,\overline{\lambda_i^n}) + r(n) \tag{8}$$

with $a \in \mathbb{R}$, $c_i \in \mathbb{C} \setminus \mathbb{R}$ and $|\lambda_i| = 1$ for every $1 \leq i \leq m$, and $|r(n)|$ exponentially decaying. Crucially, since 1 is not a dominant characteristic root, its effect is enveloped in $r(n)$. Specifically, we observe that the analysis of effective Ultimate Positivity in [15] only relies on Proposition 7. Since this holds in the case at hand, we can effectively decide Ultimate Positivity when 1 is not a dominant characteristic root.

Finally, if 1 is a dominant characteristic root, the exponential polynomial of $\mathbf{u}$ can be written as

$$u_n = A(n) + \sum_{i=1}^{m} (c_i \lambda_i^n + \overline{c_i}\,\overline{\lambda_i^n}) + r(n). \tag{9}$$

We observe that in this case, u_n is ultimately positive iff it diverges (indeed, clearly $|u_n| \to \infty$). Thus, we can reduce the problem to divergence, and proceed with the analysis as in Section 3 Case 2.

This concludes the proof that Ultimate Positivity is effectively decidable for simple inhomogeneous LRS of order at most 8. ◀

Similarly to Theorem 18, we are able to conclude the following result, whose proof can be found in the full version.

▶ **Theorem 19.** *Ultimate Positivity is decidable in polynomial time for simple inhomogeneous LRS of any order.*

Finally, using Theorem 18, we can solve the Positivity problem (see the full version for the proof).

▶ **Theorem 20.** *Positivity is decidable for simple inhomogeneous LRS of order* 8 *or less, with complexity in* $\mathbf{coNP}^{\mathbf{PosSLP}}$.

4.2 Lower Bounds

We now turn to study lower bounds, proving analogous results to Theorem 16 for inhomogeneous LRS. Similarly to [15], the hardness results we present are with respect to long standing open problems in Diophantine approximation. Before stating our results, we require some definitions from Diophantine approximation. We refer the reader to [13, 15] for comprehensive references.

For any $x \in \mathbb{R}$, we define the *Lagrange constant* of x as

$$L_\infty(x) = \inf\{c \in \mathbb{R} : |x - \frac{n}{m}| \leq \frac{c}{m^2} \text{ for infinitely many } m, n \in \mathbb{Z}\}$$

and the *approximation type* of x as

$$L(x) = \inf\{c \in \mathbb{R} : |x - \frac{n}{m}| \leq \frac{c}{m^2} \text{ for some } m, n \in \mathbb{Z}\}$$

For the vast majority of transcendental numbers, the Lagrange constant and the approximation type are unknown, despite significant work [8, 15], and the problem of computing them is a major open problem. In the following, we show that the decidability of Ultimate Positivity (resp. Positivity) for inhomogeneous LRS of order 5 or more would imply a major breakthrough in computing the Lagrange constant (resp. approximation type) for a large class of transcendental numbers.

▶ **Theorem 21.** *If Ultimate Positivity is decidable for inhomogeneous rational LRS of order at least 5 then there is an algorithm that computes the Lagrange constant of any number* $\theta/2\pi$ *such that* $e^{i\theta}$ *has rational real and imaginary parts.*

Proof. In [15], it is shown that deciding Ultimate Positivity of the homogeneous LRS of order 6 given by

$$u_n = r \sin n\theta - n(1 - \cos n\theta) \text{ and } v_n = -r \sin n\theta - n(1 - \cos n\theta)$$

for every $r \in \mathbb{Q}$ such that $r > 0$ and $\theta \in (0, 2\pi)$ such that $e^{i\theta}$ has rational real and imaginary parts would allow one to compute $L_\infty(\theta/2\pi)$.

We observe that both sequences u_n and v_n fall under the premise of Proposition 4. Thus, by applying Proposition 4, we obtain an equivalent inhomogeneous LRS of order 5, concluding the proof. ◀

A similar proof, using the results of [15], gives us also the following theorem.

▶ **Theorem 22.** *If Positivity is decidable for inhomogeneous rational LRS of order at least 5 then there is an algorithm that computes the approximation type of any number $\theta/2\pi$ such that $e^{i\theta}$ has rational real and imaginary parts.*

References

1. S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015.
2. Alan Baker and Gisbert Wüstholz. Logarithmic forms and group varieties. *J. reine angew. Math*, 442(19-62):3, 1993.
3. W. J. Baumol. *Economic Dynamics. An Introduction.* Macmillan, 1970.
4. Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real algebraic geometry*, volume 36. Springer Science & Business Media, 2013.
5. Mark Braverman. Termination of integer linear programs. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 372–385, 2006.
6. John W.S. Cassels. *An Introduction to Diophantine Approximation.* Cambridge University Press, 1965.
7. Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.
8. Thomas W Cusick and Mary E Flahive. *The Markoff and Lagrange spectra.* Number 30 in Mathematical Surveys and Monographs. American Mathematical Soc., 1989.
9. Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. *Recurrence Sequences*, volume 104 of *Mathematical surveys and monographs.* American Mathematical Society, 2003.
10. J.-H. Evertse. On sums of S-units and linear recurrences. *Compositio Math.*, 53(2):225–244, 1984.
11. David W Masser. Linear relations on algebraic groups. *New Advances in Transcendence Theory*, pages 248–262, 1988.
12. M. Mignotte. A note on linear recursive sequences. *J. Austral. Math. Soc.*, 20(2):242–244, 1975.
13. Ivan Morton Niven. *Diophantine approximations.* Courier Corporation, 2008.
14. Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014.
15. Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379, 2014.

16 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 330–341, 2014.

17 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015.

18 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. *Journal of symbolic computation*, 13(3):255–299, 1992.

19 T. N. Shorey and C. L. Stewart. On the Diophantine equation $ax^{2t} + bx^t y + cy^2 = d$ and pure powers in recurrence sequences. *Math. Scand.*, 52(1):24–36, 1983.

20 Alfred Tarski. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society*, 59, 1951.

21 A. J. van der Poorten and H. P. Schlickewei. Additive relations in fields. *J. Austral. Math. Soc. Ser. A*, 51(1):154–170, 1991.