27th International Conference on Concurrency Theory

CONCUR'16, August 23–26, 2016, Québec City, Canada

Edited by

Josée Desharnais

Radha Jagadeesan

Editors

Josée Desharnais
Université Laval
Québec
josee.desharnais@ift.ulaval.ca

Radha Jagadeesan
DePaul University
Chicago
rjagadeesan@cs.depaul.edu

ACM Classification 1998
D. Software, E. Data, F. Theory of Computation

ISBN 978-3-95977-017-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-017-0.

Publication date
August, 2016

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.2016.0

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

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Distributed Systems

Distances for Probabilistic Systems

Categories

Timed and Parametrized Systems

Logic

Probability

Preface

This volume contains the proceedings of the 27th Conference on Concurrency Theory (CONCUR 2016), which was hosted by Université Laval, in Quebec City, Canada from 23-26 August 2016. This year, CONCUR was co-located with the 13th International Conference on Quantitative Evaluation of SysTems (QEST) and the 14th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS) and two workshops, EXPRESS/SOS and TRENDS.

The aim of the CONCUR conferences is to bring together researchers, developers and students in order to contribute to the development and dissemination of the theory of concurrency and its applications. More than twenty five years after our first meeting in 1990, it is still the reference annual event for researchers in this field. This edition of the conference attracted 120 submissions of abstracts. 100 full papers were submitted for review, of which the Program Committee selected 34 papers for presentation at the conference. Most submissions were reviewed by four reviewers, aided by the generous help provided by external reviewers. The Conference Chairs warmly thank all the members of the Program Committee and all the additional reviewers for their excellent work and the constructive discussions. It is our hope that all authors were benefited as a result of these efforts. The full list of reviewers is available as part of these proceedings.

The program was enhanced by invited talks from Scott Smolka (joint invited speaker of QEST and FORMATS) , Vincent Danos, Francesca Rossi and Marc Shapiro. These talks cover a broad range of topics from traditional concurrency theory and distributed systems through reasoning about collective decision support systems in the context of autonomous AI agents. Their abstracts and invited papers (in some cases) are available as part of these proceedings.

Continuing the change made last year, CONCUR proceedings are available for open access via LIPIcs.

Last, but not least, we thank the authors and the participants for their enthusiastic participation.

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan

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

Committees

Program Committee

Alessandro Abate

Gui Agha

Nathalie Bertrand

Luis Caires

Véronique Cortier

Josée Desharnais

Yuan Feng

Alexey Gotsman

Éric Goubault

Petr Jancar

Radha Jagadeesan

Anna Ingolfsdottir

Naoki Kobayashi

Barbara König

Jean Krivine

Jérôme Leroux

Alberto Lluch Lafuente

Mohammadreza Mousavi

John Mullins

Uwe Nestmann

Gethin Norman

Catuscia Palamidessi

Jun Pang

Joachim Parrow

Daniela Petrisan

Pavel Sobocinski

Ana Sokolova

Nadia Tawbi

Mirco Tribastone

Ashutosh Trivedi

Franck van Breugel

Rob van Glabbeek

Gianluigi Zavattaro

Lijun Zhang

Co-Chairs

Josée Desharnais

Radha Jagadeesan

Steering Committee

Jos Baeten

Javier Esparza

Joost-Pieter Katoen

Kim G. Larsen

Ugo Montanari

Scott Smolka

Organizing Committee

Andrew Bedford (Laval University)

Josée Desharnais (Laval University)

Raymond Poirier (ITIS)

Gilles Rioux (ITIS)

Nadia Tawbi (Laval University)

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Samy Abbes
Parosh Aziz Abdulla
Luca Aceto
S. Akshay
Oana Andrei
Sebastian Arming
Clément Aubert
Souheib Baarir
Giorgio Bacci
Eric Badouel
Franco Barbanera
Kamel Barkaoui
Massimo Bartoletti
Francesco Belardinelli
Harsh Beohar
Giovanni Bernardi
Luca Bernardinello
Marco Bernardo
Clara Bertolissi
Devendra Bhave
Filippo Bonchi
Marcello Bonsangue
Laura Bozzelli
Tomas Brazdil
Romain Brenguier
Flavien Breuvart
Paul-David Brodmann
Roberto Bruni
Benjamin Cabrera
Yongzhi Cao
Franck Cassez
Valentina Castiglioni
Dario Cattaruzza
Andrew Cave
Pavol Cerny
Aleksandar Chakarov
Minas Charalambides
Thomas Chatain
Taolue Chen
Corina Cirstea
Pierre Clairambault
Lorenzo Clemente
Christian Colombo
João Costa Seco
Ioana Cristescu
Luís Cruz-Filipe
Pedro Da Rocha Pinto
Ferruccio Damiani
Pallab Dasgupta
Frank De Boer
Romain Demangeon
Stéphane Demri
Yuxin Deng
Pierre-Malo Denielou
Jules Desharnais
Raymond Devillers
Cinzia Di Giusto
Mike Dodds
Simon Doherty
Pierre Donat-Bouillud
Brijesh Dongol
Laurent Doyen
Cezara Dragoi
Derek Dreyer
Jérémy Dubut
Constantin Enea
Javier Esparza
Dirk Fahland
Jerome Feret
Norm Ferns
Carla Ferreira
Nathanael Fijalkow
Wan Fokkink
Paulin Fournier
Adrian Francalanza
Ignacio Fábregas
Álvaro García-Pérez
Blaise Genest
Dan Ghica
Elena Giachino
Hugo Gimbert
Ramūnas Gutkovas
Andreas Haas
Sofie Haesaert
Matthew Hague
Ernst Moritz Hahn
Emmanuel Haucourt
Fei He
Pierre-Cyrille Heam
Tobias Heindel

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keijo Heljanko
Frédéric Herbreteau
Tom Hirschowitz
Piotr Hofman
Hossein Hojjat
Andreas Holzer
Damien Imbs
Swen Jacobs
Alan Jeffrey
Bengt Jonsson
Aleksandra Jovanovic
Tim Jungnickel
Thierry Jéron
David S. Karcher
Jeroen J.A. Keiren
Daniel Khankin
Artem Khyzha
Stefan Kiefer
Dileep Kini
Helene Kirchner
Hanna Klaudel
Bartek Klin
Sophia Knight
Igor Konnov
Paraschos Koutris
Dimitrios Kouzapas
Jan Kretinsky
Shankara Narayanan Krishna
Clemens Kupke
Sebastian Küpper
Ivan Lanese
Julien Lange
Francois Laroussinie
Fribourg Laurent
Ranko Lazic
Matias David Lee
Stephane Lengrand
Luis Llana
Andreas Lochbihler
Delphine Longuet
Bas Luttik
Luc Maranget
Radu Mardare
Richard Mayr
Hernan Melgratti
Stephan Merz
Roland Meyer
Lukasz Mikulski
Dimitrios Milios
Dale Miller
Samuel Mimram
Andrzej Mizera
Sergio Mover
Muhammad Najib
Laura Nenzi
Van Chan Ngo
Karl Palmskog
Dirk Pattinson
Guillermo Perez
Kirstin Peters
Iain Phillips
Hernan Ponce-De-Leon
Andrei Popescu
Damien Pous
Marc Pouzet
Nuno Preguiça
Tobias Prehn
Jorge A. Pérez
Amgad Rady
Jean-François Raskin
Julian Rathke
Ahmed Rezine
Christina Rickmann
James Riely
Cesar Rodriguez
Adam Rogalewicz
Fernando Rosa-Velardo
Jurriaan Rot
Luca Roversi
Eric Ruppert
Gwen Salaün
Arnaud Sangnier
Ocan Sankur
Zdenek Sawa
Alceste Scalas
Philippe Schnoebelen
Stefan Schwoon
Ilya Sergey
Olivier Serre
Ali Sezgin
Mahsa Shirmohammadi
Natalia Sidorova
Fu Song
Jiri Srba
B Srivathsan
Daniel Stan

Guoxin Su
Kohei Suenaga
Chamseddine Talhi
Qiyi Tang
Maurice H. Ter Beek
Pascal Tesson
Simone Tini
Bernardo Toninho
Max Tschaikowski
Takeshi Tsukada
Andrea Turrini
Nikos Tzevelekos
Viktor Vafeiadis
Frank Valencia
Antti Valmari
Jaco van de Pol
Jan Martijn Van Der Werf
Andrea Vandin
Daniele Varacca
Björn Victor
Valeria Vignudelli
Hagen Voelzer
Walter Vogler
Christoph Wagner
Tjark Weber
Matthias Weidlich
Arno Wilhelm-Weidner
Dominik Wojtczak
Karsten Wolf
James Worrell
Zhilin Wu
Shenggang Ying
Fabio Zanasi
Naijun Zhan
Yi Zhang
Chunlai Zhou
Roberto Zunino
Aditya Zutshi
Johannes Åman Pohjola
Vladimír Štill

List of Authors

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

S. Akshay
IIT Bombay
India
akshayss@cse.iitb.ac.in

Shaull Almagor
Hebrew University
Israel
shaull.almagor@mail.huji.ac.il

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

Giorgio Bacci
Aalborg University
Denmark
grbacci@cs.aau.dk

Giovanni Bacci
Aalborg University, Denmark
Denmark
giovbacci@cs.aau.dk

Giovanni Bernardi
IMDEA Software Institute
Spain
bernargi@tcd.ie

Nathalie Bertrand
Inria
France
nathalie.bertrand@inria.fr

Simon Bliudze
EPFL
Switzerland
simon.bliudze@epfl.ch

Borzoo Bonakdarpour
McMaster University
Canada
borzoo@mcmaster.ca

Ahmed Bouajjani
LIAFA, University Paris Diderot
France
abou@liafa.univ-paris-diderot.fr

Tomas Brazdil
Masaryk University
Czech Republic
xbrazdil@fi.muni.cz

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

Romain Brenguier
University of Oxford
United Kingdom
romain.brenguier@cs.ox.ac.uk

Franck van Breugel
York University
Canada
franck@cse.yorku.ca

Marco Carbone
IT University of Copenhagen
Denmark
carbonem@itu.dk

Simon Castellan
ENS Lyon
France
simon.castellan@ens-lyon.fr

Valentina Castiglioni
University of Insubria
Italy
v.castiglioni2@uninsubria.it

Konstantinos Chatzikokolakis
CNRS & LIX, Ecole Polytechnique
France
kostas@chatzi.org

James Cheney
University of Edinburgh
United Kingdom
jcheney@inf.ed.ac.uk

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pierre Clairambault
CNRS and ENS de Lyon
France
pierre.clairambault@ens-lyon.fr

Przemyslaw Daca
IST Austria
Austria
przemek@ist.ac.at

Fredrik Dahlqvist
University College London
United Kingdom
f.p.h.dahlqvist@gmail.com

Vincent Danos
Ecole Normale Supérieure, Paris
United Kingdom
vincent.danos@gmail.com

Amélie David
CNRS – LSV & UParis-Saclay
France
amelie.david@lsv.fr

Jérémy Dubut
LSV, ENS Cachan
France
dubut@lsv.ens-cachan.fr

Javier Esparza
TUM
Germany
esparza@in.tum.de

Bernd Finkbeiner
Universität des Saarlandes
Germany
finkbeiner@cs.uni-saarland.de

Vojtech Forejt
Oxford University
United Kingdom
forejt@fi.muni.cz

Pierre Fraigniaud
LIAFA
France
pierre.fraigniaud@liafa.univ-paris-diderot.fr

Deepak Garg
Max Planck Institute for Software Systems
Germany
dg@mpi-sws.org

Ilias Garnier
University of Edinburgh
France
ilias.gar@gmail.com

Paul Gastin
LSV, ENS Cachan
France
paul.gastin@lsv.fr

Daniel Gebler
VU University Amsterdam
Netherlands
e.d.gebler@vu.nl

Alexey Gotsman
IMDEA Software Institute
Spain
alexey.gotsman@imdea.org

Eric Goubault
LIX, Ecole Polytechnique
France
eric.goubault@polytechnique.edu

Jean Goubault-Larrecq
LSV, ENS Cachan, CNRS, INRIA Futurs
France
goubault@lsv.ens-cachan.fr

Serge Haddad
LSV, ENS Cachan, CNRS, INRIA
France
haddad@lsv.fr

Christopher Hahn
Universität des Saarlandes
Germany
s9chhahn@stud.uni-saarland.de

Ichiro Hasuo
Department of Computer Science, University of Tokyo
Japan
ichiro@is.s.u-tokyo.ac.jp

Daniel Hausmann
Friedrich-Alexander-Universität Erlangen-Nürnberg
Germany
daniel.hausm4nn@gmail.com

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

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

Andreas Holzer
University of Toronto
Canada
andreas.holzer81@gmail.com

Ross Horne
Nanyang Technological University
Singapore
ross.horne@gmail.com

Loig Jezequel
Université de Nantes
France
loig.jezequel@irccyn.ec-nantes.fr

Christoph Kirsch
University of Salzburg
Austria
ck@cs.uni-salzburg.at

Igor Konnov
TU Wien
Austria
konnov@forsyte.at

Tomer Kotek
TU Wien
Austria
kotek@forsyte.at

Jan Kretinsky
Technical University of Munich
Germany
xkretins@fi.muni.cz

Shankara Narayanan Krishna
IIT Bombay
India
krishnas@cse.iitb.ac.in

Antonin Kucera
Masaryk University
Czech Republic
tony@fi.muni.cz

Denis Kuperberg
TUM
Germany
denis.kuperberg@gmail.com

Orna Kupferman
Hebrew University
Israel
orna@cs.huji.ac.il

Francois Laroussinie
LIAFA, Univ. Paris 7, CNRS
France
francoisl@liafa.univ-paris-diderot.fr

Kim Guldstrand Larsen
Computer Science, Aalborg University
Denmark
kgl@cs.aau.dk

Antonia Lechner
University of Oxford
United Kingdom
antonia.lechner@cs.ox.ac.uk

Engel Lefaucheux
IRISA
France
engel.lefaucheux@irisa.fr

Didier Lime
Ecole Centrale de Nantes, IRCCyN
France
didier.lime@ec-nantes.fr

Sam Lindley
University of Edinburgh
United Kingdom
sam.lindley@ed.ac.uk

Michael Lippautz
Google Inc.
Germany
michael.lippautz@gmail.com

Radu Mardare
Aalborg University
Denmark
mardare@cs.aau.dk

Nicolas Markey
LSV, CNRS & ENS Cachan
France
markey@lsv.fr

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

Fabrizio Montesi
University of Southern Denmark
Denmark
famontesi@gmail.com

Anca Muscholl
TUM
Germany
anca@labri.fr

Uwe Nestmann
Technische Universität Berlin
Germany
uwe.nestmann@tu-berlin.de

Tuan Phong Ngo
Uppsala University
Sweden
tuan-phong.ngo@it.uu.se

Petr Novotný
IST Austria
Austria
petr.novotny.mail@gmail.com

Joel Ouaknine
Oxford University
United Kingdom
joel@cs.ox.ac.uk

Catuscia Palamidessi
INRIA & LIX, Ecole Polytechnique
France
catuscia@lix.polytechnique.fr

Hannes Payer
Google Inc.
Germany
hannes.payer@gmail.com

Roly Perera
University of Glasgow
United Kingdom
roly.perera@dynamicaspects.org

Marco Peressotti
DiMA, University of Udine
Italy
marco.peressotti@uniud.it

Tatjana Petrov
IST Austria
Austria
tatjana.petrov@gmail.com

Amaury Pouly
University of Oxford
United Kingdom
amaury.pouly@cs.ox.ac.uk

Sergio Rajsbaum
Instituto de Matematicas, UNAM
Mexico
sergio.rajsbaum@gmail.com

Christina Rickmann
Technische Universität Berlin
Germany
c.rickmann@tu-berlin.de

David A. Rosenblueth
Universidad Nacional Autonoma de Mexico
Mexico
drosenbl@servidor.unam.mx

Uli Schlachter
Uni Oldenburg
Germany
uli.schlachter@informatik.uni-oldenburg.de

Stefan Schmid
TU Berlin & Telekom Innovation
Laboratories (T-Labs)
Germany
stefan.schmid@tu-berlin.de

Lutz Schröder
FAU Erlangen-Nürnberg
Germany
lutz.schroeder@fau.de

Carsten Schuermann
IT University of Copenhagen
Denmark
carsten@itu.dk

Ali Sezgin
University of Cambridge
United Kingdom
as2418@cam.ac.uk

Shunsuke Shimizu
The University of Tokyo
Japan
shunsuke@is.s.u-tokyo.ac.jp

Joseph Sifakis
EPFL
Switzerland
joseph.sifakis@epfl.ch

Ana Sokolova
University of Salzburg
Austria
anas@cs.uni-salzburg.at

Jan Sürmeli
Humboldt-Universität zu Berlin
Germany
suermeli@informatik.hu-berlin.de

Simone Tini
Univerisity of Insubria
Italy
simone.tini@uninsubria.it

Corentin Travers
CNRS and University of Bordeaux
France
corentin.travers@labri.fr

Marvin Triebel
Humboldt-Universität zu Berlin
Germany
triebel@informatik.hu-berlin.de

Natsuki Urabe
The University of Tokyo
Japan
urabenatsuki@is.s.u-tokyo.ac.jp

Helmut Veith
TU Wien
Austria
veith@forsyte.at

Yaron Velner
Hebrew University
Israel
yaron.welner@mail.huji.ac.il

Valeria Vignudelli
University of Bologna & INRIA
Italy
valeria.vignudelli@gmail.com

Philip Wadler
University of Edinburgh
United Kingdom
wadler@inf.ed.ac.uk

Christoph Wagner
TU Berlin
Germany
christoph.wagner@tu-berlin.de

Igor Walukiewicz
TUM
Germany
igw@labri.fr

Qiang Wang
EPFL
Switzerland
qiang.wang@epfl.ch

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

Bayesian Inversion by ω-Complete Cone Duality*

Fredrik Dahlqvist[1], Vincent Danos[2], Ilias Garnier[3], and Ohad Kammar[4]

1 **University College London**
f.dahlqvist@ucl.ac.uk
2 **Ecole Normale Supérieure**
vincent.danos@ens.fr
3 **University of Edinburgh**
igarnier@inf.ed.ac.uk
4 **University of Oxford**
ohad.kammar@cl.cam.ac.uk

Abstract

The process of inverting Markov kernels relates to the important subject of Bayesian modelling and learning. In fact, Bayesian update is exactly kernel inversion. In this paper, we investigate how and when Markov kernels (aka stochastic relations, or probabilistic mappings, or simply kernels) can be inverted. We address the question both directly on the category of measurable spaces, and indirectly by interpreting kernels as Markov operators:

- For the direct option, we introduce a typed version of the category of Markov kernels and use the so-called 'disintegration of measures'. Here, one has to specialise to measurable spaces borne from a simple class of topological spaces -e.g. Polish spaces (other choices are possible). Our method and result greatly simplify a recent development in Ref. [4].
- For the operator option, we use a cone version of the category of Markov operators (kernels seen as predicate transformers). That is to say, our linear operators are not just continuous, but are required to satisfy the stronger condition of being ω-chain-continuous.[1] Prior work shows that one obtains an adjunction in the form of a pair of contravariant and inverse functors between the categories of L_1- and L_∞-cones [3]. Inversion, seen through the operator prism, is just adjunction.[2] No topological assumption is needed.
- We show that both categories (Markov kernels and ω-chain-continuous Markov operators) are related by a family of contravariant functors T_p for $1 \leq p \leq \infty$. The T_p's are Kleisli extensions of (duals of) conditional expectation functors introduced in Ref. [3].
- With this bridge in place, we can prove that both notions of inversion agree when both defined: if f is a kernel, and $f^\dagger$ its direct inverse, then $\mathsf{T}_\infty(f)^\dagger = \mathsf{T}_1(f^\dagger)$.

1998 ACM Subject Classification Semantics of programming languages

Keywords and phrases probabilistic models, Bayesian learning, Markov operators

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.1

Category Invited Paper

* This work was funded partly through the RULE/320823 ERC project.
[1] This stronger continuity condition derives from domain-theoretic ideas and is due to Selinger [14].
[2] A similar L_p/L_q duality result also exists [2], and we will investigate it in later work.

1 Introduction

Before we get in the technicalities, we review informally the central notions of interest: kernels, kernels as operators, and Bayesian update. We also take the opportunity to introduce some of the notations used in the remainder of the paper.

A kernel is a (measurable) map from some measurable space X to the set of probability distributions on another measurable space Y. Probability distributions have to be equipped with a set of mesasurables for this to make sense. We will write $X \to \mathsf{G}Y$ or equivalently $X \to\!\!\!\!\bullet\, Y$ for the type of such X, Y kernels, where $\mathsf{G}X$ is the set of probability distributions over X. With the appropriate constructions, G is a monad over **Mes**, the category of measurable spaces and measurable maps [9].

Kernels are often used as models of probabilistic behaviour. If $X = Y$ is finite, then this is but the familiar notion of probabilistic state machine (or finite discrete-time Markov chain), where one jumps probabilistically from one state to the next with no dependence on the past. We say a kernel $f : X \to GY$ is deterministic[3] if it factorises in **Mes** as $f = \delta_Y \circ f'_d$ for some $f_d : X \to Y$, where $\delta_X : X \to\!\!\!\!\bullet\, X$ sends x to δ_x the Dirac measure at x. Intuively determinism means that f allows only one possible jump at each x in X. In particular, δ_X is a deterministic kernel itself (with an identical jump).

One can also use a kernel as a family of probabilities over Y parameterised by X. Each probability in the range of the kernel can be thought as a competing description of a hidden true probability. Given an additional probability p on the parameter space X, a *prior*, one can specify how much 'trust' one has in any particular description offered. This is the Bayesian model of quantification of uncertain probabilities: the prior describes our beliefs and 'Bayesian update' is a process by which new data (minted by the true random source) can be incorporated to modify the prior, and update our beliefs. The hope is that in the long run the successive priors will take us closer to the truth.

Kernels as operators

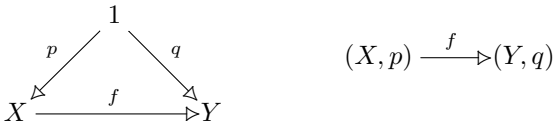
Yet another standpoint on kernels is to look at them as linear maps. Indeed, a finite $f : X \to \mathsf{G}Y$ can be seen as a transition matrix $\mathsf{T}(f)$ of type $X \times Y$ with the x, y-entry specifying the probability that being at x in X one jumps to y in Y.[4] Thus, one can think of $f : X \to \mathsf{G}Y$ as a linear map from the free vector space over Y to that over X. Clearly $\mathsf{T}(\delta_X) = I_X$. In particular, a probability p over X, ie a map from $1 \to \mathsf{G}X$, is a matrix $\mathsf{T}(p)$ of type $1 \times X$ (a row vector). This new standpoint gives a ready access to the key operation of kernel composition, written $\circ_{\mathbf{G}}$. Because G is a monad (known as the Giry monad), one knows how to compose kernels for general reasons using the so-called *Kleisli composition* (see Section 2). In the operator interpretation, Kleisli composition is just plain matrix multiplication. E.g. the composition $f \circ_{\mathbf{G}} p$ of p and f is represented as $\mathsf{T}(p)\mathsf{T}(f)$, a new row vector of type $1 \times Y$. The T (contravariant) functor can be extended to arbitrary measurable kernels, using the machinery of Banach cones of real functions (see Section 4.4).[5]

[3] Or a deterministic map, following the terminology of Lawvere in his seminar handout notes on the "category of probabilistic mappings" (1962).

[4] Of course the transpose representation is also possible as will be clear in the duality of the operator interpretation.

[5] This predicate transformer view was first championed by Kozen for probabilistic programs [12].

Importantly, we will deal not with 'naked' Markov kernels, but with 'typed' ones of the form:[6]

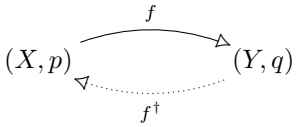


where the triangle (above, left) is drawn in the Kleisli category of G in **Mes** and assumed to commute. The simpler diagram (above, right) is just a more compact notation for the same. (Recall that we use blocky arrows to remind us that these are Kleisli arrows.) Either diagram means that in addition to f, we are given probabilities p on X, q on Y with $f \circ_{\mathbf{G}} p = q$.

This category of typed kernels has a natural subcategory where f is deterministic. The typing simply amounts to saying that q is the push-forward of p along g.[7] E.g. $\delta_X : (X, p) \to\!\!\!\triangleright (X, p)$ for any p in GX. This subcategory is (equivalent to) the familiar category of probabilistic triples and measurable measure-preserving maps.[8]

Bayesian update or inversion

Our main question is as folllows. Given a typed kernel f, we wish to build and characterise a 'weak' inverse $f^\dagger$:



In the Bayesian world, $f^\dagger$ should be the update map we mentioned above. Given a data input y in Y, this map returns a *posterior* $f^\dagger(y)$ which represents our updated set of beliefs. It is therefore central to the theory to obtain good and general descriptions of $f^\dagger$. We will access such descriptions of $f^\dagger$ following two routes. The direct route uses the non-trivial notion of disintegration (aka regular conditional probabilities) which solves the inversion problem in the special case of a deterministic f. A clever construction based on couplings allows one to generalise it to any kernel. This is done in Section 3.3 and follows the construction of Ref. [4] while being markedly simpler. The other route goes the operator way. As alluded to in the abstract, we use a domain theoretic variant of the usual interpretation which carries a perfect duality (which one does not have in the standard interpretation). Then, as the notation suggests, we can just define $\mathsf{T}(f)^\dagger$ as the adjoint of $\mathsf{T}(f)$ (this is done in Section 4). Bayesian inversion is cone duality! Whence the title. We then show that through our functorial bridge, T (really a family T_p), both routes agree (Section 5).

The paper is almost self-contained. The only pieces of mathematics borrowed are the disintegration and the cone duality results and some of the most basic definitions. We now turn to the technical preliminaries.

---

[6] In category-theoretical words, our arrows are in the 'category under 1' of the Kleisli of G.

[7] Because in this case $f \circ_{\mathbf{G}} p = (\delta \circ f_d) \circ_{\mathbf{G}} p = \mu \circ \mathsf{G}(\delta \circ f_d) \circ p = \mu \circ \mathsf{G}(\delta) \circ \mathsf{G}(f_d) \circ p = \mathsf{G}(f_d) \circ p$; as follows from the monad structure equation $\mu \circ \mathsf{G}(\delta) = I$.

[8] Richer categories of kernels were considered before; for instance, to obtain a notion of almost sure bisimilarity on labelled kernels (aka labelled Markov processes) Ref. [6, Section 7] equips kernels with ideals of negligibles.

2 Preliminaries

Measurable and Polish spaces

We refer to Ref. [1] for the definitions of measurable spaces and maps, and Ref. [11] for an introduction to the theory of Polish spaces (completely metrisable and separable topological spaces). Where convenient, we will denote measurable spaces and related structures by their underlying set. If X is a set, (Y, Λ) a measurable space and $f : X \to Y$ a function, we denote by $\sigma(f)$ its *initial σ-algebra*, which is the smallest σ-algebra that makes it measurable. As seen earlier, the category of measurable spaces and measurable maps is denoted by **Mes**, and that of Polish spaces and continuous maps by **Pol**. There is a functor $\mathcal{B} : \mathbf{Pol} \to \mathbf{Mes}$ associating any Polish space to the measurable space with same underlying set equipped with the 'Borel' σ-algebra (generated by open sets), and interpreting continuous maps as measurable ones. Measurable spaces in the range of $\mathcal{B}$ are the *standard Borel spaces*.

A *measure* p on a measurable space (X, Σ) is a set function $\Sigma \to \mathbb{R}$ which is σ-additive and such that $p(\emptyset) = 0$. One says p is a *finite measure* whenever $p(X) < \infty$, and a probability measure if $p(X) = 1$. A property holds p-almost surely (p-a.s) if its negation holds on a set of measure 0. A *measure space* is a triple (X, Σ, p) such that (X, Σ) is a measurable space and p is a finite measure on (X, Σ). We denote by $p|_\Lambda$ the restriction of p to a sub-σ-algebra $\Lambda \subseteq \Sigma$. A Borel measurable real-valued function $f : (X, \Sigma, p) \to \mathbb{R}$ is called *integrable* if $\int_X |f| \, dp < \infty$.

Radon-Nikodym and conditional expectations

Let (X, Σ) be some measurable space. For p, q finite measures, we say that p is absolutely continuous with respect to q if for all B measurable, $q(B) = 0$ implies $p(B) = 0$. This will be denoted by $p \ll q$. The Radon-Nikodym theorem tells us that we can express p in terms of its *derivative* with respect to q:

▶ **Theorem 1** (Radon-Nikodym). *If $p \ll q$ there exists a q-a.s. unique positive integrable function denoted by $\frac{dp}{dq} : (X, \Sigma, q) \to \mathbb{R}$ such that $p = B \mapsto \int_B \frac{dp}{dq} dq$.*

The function $\frac{dp}{dq}$ is called the Radon-Nikodym derivative of p with respect to q.

Let us denote $f \cdot p = B \in \Lambda \mapsto \int_B f \, dp$. Clearly $f \cdot p \ll p$. The following two identities follow from Theorem 1: (i) $\frac{df \cdot p}{dp} = f$ (ii) $\frac{dp}{dq} \cdot q = p$. We refer the reader to [1] for further facts about Radon-Nikodym derivatives. Conditional expectations can be implemented in terms of Radon-Nikodym derivatives.

▶ **Definition 2** (Conditional expectation ([10])). Let (X, Σ, p) be a measure space and let $\Lambda \subseteq \Sigma$ be a sub-σ-algebra. The *conditional expectation* of an integrable function $f : (X, \Sigma, p) \to \mathbb{R}$ with respect to Λ is the p-a.s. unique integrable function $\mathbb{E}[f \mid \Lambda] : (X, \Lambda, p|_\Lambda) \to \mathbb{R}$ that verifies for all $B \in \Lambda$ the identity $\int_B \mathbb{E}[f \mid \Lambda] \, dp = \int_B f \, dp$.

Theorem 1 implies the existence of conditional expectations: letting $f \cdot p = B \in \Lambda \mapsto \int_B f \, dp$, we have for all $B \in \Lambda$ the characteristic identity $\int_B \frac{df \cdot p}{dp|_\Lambda} \, dp|_\Lambda = \int_B f \cdot p = \int_B f dp$.

Probability functors

The endofunctor $\mathsf{G} : \mathbf{Mes} \to \mathbf{Mes}$ associates to any measurable space X the set of all probability measures on X with the smallest σ-algebra that makes the evaluation functions $ev_B : \mathsf{G}(X) \to \mathbb{R} = p \mapsto p(B)$ measurable, for B a measurable set in X. If $f : X \to Y$ is measurable, the action of the functor is defined by $\mathsf{G}(f)(P) = P \circ f^{-1}$. This functor can be

endowed with the structure of the Giry monad $(\mathsf{G}, \mu, \delta)$. The multiplication $\mu : \mathsf{G}^2 \Rightarrow \mathsf{G}$ is defined at a component X by $\mu_X(P)(B) = \int_{\mathsf{G}(X)} ev_B dP$ while the unit $\delta : Id \Rightarrow \mathsf{G}$ is defined at X by $\delta_X(x) = \delta_x$, where $\delta_x(B) = 1$ if and only if $x \in B$.

The Kleisli category of G will be denoted by $\mathcal{K}\ell$. It has the same objects as **Mes**. For all X, Y measureable spaces, a Kleisli arrow $f : X \to Y$ in $\mathcal{K}\ell$ is a kernel $f : X \to \mathsf{G}(Y)$ in **Mes**. The Kleisli composition of the kernel $f : X \to Y$ with $g : Y \to Z$ is given by $g \circ_{\mathbf{G}} f = \mu_Y \circ \mathsf{G}(g) \circ f$.

3 Bayesian inversion

Let D be a space representing some space of *data* and let $t \in \mathsf{G}(D)$ be the *truth*, a unknown probability measure that we wish to discover by sampling repeatedly from it. In order to make this search analytically or computationally tractable, or more generally to reflect some additional knowledge or assumptions held about the truth, one might wish to parameterise the search through a space H of *parameters* and a measurable *likelihood function* $f : H \to D$. The uncertainty about which parameter best matches the truth is represented by a probability $p \in \mathsf{G}(H)$ called the *prior*. The composite of the two arrows $q = f \circ_{\mathbf{G}} p$ is called the *marginal likelihood*.

Bayesian inversion is the construction from these data of a *posterior map* $g : D \to H$, also called the *inference map*. Upon observing a sample $d \in D$, this inference map will produce an updated prior $g(d)$. In good cases (e.g. H and D finite, and q absolutely continuous w.r.t. t), sampling independently and identically from the truth t and iterating this Bayesian update will make the marginal likelihood converge (in some topology to be chosen carefully) to t. The key step in the above process is the construction of the posterior g, which relies crucially on *disintegrations*.

Culbertson & Sturtz give in [4] a nice categorical account of Bayesian inversion in a setting close to $\mathcal{K}\ell$. In the following, we provide a streamlined view of their work by defining a category of kernels where disintegration and Bayesian inversion admit rather elegant statements.

3.1 Categories of kernels

Let $F : \mathbf{Mes} \to \mathcal{K}\ell$ be the functor embedding **Mes** into the Kleisli category of G. It acts identically on spaces and maps measurable arrows $f : X \to Y$ to Kleisli arrows $F(f) = \delta_Y \circ f$. $1 \downarrow F$ is the category having as objects probabilities $p : 1 \to X$, denoted by (X, p), and as morphisms $f : (X, p) \to_\delta (Y, q)$ degenerate Kleisli arrows $F(f) : X \to Y$ such that $q = F(f) \circ_{\mathbf{G}} p = \mathsf{G}(f)(p)$. As said, these correspond to the usual notion of measure-preserving map. $1 \downarrow \mathcal{K}\ell$ is the category having the same objects as $1 \downarrow F$ but where arrows are non-degenerate, i.e. an arrow from (X, p) to (Y, q) as above is *any* Kleisli arrow $f : X \to Y$ such that $q = f \circ_{\mathbf{G}} p$. Clearly, $1 \downarrow F$ is a subcategory of $1 \downarrow \mathcal{K}\ell$ with the same objects (aka lluf).

The following result ensures that for an arrow $f : (X, p) \to (Y, q)$, there are p-negligibly many points jumping to q-negligible sets (it corresponds to the condition of *non-singularity* of [3]).

▶ **Lemma 3.** *If $f : (X, p) \to (Y, q)$ is an arrow in $1 \downarrow \mathcal{K}\ell$, then $f(x) \ll q$ p-a.s.*

Proof. By definition of $1 \downarrow \mathcal{K}\ell$, $q(B) = \int_X f(x)(B)\, dp$. Assume $q(B) = 0$, then having $f(x)(B) > 0$ on a set of strictly positive p-measure implies that the integral is strictly positive, yielding a contradiction. ◀

For all objects $(X,p),(Y,q)$, let $R_{(X,p),(Y,q)}$ be the smallest equivalence relation on $\mathsf{Hom}_{1\downarrow\mathcal{K}\ell}(X,Y)$ such that $(f,f') \in R_{(X,p),(Y,q)}$ if f and f' are p-a.s. equal.

▶ **Lemma 4.** *R defines a congruence relation on $1\downarrow\mathcal{K}\ell$.*

Proof. We must show that for all $g : (X',p') \nrightarrow (X,p)$ and all $h : (Y,q) \nrightarrow (Y',q')$, $(h\circ_{\mathbf{G}} f\circ_{\mathbf{G}} g, h\circ_{\mathbf{G}} f'\circ_{\mathbf{G}} g) \in R_{(X',p'),(Y',q')}$. First, let us prove that $(f\circ_{\mathbf{G}} g, f'\circ_{\mathbf{G}} g) \in R_{(X',p'),(X,p)}$. By Lemma 3, $g(x') \ll p$ p'-a.s., hence the following equation holds for p'-almost all x':

$$(f\circ_{\mathbf{G}} g)(x') = B_Y \mapsto \int_{x\in X} f(x)(B_Y)dg(x') = B_Y \mapsto \int_{x\in X} f'(x)(B)dg(x') = (f'\circ_{\mathbf{G}} g)(x')$$

It remains to prove that $h\circ_{\mathbf{G}} f\circ_{\mathbf{G}} g$ is p'-a.s. equal to $h\circ_{\mathbf{G}} f'\circ_{\mathbf{G}} g$. We have:

$$\begin{array}{rcll}(h\circ_{\mathbf{G}} f\circ_{\mathbf{G}} g)(x') & = & B_{Y'} \mapsto \int_{y'\in Y'} h(y')(B_{Y'})d(f\circ_{\mathbf{G}} g)(x') & \\ & = & B_{Y'} \mapsto \int_{y'\in Y'} h(y')(B_{Y'})d(f'\circ_{\mathbf{G}} g)(x') & p'\text{-a.s.} \\ & = & (h\circ_{\mathbf{G}} f'\circ_{\mathbf{G}} g)(x') & p'\text{-a.s.}\end{array}$$

which concludes the proof. ◀

This congruence relation allows us to consider R-equivalence classes of $1\downarrow\mathcal{K}\ell$ arrows as proper morphisms in the corresponding quotient category (Section 2.8, [13]):

▶ **Definition 5.** The category $\mathbf{Krn}$ is the quotient category $(1\downarrow\mathcal{K}\ell)/R$, with subcategory $\mathbf{Krn}_\delta = (1\downarrow F)/R$.

In other terms, an arrow $f : (X,p) \nrightarrow (Y,q)$ in $\mathbf{Krn}$ is an equivalence class of kernels that are p-a.s. equal.

3.2 Disintegrations

Disintegrations are also called *regular conditional probabilities* and correspond to measurable families of conditional probabilities. Working in the setting of standard Borel spaces ensures their existence, and the corresponding statement admits a particularly elegant form in $\mathbf{Krn}$:

▶ **Theorem 6** (Disintegration, [8])**.** *Let X and Y be standard Borel spaces, and let $f : (X,p) \nrightarrow_\delta (Y,q)$ be an arrow in $\mathbf{Krn}_\delta$. There exists a unique $\mathbf{Krn}$ arrow $f^\dagger : (Y,q) \nrightarrow (X,p)$ that verifies $f^\dagger(y)(f^{-1}(\{y\})) = 1$ q-a.s.*

We call $f^\dagger$ the *disintegration* of p along f. We will show in Section 5 that disintegrations and more generally Bayesian inverses are *adjoints*, hence the use of the $-^\dagger$ notation. The last condition can be equivalently stated as the fact that $f\circ f^\dagger = id_{(Y,q)}$. In order to bridge our crisp statement of Theorem 6 with the usual measure-theoretic one, let us unfold the objects at play. If we inspect the type of the arrows $f : (X,p) \nrightarrow_\delta (Y,q)$ and $f^\dagger : (Y,q) \nrightarrow (X,p)$ we see that by definition of composition in $\mathcal{K}\ell$, we have the equation $q = (\mu_Y \circ \mathsf{G}(\delta_Y \circ f))(p) = \mathsf{G}(f)(p)$. The existence of the disintegration arrow $f^\dagger : (Y,q) \nrightarrow (X,p)$ implies the equation $p = (\mu_Y \circ \mathsf{G}(f^\dagger))(q)$ which corresponds to

$$\begin{array}{rcll} p & = & B \mapsto \int_{\mathsf{G}(Y)} ev_B \, d\mathsf{G}(f^\dagger)(q) & \\ & = & B \mapsto \int_{y\in Y} f^\dagger(y)(B)\, dq & \text{(Change of variables)}\end{array} \tag{1}$$

We recall that the uniqueness of $f^\dagger$ claimed in Theorem 6 is really that of a q-equivalence class of kernels. Note also that disintegrations do not in general exist in $\mathbf{Pol}$ as they need not be continuous (even when disintegrating along a continuous map).

Disintegration, as said above, is a measurable family of conditional probabilities. The sub-σ-algebras against which the conditionings are performed are encoded through the measurable map f along which the disintegration is computed. Simple calculations make explicit how conditional expectation underpins disintegration: for all $h : (X, p) \to \mathbb{R}$ integrable,

$$\begin{array}{rcll} \int_X h\, dp & = & \int_{y\in Y} \left(\int_X h\, df^\dagger(y)\right) dq & \text{(Equation 1)} \\ & = & \int_{y\in Y} \left(\int_{f^{-1}(y)} h\, df^\dagger(y)\right) dq & \text{(Theorem 6)} \end{array}$$

The last equation corresponds to the usual measure-theoretic characteristic identity of disintegrations. Let us consider a measurable set $B \in \sigma(f)$ and let us apply this identity to the function $h \cdot \mathbb{1}_B$. Applying a change of variables on $q = \mathsf{G}(f)(p)$, we get:

$$\begin{array}{rcll} \int_B h\, dp & = & \int_{x\in X} \left(\int_{f^{-1}(f(x))} h \cdot \mathbb{1}_B\, df^\dagger(f(x))\right) dp & \text{(Change of variables)} \\ & = & \int_{x\in B} \left(\int_{f^{-1}(f(x))} h\, df^\dagger(f(x))\right) dp & (\mathbb{1}_B \text{ constant on fibers}) \end{array}$$

We recognise the characteristic identity of conditional expectations (Definition 2). This implies that the following identity holds p-almost everywhere:

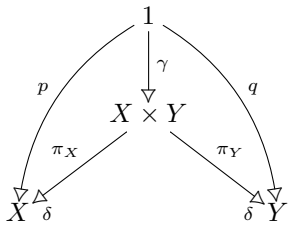
$$\mathbb{E}\left[h \mid \sigma(f)\right] = x \mapsto \int_{f^{-1}(f(x))} h\, df^\dagger(f(x)) \tag{2}$$

3.3 Bayesian inversion

Bayesian inversion is a reformulation of the disintegration theorem where the map f is allowed to be any arrow of **Krn** (and not just a deterministic one). To formulate our *Bayesian inversion theorem* we will define two **Set**-valued functors and two natural transformations between them. The first functor is simply the functor $\mathsf{Hom}((X, p), -) : \mathbf{Krn} \to \mathbf{Set}$ for a given (X, p) in **Krn**. For notational clarity, we will abbreviate objects $(X, p), (Y, q), (Y', q')$ in **Krn** to X, Y, Y' with the understanding that they come equipped with measures p, q, q'. It is useful to explicitly write the action of $\mathsf{Hom}(X, -)$ on morphisms $g : Y \nrightarrow Y'$. By definition $\mathsf{Hom}(X, -)(g) : \mathsf{Hom}(X, Y) \to \mathsf{Hom}(X, Y')$ maps $f \in \mathsf{Hom}(X, Y)$ to the kernel:

$$\mathsf{Hom}(X, -)(g)(f) \triangleq g \circ_{\mathbf{G}} f : X \nrightarrow Y' \text{ defined by } (g \circ_{\mathbf{G}} f)(x)(B_{Y'}) = \int_{y\in Y} g(y)(B_{Y'})\, df(x)$$

Given X in **Krn**, our second functor $\Gamma(X, -) : \mathbf{Krn} \to \mathbf{Set}$ is defined on objects as follows: we define $\Gamma(X, Y) \subseteq \mathsf{G}(X \times Y)$ to be the set of *couplings* of p and q, corresponding to measures γ such that $\mathsf{G}(\pi_X)(\gamma) = p$ and $\mathsf{G}(\pi_Y)(\gamma) = q$. Couplings corresponds to elements γ such that the following diagram commutes in $\mathcal{K}\ell$:



(3)

On morphisms $g : Y \nrightarrow Y'$, $\Gamma(X, g)$ is defined as the map $\Gamma(X, g) = \otimes \circ_{\mathbf{G}} (\delta \times g) \circ_{\mathbf{G}} -$ such that $\Gamma(X, g)(\gamma)$ is the composite

$$1 \xrightarrow{\gamma} X \times Y \xrightarrow{\otimes\circ(\delta_X \times g)} X \times Y'$$

where $\otimes : \mathsf{G}X \times \mathsf{G}Y' \to \mathsf{G}(X \times Y')$ is the product measure bifunctor and δ_X is the Giry unit at X. By unravelling the definitions we get

$$\Gamma(X,g)(\gamma)(B_X \times B_{Y'}) = \int_{(x,y)\in X\times Y} \delta_X(x)(B_X) \cdot g(y)(B_{Y'})\, d\gamma$$

The proof that $\Gamma(X,-)$ commutes with composition follows from the disintegration theorem. We now define a transformation $\alpha^X : \mathsf{Hom}(X,-) \to \Gamma(X,-)$ defined at Y by

$$\alpha^X_Y(f)(B_X \times B_Y) = \int_{x\in B_X} f(x)(B_Y)\, dp$$

▶ **Proposition 7.** *α^X is natural*

Proof. Let $g : (Y,q) \to (Y',q')$, we calculate

$$\begin{aligned}
\Gamma(X,g)(\alpha^X_Y(f))(B_X \times B_{Y'}) &\overset{(1)}{=} \int_{(x,y)\in X\times Y} \delta_X(x)(B_X) g(y)(B_{Y'})\, d\alpha^X_Y(f) \\
&\overset{(2)}{=} \int_{x\in X} \delta_X(x)(B_X) \left(\int_{y\in Y} g(y)(B_{Y'}) df(x) \right) dp \\
&= \int_{x\in B_X} \left(\int_{y\in Y} g(y)(B_{Y'})\, df(x) \right) dp \\
&\overset{(3)}{=} \alpha^X_{Y'}(\mathsf{Hom}(X,g)(f))(B_X \times B_{Y'})
\end{aligned}$$

where (1) follows from the definition of Γ, (2) follows from the fact that α^X_Y constructs a coupling in an explicitly disintegrated form, and (3) follows from the definition of α^X and $\mathsf{Hom}(X,-)$. ◀

Our second natural transformation goes in the opposite direction and is given by the disintegration along the first projection (which exists by Theorem 6), i.e. we define $D^X : \Gamma(X,-) \to \mathsf{Hom}(X,-)$ at Y in **Krn** by:

$$D^X_Y(\gamma) = \mathsf{G}(\pi_Y) \circ \pi_X^\dagger, \text{ such that } \gamma(B_X \times B_Y) = \int_{B_X} D^X_Y(\gamma)(x)(B_Y) dp(dx)$$

▶ **Proposition 8.** *D^X is natural.*

Proof. Let $g : (Y,q) \to (Y',q')$ and $\gamma \in \Gamma(X,Y)$. For notational clarity, for any $h \in \mathsf{Hom}(X,Y)$ let us write $\tilde{h} = \mathsf{Hom}(X,-)(g)(h)$, and let us define $f \triangleq D^X_Y(\gamma)$. We now calculate:

$$\begin{aligned}
\Gamma(X,g)(\gamma)(B_X \times B_{Y'}) &\overset{(1)}{=} \int_{(x,y)\in X\times Y} \delta_X(x)(B_X) g(y)(B_{Y'})\, d\gamma \\
&\overset{(2)}{=} \int_{x\in X} \delta_X(x)(B_X) \left(\int_{y\in Y} g(y)(B_{Y'})\, df(x) \right) dp \\
&= \int_{x\in B_X} \left(\int_{y\in Y} g(y)(B_{Y'})\, df(x) \right) dp \\
&\overset{(3)}{=} \int_{x\in B_X} \tilde{f}(x)(B_{Y'})\, dp
\end{aligned}$$

where (1) follows by definition of $\Gamma(X,-)$, (2) is by definition of f and the Disintegration Theorem 6, and (3) by definition of $\mathsf{Hom}(X,-)$. It follows immediately that $\tilde{f}$ factors through the disintegration of $\Gamma(X,g)(\gamma)$ along the first projection, i.e. that

$$D^X_{Y'}(\Gamma(X,g)(\gamma)) = \mathsf{Hom}(X,g)(D^X_Y(\gamma))$$

◀

▶ **Theorem 9** (Bayesian Inversion Theorem). *There exists a bijection:*

$$-^{\dagger} : \mathsf{Hom}_{\mathbf{Krn}}((X,p),(Y,q)) \to \mathsf{Hom}_{\mathbf{Krn}}((Y,q),(X,p))$$

Proof. It is immediate from the definitions above that $\alpha^X_Y : \mathsf{Hom}(X,Y) \to \Gamma(X,Y)$ and $D^X_Y : \Gamma(X,Y) \to \mathsf{Hom}(X,Y)$ are inverse of one another, and thus bijective. Moreover, it is also clear that the permutation map $\mathsf{G}(\pi_2 \times \pi_1) : \Gamma(X,Y) \to \Gamma(Y,X)$ being its own inverse is a bijection. It follows that $\mathsf{Hom}(X,Y) \simeq \Gamma(X,Y) \simeq \Gamma(Y,X) \simeq \mathsf{Hom}(Y,X)$. And we can set $-^{\dagger} = D^Y_X \circ \mathsf{G}(\pi_2 \times \pi_1) \circ \alpha^X_Y$. ◀

4 ω-complete normed cones

We recall some facts pertaining to the categories of ω-complete normed cones introduced in [14]. The main object of this section is to give a functional analytic account of kernels as operators on ω-complete normed cones, in the style of [3]. We improve on the latter by presenting the transformation from kernels to operators functorially. In Section 5, we will use the machinery developed here to interpret Bayesian inversion in this domain-theoretic setting. We first recall some general definitions about ω-complete normed cones and the associated category $\omega\mathbf{CC}$. We then concentrate on the duality existing between the subcategories of cones of integrable and bounded functions. Proofs not provided here can be found in Ref. [3].

4.1 Basic definitions

Cones are axiomatisations of the positive elements of (real) vector spaces.

▶ **Definition 10** (Cones). A *cone* $(V,+,\cdot,0)$ is a set V together with an associative and commutative operation $+$ with unit 0 and with a multiplication by real positive scalars $\cdot$ distributive over $+$. We have two more axioms: the *cancellation law* $\forall u,v,w \in V, v+u = w+u \Rightarrow v = w$ and the *strictness* $\forall v,w \in V, v+w = 0 \Rightarrow v = w = 0$.

Any cone C admits a natural partial order structure $\leq$ defined as follows: $u \leq v$ if and only if there exists w such that $v = u + w$. We will consider normed cones which are complete with respect to increasing sequences (*chains*) in this order which are of bounded norm.

▶ **Definition 11** (Normed cones, ω-complete). A *normed cone* C is a cone together with a function $\|-\| : C \to \mathbb{R}_+$ satisfying (i) $\forall v \in C, \|v\| = 0 \Leftrightarrow v = 0$; (ii) $\forall r \in \mathbb{R}_+, v \in C, \|r \cdot v\| = r\,\|v\|$; (iii) $\forall u,v \in C, \|u+v\| \leq \|u\| + \|v\|$; (iv) $u \leq v \Rightarrow \|u\| \leq \|v\|$. A cone is *$\omega$-complete* if (i) for all chain $(u_n)_{n\in\mathbb{N}}$ such that $\{\|u_n\|\}_{n\in\mathbb{N}}$ is bounded, there exists a least upper bound (lub) $\bigvee_n u_n$ and (ii) $\|\bigvee_n u_n\| = \bigvee_n \|u_n\|$.

Note that the norm $\|-\|$ is ω-continuous. All the cones we are going to consider in the following are ω-complete and normed. ω-continuous linear maps form the natural notion of morphism between such structures. Note that linearity implies monotonicity in the natural order.

▶ **Definition 12** (ω-continuous linear maps). For C, D ω-complete normed cones, an ω-continuous linear map $f : C \to D$ is a linear map such that for every chain $(u_n)_{n\in\mathbb{N}}$ for which $\bigvee_n u_n$ exists, $\bigvee_n f(u_n)$ exists and is equal to $f(\bigvee_n u_n)$.

The dual of an ω-complete normed cone is defined in the usual way. We will admit the following result:

▶ **Proposition 13** (ω-complete dual). *If C is an ω-complete normed cone, the cone of ω-continuous linear maps $\{\phi : C \to \mathbb{R}_+\}$ with the norm $\|\phi\| = \inf_v \{c \geq 0 \mid |\phi(v)| \leq c\,\|v\|\}$ is ω-complete.*

ω-complete normed cones and ω-continuous linear maps form a category denoted by $\omega\mathbf{CC}$. The dual operation gives rise to a contravariant endofunctor $-^* : \omega\mathbf{CC} \to \omega\mathbf{CC}$. If $f : C \to D$ is an $\omega\mathbf{CC}$ arrow, $f^* : D^* \to C^*$ is defined by $f^*(\phi) = \phi \circ f$. For all $\phi \in D^*$ and $x \in C$, one has

$$\begin{array}{rcll} \|f^*(\phi)(x)\| & = & \|(\phi \circ f)(x)\| & \\ & \leq & \|\phi\|\,\|f(x)\| & (\phi\ \omega\text{-continuous}) \\ & \leq & \|\phi\|\,\|f\|\,\|x\| & (f\ \omega\text{-continuous}) \end{array}$$

Therefore, $\|f^*\| \leq \|f\|$. We now introduce the cones we are going to work with in the remainder of the paper.

4.2 Cones of measures and of measurable functions

Let us fix a measure space (X, Σ, p) with p finite. Much of the constructions in the rest of the paper rely on dualities between the cones of measurable functions $L_1^+(X, \Sigma, p), L_\infty^+(X, \Sigma, p)$ and cones of measures $\mathcal{M}^{\ll p}(X, \Sigma), \mathcal{M}^p_{UB}(X, \Sigma)$. Let us introduce these cones in more detail.

Cones of measurable functions

Two positive measurable maps $f, f' : (X, \Sigma, p) \to \mathbb{R}_+$ are said to be p-equivalent if $p\{x \mid f(x) \neq f'(x)\} = 0$. Such as map $f : (X, \Sigma) \to \mathbb{R}$ is p-integrable if $\int_X f\,dp < \infty$. Clearly, being p-integrable is preserved by p-equivalence. The elements of the cone $L_1^+(X, \Sigma, p)$ are p-equivalence classes of real-valued integrable maps. $L_1^+(X, \Sigma, p)$ is normed by $\|f\|_1 = \int_X f\,dp$. The dominated convergence theorem implies that $L_1^+(X, \Sigma, p)$ is an ω-complete normed cone.

A positive measurable map f is p-essentially bounded if there exists $C \geq 0$ such that $p\{x \mid f(x) > C\} = 0$. The elements of the cone $L_\infty^+(X, \Sigma, p)$ are p-equivalence classes of real-valued essentially bounded maps. The norm is given by $\|f\|_\infty = \inf\{C \geq 0 \mid f(x) \leq C\ p\text{-a.s.}\}$.

Cones of measures

Closely related to the cones above are cones of absolutely continuous measures $\mathcal{M}^{\ll p}(X, \Sigma)$ and bounded measures $\mathcal{M}^p_{UB}(X, \Sigma)$. $\mathcal{M}^{\ll p}(X, \Sigma)$ is the cone of finite measures which are absolutely continuous with respect to p, with norm given by $\|q\|_{\ll} = q(X)$. $\mathcal{M}^p_{UB}(X, \Sigma)$ is the cone of finite measures which are uniformly bounded by a finite multiple of p, with norm given by $\|q\|_{UB} = \inf\{c \geq 0 \mid q \leq cp\}$. The ω-completeness of these cones will appear as a byproduct of the duality to be proved in the next.

4.3 Duality between L_1 and L_∞ cones

In the following, we will denote by $\mathbf{L}_1^+$ the full subcategory of $\omega\mathbf{CC}$ having as objects cones $L_1^+(X, \Sigma, p)$ and by $\mathbf{L}_\infty^+$ the full subcategory of $\omega\mathbf{CC}$ having as objects cones $L_\infty^+(X, \Sigma, p)$. As indicated before, the construction of the duality goes through cones of absolutely continuous measures $\mathcal{M}^{\ll p}(X, \Sigma)$ and bounded measures $\mathcal{M}^p_{UB}(X, \Sigma)$.

▶ **Theorem 14.** *$\mathcal{M}^{\ll p}(X, \Sigma)$ is isometrically isomorphic to $L_1^+(X, \Sigma, p)$ and $\mathcal{M}^p_{UB}(X, \Sigma)$ is isometrically isomorphic to $L_\infty^+(X, \Sigma, p)$.*

Proof. The Radon-Nikodym derivative induces a map $q \in \mathcal{M}^{\ll p}(X,\Sigma) \mapsto \frac{dq}{dp}$ which is linear, injective and norm-preserving, as is its inverse map $u \in L_1^+(X,\Sigma,p) \mapsto u \cdot p$. Therefore, $\mathcal{M}^{\ll p}(X,\Sigma)$ is isometrically isomorphic to $L_1^+(X,\Sigma,p)$. The isomorphism between $\mathcal{M}^p_{UB}(X,\Sigma)$ and $L_\infty^+(X,\Sigma,p)$ is proved similarly. We only show that $\frac{d-}{dp}$ is norm preserving. For $q \in \mathcal{M}^p_{UB}(X,\Sigma)$, we have $\left\|\frac{dq}{dp}\right\|_\infty = \inf\left\{C \geq 0 \mid \frac{dq}{dp} \leq C \; p\text{-a.s.}\right\}$. If $\frac{dq}{dp} \leq C$ then $q \leq C \cdot p$. By definition, the least such C is $\|q\|_{UB}$. ◀

As a consequence, both cones of finite measures are ω-complete. We are now in position to state the following variant of the Riesz representation theorem for ω-complete cones:

▶ **Theorem 15.** *We have the following isometric isomorphisms:* $L_1^{+,*}(X,\Sigma,p) \cong \mathcal{M}^p_{UB}(X,\Sigma)$ *and* $L_\infty^{+,*}(X,\Sigma,p) \cong \mathcal{M}^{\ll p}(X,\Sigma)$.

We will only prove the first isomorphism, the second one being proved similarly.

Proof. We construct an isometric isomorphism $\iota : L_1^{+,*}(X,\Sigma,p) \to \mathcal{M}^p_{UB}(X,\Sigma)$. Let us set $\iota(\phi) = B \mapsto \phi(\mathbb{1}_B)$. It is clearly linear. Let us show that $\iota(\phi)$ is countably additive. For $(B_n)_{n\in\mathbb{N}}$ a countable family of pairwise disjoint subsets, we have $\iota(\phi)(\cup_{n\in\mathbb{N}} B_n) = \phi(\mathbb{1}_{\cup_{n\in\mathbb{N}} B_n})$. Clearly, the family $\lambda_k = \sum_{i=1}^k \mathbb{1}_{B_n}$ is increasing, measurable and is bounded by $\mathbb{1}_X$. Therefore $\bigvee \lambda_k = \sum_n \mathbb{1}_{B_n}$ exists and by ω-continuity of ϕ, $\phi(\mathbb{1}_{\cup_{n\in\mathbb{N}} B_n}) = \sum_n \phi(\mathbb{1}_{B_n})$. ι is injective: let $v \in L_1^+(X,\Sigma,p)$ s.t. $\phi(v) \neq \phi'(v)$. v, being integrable can be approximated by an increasing sequence of simple functions. We deduce there must exists B such that $\phi(\mathbb{1}_B) \neq \phi'(\mathbb{1}_B)$. Let us prove that $\iota(\phi) \in \mathcal{M}^p_{UB}(X,\Sigma)$. A monotone convergence argument shows that for all $v \in L_1^+(X,\Sigma,p)$, we have the identity $\phi(v) = \int_X v \, d\iota(\phi) < \infty$. We also have $p(B) = 0 \Rightarrow \iota(\phi)(B) = 0$ hence $\iota(\phi) \ll p$. We deduce from the two previous facts that $\frac{d\iota(\phi)}{dp} \in L_\infty^+(X,\Sigma,q)$, which by Theorem 14 implies that $\iota(\phi) \in \mathcal{M}^p_{UB}(X,\Sigma)$. It remains to prove that ι is surjective: for $q \in \mathcal{M}^p_{UB}(X,\Sigma)$, we have trivially for $\phi_q = v \mapsto \int_X v \, dq$ that $\iota(\phi_q) = q$. ◀

An immediate consequence is that $L_1^{+,*}(X,\Sigma,p) \cong L_\infty^+(X,\Sigma,p)$ and reciprocally. This is notoriously false in the case of general Banach spaces. What makes everything work here is that we restrict to ω-continuous linear functionals.

Theorem 15 gives rise to a *pairing* between the spaces of integrable and bounded functions.

▶ **Definition 16** (Pairing). The *pairing* of $L_\infty^+(X,\Sigma,p)$ and $L_1^+(X,\Sigma,p)$ is a bilinear map $\langle\cdot,\cdot\rangle_X : L_\infty^+(X,\Sigma,p) \times L_1^+(X,\Sigma,p) \to \mathbb{R}$ given by $\langle u,v\rangle_X = \int_X u \cdot v \, dp$. It is continuous and ω-continuous in both arguments.

This pairing gives rise to a notion of *adjoint*:

▶ **Proposition 17** (Adjoints). *The duality functor* $-^* : \omega\mathbf{CC} \to \omega\mathbf{CC}$ *restricts to a contravariant functor* $-^\dagger : \mathbf{L}_1^+ \to \mathbf{L}_\infty^+$ *such that for all* $\mathbf{L}_1^+$ *arrow* $A : L_1^+(X,\Sigma,p) \to L_1^+(Y,\Lambda,q)$, $A^\dagger : L_\infty^+(Y,\Lambda,q) \to L_\infty^+(X,\Sigma,p)$ *is the unique adjoint arrow such that for all* $u \in L_1^+(X,\Sigma,p), v \in L_\infty^+(Y,\Lambda,q)$, $\langle v, Au\rangle_Y = \langle A^\dagger v, u\rangle_X$.

Conversely, for all $A : L_\infty^+(Y,\Lambda,q) \to L_\infty^+(X,\Sigma,p)$ *there is a unique adjoint* ${}^\dagger A : L_1^+(X,\Sigma,p) \to L_1^+(Y,\Lambda,q)$ *such that the equation above is verified.*

We will not prove Proposition 17 here but simply sketch how adjoints are constructed. Given $A : L_1^+(X,\Sigma,p) \to L_1^+(Y,\Lambda,q)$ and using that $L_\infty^+(X,\Sigma,p) \cong L_1^{+,*}(X,\Sigma,p)$ we set:

$$A^\dagger = v \in L_\infty^+(Y,\Lambda,q) \mapsto \left(u \in L_1^+(X,\Sigma,p) \mapsto \langle v, Au\rangle\right) \tag{4}$$

Note that $A^\dagger(v)$ is written in dual form. The adjoint of ${}^\dagger A : L_\infty^+(Y,\Lambda,q) \to L_\infty^+(X,\Sigma,p)$ is defined similarly. Finally, observe that setting Y to be the one point space in Proposition 17 implies Theorem 15.

4.4 Operator interpretations of kernels

In [3], it is shown that kernels that respect a condition of "non-singularity" correspond to particular linear operators between cones. This view on kernels allows to leverage the language of functional analysis to approximate and reason on these objects. We improve on this by showing that the functional interpretation of a non-singular kernel corresponds to functors defined on $\mathbf{Krn}$ and valued in $\mathbf{L}_1^+$ and $\mathbf{L}_\infty^+$. These new developments will be put to use in Section 5 where we will show that Bayesian inversion (Theorem 9) maps through this functorial correspondence to the adjunction of Proposition 17.

We introduce contravariant functors $\mathsf{T}_\infty : \mathbf{Krn} \to \mathbf{L}_\infty^+$ and $\mathsf{T}_1 : \mathbf{Krn} \to \mathbf{L}_1^+$ mapping kernels into norm 1 ω-continuous operators. T_∞ is defined on objects (X,p) by $\mathsf{T}_\infty(X,p) = L_\infty^+(X,p)$, while $\mathsf{T}_1(X,p) = L_1^+(X,p)$. For $f : (X,p) \to\!\!\!\!\bullet\ (Y,q)$ a $\mathbf{Krn}$ arrow, we set

$$\mathsf{T}_\infty(f) = v \in L_\infty^+(Y,q) \mapsto \left(x \mapsto \int_Y v \, df(x) \right). \tag{5}$$

Note that this is well-defined, as $f(x)$ is p-a.s. absolutely continuous with respect to q (Lemma 3). $\mathsf{T}_1(f)$ is defined similarly but acts on $L_1^+(Y,q)$.

▶ **Proposition 18.** *T_∞ and T_1 are functors $\mathsf{T}_\infty : \mathbf{Krn} \to \mathbf{L}_\infty^+$ and $\mathsf{T}_1 : \mathbf{Krn} \to \mathbf{L}_1^+$ ranging in operators of norm 1.*

Proof. We first check that T_∞ and T_1 are well-typed. Let us fix $f : (X,p) \to\!\!\!\!\bullet\ (Y,q)$ in $\mathbf{Krn}$. $\mathsf{T}_\infty(f)$ and $\mathsf{T}_1(f)$ as defined above are clearly linear. We start with T_1. We have for all $v \in L_1^+(Y,q)$ that

$$\|\mathsf{T}_1(f)(v)\|_1 = \int_X \left(\int_Y v \, df(x) \right) dp = \int_Y v \, d(\mu_Y \circ \mathsf{G}(f))(p) = \int_Y v \, dq = \|v\|_1$$

Therefore $\|\mathsf{T}_1(f)\| = 1$. Prop. 5.2 of [3] ensures that $\mathsf{T}_1(f)$ is ω-continuous. Let us treat the case of T_∞. For all $v \in L_\infty^+(Y,q)$, the inequality $\int_Y v \, df(x) \leq \|v\|_\infty$ is verified. This implies

$$\|\mathsf{T}_\infty(f)(v)\|_\infty = \inf \left\{ C \geq 0 \mid \int_Y v \, df(x) \leq C \; p\text{-a.s.} \right\} \leq \|v\|_\infty$$

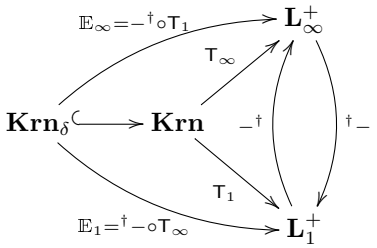
therefore $\|\mathsf{T}_\infty(f)\| \leq 1$. The upper bound is reached for $v = \mathbb{1}_Y$, therefore $\|\mathsf{T}_\infty(f)\| = 1$. ω-continuity of $\mathsf{T}_\infty(f)$ follows from the dominated convergence theorem. In the following, both T_1 and T_∞ are denoted by T. If we denote by $id' = \delta \circ id$ an identity in $\mathcal{K}\ell$, $\mathsf{T}(id')(v) = v$. Let $f : (X,p) \to\!\!\!\!\bullet\ (Y,q)$, $g : (Y,q) \to\!\!\!\!\bullet\ (Z,r)$ be two arrows in $\mathbf{Krn}$. By definition of T and using naturality of μ, we get:

$$\mathsf{T}(g \circ f)(v)(x) = \int_Z v \, d(g \circ_M f(x)) = \int_{y \in Y} \left(\int_Z v \, dg(y) \right) df(x) = (\mathsf{T}(f)\mathsf{T}(g))(v)(x)$$

Therefore, T is a well-defined functor. ◀

We will call operators in the range of T_∞ and T_1 *abstract Markov kernels*. Postcomposing T_∞ with ${}^\dagger -$ (or T_1 with $-^\dagger$) yields the covariant ("forward") interpretation of $\mathbf{Krn}$ arrows, called *Markov operators*. The restriction of T_∞ to $\mathbf{Krn}_\delta$ is familiar, as it reduces in this case to the precomposition functor $P_\infty : \mathbf{Krn}_\delta \to \mathbf{L}_\infty^+$ acting on $\mathbf{Krn}_\delta$ (hence deterministic) arrows $f : (X,p) \to_\delta (Y,q)$ by $P_\infty(f) = v \mapsto v \circ f$. Postcomposing P_∞ with ${}^\dagger -$ yields the conditional expectation functor $\mathbb{E}_1 : \mathbf{L}_1^+ \to \mathbf{L}_1^+$. Similarly, functors P_1 and $\mathbb{E}_\infty$ can are constructed by considering T_1 instead of T_∞. We invite the reader to compare our developments with

Section 3 and 4 of [3] for more details. We sum up the developments so far in the following (non-commuting) diagram:



(6)

We conclude this section by indicating that the duality between $\mathbf{L}^+_1$ and $\mathbf{L}^+_\infty$ generalises to arbitrary dual pairs $\mathbf{L}^+_p, \mathbf{L}^+_q$ [2]. We conjecture that the functors $\mathsf{T}_1, \mathsf{T}_\infty$ have counterparts in this more general setting. In the next section, we give a functional interpretation of Bayesian inversion through these functors.

5 Bayesian inversion as duality

As shown in Section 3, Bayesian inversion is a symmetrised disintegration, which by Equation 2 corresponds to a measurable family of conditional probabilities. As these are a fundamental tool of the modern probabilistic toolkit, a natural question is to find a corresponding process in the functional analytic setting of norm-1 operators between the ω-complete cones L^+_1 and L^+_∞. It is well-known that conditional expectation can be framed as a projection operator (e.g. in the L_2 case, see [10]). The result we are about to prove provides a fresh perspective on this classical problem: the Bayesian inverse of a kernel corresponds to the adjoint of its functional form.

▶ **Theorem 19** (Inversion as duality). *Let X, Y be standard Borel spaces and $f : (X, p) \nrightarrow (Y, q)$ a* **Krn** *arrow. We have:*

$$^\dagger\mathsf{T}_\infty(f) = \mathsf{T}_1(f^\dagger)$$

I.e. $\mathsf{T}_\infty(f)$ is adjoint to $\mathsf{T}_1(f^\dagger)$.

Proof. Let us recall the types of the objects:

$$\begin{array}{rcl} \mathsf{T}_\infty(f) & : & L^+_\infty(Y,q) \to L^+_\infty(X,p) \\ {}^\dagger\mathsf{T}_\infty(f) & : & L^+_1(X,p) \to L^+_1(Y,q) \end{array}$$

It is enough to prove that for all $u \in L^+_1(Y,q)$, for all $v \in L^+_\infty(X,p)$,

$$\langle v, \mathsf{T}_\infty(f)(u)\rangle_X = \langle \mathsf{T}_1(f^\dagger)(v), u\rangle_Y$$

Unfolding, we must prove:

$$\begin{array}{rcl} & \int_X v \mathsf{T}_\infty(f)(u) dp & = & \int_Y \mathsf{T}_1(f^\dagger)(v) u dq \\ \Leftrightarrow & \int_{x\in X} v(x) \left(\int_Y u df(x)\right) dp & = & \int_{y\in Y} \left(\int_X v\, df^\dagger(y)\right) u(y) dq \\ \Leftrightarrow & \int_{x\in X} \left(\int_{y\in Y} v(x)u(y) df(x)\right) dp & = & \int_{y\in Y} \left(\int_{x\in X} v(x)u(y) df^\dagger(y)\right) dq \end{array}$$

Let $\gamma \in \Gamma(p,q)$ be the coupling corresponding to $f : (X,p) \nrightarrow (Y,q)$. Continuing the string of equivalences above, we must prove:

$$\int_{x\in X} \left(\int_{y\in Y} v(x)u(y) df(x)\right) d\mathsf{G}(\pi_X)(\gamma) = \int_{y\in Y} \left(\int_{x\in X} v(x)u(y) df^\dagger(y)\right) d\mathsf{G}(\pi_Y)(\gamma)$$

Now, recall that by Equation 1, the disintegration theorem gives us for all integrable $e : (X \times Y, \gamma) \to \mathbb{R}$ that

$$\begin{array}{rcll} \int_{X\times Y} e d\gamma & = & \int_{x\in X} \left(\int_{\pi_X^{-1}(x)} e \, df(x) \right) d\mathsf{G}(\pi_X)(\gamma) & \text{(disintegrating } \gamma \text{ along } \pi_X) \\ & \overset{1}{=} & \int_{x\in X} \left(\int_Y e(x,\_) \, df(x) \right) d\mathsf{G}(\pi_X)(\gamma) & \text{(disintegrations live on the fiber)} \\ & = & \int_{y\in Y} \left(\int_{\pi_Y^{-1}(y)} e \, df^\dagger(y) \right) d\mathsf{G}(\pi_Y)(\gamma) & \text{(disintegrating } \gamma \text{ along } \pi_Y) \\ & \overset{2}{=} & \int_{y\in Y} \left(\int_X e(\_,y) \, df^\dagger(y) \right) d\mathsf{G}(\pi_Y)(\gamma) & \text{(disintegrations live on the fiber)} \end{array}$$

Taking $e(x,y) = v(x)u(y)$ and using equations marked 1 and 2 above concludes the proof. ◀

Some comments are in order. Note that the disintegration and Bayesian inversion theorems rely on some strong assumptions on the underlying spaces–here, we assume the spaces to be standard Borel; Culbertson & Sturtz [4] work in the setting of perfect measure spaces and equiperfect kernels. However, the cone duality works for any measure space! We conjecture that these regularity conditions are necessary if one wishes to extract a measurable kernel from a Markov operator or dually from an abstract Markov kernel.

6 Conclusion

We have established that the functional representation of measurable kernels as operators acting on ω-complete cones presented in [3] is functorial. Two variants of the functor exist, mapping kernels to operators acting either on bounded functions or on integrable ones. The category of 'typed' kernels on which these functors are defined allows to state elegantly the famous disintegration theorem and its generalisation, Bayesian inversion. What's more, we uncovered the categorical underpinnings of Bayesian inversion as particular natural transformations mapping kernels to couplings and reciprocally. Finally, we have shown that Bayesian inversion amounts in the functional world to adjunction.

Several further developments suggest themselves. First of all, It remains to be seen whether our construction generalises from the duality L_1^+/L_∞^+ to arbitrary pair of dual cones L_p^+/L_q^+ (e.g. the pair $p = q = 2$ which allows one to talk about reversible kernels), and can prove a stronger statement, namely $\mathsf{T}_p(f^\dagger) = \mathsf{T}_q(f)^\dagger$ for *all* conjugate exponents p, q. Another line of thought is to connect these results with some of the authors recent's work [7, 5]. In particular, instantiating the kernel-theoretic framework with the Dirichlet process [7], might provide insight into the operator-theoretic counterpart of so-called nonparametric methods in Bayesian learning. On a different note, this process has the type of a *natural* and *continuous* kernel and admits a convenient finitary characterisation. We are eager to study how these properties map through the operator interpretation.

References

1 C. Aliprantis and K. Border. *Infinite dimensional analysis*, volume 32006. Springer, 1999.

2 C. Badescu and P. Panangaden. Abstract Markov processes with L_p spaces. Private communication, 2012.

3 P. Chaput, V. Danos, P. Panangaden, and G. Plotkin. Approximating Markov Processes by averaging. *Journal of the ACM*, 61(1), January 2014. 45 pages.

4 J. Culbertson and K. Sturtz. A categorical foundation for Bayesian probability. *Applied Categorical Structures*, pages 1–16, 2012.

5 F. Dahlqvist, V. Danos, and I. Garnier. Giry and the machine. Feb 2016. To appear in the proceedings of MFPS XXXII.

6 V. Danos, J. Desharnais, F. Laviolette, and P. Panangaden. Bisimulation and cocongruence for probabilistic systems. *Information and Computation*, 204(4):503–523, 2006.

7 V. Danos and I. Garnier. Dirichlet is natural. *Electronic Notes in Theoretical Computer Science*, 319:137 – 164, 2015. MFPS XXXI.

8 C. Dellacherie and P.A. Meyer. *Probabilities and Potential, C: Potential Theory for Discrete and Continuous Semigroups*. North-Holland Mathematics Studies. Elsevier Science, 2011.

9 M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, number 915 in Lecture Notes In Math., pages 68–85. Springer-Verlag, 1981.

10 O. Kallenberg. *Foundations of Modern Probability*. Springer, 1997.

11 A. S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Text in Mathematics*. Springer, 1995.

12 D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

13 S. Mac Lane. *Categories for the Working Mathematician*, volume 111 of *Graduate Texts in Mathematics*. Springer, 1998.

14 P. Selinger. Towards a semantics for higher-order quantum computation. In *Proceedings of the 2nd International Workshop on Quantum Programming Languages, TUCS General Publication*, volume 33, pages 127–143, 2004.

Ethical Preference-Based Decision Support Systems*

Francesca Rossi

IBM T.J. Watson Research Center
(on leave from the University of Padova, Italy)
frossi@it.ibm.com

Abstract

The future will see autonomous intelligent systems acting in the same environment as humans, in areas as diverse as driving, assistive technology, and health care. Think of self-driving cars, companion robots, and medical diagnosis support systems. Also, humans and machines will often need to work together and agree on common decisions. Thus hybrid collective decision making systems will be in great need. In these scenarios, both machines and collective decision making systems should follow some form of moral values and ethical principles (appropriate to where they will act but always aligned to humans'). In fact, humans would accept and trust more machines that behave as ethically as other humans in the same environment. Also, these principles would make it easier for machines to determine their actions and explain their behavior in terms understandable by humans. Moreover, often machines and humans will need to make decisions together, either through consensus or by reaching a compromise. This would be facilitated by shared moral values and ethical principles. In this paper we introduce some issues in embedding morality into intelligent systems. A few research questions are defined, with the hope that the discussion raised by the questions will shed some light onto the possible answers.

1998 ACM Subject Classification I.2 Artificial Intelligence

Keywords and phrases preferences, decision making, multi-agent systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.2

Category Invited Paper

1 Motivation and Introduction

How do humans or machines make a decision? Whenever we make a decision, we consider our preferences over the possible options. Also, in a social context, collective decisions are made by aggregating the preferences of the individuals. AI systems that support individual and collective decision making have been studied for a long time, and several preference modelling and reasoning frameworks have been defined and exploited in order to provide rationality to the decision process and its result.

However, little effort has been devoted to understand whether this decision process, or its result, is ethical or moral. Rationality does not imply morality. How can we embed morality into a decision process? And how do we ensure that the decision we make, as an individual or a collectivity of individuals, are moral? In other words, how do we pass from the individuals' personal preferences to moral behaviour and decision making?

* This work is partially supported by the project "Safety constraints and ethical principles in collective decision making systems" funded by the Future of Life Institute.

When we pass from humans to AI systems, the task of modelling and embedding morality and ethical principles is even more vague and elusive. Are the existing ethical theories applicable also to AI systems? On one hand, things seem easier since we can narrow the scope of an AI system, so that the contextual information can help us in define the correct moral values it should work according to. However, it is not clear what moral values we should embed in the system, nor how to embed them. Should we code them in a set of rules, or should we let the system learn the values by observing us humans?

Preferences and ethical theories are not that different in one respect: they both define priorities over actions. So, can we use existing preference formalisms to also model ethical theories? We discuss how to exploit and adapt current preference formalisms in order to model morality and ethics theories, as well as the dynamic integration of moral code into personal preferences. We also discuss the use of meta-preferences, since morality seems to need a way to judge preferences according to their morality level.

It is imperative that we build intelligent systems which behave morally. To work and live with us, we need to trust such systems, and this requires that we are "reasonably" sure that it behaves morally, according to values that are aligned to the human ones. Otherwise, we would not let a robot take care of our elderly people or our kids, nor a car to drive for us, nor we would listen to a decision support system in any healthcare scenario. Of course the word "reasonable" makes sense when the application domain does not include critical situations (like suggesting a friend on a social media or a movie in an online selling system). But when the AI system is helping (or replacing) humans in critical domains such as healthcare, then we need to have a guarantee that nothing morally wrong will be done.

2 Preference modelling and reasoning

Preferences have been studied for a long time in AI, both in the area of knowledge representation and in multi-agent systems. Several frameworks have been defined to model different kinds of preferences, such as qualitative (as in, e.g., "I prefer blue to red") and quantitative ones (as in, e.g., "I give 5 stars to Breakfast at Tiffany's and 2 stars to Terminator"). In general preferences are defining an ordering over a set of options. This order can be total and strict, but in practice it may have a lot of ties and incomparability.

When the set of options is very large, and each option is defined by a set of features (such as a car, which can be defined by it model, its colour, its engine, etc.), preferences can be expressed over single features of small sets of them, rather than entire options (as in, e.g., "If I buy a convertible, I prefer it to be red rather than white"). This allows for a faster and easier preference specification phase, as well as for more efficient preference elicitation. Several ways have been defined to pass from such compact ways to model preferences over features to the preference ordering over the options. However, it is possible to reason about such preferences without generating the exponentially large ordering over the options, which makes preferences reasoning tractable in some cases. Examples of framework to do this are constraints [19], soft constraints [13] and CP-nets [3].

Once an individual's preferences over the possible options are specified, we need to be able to find the most preferred option, or the next best option, or to compare two options that may be presented to us. Several algorithms to performs such tasks have been defined [4, 3].

When individuals, or AI systems, are part of a social environment and need to make collective decisions, individual's preferences are aggregated (for example via some voting rule) and an option is chosen for the whole group. Many voting rules have been defined and studied, as well as their properties [2]. Issues such as manipulation, control, bribery, as well

as properties such as fairness and unanimity have long being investigated, in order to define decision support systems that behave as desired [1, 6, 5, 22, 10, 16, 17, 7, 12, 18, 11].

3 From preferences to morality

To trust an AI system, like a companion robot or a self-driving car, we need to be reasonably sure that it behaves morally, according to values that are aligned to the human ones. Otherwise, we would not let a robot take care of our elderly people or our kids, nor a car to drive for us, nor we would listen to a decision support system in any healthcare scenario. So it is imperative that we understand how to provide AI systems with morality [14, 21, 9].

Morality and ethical behaviour are based on prioritising actions on the basis of what is morally right or wrong. Many ethical theories have been defined and studied in the psychology literature. They include the following ones:

- Consequentialism: Action consequences are evaluated interns of a scale of good and bad, and an agent should choose the action that minimise the bad and maximises the good.
- Virtue Ethics: An agent should choose actions that satisfy some pre-defined set of virtues
- Deontologism: Actions are predefined as good or bad, and an agent should choose the best action, no matter the consequences.

No matter which ethical theory one decides to use, the notion of right and wrong of course depends on the context in which humans (or machines) function, so formally an ethical theory can be defined as a function from a context to a partial ordering over actions. Indeed, usually we have a partial order over actions, since some actions could be incomparable to others. As one may notice by looking at the previous section on preferences, this is not that different from what preferences define: a partial order over possible options (of actions, or decisions in general). So it makes sense to investigate the possible use of preference frameworks in modelling and embedding morality into AI systems.

Research question 1: Are existing preference modelling and reasoning frameworks ready to be used also to model and reason with ethical principles and moral code, or we need to adapt them or invent new ones?

If we had the "moral" partial order and the "preference" partial order for each individual, one could try to merge them in some way, to obtain a "moral preference ordering". For example, two CP-nets modelling the moral and the preference orderings could be syntactically or semantically merged via operators that could give priority to the moral CP-net and let the preference one dictate the behaviour only when it is not in conflict with the moral one. The technical details have not been spelled out yet, but one could imagine several reasonable ways of doing this.

Research question 2: Given a moral and an ethical ordering over actions, how to combine them? Given such orderings in the forms of CP-nets or soft constraints, or other compact formalisms to model preferences, how to combine them? What properties should we desire about their combination?

However, knowing the preferences of an individual is already a difficult task. Elicitation and learning framework have bee proposed in order to do that in a way that is most faithful to the "real" preferences of the individual. Knowing the moral ordering of an individual is even more difficult. And this is even more so when we are in a social context, since this

may make individuals change their moral attitudes over time because of social interaction. The existing approaches to define ethical principles in AI systems range from trying to code ethical principles in the form of rules, to letting the system "learn" such principles from a (possibly supervised) observation of the behaviour of humans in similar settings. Some AI systems try to list the set of rules to use in self-driving cars to solve ethical dilemmas like the trolley problem. However, such approaches are usually not general, since it is unfeasible to foresee all possible situations in a very wide scenario. On the other hand, other approaches use, for example, inverse reinforcement learning [15] to try to learn morality from human behaviour. I personally feel that the best results could be obtained by combining these two approaches, although it is not clear yet how to do it best.

Research question 3: How to combine bottom-up learning approaches with top-down rule-based approaches in defining ethical principles for AI systems?

Research question 4: Recently, the most successful AI systems are based on statistical machine learning approaches that, by their nature, do not provide a natural way to explain or justify their decisions (or suggestions), nor they assure optimality. If we employ this approach also for embedding morality into a machine, how are we going to prove that nothing morally wrong will happen?

4 Morality by meta-preferences

As mentioned above, in a social context, individual preferences are transformed little by little by incorporating reasonable elements from the societal interaction with other members of the group. This is often called "reconciliation" of individual preferences with social reason, and takes place in the context of collective choice. To be able to describe the dynamic moving from one preference ordering over the next one (in time), and to make sure that the later preference orderings are indeed better in terms of morality, one needs to have a way to judge preferences according to some notion of good and bad (in any of the above mentioned ethical theories). Indeed, Sen [20] claims that morality requires judgement among preferences. To account for this, he introduced the notion of metaranking (that is, preferences over preferences) which enables to formalise individual preference modifications. A moral code could then be defined as ranking of preference rankings. That is, the moral code is defined by a structure that, by employing notions such as distance, is able to rank preferences according to their morality level.

The distance intrinsic in the moral code can then be useful in measuring the deviation of any social or individual action from the moral code itself.

Research question 5: Given a moral code, in a social choice context, where individuals submit their preference ordering and the result is a collective preference ordering, how to measure the deviation of the collective ordering from a moral code? And how to measure the deviation of individuals from a collective moral code?

If an individual modifies its preference ordering from a morally low to a morally higher ordering, we should want to use collective decision making system in which such a move leads to collective actions of higher morality. That is, some form of monotonicity should be desired.

Research question 6: Which properties should be desired in a moral preference aggregation environment?

5 Morality in narrow AI systems

In [8] it is shown that human moral judgment doesn't come from a dedicated moral system, but it is rather the product of the interaction of many general-purpose brain networks, each working and being useful in narrow contexts. So it seems that humans need a general purpose brain in order to be moral. Is it true also for AI systems?

Research question 7: Can narrow AI systems be moral? If humans bring all of their general intelligence to bear when making moral decisions, even fairly simple ones, does that that mean that we have to solve Artificial General Intelligence in order to produce something useful?

6 Concurrency in moral collective decision making

Collective decision making has to do with several agents (machines or humans) that express their preferences and, based on them, give their opinions over the alternative options for the collective decision. The agents are acting in parallel and independently, submitting more and more information about their preferences as time passes. It is easy to see that this can be faithfully modelled by a concurrent environment where preference data is accumulated over time, being generated by some agent and incrementally used by the other agents that are influenced or need to react to the opinions of others. When enough preferences are provided by the concurrent agents, a centralised agent make a collective decision based on what has been accumulated. The addition of ethical/moral preferences can easily be cast into this framework, by adding another concurrent agent and/or by modifying the preference aggregation agent.

The formalisation of an ethical collective decision making system in terms of concurrent agents can be very helpful in terms of the study of the properties of the resulting system. The extensive literature on theoretical properties of concurrent systems can be of great help in both defining the interesting properties for collective decision making and in studying their presence in specific systems. The morality of a system could be modelled as one of those properties, or a collection of them, thus giving rise to a formal treatment of morality in collective decision making.

Research question 8: How to model an ethical collective decision making system as a concurrent system? How to translate properties typically studied in concurrent systems (such as fairness etc.) into interesting properties for decision systems? What set of formal properties could faithfully model morality?

7 Conclusions

Intelligent systems are going to be more and more pervasive in our everyday lives. To name just a few applications, they will take care of elderly people and kids, they will drive for us, and they will suggest doctors how to cure a disease. However, we cannot let them do all this very useful and beneficial tasks if we don't trust them. To build trust, we need to be sure

that they act in a morally acceptable way. So it is important to understand how to embed moral values into intelligent machines.

Existing preference modelling and reasoning framework can be a starting point, since they define priorities over actions, just like an ethical theory does. However, many more issues are involved when we mix preferences (that are at the core of decision making) and morality, both at the individual level and in a social context.

Concurrency theory can be useful in both modelling ethical collective decision making systems and in studying their properties.

References

1 S. Airiau, U. Endriss, U. Grandi, D. Porello, and J. Uckelman. Aggregating dependency graphs into voting agendas in multi-issue elections. In *Proceedings of IJCAI 2011*, pages 18–23, 2011.

2 K. J. Arrow, A. K. Sen, and K. Suzumura. *Handbook of Social Choice and Welfare.* North-Holland, 2002.

3 C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR*, 21:135–191, 2004.

4 R. I. Brafman, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. Finding the next solution in constraint- and preference-based knowledge representation formalisms. In *Proceedings of KR 2010*, 2010.

5 V. Conitzer, J. Lang, and L. Xia. Hypercubewise preference aggregation in multi-issue domains. In *Proceedings of IJCAI 2011*, pages 158–163, 2011.

6 H. Fargier, J. Lang, J. Mengin, and N. Schmidt. Issue-by-issue voting: an experimental evaluation. In *Proceedings of MPREF 2012*, 2012.

7 C. Gonzales, P. Perny, and S. Queiroz. Preference aggregation with graphical utility models. In *Proceedings of AAAI 2008*, pages 1037–1042, 2008.

8 Joshua Greene. The cognitive neuroscience of moral judgment and decision making. In *The Cognitive Neurosciences V (ed. M.S. Cazzaniga)*. MIT Press, 2014.

9 Joshua Greene, Francesca Rossi, John Tasioulas, Kristen Brent Venable, and Brian Williams. Embedding ethical principles in collective decision support systems. In *Proceedings AAAI 2016*. AAAI Press, 2016.

10 J. Lang, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Winner determination in sequential majority voting. In *Proceedings of IJCAI 2007*, pages 1372–1377, 2007.

11 J. Lang and L. Xia. Sequential composition of voting rules in multi-issue domains. *Mathematical social sciences*, 57:304–324, 2009.

12 A. Maran, N. Maudet, M. S. Pini, F. Rossi, and K. B. Venable. A framework for aggregating influenced CP-nets and its resistance to bribery. In *Proceedings of AAAI 2013*, 2013.

13 P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In P. Van Beek F. Rossi and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2005.

14 Bert Musschenga and Anton (eds.) van Harskamp. *What Makes Us Moral? On the capacities and conditions for being moral*. Springer, 2013.

15 Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, 2000.

16 M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Incompleteness and incomparability in preference aggregation: Complexity results. *Artif. Intell.*, 175(7-8):1272–1289, 2011.

17 G. Dalla Pozza, M. S. Pini, F. Rossi, and K. B. Venable. Multi-agent soft constraint aggregation via sequential voting. In *Proceedings of IJCAI 2011*, pages 172–177, 2011.

18 K. Purrington and E. H. Durfee. Making social choices from individuals' CP-nets. In *Proceedings of AAMAS 2007*, pages 1122–1124, 2007.

19 F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

20 Amartya Sen. Choice, ordering and morality. In *Practical Reason, Korner S. (ed)*. Oxford, 1974.

21 Wendell Wallach and Colin Allen. *Moral Machines*. Oxford, 2009.

22 L. Xia and V. Conitzer. Strategy-proof voting rules over multi-issue domains with restricted preferences. In *Proceedings of WINE 2010*, pages 402–414, 2010.

Consistency in 3D*

Marc Shapiro[1], Masoud Saeida Ardekani[2], and Gustavo Petri[3]

1 **Sorbonne-Universités-UPMC-LIP6 & Inria Paris**
2 **Purdue University** †
3 **IRIF, Université Paris Diderot**

Abstract

Comparisons of different consistency models often try to place them in a linear strong-to-weak order. However this view is clearly inadequate, since it is well known, for instance, that Snapshot Isolation and Serialisability are incomparable. In the interest of a better understanding, we propose a new classification, along three dimensions, related to: a total order of writes, a causal order of reads, and transactional composition of multiple operations. A model may be stronger than another on one dimension and weaker on another. We believe that this new classification scheme is both scientifically sound and has good explicative value. The current paper presents the three-dimensional design space intuitively.

1998 ACM Subject Classification C.2.4 Distributed databases; D.1.3 Concurrent programming; D.2.4 Software/Program Verification; E.1 Distributed data structures

Keywords and phrases Consistency models; Replicated data; Structural invariants; Correctness of distributed systems;

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.3

Category Invited Paper

1 Introduction

A distributed database maintains data scattered and replicated across nodes separated by networks that are inherently slow and unreliable. In this context, designers face an inherent trade-off between system cost and application cost. In particular, the CAP theorem [13] shows that, when failures can partition the network (P), a database can either be strongly consistent (C) or available (A), but not both. Strong consistency masks parallelism and failures from the application, at the cost of constant synchronisation, which translates to high latency and even stalling when the network is down (CP). A model with weaker consistency significantly improves availability, performance and cost (AP), but increases the opportunities for subtle yet potentially catastrophic application errors.

This trade-off has spurred a lot of creativity. A dizzying number of consistency designs are available, as theoretical models, protocol designs, and implemented systems. Note however that, among the many options, not all are related to CAP.

In order to develop high-performance yet correct distributed applications, we need a better understanding, in particular how an application's needs relate to consistency. How does a particular application behave in a particular consistency model? What are its pros and cons? This paper aims to clarify this crowded space.

* This research is supported in part by European FP7 project 609 551 SyncFree.
† Now at Samsung Research America

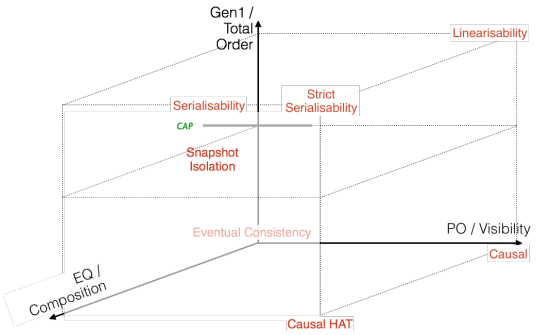


Figure 1 Some consistency models situated in the three dimensions.

The strongest consistency model, called Strict Serialisability (SSER), has three remarkable features:[1] absence of concurrent operations, i.e., transactions execute in a total order; this ordering is monotonic and respects causality; and the unit of interaction with the database, the transaction, is a composition of operations.

The thesis of this paper is that each of these features aims to guarantee a different class of application invariants. The mechanism associated with each feature has an inherent associated cost, respectively synchronisation, transitivity, and grouping.

Other consistency models differ from SSER by providing the same three features to a lesser degree, or even not at all. Relaxing a feature generally lowers its system cost but weakens the class of guaranteed invariants, increasing its cost to application programmers. Accordingly, we argue for classifying models in a three-dimensional space, along the axes of total-order, visibility, and composition. This insight is illustrated in Figure 1, and is fully detailed in Table 5.

Whereas previous surveys [1, 2, 29] are comprehensive and detailed, our focus is more pedagogical. Our three axes constitute a simplification. We do not claim to explain everything, but wish to help the reader situate a model on a mental map, glossing over details when convenient.

This paper is structured as follows. After this introduction (Section 1), Section 2 presents a generic system model. Then we define and discuss the three axes in turn: Gen1 invariants and total order in Section 3, PO invariants and visibility order in Section 4, EQ invariants and composition in Section 5. Finally, Section 6 summarises the relations between the three axes and concludes.

Inevitably, it is difficult to discuss one axis without referring to the others. We ask the reader's patience with such apparent circularities, which we do our best to minimise.

2 System model

Our model and definitions are derived from previous work [9, 14, 25, 29]. The system is composed of an unbounded set of sequential processes, uniquely identified. We divide it into an application layer running above a consistency layer. The application consists of *objects* stored in the database and of *client* processes that call operations on objects and receive results in return. Clients do not communicate directly, only via operations on shared data.

[1] We refer to Table 6 for a full list the consistency models discussed herein and the primary reference for each.

(a) Operation u decomposed into indivisible call, return, generator $u_?$ and effector $u_!$. Precondition u_{PRE} is true at the effector.

(b) The effectors of concurrent operations may execute in different orders in the general case.

Figure 2 Operation model.

The consistency layer manages state and executes the message sending, receiving and delivery events described hereafter. A consistency model consists of a set of restrictions imposed on the ordering of events, in order to *guarantee* a class of *invariants* that remains true in any execution of that model. Ideally, we would like to ensure any invariant of a sequential execution.

To simplify the discussion, there will be no failures: a message sent is eventually received, unaltered, by its destination process. We focus on safety and do not consider liveness properties.

2.1 Data

Server processes collectively implement the abstraction of a database or persistent memory. The database consists of discrete data items of *objects* x, y. The state of server i, noted σ_i, contains a copy or *replica*, noted x_i, of object x.[2]

A common object type is the *register*, which supports the read and write operations, respectively returning and completely overwriting overwrites the register's content. The state of a register depends only on the last write. However, our model is not restricted to registers. The application may store object types of arbitrary complexity, for instance a set, a stack, a table, or a tree, with their high-level operations (respectively, add and remove, push and pop, insert row, or rebalance).

2.2 Operations

We decompose an operation into indivisible asynchronous events, as illustrated in Figure 2a (a specific consistency model may place restrictions on their ordering).

The semantics of update operations is defined by a function: $\mathcal{F} \in \mathsf{Op} \to (\mathsf{State} \to \mathsf{Val} \times (\mathsf{State} \to \mathsf{State}))$ where Op is the set of operations, State the set of replica states, and Val the set of return values. Operationally, an update u starts as a *call* event, a message from client to origin replica. Delivering this message triggers an initial computation at the origin, called the *generator* $u_?$. The generator reads the state of the origin without modifying it, then:

1. Computes a return value u_{RET}, sent back to the client. When the client receives it, the update is *visible to the client*.

[2] To simplify the model, we assume *full replication*: every replica has a copy of every object.

2. Computes a state transformation, the *effector* $u_!$. The effector is sent to all replicas, including the origin itself. If and when a replica *delivers* the effector, it applies its transformation to the replica's local state, making the update *visible to the replica*.

The effector is generated based on the origin state read by the generator; we abstract this dependence with the precondition u_{PRE} of the effector. The generator can check the precondition at the origin but not at other replicas [14].

An operation reads (generator) and writes (effector) the replicas of a single object. As we shall see shortly, objects can be connected by invariants and/or transactions.

The history of a client consists of a sequence of call (sending) followed by return (delivery) events. The history of a server consists of a sequence of generator (reading and sending) events and effector (delivery and side-effect) events. The current state of a server can be identified with the sequence the effectors it has delivered. In the general case, effectors of different updates may be delivered in different orders, as illustrated in Figure 2b.

This model is very general. It abstracts away from any specific data type (from registers to complex data types with high level operations), transmission mode (state-based or operation-based), and concurrency semantics (which will be encoded into the function definition). We model operations that do not return by returning nil; we model queries that do not modify state by the skip effector; we abstract arguments away by folding them into the function definition.

The client may choose an arbitrary origin replica, not necessarily the same for successive client operations. Therefore, client-side guarantees may be weaker than those at the server [8, 27]. Conversely, we consider that the server-side guarantees are at least as strong as the client-side ones.

2.3 Executions

We define an execution as a tuple $ex = \langle R, E, \xrightarrow{\mathsf{so}}, \xrightarrow{\mathsf{ro}}, \xrightarrow{\mathsf{ext}} \rangle$ where:

(i) R is a set of replicas – which shall otherwise remain abstract.

(ii) E is a set of events, including calls, generators, effectors and returns.

(iii) $\xrightarrow{\mathsf{so}}$ is a relation among events of the *same session* [29], indicating the order in which the client issued the operations; $\xrightarrow{\mathsf{so}}$ is our abstraction of the behavior of the client.

(iv) $\xrightarrow{\mathsf{ro}}$ is a family of orders – indexed by replica name – of events that affect that replica. These events include: call, generator and return events for any operation that has this replica as its origin, and effectors of any operation that is delivered to the replica. We shall denote $\xrightarrow[r_i]{\mathsf{ro}}$ the replica order of replica r_i, and we shall overload the notation $\xrightarrow{\mathsf{ro}}$ to denote as the set-theoretic union of the replica orders of each replica, formally the relation $\bigcup_{r_i \in \mathsf{dom}(\xrightarrow{\text{ro}})} \xrightarrow[r_i]{\mathsf{ro}}$.

(v) Finally, $\xrightarrow{\mathsf{ext}}$ is an *external* order representing the real-time (otherwise called wall-clock time) in which the events occurred; we shall simply assume the existence of this order for some models, and it shall otherwise remain opaque.

Provided with the definition of executions we obtain the derived definition of visibility of operations:

$$u \xrightarrow[r_i]{\mathsf{vis}} v \iff u_! \xrightarrow[r_i]{\mathsf{ro}} v_? \qquad\qquad \xrightarrow{\mathsf{vis}} = \bigcup_{r_i \in R} \xrightarrow[r_i]{\mathsf{vis}}$$

In turn, we obtain a definition the *happens-before* order: $\xrightarrow{hb} = (\xrightarrow{\mathsf{so}} \cup \xrightarrow{\mathsf{vis}})^+$, where we denote by a superscript $^+$ the set-theoretic transitive closure of a binary relation. We speak of *transitive* visibility if $\xrightarrow{\mathsf{vis}}{}^* \subseteq \xrightarrow{\mathsf{vis}}$, and we speak of *causal* visibility if the visibility relation is consistent w.r.t. the happens-before relation: $u \xrightarrow{hb} v \Rightarrow u \xrightarrow{\mathsf{vis}} v$.

Table 1 Application assumptions (top) and robustness conditions (bottom).

Baseline	Semantic condition	$\Longrightarrow$	Reference
Sequential	Sufficient precondition	Safe	[14]
TOE	Deterministic operations	Same state	[9], [28]
0	Unspecified convergence conditions	EC	[30]
0	Monotonic semi-lattice	Monotonic SEC	[5]
CC	Commutative concurrent effectors	SEC	[25]
CC	Stable effector precondition	Gen1	[14]
SI	Materialized conflict	SI ∩ SER	[11]

0 = lowest point on all three axes. Sequential = sequential non-replicated system.

2.4 How models relate to application semantics

Applications care about consistency models for the *guarantees* that they provide. We say that model guarantees a certain class of invariants, if an invariant of that class remains true in any execution of that model without requiring additional instrumentation from the programmer.

Some guarantees refer to relations between replicas, for instance, Identical State and Convergence. However, an application-observable invariant refers to the state that is observable for a client, e.g., single-object statements such as $x > 0$ or multi-object statements such as $x = y$ or $P(x) \iff Q(y)$. As we shall discuss in detail hereafter, some consistency models guarantee some related classes of invariants.

Although it is convenient to think of the consistency and application levels as independent, this is not entirely true. The correctness of the guarantees rests on two crucial assumptions about the application:

1. A generator and an effector are functions, i.e., their result is deterministic.
2. The application is *sequentially correct*, i.e., each operation (or, in the transactional case, each transaction) in isolation maintains the application invariant. This is the C condition of ACID, called "consistency" or "correctness" in the database literature. Formally, for some invariant I, $\forall \sigma \in \mathsf{State}, u \in \mathsf{Op} : I(\sigma) \wedge u_{\text{PRE}} \implies I(\sigma \bullet u_!)$ for the single-operation case, where we denote by $\sigma \bullet u_!$ the state resulting from updating σ with the effector $u_!$.

A *robustness condition* is one by which an application, running above a less-than-perfect consistency model, can compensate for its deficiencies and support the same invariants as a stronger model. For instance, EC (Eventual Consistency) requires that the application converges, even when running above level zero on all axes (for this, see Convergent Data Types [5] or CRDTs [25]).

Fekete et al. show how the application can emulate Serialisability (SER) above Snapshot Isolation (SI) by applying some simple programming rules [11, Section 5.1].

Gotsman et al. [14] discuss under which conditions concurrent execution can maintain arbitrary application invariants (class Gen1 hereafter). They demonstrate (under certain conditions) that, if all effector preconditions u_{PRE} are stable under all concurrent updates $v_!$, then the invariant remains true, no matter what the order of delivery of effectors.

3 Gen1/Total Order Axis

This section focuses on guaranteeing invariants by restricting concurrency as summarised in Table 2. This axis orders the different consistency properties according to which events must be totally-ordered with respect to each other. Here we consider operations on a single object (the other two axes consider multiple objects). Let us now consider the different protocols

Table 2 Total-order axis. The double line marks the "CAP boundary."

Level	Guarantees	Other axes	Examples
TOG=TOE	Gen1	External visibility Transitive visibility	SSER, LIN SER
Gapless TOE	No lost updates, Identical State	External visibility Causal visibility Transitive visibility	SSI PSI NMSI
Capricious TOE	register $\implies$ Identical State	Transitive visibility	LWW
		Non-monotonic	Bayou
0 = Concurrent	Blind1	Monotonic visibility	RC EC

according to the ordering of events that they impose on the operations to the object of interest (ignoring operations on other objects, which may proceed in parallel).

Unless indicated otherwise, we assume the Monotonic Client property, which is the conjunction of the Monotonic Reads guarantee: given two operations related by the session order $v \xrightarrow{\mathsf{so}} w$, if the former "views" a third operation u (i.e., $u \xrightarrow{\mathsf{vis}} v$) then so does the latter ($u \xrightarrow{\mathsf{vis}} w$); and the Read-My-Writes guarantee: if two operations are related by the session order $u \xrightarrow{\mathsf{so}} v$, then so are they by the visibility order $u \xrightarrow{\mathsf{vis}} v$.

3.1 Same total order for generators and effectors (TOG=TOE)

The first class of models we consider are those for which there exists a Total Order relating all Effectors and Generators (TOG=TOE), let us denote this (existentially quantified) order with the arrow $\xrightarrow{\mathsf{toeg}}$.[3] These are the strongest models in the total-order axis. Evidently, there are a number of constraints that are required for $\xrightarrow{\mathsf{toeg}}$:

1. Generators and effectors are uninterrupted by other events in the order (therefore the sequences $u_? \xrightarrow{\mathsf{toeg}} v_? \xrightarrow{\mathsf{toeg}} u_!$ and $u_? \xrightarrow{\mathsf{toeg}} v_! \xrightarrow{\mathsf{toeg}} u_!$ are disallowed).
2. The visibility relation is consistent w.r.t. the total order, meaning that each generator sees exactly the effectors that precede it in this order ($u \xrightarrow{\mathsf{toeg}} v \implies u \xrightarrow{\mathsf{vis}} v$).

Table 2 presents in the cell at the first row and last column protocols that fall under this category in the total order axis.

Importantly, the existence of such an order implies that the Visibility relation, restricted to the object, is transitive since each generator must see all the effectors before it, and the effectors of an operation necessarily follow its generator. On the other hand, causality is not guaranteed unless we add the condition that the total order respects the client order ($(\xrightarrow{\mathsf{so}} \cup \xrightarrow{\mathsf{toeg}})^+$ is irreflexive). By adding this additional constraint we require the Visibility relation to be causal for this object.

3.2 TOG=TOE and Gen1 Invariants.

At this strongest point in this axis, we consider generic (arbitrary) single-item invariants, noted hereafter Gen1. For instance, a banking application may require that the balance of accounts be non-negative: $\mathsf{bal} \geq 0$. Another example: an object G that represents a graph, with the invariant that the graph forms a tree.

[3] In the interest of readability and space, we shall present some definitions intuitively instead of providing precise mathematical definitions. Their mathematical interpretation is generally self-evident.

Recall from Section 2.4 that a sequential program enforces its invariants assuming the effector-precondition u_{PRE}, which may be verified locally by the generator. In the bank account example, $\mathsf{credit(amt)}_!$ and $\mathsf{debit(amt)}_!$ respectively add or subtract amt to or from the local balance bal_i. To maintain invariant $\mathsf{bal} \geq 0$, the sequential preconditions are $\mathsf{credit}_{\text{PRE}} = \mathsf{amt} \geq 0$ and $\mathsf{debit}_{\text{PRE}} = \mathsf{bal} \geq \mathsf{amt} \geq 0$ respectively. However, under unbounded concurrency there exists no safe precondition that can be evaluated locally at the origin replicas; intuitively, enforcing this invariant requires to totally order at least some operations.

In the case of protocols respecting TOG=TOE, and assuming the system respects the ordering of operations issued by the client (the session order: $\xrightarrow{\mathsf{so}}$), any invariant that is correct for a shared memory implementation of the object – where we interpret the clients as being processes, and the database as being the shared memory – will also be respected in this case. We posit that anything provable using, for instance, the Owiki-Gries [21] logic under the shared-memory interpretation, is respected in such a model.

3.3 Total Order of Effectors (TOE): Capricious vs. Gapless

Since generators only read state without changing it, it is tempting to remove them from the total order, therefore allowing concurrency between reads and writes. We shall denote this weaker existential order as $\xrightarrow{\mathsf{toe}}$. This introduces the possibility of anomalies such as write-skew [7].

The order $\xrightarrow{\mathsf{toe}}$ may be *Capricious*: meaning that servers assign sequence numbers independently from one another. While effectors are totally ordered (i.e., each effector has a unique place in the order), they may be received in a non-increasing sequence. This conflicts with the monotonic-client requirement; as a consequence, updates might be lost, if an effector has been delivered at a replica while another effector ordered lower in $\xrightarrow{\mathsf{toe}}$ is received at a later point. This approach is used, for instance, in the Last-Writer Wins (LWW) protocol.

Alternatively, $\xrightarrow{\mathsf{toe}}$ can be *Gapless*: in this case replicas must synchronise to guarantee that the effectors are given a slot in the total order in a strictly monotonic fashion, and therefore replicas can buffer effectors until all prior updates in $\xrightarrow{\mathsf{toe}}$ have been delivered. Here lies the "CAP Line:" Capricious TOE is Available even when the network is Partitioned, whereas Gapless TOE (and of course gapless TOG=TOE) is not Available when Partitions occur.

Strictly speaking, a protocol could be both Capricious and TOG=TOE; however this combination is not very useful; therefore, to simplify the presentation, we order TOG=TOE above Gapless TOE.

3.4 TOE and Causality Based Invariants

In terms of application guarantees provided by protocols satisfying TOE guarantees we cannot generally assume that Gen1 invariants will be satisfied. On the other hand, under Causal Visibility, Rely-Guarantee based techniques can be used [14].

As it is the case with TOG=TOE models, the existence of a $\xrightarrow{\mathsf{toe}}$ order implies that the visibility relation is transitive per-object. If we additionally require that $\xrightarrow{\mathsf{toe}}$ respects the client order ($\xrightarrow{\mathsf{so}}$) we can conclude that visibility is causal per-object. An important distinction between capricious and gapless $\xrightarrow{\mathsf{toe}}$ models is that in the latter, any two replicas that have received the same updates have the exact *same state*. In contrast, capricious models cannot guarantee the same-state property.

3.5 Concurrent Effectors

At the weakest end of the total order axis, the protocol "Concurrent effectors" in Table 2 does not require any total ordering of effectors and/or generators.

Consider a register, with an invariant that refers only to the current state: e.g., register z must contain an odd number of "1" bits. To maintain it, the order and history of updates is immaterial, and it suffices that each update is individually safe. We shall denote these invariants that are *blind* to the environment and on a single object (1), Blind1 in Table 2.

4 PO/Visibility Axis

The Visibility dimension (Table 3) constitutes our second axis. It aims to guarantee invariants that require control of which effectors are visible, in which order, to generators. Whereas the first axis concerned single-object guarantees, this one connects multiple updates, system-wide. Whereas the first axis is concerned mostly about writes (effectors), this one is mostly about reads (generators). However, they are not totally independent.

4.1 PO-type invariants

The PO-type invariants discussed in this section abstract the concept of a partial order. Conventionally we will write them as $L \geq R$ and refer to the two terms as left- and right-hand-side, LHS and RHS, respectively. The prime example is program order, where each process proceeds through statements $S_1; S_2; \ldots; S_n$, left to right. This may be re-written (abusing notation somewhat) as $S_1 \Longleftarrow S_2 \Longleftarrow \ldots \Longleftarrow S_n$, i.e., executing S_i implies that S_{i-1} has executed. Similarly, write-read dependences, where $v_?$ reads the result of $u_!$, can be summarised as $u_! \Longleftarrow v_?$ and message delivery as $u_? \Longleftarrow v_!$.

Other PO-type invariants are traditional data invariants, such as "employee's salary must be less than his manager's", stock maintenance [6], or referential integrity (object x allocated $\Longleftarrow$ y points to x).

Even with unbounded concurrency, it is safe to update the objects involved in a PO-type invariant, by *first* increasing the LHS by some amount c, and *later* increasing the RHS by an amount $c' \leq c$. More generally, it is always safe to strengthen the invariant, and later weaken it assuming that the prior strengthening has been applied. This is known as the Demarcation Protocol [6] or the safe-publication idiom [1]. As a special case, c' can be null, i.e., it is safe to unilaterally increase the LHS.

We will consider different versions of the demarcation protocol according to the visibility guarantees enforced by the underlying model (as shown in Table 3). For instance, for a system enforcing causal visibility, we can operate under the causal demarcation protocol if one client does the strengthening of the invariant and notifies another client of this fact by writing on a flag. When the second client sees the effects of the update on the flag, by causality we can assume that the invariant can be safely weakened according to the prior strengthening on the other operation. A similar arguments can be made for transitive demarcation.

The above requires that updates become visible to other replicas in the same order. We discuss such protocols in the next section.

At the weakest level of the Visibility axis, labeled "Rollbacks" in Table 3, there is no required order between reads. A client could observe the effects of some update u, and later observe a state where u has not occurred. This violates the so-called Monotonic-Reads session guarantee [27]. Similarly, a client might update an object, and later observe a state of the object before the update is applied. This violates Read-My-Writes [27].

Table 3 Visibility axis.

Level	Guarantees	Other axes	Example
External	external demarcation		SSER, LIN, SSI
Trans. Vis. + Client Order = Causal Visibility	causal demarcation	TOE not TOE	PSI Causal HAT, CC
Monotonic + WR dependence = Transitive Visibility	transitive demarcation	TOE not TOE	SER, NMSI
MR + RMW = Monotonic Client	client progress		
0 = Rollbacks			Bayou

MR = Monotonic Reads. RMW = Read-My-Writes. WR = Write-Read dependency. Client Order conjoins all these relations with Write-Write dependencies [29].

Few systems are at Rollback level; most models assume what we call the Monotonic Client level, in which the client state is monotonic, ensuring both Monotonic Reads and Read-My-Writes (as defined in Section 3). In fact, client monotonicity must appear so obvious that many authors do not even state this assumption, e.g., Gray and Reuter [15]. We will follow the common practice of assuming the Monotonic Client guarantee in this paper, unless explicitly mentioned. Frigo [12] argues that non-monotonic models are "not reasonable," but some systems deliberately eschew these guarantees for the benefit of responsiveness [28].

The next-stronger level, Transitive Visibility, simply requires the visibility relation to be transitive. Given operations u and v, if the (generator of) update v reads the result of (the effector of) update u, then all clients should observe the results of u before those of v. Formally $\xrightarrow{\mathsf{vis}}{}^{*} \subseteq \xrightarrow{\mathsf{vis}}$. Note that Total Order of Effectors implies Transitive Visibility, but not vice-versa. Not all models have the Transitive Visibility property. For instance, SER has it, but not EC nor PRAM. To simplify the presentation, hereafter Transitive Visibility also includes Monotonic Client.

The next level adds Client Order (Monotonic Writes and Writes Follow Reads [27]), resulting in Causal Visibility (also called Causal Consistency or Causal Memory [3]). Formally this requires that visibility be consistent with the session order: $\xrightarrow{hb} \subseteq \xrightarrow{\mathsf{vis}}$. Transitive and Causal Visibility are partial orders. They can be further strengthened by requiring the existence of a *total* order that is causal (hence also transitive); this point meets the TOG=TOE point of the Total Order axis of Section 3.

Causal visibility is strictly stronger than transitive visibility, and is not supported by all models. As a case in point, SER does not require causal visibility: if a client calls operations u and then v, and u and v are on different object, a server (even the origin server) may execute v before u.[4]

The highest point in the Visibility axis is External Consistency. This requires that all operations are totally ordered (finding a $\xrightarrow{\mathsf{toeg}}$ as in Section 3), and that this order coincides with the external (real-time) order: $\xrightarrow{\mathsf{ext}} \subseteq \xrightarrow{\mathsf{toeg}}$. In this way, updates can be related with external events, and the causality between internal and external events is preserved.

Causal Visibility is the conjunction of the four so-called session guarantees [8]: formally, all sixteen combinations are possible. Pragmatically, however, we find that the linear presentation of Table 3 captures the important practical properties.

[4] Here we argue about operations, while serialisability is defined for transactions. The analogous argument is obvious assuming that the transactions operate on different object sets.

Table 4 Composition axis.

Level	Guarantees	Other axes
All-or-Nothing + Snapshot	EQ + Gen*	TOG=TOE
	EQ	
All-or-Nothing Effectors		
0 = Single Operation		

5 EQ/Composition Axis

Our third axis aims to guarantee some form of coupling between separate objects. It provides mechanisms to:

(i) compose together multiple updates and multiple objects dynamically, and
(ii) to close the guarantees provided by the Total Order and/or Visibility axes over the whole composition.

5.1 EQ-type and Gen* invariants

An EQ-type invariant is one that maintains an equivalence relation between objects. EQ requires to always group together updates to both objects; we call this All-Or-Nothing Effectors; intuitively, either all the updates of the composition are visible, or none is. For instance, a symmetric friendship graph $x.\mathsf{friendOf}(y) \iff y.\mathsf{friendOf}(x)$, or disjoint union to a constant set, $A \cap B = \emptyset \wedge A \cup B = C$. Notice the similarity between EQ and Blind1: neither depends on previous state, only on the current transaction (resp. operation). As it is the case for Blind1 invariants, in order to verify EQ invariants, no ordering assumptions are required from the environment, and it suffices to show that each individual transaction preserves the invariant if it was initially valid.

Consider now a generic sequential invariant over multiple objects, noted Gen*. Since multiple objects are involved, this likely requires All-Or-Nothing Effectors. Furthermore, the generators' reads will need to be mutually consistent, and served from a *consistent snapshot*. Finally, Gen* may require a total order, by the same reasoning as for Gen1 (recall Sections 3.2 and 3.3). The Transactional Composition axis serves to enforce these requirements.

Transactions support "ad-hoc" composition. For instance, when buying a ticket online, ensuring that the buyer has sufficient balance and that a ticket is available (ad-hoc Gen*), and ensuring that the money is both debited from the buyer's account and credited to the seller's (ad-hoc EQ).

5.2 (Transactional) Composition axis

For this axis we add begin and end markers to the repertoire of events uttered by a client, grouping all the intervening calls and returns into one transaction. Depending on the model, transactions may be associated with the properties "All-Or-Nothing Effectors" and "Snapshot." Table 4 shows the composition axis.

In many implementations, a server may execute a transaction speculatively, and either commit or abort at the end [23]. An aborted transaction has no effect and does not return anything. Our model considers only committed transactions.

All-Or-Nothing Effectors means that, if some effector of transaction T1 is visible to transaction T2, then all of T1's effectors are visible to T2. (This is the A in ACID, sometimes called Atomic.) TOE guarantees extend to all effectors of a transaction: if $u_!$ and $v_!$ are part

Table 5 Matrix of features and consistency models.

Total Order	Composition	Visibility: Rollbacks	Monotonic	Transitive	Causal	External
TOG=TOE	All-or-Nothing + Snapshot			SER		SSER
	All-or-Nothing Effectors					
	Single Operation				SC	LIN
Gapless TOE	All-or-Nothing + Snapshot			NMSI	PSI	SSI
	All-or-Nothing Effectors					
	Single Operation					
Capricious TOE	All-or-Nothing + Snapshot	Bayou				∅
	All-or-Nothing Effectors					∅
	Single Operation		LWW			∅
Concurrent Ops	All-or-Nothing + Snapshot				Causal HAT	∅
	All-or-Nothing Effectors		RC			∅
	Single Operation	EC	PRAM		CC	∅

of T1, $w_!$ and $t_!$ are of part of T2, and $u_! < w_!$ in the TOE, then $v_! < w_!$ and $u_! < t_!$. We may write simply $T1_! < T2_!$.

Typically, all the generators of the transaction read from a same set of effectors, called its *snapshot*. Generator order guarantees, if any, extend to the whole snapshot, i.e.,

(i) Monotonic-Client, resp. Transitive, resp. Causal Visibility: the snapshot (the set of effectors read from) is closed under the visibility order.

(ii) TOG=TOE: the generators are adjacent in the total order.

5.3 Composition: Discussion

Transactional protocols generally assume All-or-Nothing but differ in their snapshot guarantees. For instance, SER, NMSI or SI require Transitive Visibility but do not enforce client order, i.e., Monotonic Writes [15]. Indeed, these models allows a client to execute T1; T2 and the system to serialise as T2; T1 if their read-write sets are disjoint. Strong Snapshot Isolation (SSI) does ensure client order, hence Causal Visibility, as it mandates to choose a snapshot greater than any commit point when a transaction starts. The same is true of a protocol that requires external causality, such as Strict Serialisability (SSER).

In addition to the features discussed so far, snapshots may be partially ordered or totally ordered. For instance, NMSI's snapshots are partially ordered, whereas SI, SSI, and SER snapshots are totally ordered. This represents the main difference between SI and NMSI.

As a simplification, our linear axis does not differentiate between partially- and totally-ordered snapshots. Unfortunately and consequently, SI is missing from our summary table (Table 5) as it would occupy the same position as NMSI.

6 Discussion and conclusion

Our system model (Section 2) is very general. The separation between generators and effectors allows for internal parallelism; if unusual, it reflects practical implementations [23]. Our total order axis (Section 3), classifies the degree of concurrency between operations to a single object, including only effectors or also generators, and accounts for both available (capricious) and consensus-based (gapless) approaches. The other two axes introduce mechanisms that relate multiple objects; however, they serve different purposes and have different costs. Visibility order (Section 4) relates reads to writes and involves maintaining a system-wide transitive closure, and aims to support PO-type invariants. Transactions (composition,

Table 6 Cross-reference of models, protocols and systems.

Acronym	Full name	Type	Total-Order	Visibility	Composition	Ref.
Bayou	Bayou	system	Capricious TOE	Rollbacks	All-or-Nothing + Snapshot	[28]
CC	Causal Consistency	model	Concurrent Ops	Causal	Single Operation	[3]
Causal HAT	Causal Highly-Av. Txn.	model	Concurrent Ops	Causal	All-or-Nothing + Snapshot	[4]
EC	Eventual Consistency	model	Concurrent Ops	Rollbacks	Single Operation	[30]
LIN	Linearisability	model	TOG=TOE	External	Single Operation	[17]
LWW	Last-Writer Wins	protocol	Capricious TOE	Monotonic	Single Operation	[18]
NMSI	Non-Monotonic SI	model	Gapless TOE	Transitive	All-or-Nothing + Snapshot	[24]
PRAM	Pipeline RAM	model	Concurrent Ops	Monotonic	Single Operation	[20]
PSI	Parallel SI	model	Gapless TOE	Causal	All-or-Nothing + Snapshot	[26]
RC	Read Committed	model	Concurrent Ops	Monotonic	All-or-Nothing Effectors	[7]
SC	Sequential Consistency	model	TOG=TOE	Causal	Single Operation	[19]
SER	Serialisability	model	TOG=TOE	Transitive	All-or-Nothing + Snapshot	[15]
SI	Snapshot Isolation	model	Gapless TOE	Transitive	All-or-Nothing + Snapshot	[7]
SSER	Strict Serialisability	model	TOG=TOE	External	All-or-Nothing + Snapshot	[22]
SSI	Strong Snapshot Isolation	model	Gapless TOE	External	All-or-Nothing + Snapshot	[10]

Section 5) serves to enforce ad-hoc EQ and Gen*; a transaction is a one-off grouping, requested by the application.

In order to be intuitively useful, our classification simplifies the design space into three approximately linear axes (which we relate to *application invariants*). Obviously, this cannot account for the full complexity of the relations between models. We acknowledge the deficiencies of such a simplification. For instance, we flatten the visibility axis, and abusively assume that all TOG=TOE models must be gapless. We defend this simplification as practically relevant, even if not formally justified. We also ignored hybrid models, such as Update Serialisability [16].

We focus on client-monotonic models, as they are the most intuitive, and because monotonicity is trivial to implement. While the specifications of SER, NMSI, or RC do not require Monotonic visibility, all the actual implementations that we know of do provide it.

Table 5 positions some major consistency models within the three axes. Compare for instance two prominent strong consistency models: SSER and LIN. While LIN considers single operations and single objects, SSER is a transactional model requiring All-or-Nothing and Snapshot. Also notice how the visibility axis differentiates SSER from SER, and NMSI from PSI.

While our results are preliminary, we believe that this classification sheds light on the crowded space of distributed consistency guarantees, towards a better understanding of the application invariants enforced by each of them. We intend, in further work, to formalize our definitions and prove some interesting meta-properties. This work aims to be an step towards a rigorous and systematic understanding of distributed database implementations and their applications.

References

1 Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 29(12):66–76, December 1996.

2 Atul Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, Mass. Institute of Technology, Cambridge, MA, USA, March 1999.

3 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, March 1995.

4 Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. *Proc. VLDB Endow.*, 7(3):181–192, November 2013.

5 Carlos Baquero and Francisco Moura. Using structural characteristics for autonomous operation. *OSR*, 33(4):90–96, 1999.

6 Daniel Barbará-Millá and Hector Garcia-Molina. The demarcation protocol: A technique for maintaining constraints in distributed database systems. *VLDB Jrnl.*, 3(3):325–353, July 1994.

7 Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A critique of ANSI SQL isolation levels. *SIGMOD Rec.*, 24(2):1–10, May 1995.

8 J. Brzezinski, C. Sobaniec, and D. Wawrzyniak. From session causality to causal consistency. In *Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, pages 152–158, A Coruña, Spain, February 2004. Euromicro.

9 Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: Specification, verification, optimality. In *POPL*, pages 271–284, San Diego, CA, USA, January 2014.

10 Khuzaima Daudjee and Kenneth Salem. Lazy database replication with snapshot isolation. In *VLDB*, pages 715–726, 2006.

11 Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *TODS*, 30(2):492–528, June 2005.

12 Matteo Frigo. *The weakest reasonable memory model.* PhD thesis, MIT, Cambridge, MA, USA, October 1997.

13 Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

14 Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 'Cause I'm strong enough: Reasoning about consistency choices in distributed systems. In *POPL*, pages 371–384, St. Petersburg, FL, USA, 2016.

15 Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Francisco CA, USA, 1993. ISBN 1-55860-190-2.

16 R C Hansdah and Lalit M. Patnaik. Update serializability in locking. In Giorgio Ausiello and Paolo Atzeni, editors, *Int. Conf. on Database Theory*, pages 171–185, 1986.

17 Maurice Herlihy and Jeannette Wing. Linearizability: a correcteness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990.

18 Paul R. Johnson and Robert H. Thomas. The maintenance of duplicate databases. Internet Request for Comments RFC 677, Information Sciences Institute, January 1976.

19 Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. on Computers*, C-28(9):690–691, September 1979.

20 R J Lipton and J S Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, 1988.

21 Susan Owicki and David Gries. Verifying properties of parallel programs: an axiomatic approach. *CACM*, 19(5):279–285, May 1976.

22 Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, October 1979.

23 F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *J. of Dist. and Parallel Databases and Technology*, 14(1):71–98, 2003.

24 Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, pages 163–172, Braga, Portugal, October 2013. IEEE Comp. Society.

25 Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and V. Villain, editors, *SSS*, volume 6976 of *LNCS*, pages 386–400, Grenoble, France, October 2011. Springer Verlag.

26 Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li. Transactional storage for geo-replicated systems. In *SOSP*, pages 385–400, Cascais, Portugal, October 2011. Assoc. for Comp. Mach.

27 Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *PDIS*, pages 140–149, Austin, Texas, USA, September 1994.

28 Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP*, pages 172–182, Copper Mountain, CO, USA, December 1995. ACM SIGOPS, ACM Press.

29 Paolo Viotti and Marko Vukolić. Consistency in non-transactional distributed storage systems. ArXiv e-print 1512.00168, arXiv.org, December 2015. Accepted for publication in ACM Computing Surveys.

30 Werner Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, October 2008.

Love Thy Neighbor: V-Formation as a Problem of Model Predictive Control*

Junxing Yang[1], Radu Grosu[2], Scott A. Smolka[3], and Ashish Tiwari[4]

1 **Stony Brook University, USA**
2 **Stony Brook University, USA and Vienna University of Technology, Austria**
3 **Stony Brook University, USA**
4 **SRI International, USA**

Abstract

We present a new formulation of the V-formation problem for migrating birds in terms of model predictive control (MPC). In our approach, to drive a collection of birds towards a desired formation, an optimal *velocity adjustment* (acceleration) is performed at each time-step on each bird's current velocity using a model-based prediction window of T time-steps. We present both centralized and distributed versions of this approach. The optimization criteria we consider are based on fitness metrics of candidate accelerations that birds in a V-formations are known to benefit from, including *velocity matching*, *clear view*, and *upwash benefit*. We validate our MPC-based approach by showing that for a significant majority of simulation runs, the flock succeeds in forming the desired formation. Our results help to better understand the emergent behavior of formation flight, and provide a control strategy for flocks of autonomous aerial vehicles.

1998 ACM Subject Classification I.6.4 [Simulation and Modeling]: Model Validation and Analysis; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search – Control theory; J.3 [Life and Medical Sciences]: Biology

Keywords and phrases Bird flocking, V-formation, model predictive control, particle swarm optimization.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.4

Category Invited Paper

1 Introduction

It has long been observed that flocks of birds organize themselves into V-formations, particularly migrating birds traveling long distances. There are two main reasons for this behavior. The first relates to the aerodynamics of formation-flight, where birds generate an upwash region off the trailing edge of their wings, allowing birds behind them to save energy from this free lift [2, 11]. The second reason is that a V-formation provides birds with an optimum combination of a clear visual field along with visibility of lateral neighbors [5, 6].

Previous work on modeling this emergent behavior has focused on providing combinations of *dynamical flight rules* as driving forces. For example, in [4], the authors extend Reynolds' model [9] with a rule that forces a bird to move laterally away from any bird that blocks

* Research supported in part by the following grants: AFOSR FA9550-14-1-0261, NSF CPS-1446832, NSF CNS-1423298, NSF IIS-1447549, NSF CNS-1445770, and NSF CNS-1430010.

its view. This can result in multiple V-shaped clusters, but flock-wide convergence is not guaranteed. The work of [3] induces V-formations by extending Reynolds' model with a *drag reduction* rule, but the final formation tends to oscillate as birds repeatedly adjust the angle of the V. Another approach, based on three *positioning rules*, is that of [8]. Their model, however, is limited by the assumption that birds have a constant longitudinal heading. The authors of [10] attempt to improve upon this approach by handling *turning movements*. But their model also produces small clusters of birds, each of which is only moderately V-like.

We, in contrast, view the problem of V-formation as one of *optimal control*. Compared to previous work, there are no behavioral rules in our approach. Instead, we adopt the idea of model predictive control (MPC) [1]. To drive a collection of N birds towards a desired formation, an optimal *velocity adjustment* (acceleration) is performed at each time-step on each bird's current velocity using a model-based prediction window of T time-steps. This yields an optimal acceleration sequence of length T, and the first acceleration in the sequence is applied. We present both centralized and distributed versions of this approach.

The optimization we perform is based on fitness metrics that capture the essence of a V-formation, namely *Velocity Matching* (VM), *Clear View* (CV), and *Upwash Benefit* (UB). VM means that bird velocities are aligned, allowing them to maintain formation. CV requires birds to have an unobstructed frontal view, while UB models the energy saving birds obtain from the upwash regions generated by their frontal neighbors. We show by simulations that birds succeed in forming the desired formations with high probability.

2 Model Predictive Control for V-Formation

Let $\boldsymbol{x}(t)_i$, $\boldsymbol{v}(t)_i$ and $\boldsymbol{a}(t)_i$ be the vector of 2-dimensional positions, velocities and accelerations, respectively, of bird i at time t, $1 \leq i \leq N$. The following equations model the behaviors of bird i in discrete time:

$$\begin{aligned} \boldsymbol{x}(t+1)_i &= \boldsymbol{x}(t)_i + \boldsymbol{v}(t+1)_i \\ \boldsymbol{v}(t+1)_i &= \boldsymbol{v}(t)_i + \boldsymbol{a}(t)_i \end{aligned}$$

Our MPC approach uses an optimization function to find the best acceleration $\boldsymbol{a}(t)_i$ at each time-step. Each bird optimizes its own acceleration based on local information about its nearest N_R neighboring birds. It tries to find the best accelerations of all of its neighbors including itself, and uses its own component of the solution to update its velocity and position. Let $\boldsymbol{x}_{N_i}$, $\boldsymbol{v}_{N_i}$ and $\boldsymbol{a}_{N_i}$ be the vector of positions, velocities and accelerations of bird i's neighbors. We consider the following optimization problem for bird i at time t:

$$\boldsymbol{a}^*_{N_i}(t), ..., \boldsymbol{a}^*_{N_i}(t+T-1) = \underset{\boldsymbol{a}_{N_i}(t),...,\boldsymbol{a}_{N_i}(t+T-1)}{\arg\min}\ J(\boldsymbol{a}_{N_i}(t+T-1), \boldsymbol{x}_{N_i}(t+T-1), \boldsymbol{v}_{N_i}(t+T-1))$$

$$\begin{aligned} \text{subject to}\quad & \boldsymbol{x}_{N_i}(t), \boldsymbol{v}_{N_i}(t) = \texttt{Neighbors}(i, \boldsymbol{x}(t), \boldsymbol{v}(t), N_R); \\ & \forall_{\tau \in [t, t+T-1]}, \boldsymbol{x}_{N_i}(\tau+1) = \boldsymbol{x}_{N_i}(\tau) + \boldsymbol{v}_{N_i}(\tau+1), \boldsymbol{v}_{N_i}(\tau+1) = \boldsymbol{v}_{N_i}(\tau) + \boldsymbol{a}_{N_i}(\tau); \\ & \forall_{i \leq N_R}\ ||\boldsymbol{v}_{N_i}(\tau)_i|| \leq \boldsymbol{v}_{max}, ||\boldsymbol{a}_{N_i}(\tau)_i|| \leq \delta ||\boldsymbol{v}_{N_i}(\tau)_i||, \delta \in (0,1). \end{aligned}$$

where T is the prediction horizon and J is the fitness function. Function `Neighbors` returns the positions and velocities of the nearest N_R birds of bird i (including i) at time t. We place constraints on the maximum velocities and accelerations. We apply $\boldsymbol{a}(t)_i = \boldsymbol{a}^*_{N_i}(t)_i$ as the optimal acceleration for bird i at time t.

We also consider a *centralized approach* in which birds have information about the entire flock, i.e. $N_R = N$. In this case, we only need to perform one optimization for all birds at each time-step. The fitness function J consists of a sum-of-squares combination of VM, CV

and UB. Let $\boldsymbol{v}' = \boldsymbol{v}(t) + \boldsymbol{a}(t)$ and $\boldsymbol{x}' = \boldsymbol{x}(t) + \boldsymbol{v}'$ be the new velocities and positions after applying the accelerations.

$$J(\boldsymbol{a}(t), \boldsymbol{x}(t), \boldsymbol{v}(t)) = (VM(\boldsymbol{v}') - VM^*)^2 + (CV(\boldsymbol{x}', \boldsymbol{v}') - CV^*)^2 + (UB(\boldsymbol{x}', \boldsymbol{v}') - UB^*)^2$$

where $VM^* = 0$, $CV^* = 0$, $UB^* = 1$ are the optimal values in a V-formation.

3 Fitness Metrics

Velocity Matching.

The velocity-matching metric is defined as $VM(\boldsymbol{v}) = \sum_{i>j} \left(\frac{||v_i - v_j||}{||v_i|| + ||v_j||}\right)^2$ where v_i is bird i's velocity. The optimal value in a V-formation is $VM^* = 0$, as all birds will have the same velocity, enabling them to maintain formation.

Clear View.

A bird's visual field is a cone with angle θ that can be blocked by the wings of other birds. We define the clear-view metric by accumulating the percentage of a bird's visual field that is blocked by other birds:

$$B_{ij}(h_{ij}, v_{ij}) = \begin{cases} \{\alpha | \max(\frac{\pi - \theta}{2}, \operatorname{atan}(\frac{v_{ij}}{h_{ij} + w})) \le \alpha \le \min(\frac{\pi + \theta}{2}, \operatorname{atan}(\frac{v_{ij}}{h_{ij} - w}))\} \\ \qquad \text{if } (h_{ij} < w \vee \frac{h_{ij} - w}{v_{ij}} < \tan\theta) \wedge \texttt{Front}(j, i); \\ \emptyset \qquad \text{otherwise.} \end{cases}$$

$$CV_i(\boldsymbol{x}, \boldsymbol{v}) = \frac{|\bigcup_{j \neq i} B_{ij}(h_{ij}, v_{ij})|}{\theta}, \quad CV(\boldsymbol{x}, \boldsymbol{v}) = \sum_i CV_i(\boldsymbol{x}, \boldsymbol{v})$$

where w is the wing span of a bird, and h_{ij} and v_{ij} are the horizontal and vertical distances between birds i and j w.r.t. the direction of i's velocity; these distances can be computed using $\boldsymbol{x}$ and $\boldsymbol{v}$. Each $B_{ij}(h_{ij}, v_{ij})$ computes the range of i's view angle being blocked by j, and is a sub-interval of $[(\pi - \theta)/2, (\pi + \theta)/2]$. $CV_i(\boldsymbol{x}, \boldsymbol{v})$ computes the percentage of i's view that is blocked by other birds. Predicate $\texttt{Front}(j, i)$ is true when bird j is in front of bird i. The optimal value in a V-formation is $CV^* = 0$, as all birds have a clear visual field.

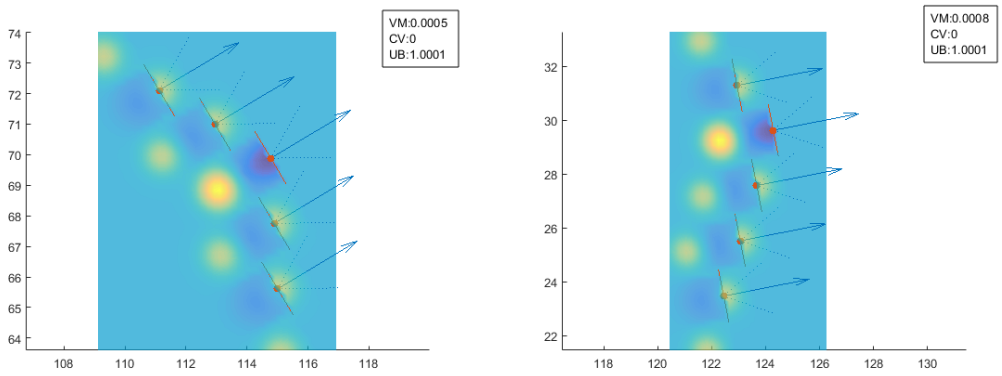
Upwash Benefit.

Upwash is generated near the wingtips of a bird, while downwash is generated near the center of a bird. We accumulate all birds' upwash benefits using a Gaussian-like model of the upwash and downwash region. The upwash and downwash a trailing bird i obtains from a preceding bird j is given by:

$$UB_{ij}(h_{ij}, v_{ij}) = \begin{cases} \frac{v_i \cdot v_j}{||v_i|| \cdot ||v_j||} S(h_{ij}) \cdot G(h_{ij}, v_{ij}, \mu_1, \Sigma_1) & \text{if } h_{ij} \ge \frac{(4-\pi)w}{8} \wedge \texttt{Front}(j, i) \\ S(h_{ij}) \cdot G(h_{ij}, v_{ij}, \mu_2, \Sigma_2) & \text{if } h_{ij} < \frac{(4-\pi)w}{8} \wedge \texttt{Front}(j, i) \\ 0 & \text{otherwise} \end{cases}$$

$$S(h_{ij}) = \texttt{erf}(2\sqrt{2}(h_{ij} - \tfrac{(4-\pi)w}{8})), \quad G(h_{ij}, v_{ij}, \mu, \Sigma) = e^{(-\frac{1}{2}([h_{ij}, v_{ij}] - \mu)^T \Sigma^{-1} ([h_{ij}, v_{ij}] - \mu))}$$

where w is the wing span, $h_{ij} = (4 - \pi)w/8$ is the boundary between the upwash and downwash regions [7], $S(h_{ij})$ is a smoothing function with $\texttt{erf}$ being the error function, and $G(h_{ij}, v_{ij}, \mu, \Sigma)$ is a Gaussian-like function. Parameters μ_1, μ_2 are chosen such that



(a) Centralized Control **(b)** Distributed Control, $N_R = 3$

Figure 1 Final formations from simulations with 5 birds. The red-filled circle and two protruding line segments represent a bird's body and wings with wing span $w = 1$. Arrows represent bird velocities. Dotted lines illustrate clear-view cones with angle $\theta = \pi/3$. A brighter background color indicates a higher upwash, while a darker background color indicates a higher downwash.

the upwash benefit is maximized when $h_{ij} = (12 + \pi)w/16$ [7] and $v_{ij} = 1$, and minimized when $h_{ij} = 0$ and $v_{ij} = 0$. Moreover, bird i only gets maximum upwash if the velocities of i and j are aligned; so the upwash is discounted by $\frac{v_i \cdot v_j}{||v_i|| \cdot ||v_j||}$. The total upwash benefit of the whole flock is $UB(\boldsymbol{x}, \boldsymbol{v}) = \sum_i (1 - min(\sum_j UB_{ij}(h_{ij}, v_{ij}), 1))$. The maximum upwash a bird can obtain is constrained to not be greater than 1. The optimal value in a V-formation $UB^* = 1$, as there is one leader that does not get any upwash.

4 Experimental Results

We used MATLAB function `particleswarm` from the Global Optimization Toolbox as the optimization algorithm. We placed a collision-avoidance constraint on the minimum distance between any two birds. The optimizer discards accelerations that will lead to collisions. The initial positions and velocities are randomly chosen with maximum velocity $\boldsymbol{v}_{max} = 5$. The bound on acceleration $\delta_{model} = 0.5$ for the model and $\delta_{plant} = 0.4$ for the plant. If the acceleration that the model produces exceeds the limit of the plant, we keep its direction and use the plant upper bound for its magnitude. We ran simulations of 50 time-steps with prediction horizon $T = 1$.

Fig. 1 shows the formations reached in the last step of simulation. We ran five simulations starting from random initial conditions for both centralized and distributed control. In both cases, four of the five simulations resulted in on-target formations. Future work will focus on further improving the success rate.

References

1 Eduardo F Camacho and Carlos Bordons Alba. *Model Predictive Control*. Springer Science & Business Media, 2013.

2 C Cutts and J Speakman. Energy savings in formation flight of pink-footed geese. *The Journal of Experimental Biology*, 189(1):251–261, 1994.

3 G Dimock and M Selig. The aerodynamic benefits of self-organization in bird flocks. *Urbana*, 51:61801, 2003.

4 Gary William Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, 1998.

5 WJ Hamilton. Social aspects of bird orientation mechanisms. *Animal Orientation and Navigation,*, pages 57–71, 1967.

6 Frank H Heppner, Jeffrey L Convissar, Dennis E Moonan Jr, and John GT Anderson. Visual angle and formation flight in Canada geese (Branta Canadensis). *The Auk,*, pages 195–198, 1985.

7 Dietrich Hummel. Aerodynamic aspects of formation flight in birds. *Journal of Theoretical Biology*, 104(3):321–347, 1983.

8 Andre Nathan and Valmir C Barbosa. V-like formations in flocks of artificial birds. *Artificial Life*, 14(2):179–188, 2008.

9 Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM Siggraph Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

10 Forrest Stonedahl and Uri Wilensky. Finding forms of flocking: Evolutionary search in ABM parameter-spaces. In *Multi-Agent-Based Simulation XI*, pages 61–75. Springer, 2011.

11 Henri Weimerskirch, Julien Martin, Yannick Clerquin, Peggy Alexandre, and Sarka Jiraskova. Energy saving in flight formation. *Nature*, 413(6857):697–698, 2001.

The Benefits of Duality in Verifying Concurrent Programs under TSO*

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[2], Ahmed Bouajjani[3], and Tuan Phong Ngo[4]

1 **Uppsala University, Sweden**
parosh@it.uu.se
2 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se
3 **IRIF, Université Paris Diderot & IUF, France**
abou@liafa.univ-paris-diderot.fr
4 **Uppsala University, Sweden**
tuan-phong.ngo@it.uu.se

Abstract

We address the problem of verifying safety properties of concurrent programs running over the TSO memory model. Known decision procedures for this model are based on complex encodings of store buffers as lossy channels. These procedures assume that the number of processes is fixed. However, it is important in general to prove correctness of a system/algorithm in a parametric way with an arbitrarily large number of processes. In this paper, we introduce an alternative (yet equivalent) semantics to the classical one for the TSO model that is more amenable for efficient algorithmic verification and for extension to parametric verification. For that, we adopt a *dual* view where *load buffers* are used instead of store buffers. The flow of information is now from the memory to load buffers. We show that this new semantics allows (1) to simplify drastically the safety analysis under TSO, (2) to obtain a spectacular gain in efficiency and scalability compared to existing procedures, and (3) to extend easily the decision procedure to the parametric case, which allows to obtain a new decidability result, and more importantly, a verification algorithm that is more general and more efficient in practice than the one for bounded instances.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Weak Memory Models, Reachability Problem, Parameterized Verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.5

1 Introduction

Most modern processor architectures execute instructions in an out-of-order manner to gain efficiency. In the context of *sequential* programming, this out-of-order execution is transparent to the programmer since one can still work under the Sequential Consistency (SC) model [24]. However, this is not true when we consider concurrent processes that share the memory. In fact, it turns out that concurrent algorithms such as mutual exclusion and producer-consumer protocols may not behave correctly any more. Therefore, program verification is a relevant (and difficult) task in order to prove correctness under the new semantics. The inadequacy of the interleaving semantics has led to the invention of new program semantics, so called

* This work was supported in part by the Swedish Research Council and carried out within the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

Weak (or relaxed) Memory Models (WMM), by allowing permutations between certain types of memory operations [7, 20, 8]. Total Store Ordering (TSO) is one of the the most common models, and it corresponds to the relaxation adopted by Sun's SPARC multiprocessors [28] and formalizations of the x86-tso memory model [26, 27]. These models put an unbounded perfect (non-lossy) *store buffer* between each process and the main memory where a store buffer carries the pending store operations of the process. When a process performs a store operation, it appends it to the end of its buffer. These operations are propagated to the shared memory non-deterministically in a FIFO manner. When a process reads a variable, it searches its buffer for a pending store operation on that variable. If no such a store operation exists, it fetches the value of the variable from the main memory. Verifying programs running on the TSO memory model poses a difficult challenge since the unboundedness of the buffers implies that the state space of the system is infinite even in the case where the input program is finite-state. Decidability of safety properties has been obtained by constructing equivalent models that replace the perfect store buffer by *lossy* channels [11, 12, 2]. However, these constructions are complicated and involve several ingredients that lead to inefficient verification procedures. For instance, they require each message inside a lossy channel to carry (instead of a single store operation) a full snapshot of the memory representing a local view of the memory contents by the process. Furthermore, the reductions involve non-deterministic guessing the lossy channel contents. The guessing is then resolved either by consistency checking [11] or by using explicit pointer variables (each corresponding to one process) inside the buffers [2], causing a serious state space explosion problem.

In this paper, we introduce a novel semantics which we call the *dual TSO* semantics. Our aim is to provide an alternative (and equivalent) semantics that is more amenable for efficient algorithmic verification. The main idea is to have *load buffers* that contain pending load operations (more precisely, values that will potentially be taken by forthcoming load operations) rather than store buffers (that contain store operations). The flow of information will now be in the reverse direction, i.e., store operations are performed by the processes atomically on the main memory, while values of variables are propagated non-deterministically from the memory to the load buffers of the processes. When a process performs a load operation it can fetch the value of the variable from the head of its load buffer. We show that the dual semantics is equivalent to the original one in the sense that any given set of processes will reach the same set of local states under both semantics. The dual semantics allows us to understand the TSO model in a totally different way compared to the classical semantics. Furthermore, the dual semantics offers several important advantages from the point of view of formal reasoning and program verification. First, the dual semantics allows transforming the load buffers to *lossy* channels without adding the costly overhead that was necessary in the case of store buffers. This means that we can apply the theory of *well-structured systems* [6, 5, 21] in a straightforward manner leading to a much simpler proof of decidability of safety properties. Second, the absence of extra overhead means that we obtain more efficient algorithms and better scalability (as shown by our experimental results). Finally, the dual semantics allows extending the framework to perform *parameterized verification* which is an important paradigm in concurrent program verification. Here, we consider systems, e.g., mutual exclusion protocols, that consist of an arbitrary number of processes. The aim of parameterized verification is to prove correctness of the system regardless of the number of processes. It is not obvious how to perform parameterized verification under the classical semantics. For instance, extending the framework of [2], would involve an unbounded number of pointer variables, thus leading to channel systems with unbounded message alphabets. In contrast, as we show in this paper, the simple nature of the dual

semantics allows a straightforward extension of our verification algorithm to the case of parameterized verification. This is the first time a decidability result is established for the parametrized verification of programs running over WMM. Notice that this result is taking into account two sources of infinity: the number of processes, and the size of the buffers.

Based on our framework, we have implemented a tool and applied it to a large set of benchmarks. The experiments demonstrate the efficiency of the dual semantics compared to the classical one (by two order of magnitude in average), and the feasibility of parametrized verification in the former case. In fact, besides its theoretical generality, parametrized verification is practically crucial in this setting: as our experiments show, it is much more efficient than verification of bounded-size instances (starting from a number of components of 3 or 4), especially concerning memory consumption (which is the critical resource).

Related Work. There have been a lot of works related to the analysis of programs running under WMM (e.g., [25, 22, 23, 17, 2, 15, 16, 13, 14, 29]). Some of these works propose precise analysis techniques for checking safety properties or stability of finite-state programs under WMM (e.g., [2, 13, 19, 4]). Others propose stateless model-checking techniques for programs under TSO and PSO (e.g., [1, 30, 18]). Different other techniques based on monitoring and testing have also been developed during these last years (e.g., [15, 16, 25]). There are also a number of efforts to design bounded model checking techniques for programs under WMM (e.g., [9, 29, 14]) which encode the verification problem in SAT/SMT.

The closest works to ours are those presented in [2, 11, 3, 12] which provide precise and sound techniques for checking safety properties for finite-state programs running under TSO. However, as stated in the introduction, these techniques are complicated and can not be extended, in a straightforward manner, to the verification of parameterized systems (as it is the case of the developed techniques for the dual TSO semantics).

In Section 7, we experimentally compare our techniques with Memorax [2, 3] which is the only precise and sound tool for checking safety properties for programs under TSO.

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* (resp. Σ^+) to denote the set of all *words* (resp. non-empty words) over Σ. Let ϵ be the empty word. The length of a word $w \in \Sigma^*$ is denoted by $|w|$ (and in particular $|\epsilon| = 0$). For every $i : 1 \leq i \leq |w|$, let $w(i)$ be the symbol at position i in w. For $a \in \Sigma$, we write $a \in w$ if a appears in w, i.e., $a = w(i)$ for some $i : 1 \leq i \leq |w|$.

Given two words u and v over Σ, we use $u \preceq v$ to denote that u is a (not necessarily contiguous) subword of v (i.e., if there is an injection $h : \{1, \ldots, |u|\} \mapsto \{1, \ldots, |v|\}$ such that: (1) $h(i) < h(j)$ for all $i < j$, and (2) for every $i \in \{1, \ldots, |u|\}$, we have $u(i) = v(h(i))$).

Given a subset $\Sigma' \subseteq \Sigma$ and a word $w \in \Sigma^*$, we use $w|_{\Sigma'}$ to denote the projection of w over Σ', i.e., the word obtained from w by erasing all the symbols that are not in Σ'.

Let A and B be two sets and let $f : A \mapsto B$ be a total function from A to B. We use $f[a \hookleftarrow b]$ to denote the function f' such that $f'(a) = b$ and $f'(a') = f(a')$ for all $a' \neq a$.

A transition system $\mathcal{T}$ is a tuple $\left(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \cup_{\mathtt{a} \in \mathtt{Act}} \xrightarrow{a}\right)$ where $\mathtt{C}$ is a (potentially infinite) set of *configurations*, $\mathtt{Init} \subseteq \mathtt{C}$ is a set of *initial configurations*, $\mathtt{Act}$ is a set of actions, and for every $a \in \mathtt{Act}$, $\xrightarrow{a} \subseteq \mathtt{C} \times \mathtt{C}$ is a *transition relation*. We use $c \xrightarrow{a} c'$ to denote that $(c, c') \in \xrightarrow{a}$. Let $\rightarrow = \cup_{a \in \mathtt{Act}} \xrightarrow{a}$. A *run* π of $\mathcal{T}$ is of the form $c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} c_n$, where $c_i \xrightarrow{a_{i+1}} c_{i+1}$ for all $i : 0 \leq i < n$. Then, we write $c_0 \xrightarrow{\pi} c_n$. We use *target* (π) to denote the configuration c_n. The run π is said to be a *computation* if $c_0 \in \mathtt{Init}$. Two runs $\pi_1 = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} c_m$ and $\pi_2 = c_{m+1} \xrightarrow{a_{m+2}} c_{m+2} \xrightarrow{a_{m+3}} \cdots \xrightarrow{a_n} c_n$ are *compatible* if $c_m = c_{m+1}$. Then, we write

$\pi_1 \bullet \pi_2$ to denote the run $\pi = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} c_m \xrightarrow{a_{m+2}} c_{m+2} \xrightarrow{a_{m+3}} \cdots \xrightarrow{a_n} c_n$. For two configurations c and c', we use $c \xrightarrow{*} c'$ to denote that $c \xrightarrow{\pi} c'$ for some run π. A configuration c is said to be *reachable* in $\mathcal{T}$ if $c_0 \xrightarrow{*} c$ for some $c_0 \in \texttt{Init}$, and a set C of configurations is said to be *reachable* in $\mathcal{T}$ if some $c \in C$ is reachable in $\mathcal{T}$.

3 Concurrent Systems

In this section, we define the syntax we use for *concurrent programs*, a model for representing communicating concurrent processes. Communication between processes is performed through a shared memory that consists of a finite number of shared variables (over finite domains) to which all processes can read and write. Then we recall the classical TSO semantics including the transition system it induces and its reachability problem. Next, we introduce the *Dual TSO* semantics and its induced transition system. Finally, we state the equivalence between the two semantics (for a given concurrent program, we can reduce its reachability problem under the classical TSO to its reachability problem under Dual TSO and vice-versa).

3.1 Syntax

Let V be a finite data domain and X be a finite set of variables. We assume w.l.o.g. that V contains the value 0. Let $\Omega(X, V)$ be the smallest set of memory operations that contains (1) *"no operation"* nop, (2) *read operation* $\mathsf{r}(x, v)$, (3) *write operation* $\mathsf{w}(x, v)$, (4) *fence operation* fence, and (5) *atomic read-write operation* $\mathsf{arw}(x, v, v')$, where $x \in X$, and $v, v' \in V$.

A *concurrent system* is a tuple $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ where for every $p : 1 \leq p \leq n$, A_p is a finite-state automaton describing the behavior of the process p. The automaton A_p is defined as a triple $\left(Q_p, q_p^{init}, \Delta_p\right)$ where Q_p is a finite set of *local states*, $q_p^{init} \in Q_p$ is the *initial* local state, and $\Delta_p \subseteq Q_p \times \Omega(X, V) \times Q_p$ is a finite set of *transitions*. We define $P = \{1, \ldots, n\}$ to be the set of process IDs, $Q := \cup_{p \in P} Q_p$ to be the set of all local states and $\Delta := \cup_{p \in P} \Delta_p$ to be the set of all transitions.

3.2 Classical TSO semantics

In the following, we recall the semantics of concurrent systems under the classical TSO model as formalized in [26, 27]. To do that, we define the set of configurations and the induced transition relation. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system.

Configurations. A *TSO-configuration* c is a triple $(\mathbf{q}, \mathbf{b}, mem)$ where (1) $\mathbf{q} : P \mapsto Q$ is the *global state* of $\mathcal{P}$ mapping each process $p \in P$ to a local state in Q_p (i.e., $\mathbf{q}(p) \in Q_p$), (2) $\mathbf{b} : P \mapsto (X \times V)^*$ gives the content of the store buffer of each process, and (3) $mem : X \mapsto V$ defines the value of each shared variable. Observe that the store buffer of each process contains a sequence of write operations, where each write operation is defined by a pair, namely a variable x and a value v that is assigned to x. The initial TSO-configuration c_{init} is defined by the tuple $(\mathbf{q}_{init}, \mathbf{b}_{init}, mem_{init})$ where, for all $p \in P$ and $x \in X$, we have that $\mathbf{q}_{init}(p) = q_p^{init}$, $\mathbf{b}_{init}(p) = \epsilon$ and $mem_{init}(x) = 0$. In other words, each process is in its initial local state, all the buffers are empty, and all the variables in the shared memory are initialized to 0. We use $\mathtt{C}_{TSO}$ to denote the set of TSO-configurations.

Transition Relation. The transition relation $\rightarrow_{TSO}$ between TSO-configurations is given by a set of rules, described in Figure 1. Here we informally explain these rules. A nop transition $(q, \mathsf{nop}, q') \in \Delta_p$ changes only the local state of the process p from q to q'. A

$$\frac{t = (q, \mathsf{nop}, q') \quad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}, mem)} \quad \text{Nop}$$

$$\frac{t = (q, \mathsf{w}(x, v), q') \quad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}[p \hookleftarrow (x, v) \cdot \mathbf{b}(p)], mem)} \quad \text{Write}$$

$$\frac{t = \mathsf{update}_p}{(\mathbf{q}, \mathbf{b}[p \hookleftarrow \mathbf{b}(p) \cdot (x, v)], mem) \xrightarrow{t}_{TSO} (\mathbf{q}, \mathbf{b}, mem[x \hookleftarrow v])} \quad \text{Update}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \quad \mathbf{q}(p) = q \quad \mathbf{b}(p)|_{\{x\} \times V} = (x, v) \cdot w}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}, mem)} \quad \text{Read-Own-Write}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \quad \mathbf{q}(p) = q \quad \mathbf{b}(p)|_{\{x\} \times V} = \epsilon \quad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}, mem)} \quad \text{Read Memory}$$

$$\frac{t = (q, \mathsf{arw}(x, v, v'), q') \quad \mathbf{q}(p) = q \quad \mathbf{b}(p) = \epsilon \quad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}, mem[x \hookleftarrow v'])} \quad \text{ARW}$$

$$\frac{t = (q, \mathsf{fence}, q') \quad \mathbf{q}(p) = q \quad \mathbf{b}(p) = \epsilon}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}[p \hookleftarrow q'], \mathbf{b}, mem)} \quad \text{Fence}$$

Figure 1 The transition relation $\rightarrow_{TSO}$ under TSO. Here $p \in P$ and $t \in \Delta_p \cup \{\mathsf{update}_p\}$ where update_p is an operation that updates the memory using the oldest message in the buffer of process p.

write transition $(q, \mathsf{w}(x, v), q') \in \Delta_p$ adds the message (x, v) to the tail of the store buffer of the process p. A memory update transition update_p can be performed at any time by removing the oldest message in the store buffer of the process p and updating the memory accordingly. For a read transition $(q, \mathsf{r}(x, v), q') \in \Delta_p$, if the store buffer of the process p contains some write operations to x, then the read value v must correspond to the value of the most recent such a write operation. Otherwise the value v of x is fetched from the memory. A fence transition may be performed by a process p only if its store buffer is empty. Finally, an atomic read-write transition $(q, \mathsf{arw}(x, v, v'), q') \in \Delta_p$ can be performed by the process p only if its store buffer is empty. This operation checks then whether the value of x is v and changes it to v'.

We use $c \rightarrow_{TSO} c'$ to denote that $c \xrightarrow{t}_{TSO} c'$ for some $t \in \Delta \cup \Delta'$ where $\Delta' := \{\mathsf{update}_p | \ p \in P\}$. The transition system induced by $\mathcal{P}$ under the classical TSO semantics is then given by $\mathcal{T}_{TSO} = (\mathsf{C}_{TSO}, \{\mathsf{c}_{init}\}, \Delta \cup \Delta', \rightarrow_{TSO})$.

The TSO Reachability Problem. A global state $\mathbf{q}_{target}$ is said to be reachable in $\mathcal{T}_{TSO}$ iff there is a TSO-configuration c of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that c is reachable in $\mathcal{T}_{TSO}$. The reachability problem for the concurrent system $\mathcal{P}$ under TSO asks, for a given global state $\mathbf{q}_{target}$, whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{TSO}$. Observe that, in the definition of the reachability problem, we require that the buffers of the configuration c must be empty instead of being arbitrary. This is only for sake of simplicity and does not constitute a restriction. Indeed, we can easily show that the "arbitrary buffer" reachability problem is reducible to the "empty buffer" reachability problem.

3.3 Dual TSO semantics

In this section, we define the Dual TSO semantics. The model has a perfect FIFO *load* buffer between the main memory and each process. The *load* buffer is used to store *potential read* operations that will be performed by the process. Each message in the load buffer of a process p is either a pair of the form (x, v) or a triple of the form (x, v, own) where $x \in X$ and $v \in V$. A message of the form (x, v) corresponds to the fact that x had the value v in the shared memory. While a message (x, v, own) corresponds to the fact that the process p has written the value v to x. A write operation $\mathsf{w}(x, v)$ of the process p immediately updates the shared memory and a message of the form (x, v, own) is then appended to the tail of the load buffer of p. Read propagation is then performed by non-deterministically choosing a variable (let's say x and its value is v in the shared memory) and appending the message (x, v) to the tail of the load buffer of p. This propagation operation speculates on a read operation of p on x that will be performed later on. Moreover, any message at the head of the load buffer can be removed at any time. A read operation $\mathsf{r}(x, v)$ of the process p can be executed if the message at the head of the load buffer (i.e., the oldest one) of p is of the form (x, v) and there is no pending message of the form (x, v', own). In the case that the load buffer contains a message belonging to p (i.e., of the form (x, v', own)), the read value must correspond to the value of the most recent message belonging to p (implicitly, this allows to simulate the Read-Own-Write transitions). A fence means that the load buffer of p must be empty before p can continue. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system.

Configurations. A *DTSO-configuration* c is a triple $(\mathbf{q}, \mathbf{b}, mem)$ where $\mathbf{q} : P \mapsto Q$ is the *global state* of $\mathcal{P}$, $\mathbf{b} : P \mapsto ((X \times V) \cup (X \times V \times \{own\}))^*$ is the content of the load buffer, and $mem : X \mapsto V$ gives the value of each variable. The initial DTSO-configuration c^D_{init} is defined by $(\mathbf{q}_{init}, \mathbf{b}_{init}, mem_{init})$ where, for all $p \in P$ and $x \in X$, we have that $\mathbf{q}_{init}(p) = q^{init}_p$, $\mathbf{b}_{init}(p) = \epsilon$ and $mem_{init}(x) = 0$. We use C_{DTSO} to denote the set of DTSO-configurations.

Transition Relation. The transition relation $\rightarrow_{DTSO}$ induced by the Dual TSO semantics is given in Figure 2. This relation is induced by members of Δ and $\Delta_{aux} := \{\mathsf{propagate}^x_p, \mathsf{delete}_p \mid p \in P,\ x \in X\}$. $\mathsf{propagate}^x_p$ is an operation that speculates on a read operation of p over x that will be executed later. This is done by appending the message (x, v) to the tail of the load buffer of p where v is the current value of x in the shared memory. The operation delete_p removes the oldest message in the load buffer of process p. A write operation $\mathsf{w}(x, v)$ updates the memory and appends the message (x, v, own) to the tail of the load buffer. A read operation $\mathsf{r}(x, v)$ checks first if the load buffer contains a message of the form (x, v', own), and in that case the read value v should correspond to the value of the most recent message of that form. If there is no message on the variable x belonging to p in its load buffer then the value v of x is fetched from the message at the head of its load buffer.

We use $c \rightarrow_{DTSO} c'$ to denote that $c \xrightarrow{t}_{DTSO} c'$ for some $t \in \Delta \cup \Delta_{aux}$. The transition system induced by $\mathcal{P}$ under the Dual TSO semantics is then given by $\mathcal{T}_{DTSO} = (\mathsf{C}_{DTSO}, \{\mathsf{c}^{\mathsf{D}}_{init}\}, \Delta \cup \Delta_{\mathsf{aux}}, \rightarrow_{DTSO})$.

The Dual TSO Reachability Problem. The reachability problem for $\mathcal{P}$ under the Dual TSO semantics is defined in a similar manner to the case of TSO. A global state $\mathbf{q}_{target}$ is said to be reachable in $\mathcal{T}_{DTSO}$ iff there is a DTSO-configuration c of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that c is reachable in $\mathcal{T}_{DTSO}$. Then, the reachability problem

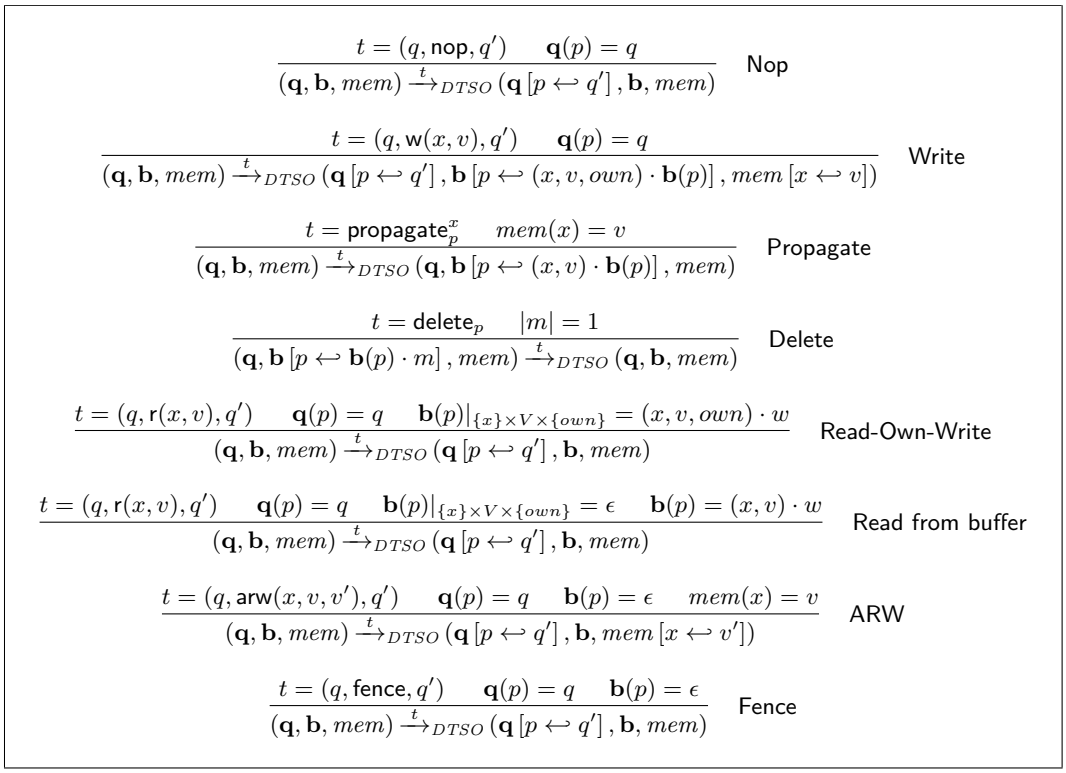


Figure 2 The induced transition relation $\rightarrow_{DTSO}$ under the Dual TSO semantics. Here $p \in P$ and $t \in \Delta_p \cup \Delta'_p$ where $\Delta'_p := \left\{\mathsf{propagate}_p^x, \mathsf{delete}_p | \ x \in X\right\}$.

consists in checking whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. The following theorem states equivalence of the reachability problems under the TSO and Dual TSO semantics.

▶ **Theorem 1.** *A global state $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{TSO}$ iff $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{DTSO}$.*

4 The Dual TSO Reachability Problem

In this section, we show the decidability of the Dual TSO reachability problem by making use of the framework of *Well-Structured Transition Systems* (WSTS) [5, 21]. First, we briefly recall the framework of WSTS and then we instantiate it to show the decidability of the Dual TSO reachability problem.

4.1 Well-Structured Transition Systems

Let $\mathcal{T} = \left(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \cup_{\mathtt{a} \in \mathtt{Act}} \xrightarrow{a}\right)$ be a transition system. Let $\sqsubseteq$ be a *well-quasi ordering* on $\mathtt{C}$. Recall that a well-quasi ordering on $\mathtt{C}$ is a binary relation over $\mathtt{C}$ that is reflexive and transitive and for every infinite sequence $(c_i)_{i \geq 0}$ of elements in $\mathtt{C}$ there exist $i, j \in \mathbb{N}$ such that $i < j$ and $c_i \sqsubseteq c_j$. A set $\mathtt{U} \subseteq \mathtt{C}$ is called *upward closed* if for every $c \in \mathtt{U}$ and $c' \in \mathtt{C}$ with $c \sqsubseteq c'$, we have $c' \in \mathtt{U}$. It is known that every upward closed set $\mathtt{U}$ can be characterised by a finite *minor set* $\mathtt{M} \subseteq \mathtt{U}$ such that: (i) for every $c \in \mathtt{U}$ there is $c' \in \mathtt{M}$ such that $c' \sqsubseteq c$, and (ii) if $c, c' \in \mathtt{M}$ and $c \sqsubseteq c'$ then $c = c'$. We use $\mathtt{min}$ to denote the function which for a given upward closed set $\mathtt{U}$ returns its minor set.

Let $\mathtt{D} \subseteq \mathtt{C}$. The upward closure of $\mathtt{D}$ is defined as $\mathtt{D}\uparrow := \{c' \in \mathtt{C} \mid \exists c \in \mathtt{D} \text{ with } c \sqsubseteq c'\}$. We also define the set of predecessors of $\mathtt{D}$ as $\mathtt{Pre}_{\mathcal{T}}(\mathtt{D}) := \{c \mid \exists c_1 \in \mathtt{D}, c \rightarrow c_1\}$. For a finite set of configurations $\mathtt{M} \subseteq \mathtt{C}$, we use *minpre* $(\mathtt{M})$ to denote $\mathtt{min}(\mathtt{Pre}_{\mathcal{T}}(\mathtt{M}\uparrow) \cup \mathtt{M}\uparrow)$.

The transition relation $\rightarrow$ is said to be *monotonic* wrt. $\sqsubseteq$ if, given $c_1, c_2, c_3 \in \mathtt{C}$ s.t. $c_1 \rightarrow c_2$ and $c_1 \sqsubseteq c_3$, we can compute a configuration $c_4 \in \mathtt{C}$ and a run π s.t. $c_3 \xrightarrow{\pi} c_4$ and $c_2 \sqsubseteq c_4$. The pair $(\mathcal{T}, \sqsubseteq)$ is called *monotonic transition system* if $\rightarrow$ is *monotonic* wrt. $\sqsubseteq$.

Given a *finite* set of configurations $\mathtt{M} \subseteq \mathtt{C}$, the *coverability problem* of $\mathtt{M}$ in the monotonic transition system $(\mathcal{T}, \sqsubseteq)$ asks whether the set $\mathtt{M}\uparrow$ is reachable in $\mathcal{T}$. For the decidability of this problem the following **three conditions** are sufficient: **(i)** For every two configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$, **(ii)** for every $c \in \mathtt{C}$, we can check whether $\{c\}\uparrow \cap \mathtt{Init} \neq \emptyset$, and **(iii)** for every $c \in \mathtt{C}$, the set *minpre* (c) is finite and computable.

The solution for the coverability problem as suggested in [5, 21] is based on a backward analysis approach. It is shown that starting from a finite set $\mathtt{M}_0 \subseteq \mathtt{C}$, the sequence $(\mathtt{M}_i)_{i \geq 0}$ with $\mathtt{M}_{i+1} :=$ *minpre* $(\mathtt{M}_i)$, for $i \geq 0$ reaches a fixpoint and is computable.

4.2 Dual TSO Transition System is a WSTS

In this section, we instantiate the framework of WSTS to show the following result:

▶ **Theorem 2.** *The Dual TSO reachability problem is decidable.*

The rest of this section is devoted to the proof of the above theorem. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system (as defined in Section 3). Let $\mathcal{T}_{DTSO} = (\mathtt{C}_{DTSO}, \{\mathtt{c}^{\mathtt{D}}_{init}\}, \Delta \cup \Delta_{\mathtt{aux}}, \rightarrow_{DTSO})$ be the transition system induced by $\mathcal{P}$ under the Dual TSO semantics (as defined in Section 3.3). In the following, we will first define a well-quasi ordering $\sqsubseteq$ on the set of DTSO-configurations (Lemma 4) such that for every two configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$. Then we show that the transition system induced under the Dual TSO semantics is monotonic wrt. to $\sqsubseteq$ (Lemma 5). We will show also that the Dual TSO reachability problem for $\mathcal{P}$ can be reduced to the coverability problem in the monotonic transition system $(\mathcal{T}_{DTSO}, \sqsubseteq)$ (Lemma 6). (Observe that this reduction is needed since we require that the load buffers are empty when defining the Dual TSO reachability problem.) The second sufficient condition (i.e., checking whether the upward closed set $\{c\}\uparrow$, with c is a DTSO-configuration, contains an initial configuration) for the decidability of the coverability problem is trivial. This check boils down to verifying whether c is an initial configuration. Finally, the computability of the set of minimal configurations for the set of predecessors of any upward closed set is stated by the following lemma:

▶ **Lemma 3.** *For any configuration c, we can compute minpre*$(\{c\})$.

Well-quasi Ordering. In the following, we define a well-quasi ordering $\sqsubseteq$ on $\mathtt{C}_{DTSO}$. Let us first introduce some notations and definitions. Consider a word $w \in ((X \times V) \cup (X \times V \times \{own\}))^*$ representing the content of a load buffer. We define an operation that divides w into a number of fragments according to the most-recent own-messages concerning each variable. We define $[w]_{own} := (w_1, (x_1, v_1, own), w_2, \ldots, w_m, (x_m, v_m, own), w_{m+1})$, where the following conditions are satisfied: (1) $x_i \neq x_j$ if $i \neq j$, (2) if $(x, v, own) \in w_i$ then $x = x_j$ for some $j < i$ (i.e., the most recent own-message on x_j occurs at position j), and (3) $w = w_1 \cdot (x_1, v_1, own) \cdot w_2 \cdots w_m \cdot (x_m, v_m, own) \cdot w_{m+1}$ (i.e., the fragments correspond to the given word w).

Let $w, w' \in ((X \times V) \cup (X \times V \times \{own\}))^*$ be two words. Let us assume that $[w]_{own} = (w_1, (x_1, v_1, own), w_2, \ldots, w_r, (x_r, v_r, own), w_{r+1})$, and $[w']_{own} = (w'_1, (x'_1, v'_1, own), w'_2, \ldots, w'_m, (x'_m, v'_m, own), w'_{m+1})$. We write $w \sqsubseteq w'$ to denote that the following conditions are satisfied: (i) $r = m$, (ii) $x'_i = x_i$ and $v'_i = v_i$ for all $i : 1 \leq i \leq m$, and (iii) $w_i \preceq w'_i$ for all $i : 1 \leq i \leq m+1$.

Consider two DTSO-configurations $c = (\mathbf{q}, \mathbf{b}, mem)$ and $c' = (\mathbf{q}', \mathbf{b}', mem')$, we extend the ordering $\sqsubseteq$ to configurations as follows: $c \sqsubseteq c'$ iff the following conditions are satisfied: (i) $\mathbf{q} = \mathbf{q}'$, (ii) $\mathbf{b}(p) \sqsubseteq \mathbf{b}'(p)$ for all process $p \in P$, and (iii) $mem' = mem$. The following lemma shows that $\sqsubseteq$ is indeed a well-quasi ordering.

▶ **Lemma 4.** *The relation $\sqsubseteq$ is a well-quasi ordering over $\mathtt{C}_{DTSO}$. Furthermore, for every two DTSO-configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$.*

Monotonicity. Given configurations $c_1 = (\mathbf{q}_1, \mathbf{b}_1, mem_1), c_2 = (\mathbf{q}_2, \mathbf{b}_2, mem_2), c_3 = (\mathbf{q}_3, \mathbf{b}_3, mem_3) \in \mathtt{C}_{DTSO}$ such that $c_1 \xrightarrow{t}_{DTSO} c_2$ for some transition $t \in \Delta_p \cup \{\mathsf{propagate}_p^x, \mathsf{delete}_p | \; x \in X\}$, with $p \in P$, and $c_1 \sqsubseteq c_3$, we will show that it is possible to compute a configuration $c_4 \in \mathtt{C}_{DTSO}$ and a run π such that $c_3 \xrightarrow{\pi}_{DTSO} c_4$ and $c_2 \sqsubseteq c_4$. To that aim, we first show that it is possible from c_3 to reach a configuration c'_3, by performing a certain number of delete_p operations, such that the process p will have the same last message in its load buffer in the configurations c_1 and c'_3 while $c_1 \sqsubseteq c'_3$. Then, from the configuration c'_3, the process p can perform the same transition t as c_1 did in order to reach the configuration c_4 such that $c_2 \sqsubseteq c_4$. Let us assume that $[\mathbf{b}_1(p)]_{own}$ is of the form $\langle w_1, (x_1, v_1, own), w_2, \ldots, w_m, (x_m, v_m, own), w_{m+1} \rangle$, and $[\mathbf{b}_3(p)]_{own}$ is of the form $\langle w'_1, (x'_1, v'_1, own), w'_2, \ldots, w'_m, (x'_m, v'_m, own), w'_{m+1} \rangle$. We define the word $w \in ((X \times V) \cup (X \times V \times \{own\}))^*$ to be the longest word such that $w'_{m+1} = w' \cdot w$ with $w_{m+1} \preceq w'$. Observe that in this case we have either $w_{m+1} = w' = \epsilon$ or $w'(|w'|) = w_{m+1}(|w_{m+1}|)$. Then, after executing a certain number $|w|$ of delete_p operations from the configuration c_3, one can obtain a configuration $c'_3 = (\mathbf{q}_3, \mathbf{b}'_3, mem_3)$ such that $\mathbf{b}_3 = \mathbf{b}'_3[p \hookleftarrow \mathbf{b}'_3(p) \cdot w]$. As a consequence, we have $c_1 \sqsubseteq c'_3$. Furthermore, since c_1 and c'_3 have the same global state, the same memory valuation, the same sequence of most-recent own messages concerning each variable, and the same last message in the load buffer of p, c'_3 can perform the transition t and reaches a configuration c_4 such that $c_2 \sqsubseteq c_4$. The following lemma shows that $(\mathcal{T}_{DTSO}, \sqsubseteq)$ is monotonic system.

▶ **Lemma 5.** *The relation $\rightarrow_{DTSO}$ is monotonic wrt. $\sqsubseteq$.*

From Reachability to Coverability. Let $\mathbf{q}_{target}$ be a global state of $\mathcal{P}$ and $\mathtt{M}_{target}$ be the set of DTSO-configurations of the form $c = (\mathbf{q}_{target}, \mathbf{b}, mem)$ with $\mathbf{b}(p) = \epsilon$ for all $p \in P$. Next, we show that the reachability problem of $\mathbf{q}_{target}$ in $\mathcal{T}_{DTSO}$ can be reduced to the coverability problem of $\mathtt{M}_{target}$ in $(\mathcal{T}_{DTSO}, \sqsubseteq)$. Recall that $\mathbf{q}_{target}$ in $\mathcal{T}_{DTSO}$ iff $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. Let us assume that $\mathtt{M}_{target}{\uparrow}$ is reachable in $\mathcal{T}_{DTSO}$. This means that there is a configuration $c \in \mathtt{M}_{target}{\uparrow}$ which is reachable in $\mathcal{T}_{DTSO}$. Let us assume that c is of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$. Then, from the configuration c, it is possible to reach the configuration $c' = (\mathbf{q}_{target}, \mathbf{b}', mem)$, with $\mathbf{b}'(p) = \epsilon$ for all $p \in P$, by performing a sequence of delete_p operations to empty the load buffer of each process. It is then easy to see that $c' \in \mathtt{M}_{target}$ and so $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. The other direction of the following lemma is trivial since $\mathtt{M}_{target} \subseteq \mathtt{M}_{target}{\uparrow}$.

▶ **Lemma 6.** *$\mathtt{M}_{target}{\uparrow}$ is reachable in $\mathcal{T}_{DTSO}$ iff $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$.*

5 Parameterized Concurrent Systems

Let V be a finite data domain and X be a finite set of variables ranging over V. A *parameterized concurrent system* (or simply a *parameterized system*) consists of an unbounded number of identical processes running under the Dual TSO semantics. Formally, a parameterized system $\mathcal{S}$ is defined by a finite-state automaton $A = \left(Q, q^{init}, \Delta\right)$ uniformly describing the behavior of each process. An *instance* of $\mathcal{S}$ is a concurrent system $\mathcal{P} = (A_1, A_2, \ldots, A_n)$, for some $n \in \mathbb{N}$, where for every $p : 1 \leq p \leq n$, we have $A_p = A$. In other words, it consists of a finite set of processes each running the same code defined by A. Observe that considering that all processes run the same code is not a real restriction. In fact, the case where the processes run (finitely many) different finite-state automata $A_1, A_2, \ldots, A_m$ can be easily encoded in our model by constructing an extended automaton A which represents the *union* of these automata $A_1, A_2, \ldots, A_m$. We use $Inst(\mathcal{S})$ to denote all possible instances of $\mathcal{S}$. We use $\mathcal{T}_{\mathcal{P}} = (\mathtt{C}_{\mathcal{P}}, \mathtt{Init}_{\mathcal{P}}, \mathtt{Act}_{\mathcal{P}}, \rightarrow_{\mathcal{P}})$ to denote the transition system induced by an instance $\mathcal{P}$ of $\mathcal{S}$ under the Dual TSO semantics.

A parameterized configuration α is a pair (P, c) where $P = \{1, \ldots, n\}$, with $n \in \mathbb{N}$, is the set of process IDs and c is a DTSO-configuration of an instance $\mathcal{P}$ of $\mathcal{S}$ of the form $(A_1, A_2, \ldots, A_n)$. The parameterized configuration $\alpha = (P, c)$ is said to be initial if c is an initial configuration of $\mathcal{P}$ (i.e., $c \in \mathtt{Init}_{\mathcal{P}}$). We use $\mathtt{C}$ (resp. $\mathtt{Init}$) to denote the set of all the parameterized configurations (resp. initial configurations) of $\mathcal{S}$.

Let $\mathtt{Act}$ denote the set of actions of all possible instances of $\mathcal{S}$ (i.e., $\mathtt{Act} = \cup_{\mathcal{P} \in Inst(\mathcal{S})} \mathtt{Act}_{\mathcal{P}}$). We define a transition relation $\rightarrow$ on parameterized configurations such that $(P, c) \xrightarrow{t} (P', c')$ for some action $t \in \mathtt{Act}$ iff $P' = P$ and there is an instance $\mathcal{P}$ of $\mathcal{S}$ such that $t \in \mathtt{Act}_{\mathcal{P}}$ and $c \rightarrow_{\mathcal{P}} c'$. The transition system induced by $\mathcal{S}$ is given by $\mathcal{T} = (\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \rightarrow)$.

In the following we extend the definition of the Dual TSO reachability problem to the case of parameterized systems. A global state $\mathbf{q}_{target} : P' \mapsto Q$ is said to be reachable in $\mathcal{T}$ if there exists a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$. Then, the reachability problem consists in checking whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}$. In other words, the Dual TSO reachability problem for parameterized systems asks whether there is an instance of the parameterized system that reaches a configuration with a number of processes in certain given local states.

6 Decidability of the Parameterized Verification Problem

We prove hereafter the following fact:

▶ **Theorem 7.** *The Dual TSO reachability problem for parameterized systems is decidable.*

Let $\mathcal{S} = \left(Q, q^{init}, \Delta\right)$ be a parameterized system and $(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \rightarrow)$ be its induced transition system. The proof of Theorem 7 is done by instantiating the framework of WSTS. Following that framework, we will first define an ordering that we denote by $\trianglelefteq$ on the set of parameterized configurations and show the monotonicity of the the relation $\rightarrow$ wrt. this ordering (see Lemma 9 and Lemma 10). Then we will show that the Dual TSO reachability problem for $\mathcal{S}$ can be reduced to the coverability problem in the monotonic transition system $(\mathcal{T}, \trianglelefteq)$ (Lemma 11). The second sufficient condition (i.e., checking whether the upward closed set $\{\alpha\}\uparrow$, with α is a parameterized configuration, contains an initial configuration) for the decidability of the coverability problem is trivial. This check boils down to whether the configuration α is initial. Finally, the last sufficient condition (i.e., computing the set

of minimal configurations for the set of predecessors of any upward closed set) for the decidability of the coverability problem is stated by the following lemma:

▶ **Lemma 8.** *For any parameterized configuration α, we can compute minpre($\{\alpha\}$).*

Well-quasi Ordering. Let $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$ and $\alpha' = (P', (\mathbf{q}', \mathbf{b}', mem'))$ be two parameterized configurations. We define the ordering $\trianglelefteq$ on the set of parameterized configurations as follows: $\alpha \trianglelefteq \alpha'$ iff the following conditions are satisfied: (1) $mem = mem'$ and (2) there is an injection $h : \{1, \ldots, |P|\} \mapsto \{1, \ldots, |P'|\}$ such that (i) $p < p'$ implies $h(p) < h(p')$, and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}(p) = \mathbf{q}'(h(p))$ and $\mathbf{b}(p) \sqsubseteq \mathbf{b}'(h(p))$. The following lemma states that $\trianglelefteq$ is indeed a well-quasi ordering.

▶ **Lemma 9.** *The relation $\trianglelefteq$ is a well-quasi ordering over $\mathtt{C}$. Furthermore, for every two parameterized configurations α and α', it is decidable whether $\alpha \trianglelefteq \alpha'$.*

Monotonicity. Let $\alpha_1 = (P, (\mathbf{q}_1, \mathbf{b}_1, mem_1))$, $\alpha_2 = (P, (\mathbf{q}_2, \mathbf{b}_2, mem_2))$ and $\alpha_3 = (P', (\mathbf{q}_3, \mathbf{b}_3, mem_3))$ be parameterized configurations. Furthermore, we assume that $\alpha_1 \trianglelefteq \alpha_3$ and $\alpha_1 \xrightarrow{t} \alpha_2$ for some transition t. Since $\alpha_1 \trianglelefteq \alpha_3$, there is an injection function $h : \{1, \ldots, |P|\} \mapsto \{1, \ldots, |P'|\}$ such that (i) $p < p'$ implies $h(p) < h(p')$, and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}_1(p) = \mathbf{q}_3(h(p))$ and $\mathbf{b}_1(p) \sqsubseteq \mathbf{b}_3(h(p))$. We define the parameterized configuration α' from α_3 by only keeping the local state and load buffers of processes in $h(P)$. Formally, $\alpha' = (P, (\mathbf{q}', \mathbf{b}', mem'))$ is defined as follows: (i) $mem' = mem_3$ and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}'(p) = \mathbf{q}_3(h(p))$ and $\mathbf{b}'(p) = \mathbf{b}_3(h(p))$. (Observe that $(\mathbf{q}_1, \mathbf{b}_1, mem_1) \sqsubseteq (\mathbf{q}', \mathbf{b}', mem')$). Since the relation $\rightarrow_{DTSO}$ is monotonic wrt. the ordering $\sqsubseteq$ (see Lemma 5), there is a Dual TSO-configuration $(\mathbf{q}'', \mathbf{b}'', mem'')$ such that $(\mathbf{q}', \mathbf{b}', mem') \rightarrow^*_{DTSO} (\mathbf{q}'', \mathbf{b}'', mem'')$ and $(\mathbf{q}_2, \mathbf{b}_2, mem_2) \sqsubseteq (\mathbf{q}'', \mathbf{b}'', mem'')$. Consider now the parameterized configuration $\alpha_4 = (P', (\mathbf{q}_4, \mathbf{b}_4, mem_4))$ such that $mem'' = mem_4$, (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}''(p) = \mathbf{q}_4(h(p))$ and $\mathbf{b}''(p) = \mathbf{b}_4(h(p))$, and (iii) for $p \in (\{1, \ldots, |P'|\} \setminus \{h(1), \ldots, h(|P|)\})$, we have $\mathbf{q}_4(p) = \mathbf{q}_3(p)$ and $\mathbf{b}_4(p) = \mathbf{b}_3(p)$. It is easy then to see that $\alpha_2 \trianglelefteq \alpha_4$ and $\alpha_3 \rightarrow^* \alpha_4$. Hence, we obtain the following result:

▶ **Lemma 10.** *The relation $\rightarrow$ is monotonic wrt. $\trianglelefteq$.*

From Reachability to Coverability. Let $\mathbf{q}_{target} : P' \mapsto Q$ be a global state. Let $\mathtt{M}_{target}$ be the set of parameterized configurations of the form $\alpha = (P', (\mathbf{q}_{target}, \mathbf{b}, mem))$ with $\mathbf{b}(p) = \epsilon$ for all $p \in P'$. In the following, we show that $\mathtt{M}_{target}\uparrow$ is reachable in $\mathcal{T}$ iff there is a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$.

Let us assume that there is a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$. It is then easy to show that $\alpha \in \mathtt{M}_{target}\uparrow$.

Now let us assume that there is $\alpha' = (P'', (\mathbf{q}', \mathbf{b}', mem')) \in \mathtt{M}_{target}\uparrow$ which is reachable in $\mathcal{T}$. From the configuration α', it is possible to reach the configuration $\alpha'' = (P'', (\mathbf{q}', \mathbf{b}'', mem'))$, with $\mathbf{b}''(p) = \epsilon$ for all $p \in P''$, by performing a sequence of delete_p operations to empty the load buffer of each process. Since $\alpha' \in \mathtt{M}_{target}\uparrow$, we have $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}'(1) \cdots \mathbf{q}'(|P''|)$. Hence, α'' is a witness of the state reachability problem.

▶ **Lemma 11.** *$\mathbf{q}_{target}$ is reachable in $\mathcal{T}$ iff $\mathtt{M}_{target}\uparrow$ is reachable in $\mathcal{T}$.*

Table 1 Comparison between Dual-TSO and Memorax: The columns #P, #T and #C give the number of processes, the running time in seconds and the number of generated configurations, respectively. If a tool runs out of time, we put TO in the #T column and • in the #C column.

Program	#P	Dual-TSO		Memorax	
		#T	#C	#T	#C
SB	5	0.3	10641	559.7	10515914
LB	3	0.0	2048	71.4	1499475
WRC	4	0.0	1507	63.3	1398393
ISA2	3	0.0	509	21.1	226519
RWC	5	0.1	4277	61.5	1196988
W+RWC	4	0.0	1713	83.6	1389009
IRIW	4	0.0	520	34.4	358057
Nbw_w_wr	2	0.0	222	10.7	200844
Sense_rev_bar	2	0.1	1704	0.8	20577
Dekker	2	0.1	5053	1.1	19788
Dekker_simple	2	0.0	98	0.0	595
Peterson	2	0.1	5442	5.2	90301
Peterson_loop	2	0.2	7632	5.6	100082
Szymanski	2	0.6	29018	1.0	26003
MP	4	0.0	883	TO	•
Ticket_spin_lock	3	0.9	18963	TO	•
Bakery	2	2.6	82050	TO	•
Dijkstra	2	0.2	8324	TO	•
Lamport_fast	3	17.7	292543	TO	•
Burns	4	124.3	2762578	TO	•

7 Experimental Results

We have implemented our techniques described in Section 4 and Section 6 in an open-source tool called Dual-TSO[1]. The tool checks the state reachability problems for (parameterized) concurrent systems under the Dual TSO semantics. We compare our tool with Memorax [2, 3] which is the *only precise and sound tool* for deciding the state reachability problem of concurrent systems under TSO. Observe that Memorax cannot handle parameterized verification. All experiments are performed on an Intel x86-32 Core2 2.4 Ghz machine and 4GB of RAM.

In the following, we present two sets of results. The first set concerns the comparison of Dual-TSO with Memorax (see Table 1). The second set shows the benefit of the parameterized verification compared to the use of the state reachability when increasing the number of processes (see Figure 3 and Table 2). Our examples are from [2, 10, 13, 4, 25]. In all experiments, we set up the time out to 600 seconds.

Table 1 presents a comparison between Dual-TSO and Memorax on a representative sample of 20 benchmarks. In all these examples, Dual-TSO and Memorax return the same result for the state reachability problem (except 6 examples where Memorax runs out of time). In the examples where the two tools return, Dual-TSO out-performs Memorax and generates fewer configurations (and so uses less memory). Indeed, Dual-TSO is 600 times faster than

[1] https://www.it.uu.se/katalog/tuang296/dual-tso

Table 2 Parameterized verification with Dual-TSO.

Program	Dual-TSO	
	#T	#C
SB	0.0	147
LB	0.6	1028
MP	0.0	149
WRC	0.8	618
ISA2	4.3	1539
RWC	0.2	293
W+RWC	1.5	828
IRIW	4.6	648

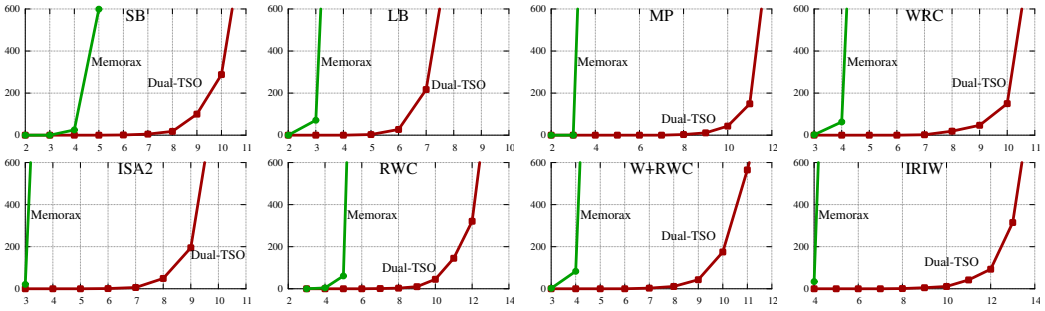


Figure 3 Running time of Memorax and Dual-TSO by increasing number of processes. The x axis is number of processes, the y axis is running time in seconds.

Memorax and generates 277 times fewer configurations on average.

The second set compares the scalability of Memorax and Dual-TSO while increasing the number of processes. The results are given in Fig. 3. We observe that Dual-TSO scales better than Memorax in all these examples. In fact, Memorax can only handle the examples with at most 5 processes. Table 2 presents the running time and the number of generated configurations when checking the state reachability problem for the parameterized version of these examples. We observe that the verification of these parameterized systems is much more efficient than verification of bounded-size instances (starting from a number of processes of 3 or 4), especially concerning memory consumption (which is given in terms of number of generated configurations). The reason behind is that the size of the generated minor sets in the analysis of a parameterized system is usually smaller than the size of the generated configurations during the analysis of an instance of the system with a large number of processes.

8 Conclusion

In this paper, we have presented an alternative (yet equivalent) semantics to the classical one for the TSO model. This new semantics allows us to understand the TSO model in a totally different way compared to the classical semantics. Furthermore, the proposed semantics offers several important advantages from the point of view of formal reasoning and program verification. First, the dual semantics allows transforming the load buffers to *lossy* channels without adding the costly overhead that was necessary in the case of store buffers. This means that we can apply the theory of *well-structured systems* [6, 5, 21] in a

straightforward manner leading to a much simpler proof of decidability of safety properties. Second, the absence of extra overhead means that we obtain more efficient algorithms and better scalability (as shown by our experimental results). Finally, the dual semantics allows extending the framework to perform *parameterized verification* which is an important paradigm in concurrent program verification.

In the future, we plan to apply our techniques to more memory models and to combine with predicate abstraction for handling programs with unbounded data domain.

References

1. P. Abdulla, S. Aronis, M.F. Atig, B. Jonsson, C. Leonardsson, and K. Sagonas. Stateless model checking for TSO and PSO. In *TACAS*, volume 9035 of *LNCS*, pages 353–367. Springer, 2015.
2. P.A. Abdulla, M.F. Atig, Y.F. Chen, C. Leonardsson, and A. Rezine. Counter-example guided fence insertion under TSO. In *TACAS 2012*, pages 204–219, 2012.
3. P.A. Abdulla, M.F. Atig, Y.F. Chen, C. Leonardsson, and A. Rezine. Memorax, a precise and sound tool for automatic fence insertion under TSO. In *TACAS*, pages 530–536, 2013.
4. P.A. Abdulla, M.F. Atig, and N.T. Phong. The best of both worlds: Trading efficiency and optimality in fence insertion for TSO. In *ESOP 2015*, pages 308–332, 2015.
5. P.A. Abdulla, K. Cerans, B. Jonsson, and Y.K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE Computer Society, 1996.
6. Parosh Aziz Abdulla. Well (and better) quasi-ordered transition systems. *Bulletin of Symbolic Logic*, 16(4):457–515, 2010.
7. S. Adve and K. Gharachorloo. Shared memory consistency models: a tutorial. *Computer*, 29(12), 1996.
8. S. Adve and M. D. Hill. Weak ordering - a new definition. In *ISCA*, 1990.
9. J. Alglave, D. Kroening, and M. Tautschnig. Partial orders for efficient bounded model checking of concurrent software. In *CAV*, volume 8044 of *LNCS*, pages 141–157, 2013.
10. Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM TOPLAS*, 36(2):7:1–7:74, 2014.
11. M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, 2010.
12. 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.
13. Ahmed Bouajjani, Egor Derevenetc, and Roland Meyer. Checking and enforcing robustness against TSO. In *ESOP*, volume 7792 of *LNCS*, pages 533–553. Springer, 2013.
14. S. Burckhardt, R. Alur, and M. M. K. Martin. CheckFence: checking consistency of concurrent data types on relaxed memory models. In *PLDI*, pages 12–21. ACM, 2007.
15. Sebastian Burckhardt and Madanlal Musuvathi. Effective program verification for relaxed memory models. In *CAV*, volume 5123 of *LNCS*, pages 107–120. Springer, 2008.
16. Jacob Burnim, Koushik Sen, and Christos Stergiou. Testing concurrent programs on relaxed memory models. In *ISSTA*, pages 122–132. ACM, 2011.
17. A. Marian Dan, Y. Meshman, M. T. Vechev, and E. Yahav. Predicate abstraction for relaxed memory models. In *SAS*, volume 7935 of *LNCS*, pages 84–104. Springer, 2013.
18. Brian Demsky and Patrick Lam. Satcheck: Sat-directed stateless model checking for SC and TSO. In *OOPSLA 2015*, pages 20–36. ACM, 2015.
19. Egor Derevenetc and Roland Meyer. Robustness against Power is PSpace-complete. In *ICALP (2)*, volume 8573 of *LNCS*, pages 158–170. Springer, 2014.
20. M. Dubois, C. Scheurich, and F. A. Briggs. Memory access buffering in multiprocessors. In *ISCA*, 1986.

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

22 Michael Kuperstein, Martin T. Vechev, and Eran Yahav. Automatic inference of memory fences. In *FMCAD*, pages 111–119. IEEE, 2010.

23 Michael Kuperstein, Martin T. Vechev, and Eran Yahav. Partial-coherence abstractions for relaxed memory models. In *PLDI*, pages 187–198. ACM, 2011.

24 L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comp.*, C-28(9), 1979.

25 Feng Liu, Nayden Nedev, Nedyalko Prisadnikov, Martin T. Vechev, and Eran Yahav. Dynamic synthesis for relaxed memory models. In *PLDI '12*, pages 429–440, 2012.

26 S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-tso. In *TPHOL*, 2009.

27 P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen. x86-tso: A rigorous and usable programmer's model for x86 multiprocessors. *CACM*, 53, 2010.

28 D. Weaver and T. Germond, editors. *The SPARC Architecture Manual Version 9*. PTR Prentice Hall, 1994.

29 Y. Yang, G. Gopalakrishnan, G. Lindstrom, and K. Slind. Nemos: A framework for axiomatic and executable specifications of memory consistency models. In *IPDPS*. IEEE, 2004.

30 N. Zhang, M. Kusano, and C. Wang. Dynamic partial order reduction for relaxed memory models. In *PLDI*, pages 250–259. ACM, 2015.

Local Linearizability for Concurrent Container-Type Data Structures*

Andreas Haas[1], Thomas A. Henzinger[2], Andreas Holzer[3], Christoph M. Kirsch[4], Michael Lippautz[5], Hannes Payer[6], Ali Sezgin[7], Ana Sokolova[8], and Helmut Veith[9]

1 **Google Inc.**
2 **IST Austria, Austria**
3 **University of Toronto, Canada**
4 **University of Salzburg, Austria**
5 **Google Inc.**
6 **Google Inc.**
7 **University of Cambridge, UK**
8 **University of Salzburg, Austria**
9 **Vienna University of Technology, Austria and**
Forever in our hearts

Abstract

The semantics of concurrent data structures is usually given by a sequential specification and a consistency condition. Linearizability is the most popular consistency condition due to its simplicity and general applicability. Nevertheless, for applications that do not require all guarantees offered by linearizability, recent research has focused on improving performance and scalability of concurrent data structures by relaxing their semantics.

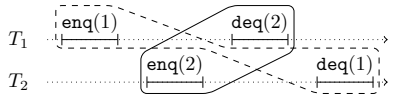
In this paper, we present local linearizability, a relaxed consistency condition that is applicable to *container-type* concurrent data structures like pools, queues, and stacks. While linearizability requires that the effect of each operation is observed by all threads at the same time, local linearizability only requires that for each thread T, the effects of its local insertion operations and the effects of those removal operations that remove values inserted by T are observed by all threads at the same time. We investigate theoretical and practical properties of local linearizability and its relationship to many existing consistency conditions. We present a generic implementation method for locally linearizable data structures that uses existing linearizable data structures as building blocks. Our implementations show performance and scalability improvements over the original building blocks and outperform the fastest existing container-type implementations.

1998 ACM Subject Classification D.3.1 [Programming Languages]: Formal Definitions and Theory—Semantics; E.1 [Data Structures]: Lists, stacks, and queues; D.1.3 [Software]: Programming Techniques—Concurrent Programming

Keywords and phrases (concurrent) data structures, relaxed semantics, linearizability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.6

* This work has been supported by the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund (FWF): S11402-N23, S11403-N23, S11404-N23, S11411-N23), a Google PhD Fellowship, an Erwin Schrödinger Fellowship (Austrian Science Fund (FWF): J3696-N26), EPSRC grants EP/H005633/1 and EP/K008528/1, the Vienna Science and Technology Fund (WWTF) trough grant PROSEED, the European Research Council (ERC) under grant 267989 (QUAREM) and by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award).



The thread-induced history of thread T_1 is enclosed by a dashed line while the thread-induced history of thread T_2 is enclosed by a solid line.

Figure 1 Local Linearizability.

1 Introduction

Concurrent data structures are pervasive all along the software stack, from operating system code to application software and beyond. Both correctness and performance are imperative for concurrent data structure implementations. Correctness is usually specified by relating concurrent executions, admitted by the implementation, with sequential executions, admitted by the sequential version of the data structure. The latter form the *sequential specification* of the data structure. This relationship is formally captured by *consistency conditions*, such as linearizability, sequential consistency, or quiescent consistency [22].

Linearizability [23] is the most accepted consistency condition for concurrent data structures due to its simplicity and general applicability. It guarantees that the effects of all operations by all threads are observed consistently. This imposes the need of extensive synchronization among threads which may in turn jeopardize performance and scalability. In order to enhance performance and scalability of implementations, recent research has explored relaxed sequential specifications [20, 35, 2], resulting in well-performing implementations of concurrent data structures [2, 16, 20, 25, 33, 6]. Except for [24], the space of alternative consistency conditions that relax linearizability has been left unexplored to a large extent. In this paper, we explore (part of) this gap by investigating *local linearizability*, a novel consistency condition that is applicable to a large class of concurrent data structures that we call *container-type* data structures, or *containers* for short. Containers include pools, queues, and stacks. A fine-grained spectrum of consistency conditions enables us to describe the semantics of concurrent implementations more precisely, e.g. in the extended version of our paper [15] we show that work stealing queues [30] which could only be proven to be linearizable wrt pool are actually locally linearizable wrt double-ended queue.

Local linearizability is a (thread-)local consistency condition that guarantees that insertions *per thread* are observed consistently. While linearizability requires a consistent view over all insertions, we only require that projections of the global history—so called *thread-induced histories*—are linearizable. The induced history of a thread T is a projection of a program execution to the insert-operations in T combined with all remove-operations that remove values inserted by T irrespective of whether they happen in T or not. Then, the program execution is locally linearizable iff each thread-induced history is linearizable. Consider the example (sequential) history depicted in Figure 1. It is not linearizable wrt a queue since the values are not dequeued in the same order as they were enqueued. However, each thread-induced history is linearizable wrt a queue and, therefore, the overall execution is locally linearizable wrt a queue. In contrast to semantic relaxations based on relaxing sequential semantics such as [20, 2], local linearizability coincides with *sequential correctness* for single-threaded histories, i.e., a single-threaded and, therefore, sequential history is locally linearizable wrt a given sequential specification if and only if it is admitted by the sequential specification.

Local linearizability is to linearizability what coherence is to sequential consistency. Coherence [19], which is almost universally accepted as the absolute minimum that a shared memory system should satisfy, is the requirement that there exists a unique global order per shared memory location. Thus, while all accesses by all threads to a given memory location have to conform to a unique order, consistent with program order, the relative ordering of accesses to multiple memory locations do not have to be the same. In other words, coherence is sequential consistency per memory location. Similarly, local linearizability is linearizability per local history. In our view, local linearizability offers enough consistency for the correctness of many applications as it is the local view of the client that often matters. For example, in a locally linearizable queue each client (thread) has the impression of using a perfect queue—no reordering will ever be observed among the values inserted by a single thread. Such guarantees suffice for many e-commerce and cloud applications. Implementations of locally linearizable data structures have been successfully applied for managing free lists in the design of the fast and scalable memory allocator scalloc [5]. Moreover, except for fairness, locally linearizable queues guarantee all properties required from Dispatch Queues [1], a common concurrency programming mechanism on mobile devices.

In this paper, we study theoretical and practical properties of local linearizability. Local linearizability is compositional—a history over multiple concurrent objects is locally linearizable iff all per-object histories are locally linearizable (see Thm. 12) and locally linearizable container-type data structures, including queues and stacks, admit only "sane" behaviours—no duplicated values, no values returned from thin air, and no values lost (see Prop. 4). Local linearizability is a weakening of linearizability for a natural class of data structures including pools, queues, and stacks (see Sec. 4). We compare local linearizability to linearizability, sequential, and quiescent consistency, and to many shared-memory consistency conditions.

Finally, local linearizability leads to new efficient implementations. We present a generic implementation scheme that, given a linearizable implementation of a sequential specification S, produces an implementation that is locally linearizable wrt S (see Sec. 6). Our implementations show dramatic improvements in performance and scalability. In most cases the locally linearizable implementations scale almost linearly and even outperform state-of-the-art pool implementations. We produced locally linearizable variants of state-of-the-art concurrent queues and stacks, as well as of the relaxed data structures from [20, 25]. The latter are relaxed in two dimensions: they are locally linearizable (the consistency condition is relaxed) and are out-of-order-relaxed (the sequential specification is relaxed). The speedup of the locally linearizable implementation to the fastest linearizable queue (LCRQ) and stack (TS Stack) implementation at 80 threads is 2.77 and 2.64, respectively. Verification of local linearizability, i.e. proving correctness, for each of our new locally linearizable implementations is immediate, given that the starting implementations are linearizable.

2 Semantics of Concurrent Objects

The common approach to define the semantics of an implementation of a concurrent data structure is (1) to specify a set of valid sequential behaviors—the sequential specification, and (2) to relate the admissible concurrent executions to sequential executions specified by the sequential specification—via the consistency condition. That means that an implementation of a concurrent data structure actually corresponds to several sequential data structures, and vice versa, depending on the consistency condition used. A (sequential) data structure D is an object with a set of method calls Σ. We assume that method calls include parameters, i.e., input and output values from a given set of values. The sequential specification S of D

Table 1 The pool axioms (1), (2), (3); the queue order axiom (4); the stack order axiom (5).

(1) $\forall i, j \in \{1, \ldots, n\}.\ \mathbf{s} = m_1 \ldots m_n \ \wedge\ m_i = m_j \ \Rightarrow\ i = j$
(2) $\forall x \in V.\ \mathtt{r}(x) \in \mathbf{s} \ \Rightarrow\ \mathtt{i}(x) \in \mathbf{s} \wedge \mathtt{i}(x) \prec_{\mathbf{s}} \mathtt{r}(x)$
(3) $\forall e \in \mathtt{Emp}.\ \forall x \in V.\ \mathtt{i}(x) \prec_{\mathbf{s}} \mathtt{r}(e) \Rightarrow \mathtt{r}(x) \prec_{\mathbf{s}} \mathtt{r}(e)$
(4) $\forall x, y \in V.\ \mathtt{i}(x) \prec_{\mathbf{s}} \mathtt{i}(y) \ \wedge\ \mathtt{r}(y) \in \mathbf{s} \ \Rightarrow\ \mathtt{r}(x) \in \mathbf{s} \ \wedge\ \mathtt{r}(x) \prec_{\mathbf{s}} \mathtt{r}(y)$
(5) $\forall x, y \in V.\ \mathtt{i}(x) \prec_{\mathbf{s}} \mathtt{i}(y) \prec_{\mathbf{s}} \mathtt{r}(x) \ \Rightarrow\ \mathtt{r}(y) \in \mathbf{s} \ \wedge\ \mathtt{r}(y) \prec_{\mathbf{s}} \mathtt{r}(x)$

is a prefix-closed subset of Σ^*. The elements of S are called D-valid sequences. For ease of presentation, we assume that each value in a data structure can be inserted and removed at most once. This is without loss of generality, as we may see the set of values as consisting of pairs of elements (core values) and version numbers, i.e. $V = E \times \mathbb{N}$. Note that this is a technical assumption that only makes the presentation and the proofs simpler, it is not needed and not done in locally linearizable implementations. While elements may be inserted and removed multiple times, the version numbers provide uniqueness of values. Our assumption ensures that whenever a sequence $\mathbf{s}$ is part of a sequential specification S, then, each method call in $\mathbf{s}$ appears exactly once. An additional core value, that is not an element, is `empty`. It is returned by remove method calls that do not find an element to return. We denote by `Emp` the set of values that are versions of `empty`, i.e., $\mathtt{Emp} = \{\mathtt{empty}\} \times \mathbb{N}$.

▶ **Definition 1** (Appears-before Order, Appears-in Relation)**.** Given a sequence $\mathbf{s} \in \Sigma^*$ in which each method call appears exactly once, we denote by $\prec_{\mathbf{s}}$ the total *appears-before order* over method calls in $\mathbf{s}$. Given a method call $m \in \Sigma$, we write $m \in \mathbf{s}$ for *m appears in $\mathbf{s}$*. ◇

Throughout the paper, we will use pool, queue, and stack as typical examples of containers. We specify their sequential specifications in an *axiomatic* way [21], i.e., as sets of axioms that exactly define the valid sequences.

▶ **Definition 2** (Pool, Queue, & Stack)**.** A pool, queue, and stack with values in a set V have the sets of methods $\Sigma_P = \{\mathtt{ins}(x), \mathtt{rem}(x) \mid x \in V\} \cup \{\mathtt{rem}(e) \mid e \in \mathtt{Emp}\}$, $\Sigma_Q = \{\mathtt{enq}(x), \mathtt{deq}(x) \mid x \in V\} \cup \{\mathtt{deq}(e) \mid e \in \mathtt{Emp}\}$, and $\Sigma_S = \{\mathtt{push}(x), \mathtt{pop}(x) \mid x \in V\} \cup \{\mathtt{pop}(e) \mid e \in \mathtt{Emp}\}$, respectively. We denote the sequential specification of a pool by S_P, the sequential specification of a queue by S_Q, and the sequential specification of a stack by S_S. A sequence $\mathbf{s} \in \Sigma_P^*$ belongs to S_P iff it satisfies axioms (1) - (3) in Table 1—*the pool axioms*—when instantiating `i`() with `ins`() and `r`() with `rem`(). We keep axiom (1) for completeness, although it is subsumed by our assumption that each value is inserted and removed at most once. Specification S_Q contains all sequences $\mathbf{s}$ that satisfy the pool axioms and axiom (4)—*the queue order axiom*—after instantiating `i`() with `enq`() and `r`() with `deq`(). Finally, S_S contains all sequences $\mathbf{s}$ that satisfy the pool axioms and axiom (5)—*the stack order axiom*—after instantiating `i`() with `push`() and `r`() with `pop`(). ◇

We represent concurrent executions via concurrent histories. An example history is shown in Figure 1. Each thread executes a sequence of method calls from Σ; method calls executed by different threads may overlap (which does not happen in Figure 1). The real-time duration of method calls is irrelevant for the semantics of concurrent objects; all that matters is whether method calls overlap. Given this abstraction, a concurrent history is fully determined by a sequence of invocation and response events of method calls. We distinguish method invocation and response events by augmenting the alphabet. Let $\Sigma_i = \{m_i \mid m \in \Sigma\}$ and $\Sigma_r = \{m_r \mid m \in \Sigma\}$ denote the sets of method-invocation events and method-response events, respectively, for the method calls in Σ. Moreover, let I be the set of thread identifiers. Let $\Sigma_i^I = \{m_i^k \mid m \in \Sigma, k \in I\}$ and $\Sigma_r^I = \{m_r^k \mid m \in \Sigma, k \in I\}$

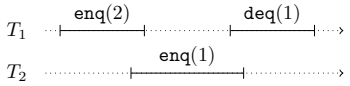
denote the sets of method-invocation and -response events augmented with identifiers of executing threads. For example, m_i^k is the invocation of method call m by thread k. Before we proceed, we mention a standard notion that we will need in several occasions.

▶ **Definition 3** (Projection). Let $\mathbf{s}$ be a sequence over alphabet Σ and $M \subseteq \Sigma$. By $\mathbf{s}|M$ we denote the projection of $\mathbf{s}$ on the symbols in M, i.e., the sequence obtained from $\mathbf{s}$ by removing all symbols that are not in M. ◇

▶ **Definition 4** (History). A (concurrent) history $\mathbf{h}$ is a sequence in $(\Sigma_i^I \cup \Sigma_r^I)^*$ where

1. no invocation or response event appears more than once, i.e., if $\mathbf{h} = m_1 \dots m_n$ and $m_h = m_*^k(x)$ and $m_j = m_*^l(x)$, for $* \in \{i, r\}$, then $h = j$ and $k = l$, and
2. if a response event m_r^k appears in $\mathbf{h}$, then the corresponding invocation event m_i^k also appears in $\mathbf{h}$ and $m_i \prec_{\mathbf{h}} m_r$. ◇

▶ **Example 5.** A queue history (left) and its formal representation as a sequence (right):



$$\texttt{enq}(2)_i^1\texttt{enq}(1)_i^2\texttt{enq}(2)_r^1\texttt{deq}(1)_i^1\texttt{enq}(1)_r^2\texttt{deq}(1)_r^1$$

A history is *sequential* if every response event is immediately preceded by its matching invocation event and vice versa. Hence, we may ignore thread identifiers and identify a sequential history with a sequence in Σ^*, e.g., $\texttt{enq}(1)\texttt{enq}(2)\texttt{deq}(2)\texttt{deq}(1)$ identifies the sequential history in Figure 1.

A history $\mathbf{h}$ is *well-formed* if $\mathbf{h}|k$ is sequential for every thread identifier $k \in I$ where $\mathbf{h}|k$ denotes the projection of $\mathbf{h}$ on the set $\{m_i^k \mid m \in \Sigma\} \cup \{m_r^k \mid m \in \Sigma\}$ of events that are local to thread k. From now on we will use the term history for well-formed history. Also, we may omit thread identifiers if they are not essential in a discussion.

A history $\mathbf{h}$ determines a partial order on its set of method calls, the precedence order:

▶ **Definition 6** (Appears-in Relation, Precedence Order). The set of method calls of a history $\mathbf{h}$ is $M(\mathbf{h}) = \{m \mid m_i \in \mathbf{h}\}$. A method call m appears in $\mathbf{h}$, notation $m \in \mathbf{h}$, if $m \in M(\mathbf{h})$. The *precedence order* for $\mathbf{h}$ is the partial order $<_{\mathbf{h}}$ such that, for $m, n \in \mathbf{h}$, we have that $m <_{\mathbf{h}} n$ iff $m_r \prec_{\mathbf{h}} n_i$. By $<_{\mathbf{h}}^k$ we denote $<_{\mathbf{h}|k}$, the subset of the precedence order that relates pairs of method calls of thread k, i.e., the program order of thread k. ◇

We can characterize a sequential history as a history whose precedence order is total. In particular, the precedence order $<_{\mathbf{s}}$ of a sequential history $\mathbf{s}$ coincides with its appears-before order $\prec_{\mathbf{s}}$. The total order for history $\mathbf{s}$ in Fig. 1 is $\texttt{enq}(1) <_{\mathbf{s}} \texttt{enq}(2) <_{\mathbf{s}} \texttt{deq}(2) <_{\mathbf{s}} \texttt{deq}(1)$.

▶ **Definition 7** (Projection to a set of method calls). Let $\mathbf{h}$ be a history, $M \subseteq \Sigma$, $M_i^I = \{m_i^k \mid m \in M, k \in I\}$, and $M_r^I = \{m_r^k \mid m \in M, k \in I\}$. Then, we write $\mathbf{h}|M$ for $\mathbf{h}|(M_i^I \cup M_r^I)$. ◇

Note that $\mathbf{h}|M$ inherits $\mathbf{h}$'s precedence order: $m <_{\mathbf{h}|M} n \;\Leftrightarrow\; m \in M \wedge n \in M \wedge m <_{\mathbf{h}} n$

A history $\mathbf{h}$ is *complete* if the response of every invocation event in $\mathbf{h}$ appears in $\mathbf{h}$. Given a history $\mathbf{h}$, $\texttt{Complete}(\mathbf{h})$ denotes the set of all *completions* of $\mathbf{h}$, i.e., the set of all complete histories that are obtained from $\mathbf{h}$ by appending missing response events and/or removing pending invocation events. Note that $\texttt{Complete}(\mathbf{h}) = \{\mathbf{h}\}$ iff $\mathbf{h}$ is a complete history.

A concurrent data structure D over a set of methods Σ is a (prefix-closed) set of concurrent histories over Σ. A history may involve several concurrent objects. Let O be a set of concurrent objects with individual sets of method calls Σ_q and sequential specifications S_q for each object $q \in O$. A history $\mathbf{h}$ over O is a history over the (disjoint) union of method

calls of all objects in O, i.e., it has a set of method calls $\bigcup_{q\in O}\{q.m \mid m \in \Sigma_q\}$. The added prefix $q.$ ensures that the union is disjoint. The *projection* of $\mathbf{h}$ to an object $q \in O$, denoted by $\mathbf{h}|q$, is the history with a set of method calls Σ_q obtained by removing the prefix $q.$ in every method call in $\mathbf{h}|\{q.m \mid m \in \Sigma_q\}$.

▶ **Definition 8** (Linearizability [23])**.** A history $\mathbf{h}$ is *linearizable* wrt the sequential specification S if there is a sequential history $\mathbf{s} \in S$ and a completion $\mathbf{h}_c \in \texttt{Complete(h)}$ such that

1. $\mathbf{s}$ is a permutation of $\mathbf{h}_c$, and
2. $\mathbf{s}$ preserves the precedence order of $\mathbf{h}_c$, i.e., if $m <_{\mathbf{h}_c} n$, then $m <_{\mathbf{s}} n$.

We refer to $\mathbf{s}$ as a *linearization* of $\mathbf{h}$. A concurrent data structure D is linearizable wrt S if every history $\mathbf{h}$ of D is linearizable wrt S. A history $\mathbf{h}$ over a set of concurrent objects O is linearizable wrt the sequential specifications S_q for $q \in O$ if there exists a linearization $\mathbf{s}$ of $\mathbf{h}$ such that $\mathbf{s}|q \in S_q$ for each object $q \in O$. ◇

3 Local Linearizability

Local linearizability is applicable to containers whose set of method calls is a disjoint union $\Sigma = \texttt{Ins} \cup \texttt{Rem} \cup \texttt{DOb} \cup \texttt{SOb}$ of insertion method calls `Ins`, removal method calls `Rem`, data-observation method calls `DOb`, and (global) shape-observation method calls `SOb`. Insertions (removals) insert (remove) a *single* value in the data set V or `empty`; data observations return a *single* value in V; shape observations return a value (not necessarily in V) that provides information on the shape of the state, for example, the size of a data structure. Examples of data observations are $\texttt{head}(x)$ (queue), $\texttt{top}(x)$ (stack), and $\texttt{peek}(x)$ (pool). Examples of shape observations are $\texttt{empty}(b)$ that returns true if the data structure is empty and false otherwise, and $\texttt{size}(n)$ that returns the number of elements in the data structure.

Even though we refrain from formal definitions, we want to stress that a valid sequence of a container remains valid after deleting observer method calls:

$$S \,|\, (\texttt{Ins} \cup \texttt{Rem}) \subseteq S. \tag{1}$$

There are also containers with multiple insert/remove methods, e.g., a double-ended queue (deque) is a container with insert-left, insert-right, remove-left, and remove-right methods, to which local linearizability is also applicable. However, local linearizability requires that each method call is either an insertion, or a removal, or an observation. As a consequence, set is not a container according to our definition, as in a set $\texttt{ins}(x)$ acts as a global observer first, checking whether (some version of) x is already in the set, and if not inserts x. Also hash tables are not containers for a similar reason.

Note that the arity of each method call in a container being one excludes data structures like snapshot objects. It is possible to deal with higher arities in a fairly natural way, however, at the cost of complicated presentation. We chose to present local linearizability on simple containers only. We present the definition of local linearizability without shape observations here and discuss shape observations in [15].

▶ **Definition 9** (In- and out-methods)**.** Let $\mathbf{h}$ be a container history. For each thread T we define two subsets of the methods in $\mathbf{h}$, called in-methods I_T and out-methods O_T of thread T, respectively:

$$\begin{aligned}
\mathrm{I}_T &= \{m \mid m \in M(\mathbf{h}|T) \cap \texttt{Ins}\} \\
\mathrm{O}_T &= \{m(a) \in M(\mathbf{h}) \cap \texttt{Rem} \mid \texttt{ins}(a) \in \mathrm{I}_T\} \cup \{m(e) \in M(\mathbf{h}) \cap \texttt{Rem} \mid e \in \texttt{Emp}\} \\
&\quad \cup \{m(a) \in M(\mathbf{h}) \cap \texttt{DOb} \mid \texttt{ins}(a) \in \mathrm{I}_T\}.
\end{aligned}$$ ◇

Hence, the in-methods for thread T are all insertions performed by T. The out-methods are all removals and data observers that return values inserted by T. Removals that remove the value `empty` are also automatically added to the out-methods of T as any thread (and hence also T) could be the cause of "inserting" `empty`. This way, removals of `empty` serve as means for global synchronization. Without them each thread could perform all its operations locally without ever communicating with the other threads. Note that the out-methods O_T of thread T need not be performed by T, but they return values that are inserted by T.

▶ **Definition 10** (Thread-induced History)**.** Let $\mathbf{h}$ be a history. The thread-induced history $\mathbf{h}_T$ is the projection of $\mathbf{h}$ to the in- and out-methods of thread T, i.e., $\mathbf{h}_T = \mathbf{h}|\,(\mathrm{I}_T \cup \mathrm{O}_T)$. ◇

▶ **Definition 11** (Local Linearizability)**.** A history $\mathbf{h}$ is locally linearizable wrt a sequential specification S if

1. each thread-induced history $\mathbf{h}_T$ is linearizable wrt S, and
2. the thread-induced histories $\mathbf{h}_T$ form a decomposition of $\mathbf{h}$, i.e., $m \in \mathbf{h} \Rightarrow m \in \mathbf{h}_T$ for some thread T.

A data structure D is locally linearizable wrt S if every history $\mathbf{h}$ of D is locally linearizable wrt S. A history $\mathbf{h}$ over a set of concurrent objects O is locally linearizable wrt the sequential specifications S_q for $q \in O$ if each thread-induced history is linearizable over O and the thread-induced histories form a decomposition of $\mathbf{h}$, i.e., $q.m \in \mathbf{h} \Rightarrow q.m \in \mathbf{h}_T$ for some thread T. ◇

Local linearizability is sequentially correct, i.e., a single-threaded (necessarily sequential) history $\mathbf{h}$ is locally linearizable wrt a sequential specification S iff $\mathbf{h} \in S$. Like linearizability [22], local linearizability is compositional. The complete proof of the following theorem and missing or extended proofs of all following properties can be found in [15].

▶ **Theorem 12** (Compositionality)**.** *A history* $\mathbf{h}$ *over a set of objects* O *with sequential specifications* S_q *for* $q \in O$ *is locally linearizable iff* $\mathbf{h}|q$ *is locally linearizable wrt* S_q *for every* $q \in O$*.*

Proof (Sketch). The property follows from the compositionality of linearizability and the fact that $(\mathbf{h}|q)_T = \mathbf{h}_T|q$ for every thread T and object q. ◀

The Choices Made. Splitting a global history into subhistories and requiring consistency for each of them is central to local linearizability. While this is common in shared-memory consistency conditions [19, 27, 28, 3, 14, 4, 18], our study of local linearizability is a first step in exploring subhistory-based consistency conditions for concurrent objects.

We chose thread-induced subhistories since thread-locality reduces contention in concurrent objects and is known to lead to high performance as confirmed by our experiments. To assign method calls to thread-induced histories, we took a data-centric point of view by (1) associating data values to threads, and (2) gathering all method calls that insert/return a data value into the subhistory of the associated thread (Def. 9). We associate data values to the thread that inserts them. One can think of alternative approaches, for example, associate with a thread the values that it removed. In our view, the advantages of our choice are clear: First, by assigning inserted values to threads, every value in the history is assigned to some thread. In contrast, in the alternative approach, it is not clear where to assign the values that are inserted but not removed. Second, assigning inserted values to the inserting thread enables eager removals and ensures progress in locally linearizable data structures. In the alternative approach, it seems like the semantics of removing `empty` should be local.

An orthogonal issue is to assign values from shape observations to threads. In [15], we discuss two meaningful approaches and show how local linearizability can be extended towards shape and data observations that appear in insertion operations of sets.

Finally, we have to choose a consistency condition required for each of the subhistories. We chose linearizability as it is the best (strong) consistency condition for concurrent objects.

4 Local Linearizability vs. Linearizability

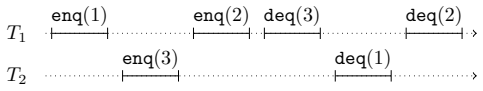
We now investigate the connection between local linearizability and linearizability.

▶ Proposition 1 (Lin 1). In general, linearizability does not imply local linearizability.

Proof. We provide an example of a data structure that is linearizable but not locally linearizable. Consider a sequential specification S_{NearlyQ} which behaves like a queue except when the first two insertions were performed without a removal in between—then the first two elements are removed out of order. Formally, $\mathbf{s} \in S_{\mathsf{NearlyQ}}$ iff

1. $\mathbf{s} = \mathbf{s_1}\mathtt{enq}(a)\mathtt{enq}(b)\mathbf{s_2}\mathtt{deq}(b)\mathbf{s_3}\mathtt{deq}(a)\mathbf{s_4}$ where $\mathbf{s_1}\mathtt{enq}(a)\mathtt{enq}(b)\mathbf{s_2}\mathtt{deq}(a)\mathbf{s_3}\mathtt{deq}(b)\mathbf{s_4} \in S_Q$ and $\mathbf{s_1} \in \{\mathtt{deq}(e) \mid e \in \mathtt{Emp}\}^*$ for some $a, b \in V$, or
2. $\mathbf{s} \in S_Q$ and $\mathbf{s} \neq \mathbf{s_1}\mathtt{enq}(a)\mathtt{enq}(b)\mathbf{s_2}$ for $\mathbf{s_1} \in \{\mathtt{deq}(e) \mid e \in \mathtt{Emp}\}^*$ and $a, b \in V$.

The example below is linearizable wrt S_{NearlyQ}. However, T_1's induced history $\mathtt{enq}(1)\mathtt{enq}(2)\mathtt{deq}(1)\mathtt{deq}(2)$ is not.



◀

The following condition on a data structure specification is sufficient for linearizability to imply local linearizability and is satisfied, e.g., by pool, queue, and stack.

▶ **Definition 13** (Closure under Data-Projection). A seq. specification S over Σ is *closed under data-projection*[1] iff for all $\mathbf{s} \in S$ and all $V' \subseteq V$, $\mathbf{s}|\{m(x) \in \Sigma \mid x \in V' \cup \mathtt{Emp}\} \in S$. ◇

For $\mathbf{s} = \mathtt{enq}(1)\mathtt{enq}(3)\mathtt{enq}(2)\mathtt{deq}(3)\mathtt{deq}(1)\mathtt{deq}(2)$ we have $\mathbf{s} \in S_{\mathsf{NearlyQ}}$, but $\mathbf{s}|\{\mathtt{enq}(x), \mathtt{deq}(x) \mid x \in \{1,2\} \cup \mathtt{Emp}\} \notin S_{\mathsf{NearlyQ}}$, i.e., S_{NearlyQ} is not closed under data-projection.

▶ Proposition 2 (Lin 2). Linearizability implies local linearizability for sequential specifications that are closed under data-projection.

Proof (Sketch). The property follows from Definition 13 and Equation (1). ◀

There exist corner cases where local linearizability coincides with linearizability, e.g., for $S = \emptyset$ or $S = \Sigma^*$, or for single-producer/multiple-consumer histories.

We now turn our attention to pool, queue, and stack.

▶ Proposition 3. The seq. specifications S_P, S_Q, and S_S are closed under data-projection.

Proof (Sketch). Let $\mathbf{s} \in S_P$, $V' \subseteq V$, and let $\mathbf{s}' = \mathbf{s}|(\{\mathtt{ins}(x), \mathtt{rem}(x) \mid x \in V' \cup \mathtt{Emp}\})$. Then, it suffices to check that all axioms for pool (Definition 2 and Table 1) hold for $\mathbf{s}'$. ◀

▶ **Theorem 14** (Pool & Queue & Stack, Lin). *For pool, queue, and stack, local linearizability is (strictly) weaker than linearizability.*

[1] The same notion has been used in [7] under the name *closure under projection*.

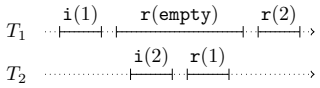


Figure 2 LL, not SC (Pool, Queue, Stack).

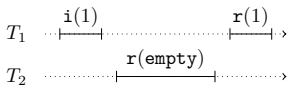


Figure 3 SC, not LL (Pool, Queue, Stack).

Proof. Linearizability implies local linearizability for pool, queue, and stack as a consequence of Proposition 2 and Proposition 3. The history in Figure 2 is locally linearizable but not linearizable wrt pool, queue and stack (after suitable renaming of method calls). ◀

Although local linearizability wrt a pool does not imply linearizability wrt a pool (Theorem 14), it still guarantees several properties that ensure sane behavior as stated next.

▶ Proposition 4 (LocLin Pool). Let $\mathbf{h}$ be a locally linearizable history wrt a pool. Then:

1. No value is duplicated, i.e., every remove method appears in $\mathbf{h}$ at most once.
2. No out-of-thin-air values, i.e., $\forall x \in V.\ \texttt{rem}(x) \in \mathbf{h} \Rightarrow \texttt{ins}(x) \in \mathbf{h} \ \wedge\ \texttt{rem}(x) \not<_{\mathbf{h}} \texttt{ins}(x)$.
3. No value is lost, i.e., $\forall x \in V.\ \forall e \in \texttt{Emp}.\ \texttt{rem}(e) <_{\mathbf{h}} \texttt{rem}(x) \Rightarrow \texttt{ins}(x) \not<_{\mathbf{h}} \texttt{rem}(e)$ and $\forall x \in V.\ \forall e \in \texttt{Emp}.\ \texttt{ins}(x) <_{\mathbf{h}} \texttt{rem}(e) \Rightarrow \texttt{rem}(x) \in \mathbf{h} \ \wedge\ \texttt{rem}(e) \not<_{\mathbf{h}} \texttt{rem}(x)$.

Proof. By direct unfolding of the definitions. ◀

Note that if a history $\mathbf{h}$ is linearizable wrt a pool, then all of the three stated properties hold, as a consequence of linearizability and the definition of S_P.

5 Local Linearizability vs. Other Relaxed Consistency Conditions

We compare local linearizability with other classical consistency conditions to better understand its guarantees and implications.

Sequential Consistency (SC). A history $\mathbf{h}$ is *sequentially consistent* [22, 26] wrt a sequential specification S, if there exists a sequential history $\mathbf{s} \in S$ and a completion $\mathbf{h}_c \in \texttt{Complete(h)}$ such that

1. $\mathbf{s}$ is a permutation of $\mathbf{h}_c$, and

2. $\mathbf{s}$ preserves each thread's program order, i.e., if $m <^T_{\mathbf{h}} n$, for some thread T, then $m <_{\mathbf{s}} n$.

We refer to $\mathbf{s}$ as a *sequential witness* of $\mathbf{h}$. A data structure D is sequentially consistent wrt S if every history $\mathbf{h}$ of D is sequentially consistent wrt S.

Sequential consistency is a useful consistency condition for shared memory but it is not really suitable for data structures as it allows for behavior that excludes any coordination between threads [34]: an implementation of a data structure in which every thread uses a dedicated copy of a sequential data structure without any synchronization is sequentially consistent. A sequentially consistent queue might always return `empty` in one (consumer) thread as the point in time of the operation can be moved, e.g., see Figure 3. In a producer-consumer scenario such a queue might end up with some threads not doing any work.

▶ **Theorem 15** (Pool, Queue & Stack, SC)**.** *For pool, queue, and stack, local linearizability is incomparable to sequential consistency.* ◀

Figures 2 and 3 give example histories that show the statement of Theorem 15. In contrast to local linearizability, sequential consistency is not compositional [22].

(Quantitative) Quiescent Consistency (QC & QQC). Like linearizability and sequential consistency, quiescent consistency [11, 22] also requires the existence of a sequential history, a *quiescent witness*, that satisfies the sequential specification. All three consistency conditions impose an order on the method calls of a concurrent history that a witness has to preserve. Quiescent consistency uses the concept of *quiescent states* to relax the requirement of preserving the precedence order imposed by linearizability. A quiescent state is a point in a history at which there are no pending invocation events (all invoked method calls have already responded). In a quiescent witness, a method call m has to appear before a method call n if and only if there is a quiescent state between m and n. Method calls between two consecutive quiescent states can be ordered arbitrarily. *Quantitative quiescent consistency* [24] refines quiescent consistency by bounding the number of reorderings of operations between two quiescent states based on the concurrent behavior between these two states.

The next result about quiescent consistency for pool is needed to establish the connection between quiescent consistency and local linearizability.

▶ Proposition 5. A pool history $\mathbf{h}$ satisfying 1.-3. of Prop. 4 is quiescently consistent. ◀

From Prop. 4 and 5 follows that local linearizability implies quiescent consistency for pool.

▶ **Theorem 16** (Pool, Queue & Stack, QC)**.** *For pool, local linearizability is (strictly) stronger than quiescent consistency. For queue and stack, local linearizability is incomparable to quiescent consistency.* ◀

Local linearizability also does not imply the stronger condition of quantitative quiescent consistency. Like local linearizability, quiescent consistency and quantitative quiescent consistency are compositional [22, 24]. For details, please see [15].

Consistency Conditions for Distributed Shared Memory. There is extensive research on consistency conditions for distributed shared memory [3, 4, 8, 14, 18, 19, 26, 27, 28]. In [15], we compare local linearizability against coherence, PRAM consistency, processor consistency, causal consistency, and local consistency. All these conditions split a history into subhistories and require consistency of the subhistories. For our comparison, we first define a sequential specification S_M for a single memory location. We assume that each memory location is preinitialized with a value $v_{init} \in V$. A read-operation returns the value of the last write-operation that was performed on the memory location or v_{init} if there was no write-operation. We denote write-operations by `ins` and read-operations by `head`. Formally, we define S_M as $S_M = \{\texttt{head}(v_{init})\}^\star \cdot \{\texttt{ins}(v)\texttt{head}(v)^i \mid i \geq 0, v \in V\}^\star$. Note that read-operations are data observations and the same value can be read multiple times. For brevity, we only consider histories that involve a single memory location. In the following, we summarize our comparison. For details, please see [15].

While local linearizability is well-suited for concurrent data structures, this is not necessarily true for the mentioned shared-memory consistency conditions. On the other hand, local linearizability appears to be problematic for shared memory. Consider the locally linearizable history in Figure 4. There, the read values oscillate between different values that were written by different threads. Therefore, local linearizability does not imply any of the

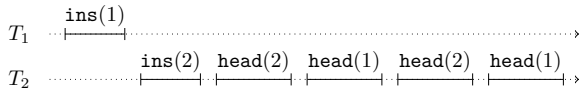


Figure 4 Problematic shared-memory history.

shared-memory consistency conditions. In [15], we further show that local linearizability is incomparable to all considered shared-memory conditions.

6 Locally Linearizable Implementations

In this section, we focus on locally linearizable data structure implementations that are generic as follows: Choose a linearizable implementation of a data structure Φ wrt a sequential specification S_Φ, and we turn it into a (distributed) data structure called LLD Φ that is locally linearizable wrt S_Φ. An LLD implementation takes several copies of Φ (that we call backends) and assigns to each thread T a backend Φ_T. Then, when thread T inserts an element into LLD Φ, the element is inserted into Φ_T, and when an arbitrary thread removes an element from LLD Φ, the element is removed from some Φ_T eagerly, i.e., if no element is found in the attempted backend Φ_T the search for an element continues through all other backends. If no element is found in one round through the backends, then we return `empty`.

▶ Proposition 6 (LLD correctness). Let Φ be a data structure implementation that is linearizable wrt a sequential specification S_Φ. Then LLD Φ is locally linearizable wrt S_Φ.

Proof. Let $\mathbf{h}$ be a history of LLD Φ. The crucial observation is that each thread-induced history $\mathbf{h_T}$ is a backend history of Φ_T and hence linearizable wrt S_Φ. ◀

Any number of copies (backends) is allowed in this generic implementation of LLD Φ. If we take just one copy, we end up with a linearizable implementation. Also, any way of choosing a backend for removals is fine. However, both the number of backends and the backend selection strategy upon removals affect the performance significantly. In our LLD Φ implementations we use one backend per thread, resulting in no contention on insertions, and always attempt a local remove first. If this does not return an element, then we continue a search through all other backends starting from a randomly chosen backend.

LLD Φ is an implementation closely related to Distributed Queues (DQs) [16]. A DQ is a (linearizable) *pool* that is organized as a single segment of length ℓ holding ℓ backends. DQs come in different flavours depending on how insert and remove methods are distributed across the segment when accessing backends. No DQ variant in [16] follows the LLD approach described above. Moreover, while DQ algorithms are implemented for a fixed number of backends, LLD Φ implementations manage a segment of variable size, one backend per (active) thread. Note that the strategy of selecting backends in the LLD Φ implementations is similar to other work in work stealing [30]. However, in contrast to this work our data structures neither duplicate nor lose elements. LLD (stack) implementations have been successfully applied for managing free lists in the fast and scalable memory allocator scalloc [5]. The guarantees provided by local linearizability are not needed for the correctness of scalloc, i.e., the free lists could also use a weak pool (pool without a linearizable emptiness check). However, the LLD stack implementations provide good caching behavior when threads operate on their local stacks whereas a weak pool would potentially negatively impact performance.

We have implemented LLD variants of strict and relaxed queue and stack implementations. None of our implementations involves observation methods, but the LLD algorithm can easily be extended to support observation methods. For details, please see [15]. Finally, let us note that we have also experimented with other locally linearizable implementations that lacked the genericity of the LLD implementations, and whose performance evaluation did not show promising results (see [15]). As shown in Sec. 4, a locally linearizable pool is not a linearizable pool, i.e., it lacks a linearizable emptiness check. Indeed, LLD implementations do not provide a linearizable emptiness check, despite of eager removes. We provide LL$^+$D Φ, a variant of LLD Φ, that provides a linearizable emptiness check under mild conditions on the starting implementation Φ (see [15] for details).

Experimental Evaluation. All experiments ran on a uniform memory architecture (UMA) machine with four 10-core 2GHz Intel Xeon E7-4850 processors supporting two hardware threads (hyperthreads) per core, 128GB of main memory, and Linux kernel version 3.8.0. We also ran the experiments without hyper-threading resulting in no noticeable difference. The CPU governor has been disabled. All measurements were obtained from the artifact-evaluated Scal benchmarking framework [10, 17, 9], where you can also find the code of all involved data structures. Scal uses preallocated memory (without freeing it) to avoid memory management artifacts. For all measurements we report the arithmetic mean and the 95% confidence interval (sample size=10, corrected sample standard deviation).

In our experiments, we consider the linearizable queues Michael-Scott queue (MS) [29] and LCRQ [31] (improved version [32]), the linearizable stacks Treiber stack (Treiber) [36] and TS stack [12], the k-out-of-order relaxed k-FIFO queue [25] and k-Stack [20] and linearizable well-performing pools based on distributed queues using random balancing [16] (1-RA DQ for queue, and 1-RA DS for stack). For each of these implementations (but the pools) we provide LLD variants (LLD LCRQ, LLD TS stack, LLD k-FIFO, and LLD k-Stack) and, when possible, LL$^+$D variants (LL$^+$D MS queue and LL$^+$D Treiber stack). Making the pools locally linearizable is not promising as they are already distributed. Whenever LL$^+$D is achievable for a data structure implementation Φ we present only results for LL$^+$D Φ as, in our workloads, LLD Φ and LL$^+$D Φ implementations perform with no visible difference.

We evaluate the data structures on a Scal producer-consumer benchmark where each producer and consumer is configured to execute 10^6 operations. To control contention, we add a busy wait of $5\mu s$ between operations. This is important as too high contention results in measuring hardware or operating system (e.g., scheduling) artifacts. The number of threads ranges between 2 and 80 (number of hardware threads) half of which are producers and half consumers. To relate performance and scalability we report the number of data structure operations per second. Data structures that require parameters to be set are configured to allow maximum parallelism for the producer-consumer workload with 80 threads. This results in $k = 80$ for all k-FIFO and k-Stack variants (40 producers and 40 consumers in parallel on a single segment), $p = 80$ for 1-RA-DQ and 1-RA-DS (40 producers and 40 consumers in parallel on different backends). The TS Stack algorithm also needs to be configured with a delay parameter. We use optimal delay ($7\mu s$) for the TS Stack and zero delay for the LLD TS Stack, as delays degrade the performance of the LLD implementation.

Figure 5 shows the results of the producer-consumer benchmarks. Similar to experiments performed elsewhere [12, 20, 25, 31] the well-known algorithms MS and Treiber do not scale for 10 or more threads. The state-of-the-art linearizable queue and stack algorithms LCRQ and TS-interval Stack either perform competitively with their k-out-of-order relaxed counter parts k-FIFO and k-Stack or even outperform and outscale them. For any imple-

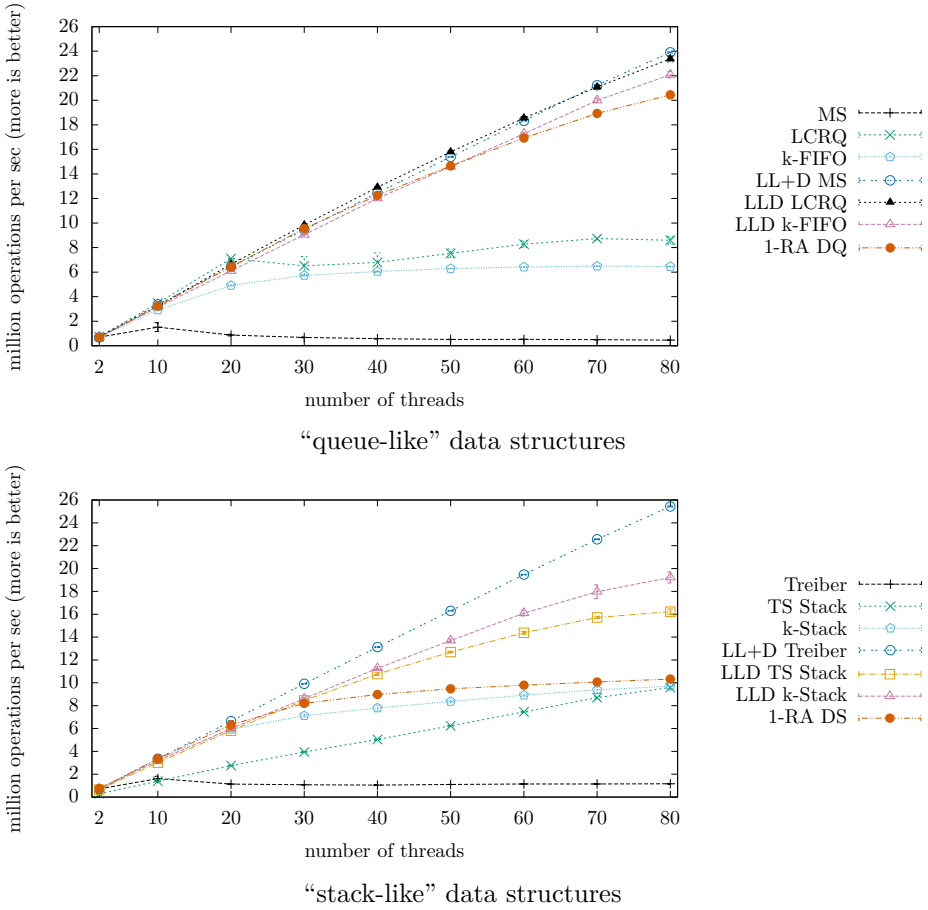


Figure 5 Performance and scalability of producer-consumer microbenchmarks with an increasing number of threads on a 40-core (2 hyperthreads per core) machine.

mentation Φ, LLD Φ and LL$^+$D Φ (when available) perform and scale significantly better than Φ does, even slightly better than the state-of-the-art pool that we compare to. The best improvement show LLD variants of MS queue and Treiber stack. The speedup of the locally linearizable implementation to the fastest linearizable queue (LCRQ) and stack (TS Stack) implementation at 80 threads is 2.77 and 2.64, respectively. The performance degradation for LCRQ between 30 and 70 threads aligns with the performance of `fetch-and-inc`—the CPU instruction that atomically retrieves and modifies the contents of a memory location—on the benchmarking machine, which is different on the original benchmarking machine [31]. LCRQ uses `fetch-and-inc` as its key atomic instruction.

7 Conclusion & Future Work

Local linearizability splits a history into a set of thread-induced histories and requires consistency of all such. This yields an intuitive consistency condition for concurrent objects that enables new data structure implementations with superior performance and scalability. Local linearizability has desirable properties like compositionality and well-behavedness for container-type data structures. As future work, it is interesting to investigate the guarantees that local linearizability provides to client programs along the line of [13].

References

1 iOS Developer Library, Concurrency Programming Guide, Dispatch Queues. URL: https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html.

2 Y. Afek, G. Korland, and E. Yanovsky. Quasi-Linearizability: Relaxed Consistency for Improved Concurrency. In *OPODIS*, pages 395–410, 2010.

3 M. Ahamad, R.A. Bazzi, R. John, P. Kohli, and G. Neiger. The Power of Processor Consistency. In *SPAA*, pages 251–260, 1993.

4 M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, and P.W. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

5 M. Aigner, C. M. Kirsch, M. Lippautz, and A. Sokolova. Fast, multicore-scalable, low-fragmentation memory allocation through large virtual memory and global data structures. In *OOPSLA*, pages 451–469, 2015.

6 D. Alistarh, J. Kopinsky, J. Li, and N. Shavit. The SprayList: A Scalable Relaxed Priority Queue. In *PPoPP*, pages 11–20, 2015.

7 A. Bouajjani, M. Emmi, C. Enea, and J. Hamza. On Reducing Linearizability to State Reachability. In *ICALP*, pages 95–107, 2015.

8 S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski. Replicated Data Types: Specification, Verification, Optimality. In *POPL*, pages 271–284, 2014.

9 POPL 2015 Artifact Evaluation Committee. POPL 2015 Artifact Evaluation. Accessed on 01/14/2015. URL: http://popl15-aec.cs.umass.edu/home/.

10 Computational Systems Group, University of Salzburg. Scal: High-Performance Multicore-Scalable Computing. URL: http://scal.cs.uni-salzburg.at.

11 J. Derrick, B. Dongol, G. Schellhorn, B. Tofan, O. Travkin, and H. Wehrheim. Quiescent Consistency: Defining and Verifying Relaxed Linearizability. In *FM*, pages 200–214, 2014.

12 M. Dodds, A. Haas, and C.M. Kirsch. A Scalable, Correct Time-Stamped Stack. In *POPL*, pages 233–246, 2015.

13 I. Filipovic, P.W. O'Hearn, N. Rinetzky, and H. Yang. Abstraction for concurrent objects. *Theor. Comput. Sci.*, 411(51-52):4379–4398, 2010.

14 J.R. Goodman. *Cache consistency and sequential consistency.* University of Wisconsin-Madison, Computer Sciences Department, 1991.

15 A. Haas, T.A. Henzinger, A. Holzer, C.M. Kirsch, M. Lippautz, H. Payer, A. Sezgin, A. Sokolova, and H. Veith. Local Linearizability. *CoRR*, abs/1502.07118, 2016.

16 A. Haas, T.A. Henzinger, C.M. Kirsch, M. Lippautz, H. Payer, A. Sezgin, and A. Sokolova. Distributed Queues in Shared Memory: Multicore Performance and Scalability through Quantitative Relaxation. In *CF*, 2013.

17 A. Haas, T. Hütter, C.M. Kirsch, M. Lippautz, M. Preishuber, and A. Sokolova. Scal: A Benchmarking Suite for Concurrent Data Structures. In *NETYS*, pages 1–14, 2015.

18 A. Heddaya and H. Sinha. Coherence, Non-coherence and Local Consistency in Distributed Shared Memory for Parallel Computing. Technical report, Computer Science Department, Boston University, 1992.

19 J.L. Hennessy and D.A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

20 T.A. Henzinger, C.M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova. Quantitative relaxation of concurrent data structures. In *POPL*, pages 317–328, 2013.

21 T.A. Henzinger, A. Sezgin, and V. Vafeiadis. Aspect-Oriented Linearizability Proofs. In *CONCUR*, pages 242–256, 2013.

22 M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

23 M. Herlihy and J.M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

24 R. Jagadeesan and J. Riely. Between Linearizability and Quiescent Consistency - Quantitative Quiescent Consistency. In *ICALP*, pages 220–231, 2014.

25 C.M. Kirsch, M. Lippautz, and H. Payer. Fast and Scalable, Lock-free k-FIFO Queues. In *PaCT*, pages 208–223, 2013.

26 L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Comput.*, 28(9):690–691, September 1979.

27 R.J. Lipton and J.S. Sandberg. PRAM: A Scalable Shared Memory. Technical Report Nr. 180, Princeton University, Department of Computer Science, 1988.

28 R.J. Lipton and J.S. Sandberg. Oblivious memory computer networking, September 28 1993. CA Patent 1,322,609.

29 M.M. Michael and M.L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *PODC*, pages 267–275, 1996.

30 M.M. Michael, M.T. Vechev, and V.A. Saraswat. Idempotent Work Stealing. In *PPoPP*, pages 45–54, 2009.

31 A. Morrison and Y. Afek. Fast Concurrent Queues for x86 Processors. In *PPoPP*, pages 103–112, 2013.

32 Multicore Computing Group, Tel Aviv University. Fast Concurrent Queues for x86 Processors. Accessed on 01/28/2015. URL: `http://mcg.cs.tau.ac.il/projects/lcrq/`.

33 H. Rihani, P. Sanders, and R. Dementiev. MultiQueues: Simpler, Faster, and Better Relaxed Concurrent Priority Queues. *CoRR*, 2014. `arXiv:1411.1209`.

34 A. Sezgin. Sequential Consistency and Concurrent Data Structures. *CoRR*, abs/1506.04910, 2015.

35 N. Shavit. Data Structures in the Multicore Age. *CACM*, 54(3):76–84, March 2011.

36 R.K. Treiber. Systems Programming: Coping with Parallelism. Technical Report RJ-5118, IBM Research Center, 1986.

Robustness against Consistency Models with Atomic Visibility

Giovanni Bernardi[1] **and Alexey Gotsman**[2]

1 **IMDEA Software Institute, Madrid, Spain**
2 **IMDEA Software Institute, Madrid, Spain**

Abstract

To achieve scalability, modern Internet services often rely on distributed databases with consistency models for transactions weaker than serializability. At present, application programmers often lack techniques to ensure that the weakness of these consistency models does not violate application correctness. We present criteria to check whether applications that rely on a database providing only weak consistency are *robust*, i.e., behave as if they used a database providing serializability. When this is the case, the application programmer can reap the scalability benefits of weak consistency while being able to easily check the desired correctness properties. Our results handle systematically and uniformly several recently proposed weak consistency models, as well as a mechanism for strengthening consistency in parts of an application.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Robustness, Replication, Consistency models, Transactions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.7

1 Introduction

To achieve scalability and availability, modern Internet services often rely on large-scale databases that replicate and partition data across a large number of nodes and/or a wide geographical span (e.g., [14, 20, 26, 3, 6, 4, 5, 11, 22, 27, 10]). The database clients invoke transactions on the data at any of the nodes, and the nodes communicate changes to each other using message passing. Ideally, we want this distributed system to provide strong guarantees about transaction processing, such as *serializability* [8]: the results of concurrently executing a set of transactions could be obtained if these transactions were executed serially in some order. Serializability is useful because it allows an application programmer to easily establish desired correctness properties. For example, to check that the transactions of an application preserve a given data integrity constraint, the programmer only needs to check that every transaction does so when executed in isolation, without worrying about concurrency. Unfortunately, achieving serializability requires excessive synchronisation among database nodes, which slows down the database and even makes it unavailable if network connections between replicas fail [17, 1]. For this reason, nowadays large-scale databases often provide weak consistency guarantees, which allow non-serializable behaviours, called *anomalies*.

As a motivating example, consider a toy on-line auction application with transactions defined by the *transactional programs* in Figure 1. The program `RegUser` creates a new user account. It manipulates the table USERS, whose rows contain a primary key (uId) and a nickname. An invocation of `RegUser(uname)` inserts a new row in USERS only if the nickname `uname` does not appear in USERS, to ensure that nicknames are unique. The program `ViewUsers` can be used to view all the users. Some databases [22, 26, 3] may allow

```
RegUser(uname)
select name
from USERS
as ret_name
where name = uname;
if ret_name defined
then abort
else
insert into USERS
values (new uId, uname);
```

```
StoreBid(iid,val)
insert into BIDS
values
(new bId, iid, val);
select nbids
from ITEMS
as n where iId = iid;
update ITEMS
set nbids = n + 1
where iId = iid;
```

```
ViewItem(iid)
select *
from ITEMS as ret_item
where iId = iid;
```

```
ViewUsers()
select *
from USERS
as ret_users;
```

USERS(uId, name)
ITEMS(iId, desc, nbids)
BIDS(bId, iId, val)

Figure 1 SQL code and table schema for an auction application. Primary keys are underlined.

executions of `RegUser` and `ViewUsers` such as the one sketched in Figure 2(c). There two invocations of `RegUser` generate the transactions T_1 and T_2; these write two rows of USERS, denoted by x and y, to register the users *Alice* and *Bob*. The program `ViewUsers`() then is invoked twice; the invocation in T_3 sees *Alice* but not *Bob*, while the invocation in T_4 sees *Bob* but not *Alice*. This result, called a *long fork* anomaly, cannot be obtained by executing the four transactions in any sequence and, hence, is not serializable.

The past few years have seen a number of proposals of new transactional consistency models for modern large-scale databases [22, 11, 26, 3, 5], differing in how much they weaken consistency, by exposing such anomalies, in exchange for improved performance. Unfortunately, application programmers often lack techniques to ensure that the weakness of these consistency models does not violate application correctness. This situation hinders the adoption of the novel consistency models by mainstream database developers and application programmers.

One way to address this problem is using the notion of application *robustness* [16, 15, 9]. An application is robust against a particular weak consistency model if it behaves the same whether using a database providing this model or serializability. If an application is robust against a given weak consistency model, then programmers can reap the performance benefits of using weak consistency while being able to easily check the desired correctness properties.

In this paper we develop criteria for checking the robustness of applications against three recently proposed consistency models—causal (aka causal+) consistency (CC) [22], prefix consistency [11] (PC) and parallel snapshot isolation [26] (PSI, aka non-monotonic snapshot isolation [3]). As a corollary of our results, we also derive an existing robustness criterion [16] for a classical model of snapshot isolation [6] (SI). Our criteria also handle variants of the consistency models that allow application programmers to request that certain transactions be executed under serializability and thereby ensure the robustness of applications that are not robust otherwise.

We handle the above four consistency models in a uniform and systematic way by exploiting a recently proposed framework [12] for declaratively specifying their semantics (Section 2). In particular, all of the consistency models that we consider guarantee the *atomic visibility* of transactions: either all or none of the writes performed by a transaction can be observed by other transactions. This allows us to simplify reasoning needed to establish robustness criteria by abstracting from internals of transactions in application executions. We first propose a *dynamic* robustness criterion that checks whether a given execution is serializable (Section 3). We formulate this criterion in terms of the dependency graph of the execution [2], describing several kinds of relationships between its transactions: an execution

is serializable if its dependency graph contains no cycles of a certain form, which we call *critical*. Criteria for robustness against different consistency models differ in which cycles are considered critical. We then illustrate how our dynamic robustness criteria on a single execution can be lifted to *static* criteria that check that all executions of a given application are serializable (Section 4).

2 Consistency Model Specifications

We start by recalling from [12] a formal model of database computations and the specifications of the consistency models that we handle. These specifications are declarative, which greatly simplifies our formal development. Nonetheless, as shown in [12], the specifications are equivalent to certain operational specifications, close to implementations.

We consider a database storing ***objects*** $\mathsf{Obj} = \{x, y, \ldots\}$, which we assume to be natural-valued. Clients interact with the database by issuing `read` and `write` operations on the objects, grouped into transactions. We denote each operation invocation by an ***event*** (ι, o), where ι is an identifier from a denumerable set $\mathsf{EventId}$, and $o \in \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}$ describes the operation invoked and its outcome: reading a value n from an object x or writing n to x. We range over events by e, f, g and denote the set of all events by Event. In the following we denote irrelevant expressions by $\_$, and write $e \vdash \mathtt{write}(x, n)$ if $e = (\_, \mathtt{write}(x, n))$ and $e \vdash \mathtt{read}(x, n)$ if $e = (\_, \mathtt{read}(x, n))$. A binary relation $<$ is a ***strict partial order*** if it is transitive and irreflexive. It is ***total*** if additionally for all elements a and b, we have $a < b$, $b < a$ or $a = b$.

▶ **Definition 1.** A ***transaction*** $T, S, \ldots$ is a pair $(E, <_{\mathsf{po}})$, where $E \subseteq \mathsf{Event}$ is a finite, non-empty set of events with distinct identifiers, and the ***program order*** $<_{\mathsf{po}}$ is a total order over E. A ***history*** H is a finite set of transactions with disjoint sets of event identifiers. An annotated history (H, level) is a pair where H is an history and $\mathsf{level} : H \to \{\mathsf{SER}, \bot\}$. An ***execution*** is a triple $X = ((H, \mathsf{level}), <_{\mathsf{hb}}, <_{\mathsf{ar}})$, where (H, level) is an annotated history, $<_{\mathsf{hb}}$ is a strict partial order over H, and $<_{\mathsf{ar}}$ is a total order over H such that $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$. We refer to $<_{\mathsf{hb}}$ and $<_{\mathsf{ar}}$ as ***happens-before*** and ***arbitration***. ◀

We denote components of an execution as in $X.H$ and use the same notation for similar structures.

A transaction records a set of operations and the order in which the client program invoked them. A history records transactions that committed in a finite database computation. For simplicity we elide the treatment of aborted and ongoing transactions, as well as infinite database computations. Annotated histories enrich histories with a function level that records which transactions the programmer requested to execute under serializability, and which transactions under the weak consistency model offered by the underlying database. Finally, executions enrich annotated histories with a happens-before order and an arbitration order, which declaratively represent internal database processing. Intuitively, $T <_{\mathsf{hb}} S$ means that S is aware of the updates performed by T, and thus the outcome of the operations in S may depend on the effects of T. We call transactions that are not related by happens-before ***concurrent***. The relationship $T <_{\mathsf{ar}} S$ means that the versions of objects written by S supersede those written by T. The constraint $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$ ensures that writes by a transaction T supersede those that T is aware of.

We use the set $\{\mathsf{CC}, \mathsf{PC}, \mathsf{PSI}, \mathsf{SI}, \mathsf{SER}\}$ to refer to the consistency models that we treat (Section 1), and we range over this set by wm. In Figure 4 we specify these consistency models as combination of the axioms in Figure 3, constraining executions. Formally, we let

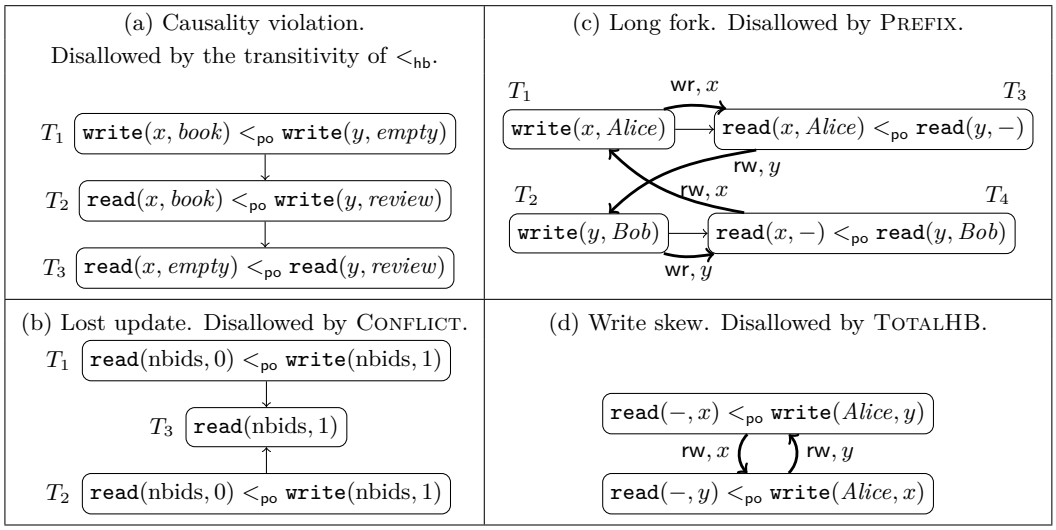


Figure 2 Non-serializable executions illustrating anomalies. Boxes represent transactions, and thin arrows between boxes represent the happens-before relation. We omit arbitration edges to avoid clutter. The thick arrows marked wr/rw are explained in Section 3.

$\forall (E, <_{\mathsf{po}}) \in H.\, \forall e \in E.\, \forall x, n.$ $e \vdash \mathtt{read}(x, n) \implies (\mathsf{before}(e, <_{\mathsf{po}}, \_\ x) = \emptyset \vee \max(\mathsf{before}(e, <_{\mathsf{po}}, \_\ x), <_{\mathsf{po}}) \vdash \_(\_, n))$	(Int)
$\forall T \in H.\, \forall x, n.$ $T \vdash \mathtt{read}(x, n) \implies ((\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset \wedge n = 0) \vee$ $\max(\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x), <_{\mathsf{ar}}) \vdash \_(\_, n))$	(Ext)
$\forall T, S \in H.\, (\exists x.\, T \vdash \mathtt{write}(x, \_) \wedge S \vdash \mathtt{write}(x, \_)) \implies T = S \vee T <_{\mathsf{hb}} S \vee S <_{\mathsf{hb}} T$	(Conflict)
$<_{\mathsf{ar}}\ ;\ <_{\mathsf{hb}} \subseteq <_{\mathsf{hb}}$	(Prefix)
$\forall T, S \in H.\, T = S \vee T <_{\mathsf{hb}} S \vee S <_{\mathsf{hb}} T$	(TotalHB)
$\forall T, S \in H.\, (\mathsf{level}(T) = \mathsf{level}(S) = \mathsf{SER}) \implies T = S \vee T <_{\mathsf{hb}} S \vee S <_{\mathsf{hb}} T$	(SerTotal)

Figure 3 Consistency axioms constraining an execution $((H, \mathsf{level}), <_{\mathsf{hb}}, <_{\mathsf{ar}})$.

the set of annotated histories allowed by a consistency model wm be given by $\mathsf{hist}(wm) = \{(X.H, X.\mathsf{level}) \mid X \models wm\}$. We now explain the axioms and the anomalies that they (dis)allow. We summarise these anomalies in Figure 2.

Given a total order $< \subseteq A \times A$ and a set $B \subseteq A$, we write $\max(B, <)$ for the element $b \in B$ such that $\forall a \in B.\, a \leq b$; if $A = \emptyset$, then $\max(B, <)$ is undefined. We define min in the obvious dual manner. In the following, when we write $\max(B, <)$ or $\min(B, <)$, we assume that they are defined. Given a partial order $< \subseteq A \times A$ and an $a \in A$, we define the downset of a as $\mathsf{before}(a, <) = \{a' \in A \mid a' < a\}$, and let $\mathsf{before}(a, <, \mathsf{op}\, x) = \mathsf{before}(a, <) \cap \{a' \in A \mid a' \vdash \mathsf{op}(x, \_)\}$.

The ***internal consistency axiom*** Int ensures that, within a transaction, the database provides sequential semantics: in a transaction $(E, <_{\mathsf{po}})$, a read event e on an object x returns the value of the last event on x preceding e. The events on x preceding e are given by the set $\mathsf{before}(e, <_{\mathsf{po}}, \_\ x)$. If in $(E, <_{\mathsf{po}})$ a read e on x is not preceded by an operation on the same object (i.e., $\mathsf{before}(e, <_{\mathsf{po}}, \_\ x) = \emptyset$), then its value is determined in terms of writes by other transactions, using the ***external consistency axiom*** Ext. To formulate

$$
\begin{aligned}
\mathsf{CC} &\equiv \textsc{Int} \wedge \textsc{Ext} \wedge \textsc{SerTotal} \\
\mathsf{PSI} &\equiv \mathsf{CC} \wedge \textsc{Conflict} \\
\mathsf{PC} &\equiv \mathsf{CC} \wedge \textsc{Prefix} \\
\mathsf{SI} &\equiv \mathsf{PSI} \wedge \textsc{Prefix} \\
\mathsf{SER} &\equiv \textsc{Int} \wedge \textsc{Ext} \wedge \textsc{TotalHB}
\end{aligned}
$$

Figure 4 Consistency model definitions.

Ext we lift the $\vdash$ notation to transactions. For given $T = (E, <_{\mathsf{po}})$, $x \in \mathsf{Obj}$ and $n \in \mathbb{N}$, we write:

- $T \vdash \texttt{write}(x, n)$ if $\max(\{e \in E \mid e \vdash \texttt{write}(x, \_)\}, <_{\mathsf{po}}) \vdash \texttt{write}(x, n)$; and
- $T \vdash \texttt{read}(x, n)$ if $\min(\{e \in E \mid e \vdash \_(x, \_)\}, <_{\mathsf{po}}) \vdash \texttt{read}(x, n)$.

According to Ext, if a transaction T reads an object x before writing to it, then the value returned by the read is determined by the transactions that happen before T and that write to x; the set of such transactions is given by $\mathsf{before}(T, <_{\mathsf{hb}}, \texttt{write}\ x)$. If this set is empty, then T reads the initial value 0; otherwise it reads the value written by the transaction from the set that is the last one in $<_{\mathsf{ar}}$. Ext guarantees the *atomic visibility* of a transaction: either all or none of its writes can be visible to another transaction. A detailed discussion on the matter can be found in [12, Section 3].

The axiom SerTotal formalises the additional guarantees provided to transactions that the application programmer required to execute on serializability, as recorded by level. We discuss this axiom in more detail below; for now we assume $\mathsf{level} = (\lambda T.\, \bot)$ for all executions.

The axioms Int, Ext and SerTotal define causal consistency (CC) [22]. This forbids the ***causality violation*** anomaly in Figure 2(a), where a user sees the review, but not the book it was associated with. This anomaly is forbidden because $<_{\mathsf{hb}}$ is transitive, so we must have $T_1 <_{\mathsf{hb}} T_3$. Since $T_1 <_{\mathsf{ar}} T_2$, the writes by T_2 supersede those by T_1, and thus Ext implies $T_3 \vdash \texttt{read}(y, \mathit{review})$ and $T_3 \vdash \texttt{read}(x, \mathit{book})$.

Causal consistency allows the ***lost update*** anomaly, illustrated by the execution in Figure 2(b). This execution may arise from the programs `ViewItem` and `StoreBid` in Figure 1, which respectively let a user query the information about an item and bid on an item. They access a table ITEMS, whose rows represent items and contain a primary key (iId), an item description (desc) and the number of existing bids (nbids). The anomaly in Figure 2(b) is caused by two invocations of the program `StoreBid` that generate the transactions T_1 and T_2, meant to increase the number of bids of an item. The two transactions read the initial number of bids for the item, namely 0, and concurrently modify it, resulting in one addition getting lost. This is observed by a third transaction T_3 generated by `ViewItem`. The lost update anomaly is disallowed by the axiom Conflict, which guarantees that transactions updating the same object are not concurrent. This axiom rules out any execution with the history in Figure 2(b). We specify parallel snapshot isolation (PSI) [26] by strengthening causal consistency with the axiom Conflict. This consistency models allows the ***long fork*** anomaly given in Figure 2(c), which we discussed in Section 1.

We specify prefix consistency (PC) [11] and snapshot isolation (SI) [6] by strengthening respectively CC and PSI via the axiom Prefix: if T observes S, then it also observes all $<_{\mathsf{ar}}$-predecessors of S, which is formalised using sequential composition ; of relations. The axiom Prefix disallows any execution with the history in Figure 2(c): T_1 and T_2 have to be related by $<_{\mathsf{ar}}$ one way or another; but then by Prefix, either T_4 has to observe *Alice* or T_3 has to observe *Bob*. Like causal consistency, prefix consistency allows the lost update

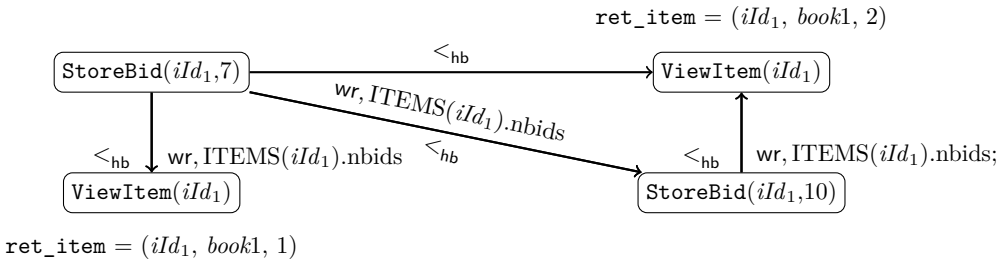


Figure 5 An execution produced by the programs in Figure 1 and its dynamic dependency graph (the latter explained in Section 3). We assume $\mathsf{level} = (\lambda T.\, \bot)$. We omit events inside transactions and only show the parameters and the return values of the corresponding programs. Since $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$, all the relevant arbitration edges coincide with the happens-before ones, and we omit the $<_{\mathsf{ar}}$ label.

anomaly in Figure 2(b). Snapshot isolation disallows it, but allows the anomaly of ***write skew***, illustrated by the execution in Figure 2(d). This execution could be produced by `RegUser` in Figure 1. The objects x and y correspond to different rows in the table USERS. Two invocations of `RegUser` generate transactions that miss each other's writes and, as a consequence, concurrently register two users with the same nickname.

We define serializability (SER) using the axiom TotalHB, which requires happens-before to be total. It disallows any execution with one of the histories in Figure 2.

Finally, the consistency models we consider include the axiom SerTotal, which requires happens-before to be total on transactions that the programmer marked as serializable. For example, in a database providing CC, the history in Figure 2(c) can be disallowed by letting $\mathsf{level}(T_1) = \mathsf{level}(T_2) = \mathsf{SER}$ and $\mathsf{level}(T_3) = \mathsf{level}(T_4) = \bot$. This is because then SerTotal forces T_1 and T_2 to be related by happens-before, and therefore either T_3 or T_4 has to observe *both* T_1 and T_2. We can disallow the history in Figure 2(b) by letting $\mathsf{level}(T_1) = \mathsf{level}(T_2) = \mathsf{SER}$ and $\mathsf{level}(T_3) = \bot$.

As the last example, we consider the execution X in Figure 5, which is produced on a PSI database by the programs `StoreBid` and `ViewItem` in Figure 1: $X \models \mathsf{PSI}$. In X the two transactions due to `StoreBid` submit bids for an item iId_1: one bid of 7 dollars and one bid of 10 dollars. The other two transactions due to `ViewItem` query the state of the item. The query on the left sees the bid of 7, but not that of 10. The query on the right sees both bids. It is easy to check that the history of this execution is serializable. As a matter of fact, the results we develop in the following sections let us show that *any* execution produced by the programs `StoreBid` and `ViewItem` under PSI has a serializable history. Hence, a database can process the corresponding transactions using the PSI concurrency control without exposing any anomalies to its users.

3 Dynamic Robustness Criteria

Our first goal is to define criteria to check whether a single execution X in one of the consistency models that we consider has a serializable history: $X.H \in \mathsf{hist}(\mathsf{SER})$. From these *dynamic* robustness criteria, in the next section we derive *static* criteria to check whether this is the case for all executions of a given application.

Our dynamic criteria are formulated in terms of *dependency graphs*, widely used in the database literature [2]. Let the set of ***labels*** L be defined as follows: $D = \{(\mathsf{wr}, x), (\mathsf{ww}, x) \mid x \in \mathsf{Obj}\}$, $L = D \cup \{(\mathsf{rw}, x) \mid x \in \mathsf{Obj}\}$. We use λ to range over L and s, t to range over

$L^\star$. A graph G is a pair $(H, \longrightarrow)$, where $H \in \mathsf{Hist}$ and $\longrightarrow \subseteq H \times L \times H$. We write $T \xrightarrow{\lambda} S \in G$ in place of $(T, \lambda, S) \in \longrightarrow$. We also use some graph-theoretic notions. A ***path*** π in G is a non-empty finite sequence of edges $T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{n-1}} T_n$. In this case we write $\pi \in G$. The path is a ***cycle*** if $T_0 = T_n$, and it is a ***simple*** cycle if all other pairs of transactions on it are distinct. We also write $T \xrightarrow{\lambda} S \in \pi$ to mean that the edge $T \xrightarrow{\lambda} S$ appears in the path π. Given a graph $G = (H, \longrightarrow)$, we denote by $\xrightarrow{s}{}^+$ the least relation such that

(i) $T \xrightarrow{\lambda}{}^+ S$ if $T \xrightarrow{\lambda} S \in G$; and

(ii) $T \xrightarrow{\lambda s}{}^+ S$ whenever $T \xrightarrow{\lambda} T' \in G$ and $T' \xrightarrow{s}{}^+ S$ for some $T' \in H$.

We denote with $\xrightarrow{s}{}^\star$ the reflexive closure of $\xrightarrow{s}{}^+$, so that $T \xrightarrow{\varepsilon}{}^\star T$ for every $T \in H$. We now define a map from executions into dependency graphs.

▶ **Definition 2.** The ***dynamic dependency graph*** of an execution $X = ((H, \_), <_{\mathsf{hb}}, <_{\mathsf{ar}})$ is $\mathsf{DDG}(X) = (H, \longrightarrow)$, where for every $x \in \mathsf{Obj}$ the relation $\longrightarrow$ contains the following triples:

read-dependency: $T \xrightarrow{\mathsf{wr},\, x} S$ if $S \vdash \mathtt{read}(x, \_)$ and $T = \max(\mathsf{before}(S, <_{\mathsf{hb}}, \mathtt{write}\ x), <_{\mathsf{ar}})$;

write-dependency: $T \xrightarrow{\mathsf{ww},\, x} S$ if $T \vdash \mathtt{write}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $T <_{\mathsf{ar}} S$;

anti-dependency: $T \xrightarrow{\mathsf{rw},\, x} S$ if $T \neq S$ and either

(i) $T \vdash \mathtt{read}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset$, or

(ii) $T' \xrightarrow{\mathsf{wr},\, x} T$ and $T' \xrightarrow{\mathsf{ww},\, x} S$ for some $T' \in H$. ◀

Thus, $T \xrightarrow{\mathsf{wr},\, x} S$ means that S reads T's write to x (cf. Ext in Figure 3), and $T \xrightarrow{\mathsf{ww},\, x} S$ means that S overwrites T's write to x. The relation $T \xrightarrow{\mathsf{rw},\, x} S$ means that S overwrites the write to x read by T (the initial value of an object is overwritten by any write to this object). In Figures 2(c), 2(d) and 5 we draw the dependency graphs with thick edges.

Dependency graphs provide a way to show that executions have serializable histories [2].

▶ **Lemma 3.** *For every X, if $X \models \textsc{Int} \wedge \textsc{Ext}$ and $\mathsf{DDG}(X)$ is acyclic, then $X.H \in \mathsf{hist}(\mathsf{SER})$.*

For instance, the history in Figure 5 is serializable. The graphs of the executions in Figure 2(c, d) contain cycles and, in fact, the histories of these executions are not serializable.

As we now show, to ensure that the history of an execution X arising from a particular consistency model is serializable, it is enough to check that $\mathsf{DDG}(X)$ does not contain cycles of a particular form, which we call *critical*. This more precise characterisation is instrumental in obtaining our static robustness criteria (Section 4).

A path π in a dynamic dependency graph G is ***chord-free*** if, whenever $u \xrightarrow{s}{}^+ v \in \pi$ for some $s \in L^n$ with $n \geq 2$, we have $\neg(u \longrightarrow v \in G)$. A path π is ***rw-minimal*** if, whenever $u \xrightarrow{\mathsf{rw},\, \_} v \in \pi$ and $u \xrightarrow{\lambda} v \in G$, we have $\lambda = (\mathsf{rw}, \_)$. The last notion forces us exclude an rw edge from π if there is another option.

▶ **Definition 4.** Given an execution X, an edge $T \xrightarrow{\lambda} S \in \mathsf{DDG}(X)$ is ***unprotected*** if either $X.\mathsf{level}(T) \neq \mathsf{SER}$ or $X.\mathsf{level}(S) \neq \mathsf{SER}$. A cycle $\pi \in \mathsf{DDG}(X)$ among transactions $T_0, T_1, \dots, T_n$ (where $T_0 = T_n$) that is simple, chord-free and rw-minimal is:

CC-**critical,** if π contains an unprotected edge $T_i \xrightarrow{\mathsf{rw},\, \_} T_{i+1}$ and an unprotected edge $T_j \xrightarrow{\lambda} T_{j+1}$ with $i \neq j$ and $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$;

PC-**critical,** if π contains an unprotected edge $T_i \xrightarrow{\mathsf{rw},\, \_} T_{i+1}$ and at least two adjacent unprotected edges with labels in $\{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$;

PSI-**critical,** if:

1. π contains at least two unprotected rw edges; and

2. for every $T_i \xrightarrow{\mathsf{rw},x} T_{i+1}$, $T_j \xrightarrow{\mathsf{rw},y} T_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$;

SI-**critical,** if:

1. π contains at least two adjacent unprotected rw edges; and

2. for every $T_i \xrightarrow{\mathsf{rw},x} T_{i+1}$, $T_j \xrightarrow{\mathsf{rw},y} T_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$. ◀

The graphs of the executions in Figure 2 (with $\mathsf{level} = (\lambda T.\, \bot)$) contain critical cycles: (c) contains a PSI-critical cycle, and (d) contains an SI-critical cycle.

▶ **Theorem 5.** *For every wm and X, if $X \models wm$, then $\mathsf{DDG}(X)$ contains a cycle if and only if it contains a wm-critical cycle.*

From Theorem 5 and Lemma 3 we obtain our dynamic robustness criterion.

▶ **Corollary 6.** *For every wm and every X, if $X \models wm$ and $\mathsf{DDG}(X)$ contains no wm-critical cycle then $X.H \in \mathsf{hist}(\mathsf{SER})$.*

We note that the above robustness criterion for SI is a variant of an existing one [16, 15]. In our setting, it is just a consequence of our novel criterion for PSI.

To prove Theorem 5 we show how the axioms in Figure 3 impact the properties of edges and paths in dependency graphs. First, observe that there is a relation between wr, ww edges and the orders $<_{\mathsf{hb}}$ and $<_{\mathsf{ar}}$: for every X and every $T, S \in X.H$, the definitions ensure that

$$\begin{aligned} T \xrightarrow{\mathsf{wr},\_} S \in \mathsf{DDG}(X) &\implies T <_{\mathsf{hb}} S; \\ (T \xrightarrow{\mathsf{wr},\_} S \in \mathsf{DDG}(X) \vee T \xrightarrow{\mathsf{ww},\_} S \in \mathsf{DDG}(X)) &\implies T <_{\mathsf{ar}} S; \\ (X \models \textsc{Conflict} \wedge T \xrightarrow{\mathsf{ww},\_} S \in \mathsf{DDG}(X)) &\implies T <_{\mathsf{hb}} S. \end{aligned}$$

These implications let us show that, under certain conditions, if two transactions in a dependency graph are connected by a path, then they are also related by happens-before or arbitration.

▶ **Lemma 7.** *For any X, $s \in L^+$ and $T \xrightarrow{s}{}^{+} S \in \mathsf{DDG}(X)$, if $X \models \textsc{SerTotal}$ then:*

1. *if all the* rw *and* ww *edges in $T \xrightarrow{s}{}^{+} S$ are protected then $T <_{\mathsf{hb}} S$;*
2. *if all the* rw *edges in $T \xrightarrow{s}{}^{+} S$ are protected then $T <_{\mathsf{ar}} S$;*
3. *if $X \models \textsc{Conflict}$ and all the* rw *edges in $T \xrightarrow{s}{}^{+} S$ are protected then $T <_{\mathsf{hb}} S$.*

The following lemma shows that, if $T \xrightarrow{\mathsf{rw},x} S$, then T cannot happen-before S: in this case T would have to read a value at least as up-to-date as that written by S, contradicting the definition of anti-dependencies.

▶ **Lemma 8.** $\forall X.\, \forall x \in \mathsf{Obj}.\, \forall T, S \in X.H.\, T \xrightarrow{\mathsf{rw},x} S \in \mathsf{DDG}(X) \implies S \not<_{\mathsf{hb}} T \wedge T \vdash \mathtt{read}(x, \_) \wedge S \vdash \mathtt{write}(x, \_)$.

Proof of Theorem 5. The *if* implication is obvious, so let us prove the *only if* implication. Suppose that the graph $\mathsf{DDG}(X)$ contains a cycle π'. From π' we can easily build a cycle

$$\pi = T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} T_n \quad (\text{where } T_0 = T_n,\ n \geq 2) \tag{1}$$

in $\mathsf{DDG}(X)$ that is simple, chord-free and rw-minimal. The argument now is a case analysis on the *wm*. Here we consider only CC and PSI and defer the full proof to [7].

Case of $wm = \mathsf{CC}$**.** Lemma 7(2) implies that π contains at least one unprotected rw edge, for otherwise $T_0 <_{\mathsf{ar}} T_0$, contradicting the irreflexivity of $X.<_{\mathsf{ar}}$. Let this edge be $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1}$. Then Lemma 8 ensures that $T_{i+1} \not<_{\mathsf{hb}} T_i$. Since π contains the non-empty

path $T_{i+1} \longrightarrow^{+} T_i$, Lemma 7(1) implies that on this path there is at least one unprotected edge $T_j \xrightarrow{\lambda} T_{j+1}$ with $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$ and $i \neq j$. It follows that π is CC-critical.

Case of $wm = \mathsf{PSI}$**.** First we prove that the cycle π contains at least two unprotected edges rw edges. Since $X \models \mathsf{PSI}$, we know that $X \models \mathsf{CC}$. Thus, the previous argument ensures that the cycle π contains at least one unprotected rw edge, say $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$. Suppose that π contains exactly one such edge. Since $X \models$ Conflict, Lemma 7(3) now ensures $T_{i+1} <_{\mathsf{hb}} T_i$. But by Lemma 8, $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$ implies $T_{i+1} \not<_{\mathsf{hb}} T_i$. The resulting contradiction shows that π must contain at least two unprotected rw edges.

Now we have to prove

$$\forall T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}, T_j \xrightarrow{\mathsf{rw},\, y} T_{j+1} \in \pi. i \neq j \implies x \neq y. \tag{2}$$

Suppose that π does not satisfy (2). Then, as $<_{\mathsf{ar}}$ is total, Definition 2 guarantees that we have either $T_{i+1} \xrightarrow{\mathsf{ww},x} T_{j+1}$ or $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. Since π is a simple cycle, in the first case we contradict either that π is chord-free or that π is rw-minimal. In the second case, we have either (a) $T_{j+1} = T_i$ or (b) $T_{j+1} \neq T_i$. If (a) holds, then we contradict that π is rw-minimal, because $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1} \in \pi$ and $T_i \xrightarrow{\mathsf{ww},\_} T_{i+1}$. If (b) holds, then the sub-path $T_{j+1} \xrightarrow{s}{}^{+} T_{i+1}$ of π contains at least two edges and it is chord-free by construction. But this contradicts $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. It follows that π satisfies (2) above, and thus it is a PSI-critical cycle. ◀

4 Static Robustness Criteria

We now illustrate how the dynamic robustness criteria (Corollary 6) can be lifted to static criteria, which allow programmers to analyse the behaviour of their applications and which can serve as a basis for static analysis tools.

We define an ***application*** $\mathcal{A}$ by a set of ***transactional programs*** $\mathtt{f}_i$, giving the code of its transactions: $\mathcal{A} = \{\mathtt{f}_1, \ldots, \mathtt{f}_n\}$ (e.g., see Figure 1). As is standard in the database literature [16], this abstracts from the rest of the application logic to focus on the parts that directly interact with the database. We call a pair $I = (\mathtt{f}, \overline{\mathtt{v}})$ of a program and a vector of its actual parameters a ***program instance***. An ***application instance*** $\mathcal{I}$ is a set of program instances, and an ***annotated application instance*** is a pair $(\mathcal{I}, \mathsf{level}_{\mathrm{S}})$, where $\mathsf{level}_{\mathrm{S}} : \mathcal{I} \to \{\mathsf{SER}, \bot\}$ defines which programs the programmer requested to execute under serializability. We first formulate criteria for checking the robustness of a particular annotated application instance, resulting from running a set of transactional programs with given parameters. We then sketch how these criteria can be generalised to whole applications.

We aim to illustrate the ideas for lifting dynamic robustness criteria to static ones in the simplest form. To this end, we abstract from the syntax of the programming language and assume that we are only given approximate information about the set of objects read or written by each transactional program. Namely, we assume a function rwsets that maps every program instance I to a triple $\mathsf{rwsets}(I) = (R^{\diamond}, W^{\diamond}, W^{\square})$. Informally, $R^{\diamond}$ and $W^{\diamond}$ are the sets of all the objects that *may* be read or written in some execution of I, and $W^{\square}$ is a set of the objects that *must* be written in any execution of I, with the proviso that $W^{\square} \subseteq W^{\diamond}$. For instance, for $I = (\mathtt{StoreBid}, \langle iId_1, 7\rangle)$ (Figure 1) we have

$$\mathsf{rwsets}(I) = (\{\text{ITEMS}(iId_1).\text{nbids}\}, \{\text{ITEMS}(iId_1).\text{nbids}, \text{BIDS}(*).*\}, \{\text{ITEMS}(iId_1).\text{nbids}\})$$

where $*$ means "all fields" or "all rows".

To formalise the meaning of the read/write sets, we define a relation that determines if a history can be produced by a given $\mathcal{I}$. We let $T \Vdash I$ for $\mathsf{rwsets}(I) = (R^{\diamond}, W^{\diamond}, W^{\square})$, if:

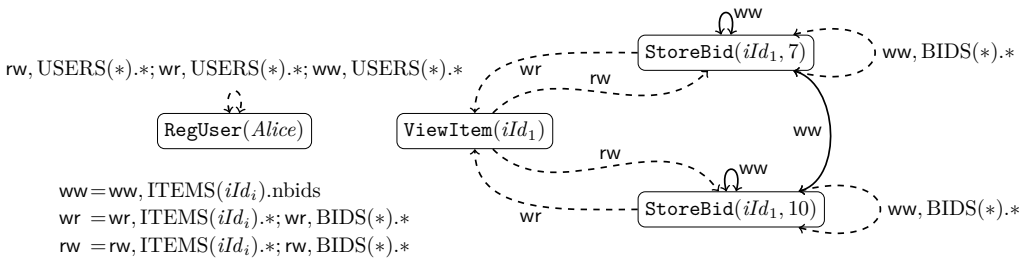


Figure 6 The static dependency graph $\mathsf{SDG}(\mathcal{I})$ of the application instance $\mathcal{I}$ defined in (3). We draw may edges with dashed arrows, and must edges with solid arrows.

(i) $T \vdash \texttt{write}(x, \_) \implies x \in W^{\Diamond}$;
(ii) $T \vdash \texttt{read}(x, \_) \implies x \in R^{\Diamond}$;
(iii) $x \in W^{\Box} \implies T \vdash \texttt{write}(x, \_)$.

We lift the relation $\Vdash$ to annotated histories and annotated application instances:

$$(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}}) \iff \forall T \in H.\, \exists I \in \mathcal{I}.\, T \Vdash I \wedge \mathsf{level}_{\mathrm{S}}(I) = \mathsf{level}(T).$$

Note that the definition of $\Vdash$ allows multiple transactions in H to be associated to a single I in $\mathcal{I}$. For example, we have $(H, (\lambda T.\, \bot)) \Vdash (\mathcal{I}, (\lambda I.\, \bot))$ for the history H in Figure 5 and

$$\mathcal{I} = \{(\texttt{RegUser}, \mathit{Alice}), (\texttt{ViewItem}, \mathit{iId}_1), (\texttt{StoreBid}, \langle \mathit{iId}_1, 7\rangle), (\texttt{StoreBid}, \langle \mathit{iId}_2, 10\rangle)\}. \quad (3)$$

We formulate our robustness criteria by adapting modal transition systems [21].

▶ **Definition 9.** The ***static dependency graph*** of an application instance $\mathcal{I}$ is a triple $\mathsf{SDG}(\mathcal{I}) = (\mathcal{I}, \dashrightarrow, \longleftrightarrow)$, where the relations $\dashrightarrow$ and $\longleftrightarrow$ are defined as follows. For every $I, J \in \mathcal{I}$, if $\mathsf{rwsets}(I) = (W_I^{\Diamond}, R_I^{\Diamond}, W_I^{\Box})$ and $\mathsf{rwsets}(J) = (W_J^{\Diamond}, R_J^{\Diamond}, W_J^{\Box})$, then:

$$I \overset{\mathsf{wr}, x}{\dashrightarrow} J \iff x \in W_I^{\Diamond} \cap R_J^{\Diamond}; \qquad I \overset{\mathsf{ww}, x}{\dashrightarrow} J \iff x \in W_I^{\Diamond} \cap W_J^{\Diamond};$$
$$I \overset{\mathsf{rw}, x}{\dashrightarrow} J \iff x \in R_I^{\Diamond} \cap W_J^{\Diamond}; \qquad I \overset{\mathsf{ww}, x}{\longleftrightarrow} J \iff x \in W_I^{\Box} \cap W_J^{\Box}.$$

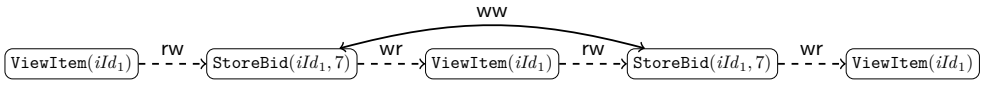
Figure 6 shows the static dependency graph of the $\mathcal{I}$ in (3). Informally, the edges of the static dependency graph $\mathsf{SDG}(\mathcal{I})$ describe possible dependencies between transactions in executions produced by $\mathcal{I}$: an edge $I \overset{\lambda}{\dashrightarrow} J$ represents a dependency that may exist, and an edge $I \overset{\lambda}{\longleftrightarrow} J$ a dependency that must exists. Formally, given an annotated application instance $(\mathcal{I}, \mathsf{level}_{\mathrm{S}})$, we say that the pair $(\mathsf{DDG}(X), X.\mathsf{level})$ is ***over-approximated*** by the pair $(\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}})$, written $(\mathsf{DDG}(X), X.\mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}})$, if for some total function $f : X.H \to \mathcal{I}$ we have:

1. $\forall T \overset{\lambda}{\longrightarrow} S \in \mathsf{DDG}(X).\, f(T) \overset{\lambda}{\dashrightarrow} f(S) \in \mathsf{SDG}(\mathcal{I})$;
2. $\forall I \overset{\lambda}{\longleftrightarrow} J \in \mathsf{SDG}(\mathcal{I})$.
   $\forall T \in f^{-1}(I).\, \forall S \in f^{-1}(J).\, T \overset{\lambda}{\longrightarrow} S \in \mathsf{DDG}(X) \vee S \overset{\lambda}{\longrightarrow} T \in \mathsf{DDG}(X)$; and
3. $\mathsf{level}(T) = \mathsf{level}_{\mathrm{S}}(f(T))$.

▶ **Lemma 10.** $\forall X.\, \forall (\mathcal{I}, \mathsf{level}_{\mathrm{S}}).\, (X.H, X.\mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}}) \implies (\mathsf{DDG}(X), \mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}})$.

We now formulate our static robustness criteria by using the same notions of paths and cycles for static dependency graphs as for dynamic ones (Section 3). Given a pair $(\mathcal{I}, \mathsf{level}_{\mathrm{S}})$ and a cycle in $\pi \in \mathsf{SDG}(\mathcal{I})$ among program instances $I_0, I_1, \ldots, I_n$ (where $I_0 = I_n$), we

say that an rw edge $I_i \xrightarrow{\mathsf{rw},x} I_{i+1} \in \pi$ is ***critical in*** π, if for all I_l, I_m in π such that $l \neq m$ and for all $t, t' \in D^\star$ such that $I_l \xrightarrow{t} {}^\star\, I_i \in \pi$ and that $I_{i+1} \xrightarrow{t'} {}^\star\, I_m \in \pi$, we have $\neg(I_l \xleftrightarrow{\mathsf{ww},\_} I_m)$. For example, the graph in Figure 6 contains the following cycle π, in which the left-most rw edge is critical, while the right-most rw edge is not critical:



(4)

▶ **Definition 11.** Given a pair $(\mathcal{I}, \mathsf{level_S})$, an edge $I_i \xrightarrow{\lambda} I_{i+1} \in \mathsf{SDG}(\mathcal{I})$ is ***unprotected*** if either $\mathsf{level_S}(I_i) \neq \mathsf{SER}$ or $\mathsf{level_S}(I_{i+1}) \neq \mathsf{SER}$. A cycle $\pi \in \mathsf{SDG}(\mathcal{I})$ among program instances $I_0, I_1, \ldots, I_n$ (where $I_0 = I_n$) is:

CC-**critical,** if π contains an unprotected edge $I_i \xrightarrow{\mathsf{rw},\_} I_{i+1}$ and an unprotected edge $I_j \xrightarrow{\lambda} I_{j+1}$ with $i \neq j$ and $\lambda \in \{(\mathsf{ww},\_), (\mathsf{rw},\_)\}$;

PC-**critical,** if π contains an unprotected edge $I_i \xrightarrow{\mathsf{rw},\_} I_{i+1}$ and at least two adjacent unprotected edges with labels in $\{(\mathsf{ww},\_), (\mathsf{rw},\_)\}$. ;

PSI-**critical,** if:

1. π contains at least two unprotected critical rw edges; and
2. for every $I_i \xrightarrow{\mathsf{rw},x} I_{i+1}$, $I_j \xrightarrow{\mathsf{rw},y} I_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$;

SI-**critical,** if:

1. π contains at least two adjacent unprotected critical rw edges; and
2. for every $I_i \xrightarrow{\mathsf{rw},x} I_{i+1}$, $I_j \xrightarrow{\mathsf{rw},y} I_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$. ◀

Note that, unlike a critical cycle in a dynamic dependency graph (Definition 4), a critical cycle in a static graph does not have to be simple. The following lemma states that $\lhd$ preserves critical cycles.

▶ **Lemma 12.** *For every wm,* (G, level) *and* $(F, \mathsf{level_S})$ *such that* $(G, \mathsf{level}) \lhd (F, \mathsf{level_S})$*, if* G *contains a wm-critical cycle, then* F *contains a wm-critical cycle.*

Lemmas 10 and 12 (which are proven in [7]) and Corollary 6 establish our static criteria.

▶ **Theorem 13.** *For every* (H, level)*,* $(\mathcal{I}, \mathsf{level_S})$ *and wm, if* $\mathsf{SDG}(\mathcal{I})$ *contains no wm-critical cycles and* $(H, \mathsf{level}) \in \mathsf{hist}(wm)$*, then whenever* $(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level_S})$*, we have* $H \in \mathsf{hist}(\mathsf{SER})$*.*

For example, let $\mathsf{level_S} = (\lambda I.\, \bot)$ and consider $\mathcal{I}$ defined by (3). The corresponding static dependency graph in Figure 6 contains PSI-critical cycles, one of which is obtained by following twice the loop $\texttt{RegUser}(\mathit{Alice}) \xrightarrow{\mathsf{rw},\,\mathrm{USERS}(\_).\mathrm{name}} \texttt{RegUser}(\mathit{Alice})$. Indeed, as we explained in Section 2, the annotated instance $(\mathcal{I}, \mathsf{level_S})$ is not robust against PSI, because it can produce the write skew anomaly in Figure 2(d). Now let $\mathsf{level'_S}(\texttt{RegUser}, \mathit{Alice}) = \mathsf{SER}$ and $\mathsf{level'_S}(\_) = \bot$ otherwise. Figure 6 contains the static dependency graph corresponding to the annotated instance $(\mathcal{I}, \mathsf{level'_S})$. This graph does not contain PSI-critical cycles. To see why, observe that in the graph there are only two kinds of cycles: the ones due to the self-loop on the node $\texttt{RegUser}(\mathit{Alice})$, and the ones that connect nodes in $\{\texttt{StoreBid}(\mathit{iId}_1, 7), \texttt{StoreBid}(\mathit{iId}_1, 10), \texttt{ViewItem}(\mathit{iId}_1)\}$. The cycles of the first kind contain only protected rw edges thanks to $\mathsf{level'_S}$, while the cycles of the second kind contain at most one critical rw edge, as sketched in (4) above. It follows that no cycle is PSI-critical, and thus by executing only the RegUser transaction on serializability, we make $\mathcal{I}$ robust. However, the graph contains a CC-critical cycle, namely the one shown in (4) above. It is CC-critical for its two rw edges are unprotected. As we explained in Section 2, under CC $\mathcal{I}$ may produce the lost update anomaly in Figure 2(b), and it is unsafe to run $\mathcal{I}$ over a CC database.

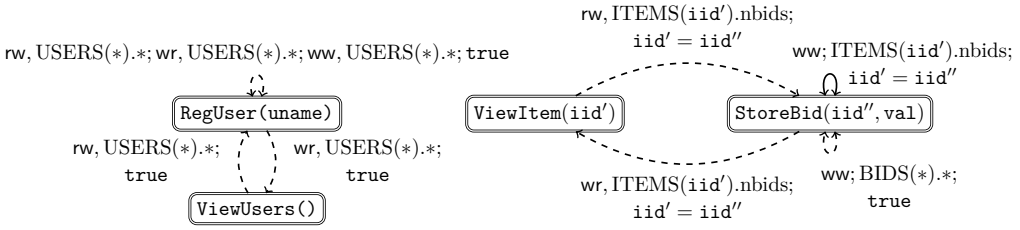
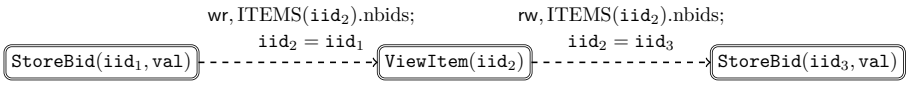


Figure 7 The summary dependency graph $\mathsf{SDG}(\mathcal{A})$, where $\mathcal{A}$ contains all the transactional programs in Figure 1.

Analysing Whole Applications. The static criteria in Theorem 13 allow a programmer to analyse the robustness of a given application instance. Analysing an application completely using the theorem requires considering an infinite number of its instances, a task best done by an automatic static analysis tool. We now sketch how ideas from abstract interpretation [13, 25] can be used to finitely represent and analyse the set of all instances of an application. In the future, this can pave the way to automating our robustness criteria in static analysis tools. Due to space constraints, we only present the concepts by an example.

We associate an application $\mathcal{A}$ with a ***summary dependency graph*** $\mathsf{SDG}(\mathcal{A})$ that summarises the static graphs of all the instances of $\mathcal{A}$. In Figure 7 we show the summary dependency graph $\mathsf{SDG}(\mathcal{A})$ for the application in Figure 1. Every program in $\mathcal{A}$ yields a ***summary node*** in the graph $\mathsf{SDG}(\mathcal{A})$, representing all instances of the program. Every edge in $\mathsf{SDG}(\mathcal{A})$ is a ***summary edge***, summarising the possible dependencies between the corresponding programs. It is annotated by a constraint relating the actual parameters of the incident programs between which the dependency exists. For example, the summary edge from `ViewItem` to `StoreBid` in Figure 7 means that for every instance $\mathcal{I}$ of $\mathcal{A}$ we have $\texttt{StoreBid}(iId_1, \_) \xrightarrow{\mathsf{rw}, x} \texttt{ViewItem}(iId_2)$ in $\mathsf{SDG}(\mathcal{I})$ iff $x \in \{\text{BIDS}(*).\text{iId}, \text{BIDS}(*).\text{val}, \text{ITEMS}(iId_1).\text{nbids}\}$ and $iId_1 = iId_2$. Similarly, the ww edge incident to `StoreBid` means that we have $\texttt{StoreBid}(iId_1, \_) \xleftrightarrow{\mathsf{ww}, x} \texttt{StoreBid}(iId_2, \_)$ in $\mathsf{SDG}(\mathcal{I})$ iff $x = \text{ITEMS}(iId_1).\text{nbids}$ and $iId_1 = iId_2$.

Definition 11 carries over to summary graphs of applications by taking into account the constraints on summary edges when checking whether a given rw edge is critical in a given cycle π, and whether the objects that appear on the rw edges of π are different. For example, consider the following cycle in the graph in Figure 7:



We consider the rw edge on this cycle not critical. This is because the constraints on the edges in the cycle imply $\texttt{iid}_1 = \texttt{iid}_3$, which satisfies the constraint on the must ww edge between `StoreBid` programs in Figure 7. For any annotated instance $(\mathcal{I}, \mathsf{level}_\mathrm{S})$ of the application $\mathcal{A}$ in Figure 1, using the adjusted Definition 11 we can check that, if $\mathsf{level}_\mathrm{S}$ maps the instances of `RegUser` and `ViewUser` in $\mathcal{I}$ to SER, then $(\mathcal{I}, \mathsf{level}_\mathrm{S})$ is robust against PSI[1].

[1] Marking `ViewUser` as SER is actually unnecessary to make this application robust under PSI, because the graph $\mathsf{SDG}(\mathcal{A})$ contains an edge $\texttt{ViewUser}() \xrightarrow{\mathsf{rw}, \text{USERS}(*).*} \texttt{RegUser}(\texttt{uname})$ which does not exist in the dependency graph of any execution of $\mathcal{A}$. This can be addressed by a more precise static analysis.

5 Related Work

In the setting of databases, application robustness was first investigated by Fekete et al. [16], who proposed a criterion for robustness against snapshot isolation (SI) [6]. Fekete then extended the criterion to SI databases allowing the programmer to request serializability for certain transactions [15], a mechanism that we also consider. Our criterion is formulated in a way similar to that of Fekete et al., using dependency graphs [2]. However, in contrast to their work, we consider more subtle models of parallel snapshot isolation, prefix consistency and causal consistency, which allow more anomalies than SI. The method we use is also different from that of Fekete et al. They consider an operational specification of SI [6], which makes the proof of the robustness criterion highly involved. In contrast, we benefit from using declarative specifications that achieve conciseness by exploiting atomic visibility of transactions [12]. This allows us to come up with robustness criteria more systematically.

Robustness has also been investigated for applications running on weak shared-memory models of common multiprocessors and programming languages (e.g., [9]). However, this line of work has not considered applications using transactions. Transactions complicate the consistency model semantics, which makes establishing robustness criteria more challenging.

Serializability of transactions in an application simplifies establishing its correctness properties, but is not necessary for this. Thus, an alternative approach to establishing application correctness is to prove its desired properties directly, without requiring the transactions to produce only serializable behaviours. Corresponding methods have been proposed for ANSI SQL isolation levels and SI by Lu et al. [23], and for PSI and some of other recent models by Gotsman et al. [18]. Such methods are complementary to ours: the conditions they require can be satisfied by more applications, but are more difficult to check than robustness.

6 Conclusion

In this paper we have made the first steps towards understanding the impact of recently-proposed transactional consistency models for large-scale databases on the correctness properties of applications using them. To this end, we have proposed criteria for checking when an application using a weak consistency model exhibits only strongly consistent behaviours. This enables programmers to check that application correctness will be preserved for a particular choice of a consistency model or transactions to be executed under serializability.

The robustness result of Fekete et al. for SI has previously given rise to automatic tools for statically detecting anomalies in applications [19]. Our work could form a basis for similar advances in databases providing weaker consistency models. Our dynamic robustness criteria are also of an independent interest: apart from serving as a basis for static analysis, such criteria can be used to optimise run-time monitoring algorithms [24, 28].

In establishing our robustness criteria, we have followed a systematic approach that exploits axiomatic specifications [12]: using the axioms of a consistency model, we have characterised the cycles allowed in dependency graphs of executions on the model, and exploited the characterisations to provide sound static analysis techniques. We hope that this method will be applicable to other consistency models being proposed for large-scale databases.

References

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

2 Atul Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. Ph.D., MIT, Cambridge, MA, USA, March 1999.

3 M. S. Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, 2013.

4 P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, Joseph M. Hellerstein, and I. Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.

5 P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Scalable atomic visibility with RAMP transactions. In *SIGMOD*, 2014.

6 H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.

7 G. Bernardi and A. Gotsman. Robustness against consistency models with atomic visibility (extended version). Available from http://software.imdea.org/∼gotsman/.

8 P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

9 A. Bouajjani, E. Derevenetc, and Roland M. Checking and enforcing robustness against TSO. In *ESOP*, 2013.

10 S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *ESOP*, 2012.

11 S. Burckhardt, S. Leijen, J. Protzenko, and M. Fähndrich. Global sequence protocol: A robust abstraction for replicated shared state. In *ECOOP*, 2015.

12 A. Cerone, G. Bernardi, and A. Gotsman. A framework for transactional consistency models with atomic visibility. In *CONCUR*, 2015.

13 P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

14 G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP*, 2007.

15 A. Fekete. Allocating isolation levels to transactions. In *PODS*, 2005.

16 A. Fekete, D. Liarokapis, D. J. O'Neil, P.E O'Neil, and D. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2), 2005.

17 S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 2002.

18 A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro. 'Cause I'm strong enough: reasoning about consistency choices in distributed systems. In *POPL*, 2016.

19 S. Jorwekar, A. Fekete, K. Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *VLDB*, 2007.

20 A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2), 2010.

21 K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, 1988.

22 W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *SOSP*, 2011.

23 S. Lu, A. J. Bernstein, and P. M. Lewis. Correct execution of transactions at different isolation levels. *IEEE Trans. Knowl. Data Eng.*, 16(9), 2004.

24 Dan R. K. Ports and Kevin Grittner. Serializable snapshot isolation in PostgreSQL. *Proc. VLDB Endow.*, 5(12), August 2012.

25 M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.

26 Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.

27 D. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, and M. K. Aguilera. Transactions with consistency choices on geo-replicated cloud storage. Technical Report MSR-TR-2013-82, Microsoft Research, 2013.

28 Kamal Zellag and Bettina Kemme. Consistency anomalies in multi-tier architectures: Automatic detection and prevention. *The VLDB Journal*, 23(1), 2014.

Optimal Assumptions for Synthesis

Romain Brenguier*

University of Oxford, UK
rbrengui@cs.ox.ac.uk

Abstract

Controller synthesis is the automatic construction a correct system from its specification. This often requires assumptions about the behaviour of the environment. It is difficult for the designer to identify the assumptions that ensures the existence of a correct controller, and doing so manually can lead to assumptions that are stronger than necessary. As a consequence the generated controllers are suboptimal in terms of robustness. In this work, given a specification, we identify the weakest assumptions that ensure the existence of a controller. We also consider two important classes of assumptions: the ones that can be ensured by the environment and assumptions that restricts only inputs of the systems. We show that optimal assumptions correspond to strongly winning strategies, admissible strategies and remorse-free strategies depending on the classes. Using these correspondences, we then propose an algorithm for computing optimal assumptions that can be ensured by the environment.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Controller synthesis, Parity games, Admissible strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.8

1 Introduction

The goal of synthesis is the implementation of a correct reactive system from its specifications. A specification is given by a ω-regular language over input and output signals of the desired system. It is realisable if we can guarantee that the sequence of inputs and outputs belong to the language. For regular languages this can be done using finite memory and thus implemented using Moore machines. Several tools have been developed to solve this problem (see for example: [3, 14]).

In general, the realisation of a specification requires some assumptions about the environment. In this work, we look for the weakest assumption that makes it realisable. For us, an assumption is weaker if it allows more behaviours. We therefore use language inclusion to compare assumptions, which means there may be incomparable and there is not a unique weakest in general. Apart from looking for just an assumption, we also consider two important classes of assumptions: ensurable and input assumptions. *Ensurable assumptions* can be ensured by the environment; in other term they cannot be falsified by a strategy of the controller. These assumptions are natural to consider when faced with a reactive environment. On the other hand, *input assumptions* are independent of the sequence of output that is produced. They are better suited to ensurable assumptions when the behaviour of the environment does not depend on the outputs of our system.

Synthesis is in general achieved by the computation of winning strategies in a game. For instance, if the specification is given by a parity automaton, we can see it has a game where the controller chooses output symbols and the adversary controls input symbols. Winning

* Work supported by EPSRC grant EP/M023656/1.

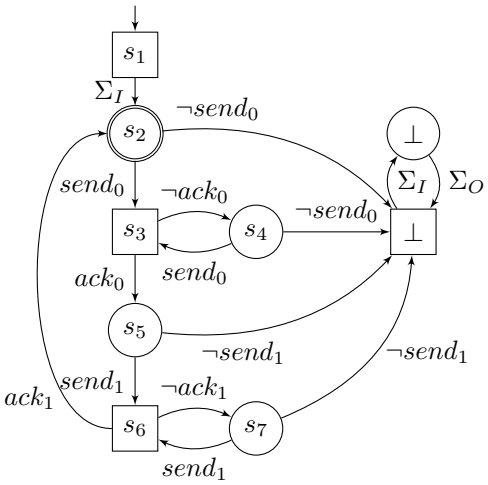


Figure 1 A Büchi automaton for the specification $\Sigma_I \cdot (send_0 \cdot (\neg ack_0 \cdot send_0)^* \cdot ack_0 \cdot send_1 \cdot (\neg ack_1 \cdot send_1)^* \cdot ack_1)^\omega$. Accepting states (with colour 0) are double lined. Square states mean that the next signal is an input, while circles mean it will be an output.

strategies in this game correspond to correct implementation of the system, and their existence implies realisability. When winning strategies do not exist, different classes have been introduced to characterise strategies that make their best effort to win. In particular, *strongly winning strategy* [10] play a winning strategy as soon as the current history (sequence of signals seen so far) makes it possible. *Admissible strategies* [1] are not *dominated* by other ones, in the sense that no strategy performs better than them against all adversary strategies. *Remorse-free strategies* [8] are such that no other strategy performs better than them against all *words* played by the adversary. We draw a link between classes of assumptions and these classes of strategies.

Example. As an example, assume we want to design a sender on a network where packets can be lost or corrupted, and our goal is to obtain a protocol similar to the classical alternating bit protocol. The outputs of the sender are actions $send_0$ and $send_1$, and the environment controls ack_0, ack_1 corresponding to acknowledgement of good reception of the packet. The specification is given by the ω-regular expression: $\Sigma_I \cdot (send_0 \cdot (\neg ack_0 \cdot send_0)^* \cdot ack_0 \cdot send_1 \cdot (\neg ack_1 \cdot send_1)^* \cdot ack_1)^\omega$. Intuitively, we have to send message with bit control 0 until receiving the corresponding acknowledgement, then do the same thing with the next message with bit control 1 and repeat this forever.

Although the implementation of the protocol seems straightforward, classical realisability fails here since if all packets are lost after some point the specification will not be satisfied, hence there is no winning strategy. To ensure realisability we have to make the assumption that a packet that is repeatedly sent will eventually be acknowledged. An admissible strategy for this specification can be implemented by a Moore machine which has the same structure as the automaton in Figure 1 with output function G such that $G(s_2) = G(s_4) = send_0$ and $G(s_5) = G(s_7) = send_1$. This implementation is natural for the given specification and corresponds indeed to the alternating bit protocol. The assumption corresponding to this strategy is the language recognised by the same automaton where we add $\bot$-states to the set of accepting states. As we will see in Theorem 23, it is an optimal ensurable assumption for the specification.

Table 1 Link between classes of assumptions and strategies.

Class of assumption:	Optimal achieved by:	
General ($\mathcal{A}$)	Strongly winning strategies	Theorem 12
Ensurable ($\mathcal{E}$)	Admissible strategies	Theorem 23
Input ($\mathcal{I}$)	Remorsefree strategies	Theorem 38

Scenarios. Specifications disallow behaviours that are not desirable. Dually, we may want to specify execution scenarios that should be possible in the synthesised system. We ask then for a system whose outcomes are all within the specifications and contains all the given scenarios. Scenarios are also a means for the user to provide feedback when the assumptions and strategies suggested by our algorithm are not the expected ones.

Generalisation. Sometimes, we already know a sufficient assumption but we want to synthesise a system which is as robust as possible by generalising this assumption. For instance, for the alternating bit protocol we could suggest as an initial assumption that two successive packets cannot be lost. With this assumption, we would offer no guarantee when than more two packets in a row are lost. By generalising the assumption, we ensure that the strategy synthesised works well under the assumption and for as many input sequences as possible. For instance, the alternating bit protocol works if an infinite number of packets are not lost.

Contribution. In this article we establish correspondences between class of assumptions and classical classes of strategies, which are summarised in Table 1.

We also show existence of optimal assumptions in most cases and give algorithms to compute optimal assumptions. In particular, we show the following properties. The existence of sufficient input assumptions compatible with a scenario is always true (Theorem 6). It is also true for safety assumptions if the scenario is itself a safety language (Theorem 7). There may exist an infinite number of optimal and ensurable-optimal assumptions (Theorem 14) and of input-optimal assumptions (Theorem 34). We can compute an optimal ensurable assumption in exponential time for parity specification and in polynomial time if we have an oracle to solve parity games (Theorem 27 and Theorem 30). There is an exponential algorithm that given a sufficient assumption, generalises it to an ensurable-optimal assumption (Theorem 33).

Comparison on previous works on assumptions for synthesis. In [7], the study is focused on safety conditions defined by forbidding edges of the automaton defining specification L. This approach depends on the choice of the automaton representing L, while ours does not. In the setting of [7], comparison between assumptions is based on the number of edges, while we compare them based on language inclusion which we find more relevant. Consider the example of Figure 2 taken from [7]. There is no winning strategy from s_1. According to [7], there are 2 minimal sufficient assumptions which are to remove either (s_3, s_6) or (s_5, s_7). However if we remove the edge from s_3 to s_6, state s_5 is no longer reachable which means that the first assumption is in fact stronger than the second one from the point of view of language inclusion. Moreover, we show that even for this restricted class of safety assumption, the claim that there is a unique non-restrictive optimal assumption [7, Theorem 5] does not hold for our notion of ensurable assumption. Consider the example of Figure 3: removing (s_3, s_6) or (s_4, s_6) is sufficient for $L = \Sigma_I \cdot (o_1 \cdot i_1 + o_2 \cdot i_2) \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$, and both these assumptions are ensurable-optimal.

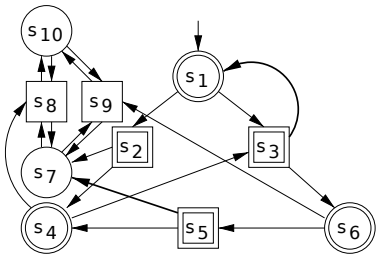


Figure 2 A game from [7].

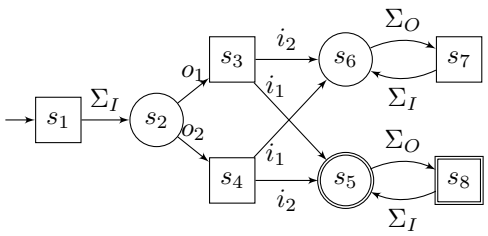


Figure 3 Automaton for specification $\Sigma_I \cdot (o_1 \cdot i_1 + o_2 \cdot i_2) \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$.

Other related works. Our goal is close to the work on assume-guarantee synthesis [6]. However assume-guarantee synthesis relies on equilibria concepts inspired by Nash equilibria. As such they assume rationality of the environment of the system and that its objective is known. By contrast, here we do not assume a rational environment and we look for the minimal assumptions about it. Closer to our work is [8] which relies on a notion of dominant strategy to obtain the weakest input assumption for which a component of the system can be implemented. A problem with this approach is that dominant strategies do not always exist. Here we characterise all minimal assumptions, and we look both at input assumptions and ensurable assumptions, which are more relevant in the context of a reactive environment.

2 Preliminaries

Given a finite alphabet Σ and an infinite word $w \in \Sigma^\omega$, we use w_i to denote the i-th symbol of w, and $w_{\leq i} = w_1 \cdots w_i$ the finite prefix of w of length i. We write $|w_{\leq i}| = i$ its length. A reactive system reads *input signals* in a finite alphabet Σ_I and produces *output signals* in a finite alphabet Σ_O. We fix these alphabets for the remainder of this paper. A *specification* of a reactive system is an ω-language $L \subseteq (\Sigma_I \cdot \Sigma_O)^\omega$. A *program* or *strategy* is a mapping $\sigma_\exists \colon (\Sigma_I \cdot \Sigma_O)^* \cdot \Sigma_I \to \Sigma_O$. An *outcome* of such a strategy $\sigma_\exists$ is a word w such that for all $i \in \mathbb{N}$, $w_{2 \cdot i+2} = \sigma_\exists(w_{\leq 2 \cdot i+1})$. We write $\mathsf{Out}(\sigma_\exists)$ for the set of outcomes of $\sigma_\exists$.

Given a specification L, the realizability problem [15] asks whether there exists a strategy $\sigma_\exists$ such that $\mathsf{Out}(\sigma_\exists) \subseteq L$. Such a strategy is said *winning* for L. The process of constructing such a strategy is called *synthesis*.

Parity automata. We assume that specifications are given by deterministic parity automata, which can recognise any ω-regular languages [13]. A *parity automaton* is given by $\langle S, s_0, \Delta, \chi \rangle$, where S is a finite set of states, $s_0 \in S$ is the initial state, $\Delta \in S \times (\Sigma_I \cup \Sigma_O) \times S$ is the transition relation, and $\chi \colon S \to \mathbb{N}$ is a colouring function. A path $\rho \in S^\omega$ is accepting if the smallest colour seen infinitely often is even (i.e. if $\min\{c \mid \forall i \in \mathbb{N}.\ \exists j \geq i.\ \chi(w_j) =$

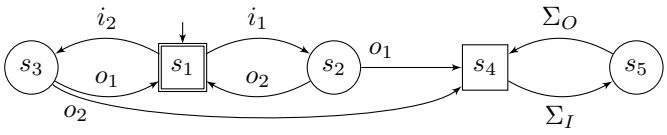


Figure 4 A Büchi automaton corresponding to specification $(i_1 \cdot o_2 + i_2 \cdot o_1)^\omega$.

$c\} \in 2 \cdot \mathbb{N}$). A word w is accepted if there is an accepting path whose labelling is w. A *Büchi automaton* is a parity automaton for which $\chi(S) \subseteq \{0,1\}$. A *safety automaton* is a Büchi automaton where states of colour 1 are absorbing. The language recognised by an automaton is the set of words it accepts. Specification can also be given in temporal logics such as LTL before being translated to an automaton representation. In some examples, we will use LTL formulas with the syntax $\mathtt{X}\phi$ meaning ϕ holds in the next state, $\phi_1 \mathtt{U} \phi_2$ meaning ϕ_1 holds until ϕ_2 holds (and ϕ_2 must hold at some point), $\mathsf{F}\phi := \mathsf{true}\ \mathtt{U}\ \phi$ and $\mathtt{G}\phi := \neg\mathsf{F}(\neg\phi)$.

Strategies. The realizability problem is best seen as a game between two players [12]. The environment chooses the input signals and the controller the output signals. We therefore also define the concept of *environment-strategy* which is a mapping $\sigma_\forall \colon (\Sigma_I \cdot \Sigma_O)^* \to \Sigma_I$. Given an environment-strategy $\sigma_\forall$, we write $\mathsf{Out}(\sigma_\forall)$ the set of words w such that for all $i \in \mathbb{N}$, $w_{2\cdot i+1} = \sigma_\forall(w_{\leq 2\cdot i})$. Given an input word $u \in {\Sigma_I}^\omega$, we write $\mathsf{Out}(\sigma_\exists, u)$ the unique outcome such that for all $i \in \mathbb{N}$, $w_{2\cdot i+1} = u_{i+1}$ and $w_{2\cdot i+2} = \sigma_\exists(w_{\leq 2\cdot i+1})$. We also write $\mathsf{Out}(\sigma_\exists, \sigma_\forall) = \mathsf{Out}(\sigma_\exists) \cap \mathsf{Out}(\sigma_\forall)$, note that it contains only one outcome. A finite prefix of an outcome is called a *history*. Given a history h, we write $\mathsf{Out}_h(\sigma_\exists)$ a word w such that for all $i \leq |h|$, $w_i = h_i$ and for all i such that $2 \cdot i + 2 > |h|$, $w_{2\cdot i+2} = \sigma_\exists(w_{\leq 2\cdot i+1})$. We write π_I and π_O the *samplings* over input and output signals respectively, that is $\pi_I \colon (\Sigma_I \cdot \Sigma_O)^\omega \to {\Sigma_I}^\omega$ is such that $\pi_I(w)_i = w_{2\cdot i-1}$ and $\pi_O \colon (\Sigma_I \cdot \Sigma_O)^\omega \to {\Sigma_O}^\omega$ is such that $\pi_O(w)_i = w_{2\cdot i}$.

Moore machine. Finite memory strategies are implemented as Moore machines. Note that the machines as we will define them read both inputs and outputs. In many application, reading outputs is unnecessary since the strategy can be left undefined on incompatible histories. However the definition we provide is coherent with our definition of strategies and makes it easier to combine strategies which may not be compatible with the same histories. A *Moore machine* is given by $\langle S_I, S_O, s_0, \delta, G\rangle$ where $S = S_I \cup S_O$ is a finite set of states, S_I is a set of input states and S_O of output states, $s_0 \in S_I$ is the initial state, $\delta \colon S \times (\Sigma_I \cup \Sigma_O) \to S$ is the transition function, and $G \colon S_O \to \Sigma_O$ is an output function. A Moore machine implements a strategy $\sigma_\exists$ where for all history $h \in (\Sigma_I \cdot \Sigma_O)^* \cdot \Sigma_I$, $\sigma_\exists(h) = G(s_{|h|})$ where for all $0 \leq i < |h|$, $s_{i+1} = \delta(s_i, h_{i+1})$.

2.1 Assumptions

An assumption $A \subseteq (\Sigma_I \cdot \Sigma_O)^\omega$ is *sufficient* for specification L if there is a strategy $\sigma_\exists$ of the controller such that any outcome either satisfies L or is not in A, i.e. $\mathsf{Out}(\sigma_\exists) \cap A \subseteq L$. In this case we also say that A is *sufficient for* $\sigma_\exists$. We look for assumptions that are the least restrictive. We say that assumption A is *less restrictive* than B if $B \subseteq A$. We say it is *strictly less restrictive* if $B \subset A$ (i.e. $B \subseteq A$ and $A \neq B$). We consider the following classes:

- An *assumption* is an ω-regular language $A \subseteq (\Sigma_I \cdot \Sigma_O)^\omega$. We write the class of all assumptions $\mathcal{A}$.

- An *input-assumption* is an assumption which concerns only inputs and does not restrict outputs. We write this class $\mathcal{I} = \{A \in \mathcal{A} \mid \forall w, w' \in (\Sigma_I \cdot \Sigma_O)^\omega.\ \pi_I(w) = \pi_I(w') \wedge w \in A \Rightarrow w' \in A\}$.
- An *ensurable assumption* is an assumption for which the environment has a winning strategy, i.e. for each $w \in (\Sigma_I \cdot \Sigma_O)^*$, if $w \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap A \neq \varnothing$ then there exists an environment strategy $\sigma_\forall$ such that $w \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap \mathsf{Out}(\sigma_\forall) \neq \varnothing$ and $\mathsf{Out}(\sigma_\forall) \subseteq A$. We write this class $\mathcal{E}$. The fact that the environment can ensure the assumption is often a requirement in synthesis (see for instance [2]).
- An *output-restrictive assumption* A is an assumption which it restricts the strategies of the controller, that is there is $w \in (\Sigma_I \cdot \Sigma_O)^*$ and $\sigma_\exists$ a strategy, such that: $A \cap w \cdot (\Sigma_I \cdot \Sigma_O)^\omega \neq \varnothing$ and $\mathsf{Out}(\sigma_\exists) \cap w \cdot (\Sigma_I \cdot \Sigma_O)^\omega \neq \varnothing$ and $A \cap \mathsf{Out}(\sigma_\exists) \cap w \cdot (\Sigma_I \cdot \Sigma_O)^\omega = \varnothing$. Intuitively an output-restrictive assumption A forbids strategy $\sigma_\exists$, so playing $\sigma_\exists$ would be a trivial way to satisfy $A \Rightarrow L$. From the point of view of synthesis it is better to avoid such assumptions. We write this class $\mathcal{R}$.
- A *safety assumption* is an assumption A for which every word not in A has a bad prefix [11], i.e. $A = (\Sigma_I \cdot \Sigma_O)^\omega \setminus \{w \mid \exists k.\ w_{\leq 2\cdot k} \in \mathsf{Bad}(A)\}$, where $\mathsf{Bad}(A) = \{h \in (\Sigma_I \cdot \Sigma_O)^* \mid h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap A = \varnothing\}$ is the set of bad prefixes of A. We write this class $\mathcal{S}$.

For a class $\mathcal{C}$ of assumptions, we say that assumption A is *$\mathcal{C}$-optimal* for L if A belongs to $\mathcal{C}$, is sufficient for L and there is no assumption $B \in \mathcal{C}$ that is strictly less restrictive than A and sufficient for L.

▶ Remark. Note that L is always a sufficient assumption, however it is too strong and will never be interesting for synthesis: if we assume that our specification always hold then any strategy would do. That is why we ask for assumptions that are as weak as possible.

We first note the following relationships between the classes of assumptions.

▶ **Lemma 1.** *Non-empty input assumptions are ensurable, i.e.* $(\mathcal{I} \setminus \{\varnothing\}) \subset \mathcal{E}$

▶ **Lemma 2.** *Ensurable assumptions are the non-output-restrictive ones:* $\mathcal{E} = \mathcal{A} \setminus \mathcal{R}$.

▶ **Example 3.** In all the examples of this article we will assume that the set of input signals is $\Sigma_I = \{i_1, i_2\}$ and the set output of output signals is $\Sigma_O = \{o_1, o_2\}$. The automaton for specification L given by formula $o_1 \mathsf{U} i_1$ is represented in Figure 5. This specification is not realisable, however several assumptions can be sufficient for it. Consider for instance the assumption A given by the LTL formula $\mathsf{F} o_1$. It is sufficient for L and is in fact sufficient for any specification since a strategy $\sigma_\exists$ which never plays o_1 has no outcome in A. To avoid this degenerate assumptions we focus on non-restrictive assumptions: $\mathsf{F} o_1$ is indeed output restrictive. On the other hand $\mathsf{F}(o_1) \Leftrightarrow \mathsf{F}(i_1)$ is ensurable because the environment can react to make the assumption hold, no matter the strategy $\sigma_\exists$ we chose. We can also check that it is sufficient for L: the strategy that always play o_1 is winning.

This assumption is fine in the context of a reactive environment, but if the environment behaves independently of the output of the system, o_1 should not appear in the assumption. Imagine the inputs are read from a file, then the environment cannot react to our outputs since the input word is already present on disk before we started producing outputs. Thus there is no way the assumption $\mathsf{F}(o_1) \Leftrightarrow \mathsf{F}(i_1)$ can be satisfied for all possible programs. In that case, the input-assumption $\mathsf{F} i_1$ which is independent from outputs is better suited.

2.2 Refinement using scenarios

As we will see in the next sections, in general there are an infinite number of incomparable optimal assumptions. This raises the problem of choosing one among all the possibilities.

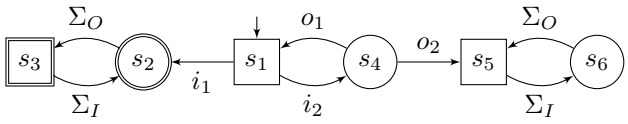


Figure 5 Büchi automaton recognising the language corresponding to LTL formula $o_1 \mathtt{U} i_1$.

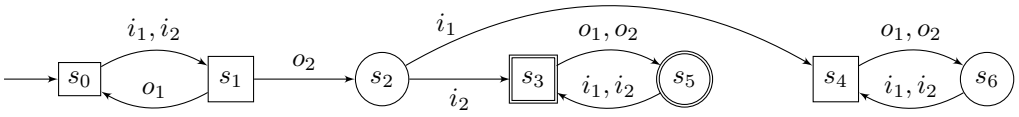


Figure 6 Büchi automaton for specification $(\neg o_2)\mathtt{U}(o_2 \wedge \mathtt{X} i_2)$.

A solution is to get some feedback from the user in the form of *scenarios*. A scenario is a behaviour that the strategy we produce should allow.

Scenario. Formally a *scenario* is given by a language $S \subseteq (\Sigma_I \cdot \Sigma_O)^\omega$. A strategy $\sigma_\exists$ is *compatible* with the set of scenarios S when $S \subseteq \mathsf{Out}(\sigma_\exists)$. Similarly S is *compatible* with specification L if $S \subseteq L$. Assumption A is *sufficient* for L and S, if there exists $\sigma_\exists$ such that $S \subseteq A \cap \mathsf{Out}(\sigma_\exists) \subseteq L$. We say that an assumption A is $\mathcal{C}$-optimal for L and S if it is sufficient for L and S and there is no $A' \in \mathcal{C}$ strictly less restrictive than A and sufficient for L and S. A scenario S is *coherent* if there is no words $w, w' \in S$ such that $w_{\leq 2\cdot i+1} = w'_{\leq 2\cdot i+1}$ and $w_{2\cdot i+2} \neq w'_{2\cdot i+2}$ for some $i \in \mathbb{N}$. If S is not coherent, then as no strategy can play both $w_{\leq 2\cdot i+2}$ and $w'_{\leq 2\cdot i+2}$, no strategy can be compatible with S. Coherence is in fact a necessary and sufficient condition for the existence of a compatible strategy.

▶ **Lemma 4.** *Scenario S is coherent if, and only if, there exists a strategy compatible with S. In particular, given a coherent scenario S and a strategy $\sigma_\exists$, the strategy $[S \to \sigma_\exists]$ is compatible with S, where:* $[S \to \sigma_\exists](h) = \begin{cases} w_{h+1} & \text{if there is } w \in S \text{ such that } h \text{ is a prefix of } w \\ \sigma_\exists(h) & \text{otherwise} \end{cases}$.

▶ **Example 5.** Consider the example in Figure 6. There are many different possible assumptions we could chose from. However if we give the scenario $\Sigma_I \cdot o_2 \cdot i_2 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ then the only ensurable-optimal assumption is $(\Sigma_I \cdot o_2 \cdot i_2 + \Sigma_I \cdot o_1 \cdot \Sigma_I) \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$. The corresponding winning strategy consists in playing o_2 for the first output.

2.3 Existence of a sufficient assumption with scenario

We show that given a scenario, there exists an input assumption which is sufficient and compatible with it. A safety assumption also exists if the scenario is itself a safety language.

▶ **Theorem 6.** *Let L be a specification and S a scenario. If S is compatible with L, then there exists an input assumption which is sufficient for L and compatible with S.*

Proof. Assume S is compatible with L and let $A = \{w \in (\Sigma_I \cdot \Sigma_O)^\omega \mid \exists w' \in S.\ \pi_I(w) = \pi_I(w')\}$. A is clearly an input assumption. We prove that for any strategy $\sigma_\exists$, A is sufficient for $[S \to \sigma_\exists]$. Let $w \in A \cap \mathsf{Out}([S \to \sigma_\exists])$. Since $w \in A$, there is $w' \in S$ such that $\pi_I(w') = \pi_I(w)$. Since $[S \to \sigma_\exists]$ is compatible with S (Lemma 4), it is compatible with w', and therefore $w' = \mathsf{Out}(\sigma_\exists, \pi_I(w)) = w$. This proves $w \in S$, and thus A sufficient for L. ◀

▶ **Theorem 7.** *Let L be a specification and S a scenario compatible with L. If S is a safety language, then there exists a safety assumption sufficient for L and compatible with S.*

▶ Remark. There are examples of scenarios which are not safety languages for which there is no sufficient safety assumption. Consider L and S both given by $\mathsf{F} i_1$. A safety assumption A compatible with S has to contain $\mathsf{F} i_1$. Assume towards a contradiction that there is $w \notin A$. Since A is a safety assumption, w has a bad prefix, that is there is i such that $w_{\leq 2\cdot i} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap A = \varnothing$. As $w_{\leq 2\cdot i} \cdot (i_1 \cdot o_1)^\omega \in S$, this is contradiction with the fact that A is compatible with S. Therefore $A = (\Sigma_I \cdot \Sigma_O)^\omega$ and this is not sufficient for L.

3 General assumptions

In this section we study general assumptions, without concern for whether there are ensurable or not. Properties established here will be useful when studying ensurable assumptions.

Necessary and sufficient assumptions. Given a specification L and a strategy $\sigma_\exists$, we say that an assumption A is *necessary* for $\sigma_\exists$ if B sufficient for $\sigma_\exists$ implies $B \subseteq A$.

▶ **Lemma 8.** *Given a strategy $\sigma_\exists$, the assumption $\mathit{GA}(\sigma_\exists) = L \cup ((\Sigma_I \cdot \Sigma_O)^\omega \setminus \mathsf{Out}(\sigma_\exists))$ is sufficient and necessary for $\sigma_\exists$.*

▶ **Corollary 9.** *If A is optimal, then there exists $\sigma_\exists$ such that $A = \mathit{GA}(\sigma_\exists)$.*

Link with strongly winning strategies. Our goal is to establish a link with the notion of strongly winning strategy. Intuitively this corresponds to the strategies that play a winning strategy whenever it is possible from the current history.

▶ **Definition 10** ([10, 4])**.** Strategy $\sigma_\exists$ is *strongly winning* when for any history h, if there exists $\sigma'_\exists$ such that $\varnothing \neq (\mathsf{Out}(\sigma'_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega) \subseteq L$, then $(\mathsf{Out}(\sigma_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega) \subseteq L$. A *subgame winning* strategy (called subgame perfect in [10]), is such that for any history h, if there exists $\sigma'_\exists$ such that $\mathsf{Out}_h(\sigma'_\exists) \subseteq L$ then $\mathsf{Out}_h(\sigma_\exists) \subseteq L$.

▶ **Lemma 11** ([10, Lemma 1])**.** *For every specification, there exists strongly winning and subgame winning strategies.*

▶ **Theorem 12.** *Let $\mathit{GA}(\sigma_\exists) = L \cup ((\Sigma_I \cdot \Sigma_O)^\omega \setminus \mathsf{Out}(\sigma_\exists))$. If strategy $\sigma_\exists$ is strongly winning for L, then $\mathit{GA}(\sigma_\exists)$ is an optimal assumption for L. Reciprocally, if A is an optimal assumption for L, then there is a strongly winning strategy $\sigma_\exists$ such that $A = \mathit{GA}(\sigma_\exists)$.*

Proof. Assume that $\sigma_\exists$ is strongly winning. First notice that by Lemma 8, $\mathsf{GA}(\sigma_\exists)$ is sufficient for $\sigma_\exists$ and thus sufficient for L. Let A be an assumption which is sufficient for L, we will prove that $\mathsf{GA}(\sigma_\exists) \not\subset A$, which shows that $\mathsf{GA}(\sigma_\exists)$ is optimal. Let $\sigma'_\exists$ be a strategy for which A is sufficient. If $A \setminus \mathsf{GA}(\sigma_\exists) = \varnothing$, then $A \subseteq \mathsf{GA}(\sigma_\exists)$ which shows the property. Otherwise there exists $w \in A \setminus \mathsf{GA}(\sigma_\exists)$. Since $w \notin \mathsf{GA}(\sigma_\exists)$ and $L \subseteq \mathsf{GA}(\sigma_\exists)$, $w \notin L$, i.e. w is losing. Since $w \notin \mathsf{GA}(\sigma_\exists)$ and $(\Sigma_I \cdot \Sigma_O)^\omega \setminus \mathsf{Out}(\sigma_\exists) \subseteq \mathsf{GA}(\sigma_\exists)$, $w \in \mathsf{Out}(\sigma_\exists)$, i.e. it is an outcome of $\sigma_\exists$. Since $A \cap \mathsf{Out}(\sigma'_\exists) \subseteq L$ and $w \in A \setminus L$, $w \notin \mathsf{Out}(\sigma'_\exists)$, i.e. it is not an outcome of $\sigma'_\exists$. Let $w_{\leq k}$ be the longest prefix of w that is compatible with $\sigma'_\exists$. Since $\sigma_\exists$ is strongly winning and w is an outcome of $\sigma_\exists$ which is losing, for all strategies $\sigma''_\exists$, either $w_{\leq k} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap \mathsf{Out}(\sigma''_\exists) = \varnothing$ or $w_{\leq k} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap \mathsf{Out}(\sigma''_\exists) \setminus L \neq \varnothing$. Since $w_{\leq k}$ is compatible with $\sigma'_\exists$, $w_{\leq k} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap \mathsf{Out}(\sigma'_\exists) \neq \varnothing$, and therefore there is an outcome w' of $\sigma'_\exists$ which is losing. Since A is sufficient for $\sigma'_\exists$, $w' \notin A$. Note that w' is not an outcome

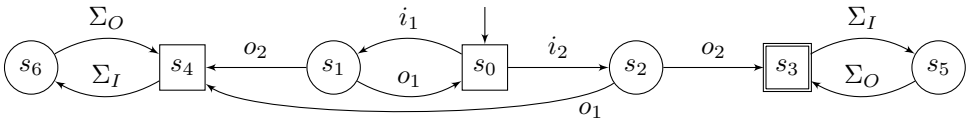


Figure 7 Büchi automaton for expression $(i_1 \cdot o_1)^* \cdot i_2 \cdot o_2 \cdot (\Sigma_I \cdot \Sigma_O)^\omega$.

of $\sigma_\exists$: $w'_{k+1} = \sigma'_\exists(w_{\leq k}) \neq \sigma_\exists(w_{\leq k})$. Hence, $w \in (\Sigma_I \cdot \Sigma_O)^\omega \setminus \mathsf{Out}(\sigma_\exists) \subseteq \mathsf{GA}(\sigma_\exists)$. Therefore $w' \in \mathsf{GA}(\sigma_\exists) \setminus A$ which proves $\mathsf{GA}(\sigma_\exists) \not\subset A$.

Let now A be an optimal assumption for L and $\sigma_\exists$ a strategy for which A is sufficient. Note that by Corollary 9, $A \subseteq \mathsf{GA}(\sigma_\exists)$. We show that $\sigma_\exists$ is strongly winning. Let h be a history such that there is $\sigma'_\exists$ such that $\varnothing \neq \mathsf{Out}(\sigma'_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq (L)$. We prove that $\mathsf{Out}(\sigma_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq L$ which shows the result. If $\mathsf{Out}(\sigma_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega = \varnothing$ the property holds and otherwise consider the strategy $\sigma_\exists\,[h \leftarrow \sigma'_\exists]$ that plays according to $\sigma_\exists$ and when h is reached shifts to $\sigma'_\exists$. Formally, given a history h':

$$\sigma_\exists\,[h \leftarrow \sigma'_\exists] = \begin{cases} \sigma'_\exists(h') & \text{if } h \text{ is a prefix of } h' \\ \sigma_\exists(h') & \text{otherwise} \end{cases}$$

Since h is compatible with $\sigma'_\exists$ and $\mathsf{Out}(\sigma'_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq L$, we also have $\sigma_\exists\,[h \leftarrow \sigma'_\exists] \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq L$. Moreover all outcomes not in $h \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ are compatible with $\sigma_\exists$. Hence $\mathsf{GA}(\sigma_\exists) \cup h \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ is sufficient for $\sigma_\exists\,[h \leftarrow \sigma'_\exists]$. By the optimality of assumption $\mathsf{GA}(\sigma_\exists)$, $\mathsf{GA}(\sigma_\exists) \not\subset \mathsf{GA}(\sigma_\exists) \cup h \cdot (\Sigma_I \cdot \Sigma_O)^\omega$. Hence $h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq \mathsf{GA}(\sigma_\exists)$. Since $\mathsf{GA}(\sigma_\exists)$ is sufficient for $\sigma_\exists$, $\mathsf{Out}(\sigma_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq L$, which shows the result. ◀

▶ **Example 13.** Consider the specification L given by expression $(i_1 \cdot o_1)^* \cdot i_2 \cdot o_2 \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ and for which a Büchi automaton is given in Figure 7. There is no winning strategy in this game, since if the input is always i_1 there is no way to satisfy the specification. However, if the current history is of the form $(i_1 \cdot o_1)^* \cdot i_2$, then controller has a winning strategy which consists in replying o_2. Strongly winning strategies must therefore present this behaviour for all $(i_1 \cdot o_1)^* \cdot i_2$ that are compatible with it. Consider strategy $\sigma_\exists$ such that if the first input is i_2, then $\sigma_\exists$ plays the winning strategy we described and otherwise always play o_2. This is a strongly winning strategy since for histories beginning with i_2 it is winning and for any other history compatible with $\sigma_\exists$ there is no winning strategy. The assumption corresponding to this strategy is $\mathsf{GA}(\sigma_\exists) = i_2 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega + i_1 \cdot (\Sigma_O \cdot \Sigma_I)^* \cdot o_1 \cdot (\Sigma_I \cdot \Sigma_O)^\omega$.

Infinity of optimal assumptions. As Figure 8 illustrates, there can be an infinite number of optimal assumptions.

▶ **Theorem 14.** *There is a specification for which there are an infinite number of optimal assumptions and an infinite number of optimal ensurable assumptions.*

Proof. Consider the game of Figure 8. In this game there are an infinite number of strongly winning strategies. They must all play o_2 in s_5 but have the choice of how long to stay in s_2. We write $\sigma^n_\exists$ the strategy that plays o_1, n times before playing o_2 (note that we could also consider strategies that depend on the choice of input in s_1, but this will not be necessary here). The sufficient hypothesis for $\sigma^n_\exists$ is $\mathsf{GA}(\sigma^n_\exists) = (\Sigma_I \cdot \Sigma_O)^\omega \setminus (\Sigma_I \cdot o_1)^n \cdot \Sigma_I \cdot o_2 \cdot i_1 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$. They are incomparable and since $\sigma^n_\exists$ are strongly winning they are all optimal. This shows that there is an infinite number of optimal assumptions. Note that these assumptions are ensurable, and therefore there also is an infinite number of ensurable-optimal assumptions. ◀

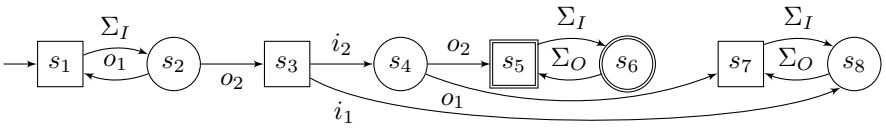


Figure 8 A parity automaton for a specification with an infinity of optimal assumptions.

Scenarios. Strategy $[S \to \sigma_\exists]$ corresponds to an optimal assumption when $\sigma_\exists$ is winning.

▶ **Lemma 15.** *If $\sigma_\exists$ is subgame winning strategy for L, then $\mathsf{GA}([S \to \sigma_\exists])$ is optimal for L and S.*

Generalisation. Assume now we are given a sufficient assumption A and want to generalise it, that is find A' optimal and such that $A \subseteq A'$. We compute $\sigma_\exists$ winning for $A \Rightarrow L$ (i.e. such that $\mathsf{Out}(\sigma_\exists) \cap A \subseteq L$ and $\sigma'_\exists$ subgame winning for L. We then define $\sigma'_\exists[A \setminus W \to \sigma_\exists]$ to be the function that maps h to $\sigma'_\exists(h)$ if h is not a prefix of a word $w \in A$ or $h \in W = \{h \mid \mathsf{Out}(\sigma'_\exists) \cap h \cdot (\Sigma_I \cdot \Sigma_O)^\omega \subseteq L\}$, and maps h to $\sigma_\exists(h)$ otherwise.

▶ **Lemma 16.** *If $\mathsf{Out}(\sigma_\exists) \cap A \subseteq L$ and $\sigma'_\exists$ is subgame winning for L, then $\mathsf{GA}(\sigma'_\exists[A \setminus W \to \sigma_\exists])$ is an optimal assumption for L and contains A.*

4 Ensurable assumptions

Consider again the solution given in Example 13. Assumption $i_2 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega + i_1 \cdot (\Sigma_O \cdot \Sigma_I)^* \cdot o_1 \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ is indeed an optimal, but it may not be what we would expect because the expression $i_1 \cdot (\Sigma_O \cdot \Sigma_I)^* \cdot o_1$ is an assumption about the controller rather than the environment. A controller which falsifies the assumption would then be considered correct. Instead of this, we would prefer an assumption which only restrict the behaviour environment. This motivates the search for nonrestrictive assumptions.

4.1 Necessary and sufficient non-restrictive assumptions

In this section, we show properties of assumptions that are not restrictive. As we have seen in Lemma 2, this coincide with ensurable assumptions for ω-regular objectives.

Given a strategy $\sigma_\exists$, the word w is *doomed for* $\sigma_\exists$ if there is an index k such that one outcome of $\sigma_\exists$ has prefix $w_{\leq k}$ and all outcome of $\sigma_\exists$ that have prefix $w_{\leq k}$ do not satisfy L. We write $\mathsf{Doomed}(\sigma_\exists)$ for the set of words that are doomed for $\sigma_\exists$ i.e. $\mathsf{Doomed}(\sigma_\exists) = \{w \mid \exists k \in 2 \cdot \mathbb{N}.\ \mathsf{Out}(\sigma_\exists) \cap w_{\leq k} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \neq \varnothing \text{ and } \mathsf{Out}(\sigma_\exists) \cap w_{\leq k} \cdot (\Sigma_I \cdot \Sigma_O)^\omega \cap L = \varnothing\}$. We consider the assumption $\mathsf{EA}(\sigma_\exists) = \mathsf{GA}(\sigma_\exists) \setminus \mathsf{Doomed}(\sigma_\exists)$.

▶ **Lemma 17.** *Let $\sigma_\exists$ be a strategy, we have the following properties:* **1.** *$\mathsf{EA}(\sigma_\exists)$ is sufficient for $\sigma_\exists$, and nonrestrictive;* **2.** *for all assumption A sufficient for $\sigma_\exists$ and not output-restrictive, we have that $A \subseteq \mathsf{EA}(\sigma_\exists)$.*

▶ **Example 18.** For the strategy $\sigma_\exists$ we defined in Example 13, the set of doomed histories is $i_1 \cdot o_2 \cdot (\Sigma_I \cdot \Sigma_O)^\omega$. Then $\mathsf{EA}(\sigma_\exists)$ is $i_2 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ which is nonrestrictive. This assumption describes better than $\mathsf{GA}(\sigma_\exists)$ the assumptions on the environment necessary to win. However it is not optimal among nonrestrictive assumptions, and we will now characterise the strategies for which $\mathsf{EA}(\sigma_\exists)$ is optimal.

4.2 Link between non-dominated strategies and optimal assumptions

We use the notion of weak dominance classical in game theory. Intuitively a strategy dominates another one if it performs at least as well against any strategy of the environment.

▶ **Definition 19** ([5]). Strategy $\sigma_\exists$ is *very weakly dominated* from history h by strategy $\sigma'_\exists$ if for all strategy $\sigma_\forall$ of the environment, $\mathsf{Out}_h(\sigma_\exists, \sigma_\forall) \in L \Rightarrow \mathsf{Out}_h(\sigma'_\exists, \sigma_\forall) \in L$. It is *weakly dominated* from h by $\sigma'_\exists$ if moreover $\sigma'_\exists$ is not very weakly dominated by $\sigma_\exists$ from h. A strategy is said *non-dominated* if no strategy weakly-dominates it from the empty history ε. A strategy is *non-subgame-dominated* if there is no strategy that weakly-dominates it from any history h. A strategy is said *dominant* if it very weakly dominates all strategies.

We can draw a link between optimal assumptions and non-dominated strategies.

▶ **Lemma 20.** *If $\mathit{EA}(\sigma_\exists) \subseteq \mathit{EA}(\sigma'_\exists)$ then $\sigma_\exists$ is very weakly dominated by $\sigma'_\exists$.*

▶ **Lemma 21.** *If $\sigma_\exists$ is very weakly dominated by $\sigma'_\exists$ then $\mathit{EA}(\sigma_\exists) \subseteq \mathit{EA}(\sigma'_\exists)$.*

▶ **Example 22.** Consider again Example 13 and a strategy $\sigma'_\exists$ which plays o_1 in s_1 and o_2 in s_2. We have that $\sigma'_\exists$ weakly dominates $\sigma_\exists$. The assumption necessary to $\sigma'_\exists$ is $\mathsf{GA}(\sigma'_\exists) = (i_1 \cdot o_1)^* \cdot (i_1 \cdot o_2 + i_2 \cdot \Sigma_O) \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ which is incomparable with $\mathsf{GA}(\sigma_\exists)$: it does not contain $(i_1 \cdot o_1)^\omega$ for instance. But $\mathsf{Doomed}(\sigma'_\exists) = \varnothing$ while $\mathsf{Doomed}(\sigma_\exists) = i_1 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega$ (which rules out $(i_1 \cdot o_1)^\omega$). So we indeed have $\mathsf{EA}(\sigma_\exists) \subset \mathsf{EA}(\sigma'_\exists)$.

▶ **Theorem 23.** *Let L be an ω-regular specification. If $\sigma_\exists$ is a non-dominated strategy for L, then $\mathit{EA}(\sigma_\exists)$ is ensurable optimal for L. Reciprocally if A is an ensurable optimal assumption for L, then there is a non-dominated strategy $\sigma_\exists$ for L such that $A = \mathit{EA}(\sigma_\exists)$.*

Proof. Let $\sigma_\exists$ be a non-dominated strategy. By Lemma 17, the assumption $\mathsf{EA}(\sigma_\exists)$ is sufficient for L and not output-restrictive. We now prove it is optimal. Let A be a nonrestrictive assumption sufficient for L and $\sigma'_\exists$ for which A is sufficient. By Lemma 8, $A \subseteq \mathsf{GA}(\sigma'_\exists)$. As $\sigma_\exists$ is not weakly dominated by $\sigma'_\exists$, either: **1.** $\sigma_\exists$ is not very weakly dominated by $\sigma'_\exists$. Then by Lemma 20 $\mathsf{EA}(\sigma_\exists) \not\subseteq \mathsf{EA}(\sigma'_\exists)$. Or **2.** $\sigma_\exists$ very weakly dominates $\sigma'_\exists$ then by Lemma 21, $\mathsf{EA}(\sigma'_\exists) \subseteq \mathsf{EA}(\sigma_\exists)$. Therefore $\mathsf{EA}(\sigma_\exists) \not\subset \mathsf{EA}(\sigma'_\exists)$ and by Lemma 17 $A \subseteq \mathsf{EA}(\sigma'_\exists)$, so $\mathsf{EA}(\sigma_\exists) \not\subset A$. This shows that $\mathsf{EA}(\sigma_\exists)$ is $\mathcal{E}$-optimal for L.

Now let $\sigma_\exists$ be a strategy such that $\mathsf{EA}(\sigma_\exists)$ is $\mathcal{E}$-optimal, we show that $\sigma_\exists$ is non-dominated. Let $\sigma'_\exists$ be a strategy which very weakly dominates $\sigma_\exists$, we prove that $\sigma_\exists$ very weakly dominates $\sigma'_\exists$. By Lemma 21, $\mathsf{EA}(\sigma_\exists) \subseteq \mathsf{EA}(\sigma'_\exists)$. Since $\mathsf{EA}(\sigma_\exists)$ is optimal, $\mathsf{EA}(\sigma_\exists) \not\subset \mathsf{EA}(\sigma'_\exists)$. Therefore $\mathsf{EA}(\sigma_\exists) = \mathsf{EA}(\sigma'_\exists)$. By Lemma 20 this implies that $\sigma'_\exists$ is very weakly dominated by $\sigma_\exists$ and shows that $\sigma_\exists$ is not weakly dominated. ◀

▶ **Corollary 24.** *If $\sigma_\exists$ is dominant, then $\mathit{EA}(\sigma_\exists)$ is the unique $\mathcal{E}$-optimal assumption.*

4.3 Computation of optimal ensurable assumptions

In parity games, deciding the existence of a winning strategy from a particular state is in the complexity class $\mathsf{NP} \cap \mathsf{coNP}$ [9]. We will show that if an algorithm for solving parity games is available, then the other operations to obtain optimal assumptions can be performed efficiently. We first construct a representation of one arbitrary non-dominated strategy. Our construction is based on the notion of memoryless strategies: given L as a parity automaton, a strategy is said *memoryless* if it only depends on the current state of the automaton, in other words it can be implemented with a Moore machine with the same structure as the given automaton.

▶ **Lemma 25.** *Given a parity automaton and a memoryless strategy $\sigma_\exists$ which ensures we are winning from each state in the winning region, we can compute in polynomial time a Moore machine implementing a memoryless non-dominated strategy $\sigma'_\exists$.*

By combining this construction with a parity automaton for L, we can build an automaton for $\mathsf{GA}(\sigma'_\exists) = L \cup ((\Sigma_I \cdot \Sigma_O)^\omega \setminus \mathsf{Out}(\sigma_\exists))$. We can then exclude doomed histories by removing transitions going to states from which there is no winning path, and obtain an automaton which recognises $\mathsf{EA}(\sigma_\exists)$.

▶ **Lemma 26.** *Given a specification as a parity automaton, and a strategy $\sigma_\exists$ as a Moore machine, we can compute in polynomial time a parity automaton recognising $\mathsf{EA}(\sigma_\exists)$.*

▶ **Theorem 27.** *Given a specification as a parity automaton, we can compute in exponential time a parity automaton of polynomial size recognising an ensurable optimal assumption. Moreover, if we have access to an oracle for computing memoryless winning strategies in parity games, our algorithm works in polynomial time.*

Proof. We first need to obtain a Moore machine for a memoryless winning strategy $\sigma_\exists$, this can be done in exponential time or constant time if we have an oracle. Then by Lemma 25, we can compute a Moore machine implementing a memoryless non-dominated strategy $\sigma'_\exists$. By Lemma 26, we can construct a parity automaton recognising $\mathsf{EA}(\sigma'_\exists)$. By Lemma 23, the language of this automaton is an ensurable optimal assumption. ◀

4.4 Scenarios

Assume now, we are given a scenario and asked for a correct system compatible with the scenario. We first characterise optimal ensurable assumptions that are needed for this.

▶ **Theorem 28.** *Let L be a specification, and S a coherent scenario compatible with L. If $\sigma_\exists$ is a non-subgame-dominated, then $\mathsf{EA}([S \to \sigma_\exists])$ is $\mathcal{E}$-optimal for L and S.*

▶ **Lemma 29.** *Given a parity automaton for a coherent scenario S and a strategy $\sigma_\exists$ we can compute in polynomial time a Moore machine for $[S \to \sigma_\exists]$.*

We can now compute an optimal ensurable assumption compatible with the scenario.

▶ **Theorem 30.** *Given a specification L and a scenario S as parity automata, we can compute in exponential time a parity automaton of polynomial size recognising an $\mathcal{E}$-optimal assumption for L and S.*

Proof. We can compute in exponential time a memoryless strategy in parity game and as seen in Lemma 25, deduce in polynomial time a memoryless non-dominated strategy $\sigma'_\exists$. From the definition of non-subgame-dominated, this strategy is in fact non-subgame-dominated. Then by Lemma 29, we can compute a Moore machine for $[S \to \sigma_\exists]$. By Theorem 28, the corresponding assumption $\mathsf{EA}([S \to \sigma_\exists])$ is ensurable optimal for L and S. By Lem 26, $\mathsf{EA}([S \to \sigma_\exists])$ can be computed in polynomial time. ◀

4.5 Generalisation

We have seen in Lemma 16 that from a winning strategy for $A \Rightarrow L$ and a strongly winning strategy for L, we could obtain a strategy $\sigma_\exists$ that has both properties. Furthermore, we can compute $\sigma'_\exists$ that is strongly non-dominated for L and define a strategy that is both non-dominated for L and winning for $A \Rightarrow L$. We define $\sigma'_\exists[A \to \sigma_\exists]$ to be the function that maps h to $\sigma_\exists(h)$ if $h \cdot \sigma_\exists(h)$ is a prefix of some $w \in A$ and maps h to $\sigma'_\exists(h)$ otherwise.

▶ **Lemma 31.** *If $\sigma_\exists$ is winning for $A \Rightarrow L$ and strongly winning for L and $\sigma'_\exists$ is strongly non-dominated for L, then $\mathsf{EA}(\sigma'_\exists[A \to \sigma_\exists])$ contains A and is ensurable-optimal for L.*

▶ **Lemma 32.** *Given strategies $\sigma'_\exists$, $\sigma_\exists$ as Moore machines and assumption A as a parity automaton, can construct in polynomial time a Moore machine for $\sigma'_\exists[A \to \sigma_\exists]$.*

▶ **Theorem 33.** *There is an exponential algorithm that given L and A sufficient for L as parity automata, computes a parity automaton whose language A' is such that $A \subseteq A'$ and A' ensurable-optimal for L.*

Proof. Assume we are given automata $\mathcal{A}_A$ for A, and $\mathcal{A}_L$ for L. We construct $\mathcal{A}_{A \Rightarrow L}$ recognising $L \cup (\Sigma_I \cdot \Sigma_O)^\omega \setminus A$, and $\sigma_\exists$ a winning strategy in $\mathcal{A}_{A \Rightarrow L}$, then compute in exponential time a memoryless strategy $\sigma'_\exists$ which is winning in $\mathcal{A}_L$ from all states from which there is a winning strategy [13]; it is in fact strongly winning. We can construct a Moore machine for $\sigma'_\exists[A \to \sigma_\exists]$ (Lemma 32), and also an automaton for $\mathsf{EA}(\sigma'_\exists[A \to \sigma_\exists])$ (Lemma 26). $\mathsf{EA}(\sigma'_\exists[A \to \sigma_\exists])$ is ensurable optimal for L and contains A (Lemma 31). ◀

5 Input-assumptions

We now focus on input assumptions. There can be an infinite number of incomparable ones that are sufficient. This can be seen in the example of Figure 6, where the specification was $(\neg o_2)\mathtt{U}(o_2 \wedge \mathtt{X} i_2)$. There, we need the assumption to tell us when exactly the first i_2 will occur. This corresponds to assumptions of the form $A_n = (\Sigma_I \cdot \Sigma_O)^\omega \setminus \{(\Sigma_I \cdot \Sigma_O)^n \cdot i_1 \cdot \Sigma_O \cdot (\Sigma_I \cdot \Sigma_O)^\omega\}$. Any such assumption will be sufficient and they are all incomparable.

▶ **Theorem 34.** *There is a specification L for which there are an infinite number of optimal input assumptions.*

We show a link between input assumptions and a class of strategies called remorsefree.

▶ **Definition 35** (Remorsefree)**.** Given a specification L, a strategy $\sigma_\exists$ is *remorsefree* if for all $\sigma'_\exists$ and $w \in \Sigma_I^\omega$, $\mathsf{Out}(\sigma'_\exists, w) \models L$ implies $\mathsf{Out}(\sigma_\exists, w) \models L$. This is the notion used in [8] for dominance. A strategy $\sigma_\exists$ is *remorsefree-admissible* if for all $\sigma'_\exists$ either $\forall w \in \Sigma_I^\omega.\ \mathsf{Out}(\sigma'_\exists, w) \models L \Rightarrow \mathsf{Out}(\sigma_\exists, w) \models L$ or $\exists w \in \Sigma_I^\omega.\ \mathsf{Out}(\sigma_\exists, w) \models L \not\Rightarrow \mathsf{Out}(\sigma'_\exists, w) \models L$.

▶ **Lemma 36.** *Given a strategy $\sigma_\exists$, if $L \neq \varnothing$ then the input-assumption $\mathsf{IA}(\sigma_\exists) = \{w \in (\Sigma_I \cdot \Sigma_O)^\omega \mid \pi_I(w) \notin \pi_I(\mathsf{Out}(\sigma_\exists) \setminus L)\}$ is sufficient for $\sigma_\exists$. Moreover if A is an input-assumption which is sufficient for $\sigma_\exists$ then $A \subseteq \mathsf{IA}(\sigma_\exists)$.*

▶ **Example 37.** Consider the game of Figure 9. In this game the only remorsefree strategy is to output o_1 at the first step. The corresponding assumption is $A = \Sigma_I \cdot \Sigma_O \cdot ((i_2 + i_3) \cdot \Sigma_O)^\omega$ while the assumption corresponding to the strategy outputting o_2 is $\Sigma_I \cdot \Sigma_O \cdot (i_3 \cdot \Sigma_O)^\omega$ which is more restrictive. The assumption A is indeed the unique optimal input-assumption.

In [8], Damm and Finkbeiner show that there is a remorsefree strategy if, and only if, there is a unique minimal assumption for L. We generalise this result using the associated notion of admissibility to characterise the minimal assumptions that are sufficient to win.

▶ **Theorem 38.** *If $\sigma_\exists$ is a remorsefree admissible strategy for L, then $\mathsf{IA}(\sigma_\exists)$ is an optimal input-assumption for L. Reciprocally if A is an optimal input-assumption for L, then there is a remorsefree admissible strategy $\sigma_\exists$, such that $A = \mathsf{IA}(\sigma_\exists)$.*

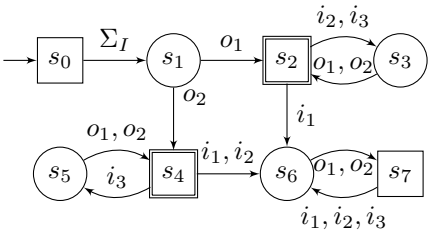


Figure 9 Büchi automaton for which the remorsefree strategy consists in outputting o_1.

Proof. Let $\sigma_\exists$ be a remorsefree-admissible strategy and A the corresponding environment assumption. Let B be such that $A \subset B$. We prove that B is not sufficient for L which will show that A is optimal. Assume towards a contradiction that B is sufficient for a strategy $\sigma'_\exists$. Since $\sigma_\exists$ is remorsefree-admissible, one of those two cases occurs: **1.** $\forall w' \in \Sigma_I^\omega$. $\mathsf{Out}(\sigma'_\exists, w') \models L \Rightarrow \mathsf{Out}(\sigma_\exists, w') \models L$. Let $w \in B \setminus A$. Since $A = \mathsf{IA}(\sigma_\exists)$, we have that $w \in \pi_I(\mathsf{Out}(\sigma_\exists) \setminus L)$. Hence $\mathsf{Out}(\sigma_\exists, w) \not\models L$, and we have that $\mathsf{Out}(\sigma'_\exists, w) \not\models L$ which shows that B is not sufficient for $\sigma'_\exists$; or **2.** $\exists w' \in \Sigma_I^\omega$. $\mathsf{Out}(\sigma_\exists, w') \models L \wedge \mathsf{Out}(\sigma'_\exists, w') \not\models L$. We have that w' belongs to A by definition, thus it belongs to B by hypothesis, and since $\mathsf{Out}(\sigma'_\exists, w') \not\models L$, B is not sufficient for $\sigma'_\exists$.

Let now A be an optimal assumption for L and $\sigma_\exists$ the corresponding strategy. We show that $\sigma_\exists$ is remorsefree-admissible. Let $\sigma'_\exists$ be another strategy. Since A is optimal, it is not strictly included in $B = \mathsf{IA}(\sigma'_\exists)$, so either $A = B$ or $A \setminus B \neq \varnothing$. **1.** If $A = B$ then we show that $\forall w \in \Sigma_I^\omega$. $\mathsf{Out}(\sigma'_\exists, w) \models L \Rightarrow \mathsf{Out}(\sigma_\exists, w) \models L$: **a.** if $w \in A = \mathsf{IA}(\sigma_\exists)$, then $\mathsf{Out}(\sigma_\exists, w) \models L$, and the implication holds. **b.** if $w \notin A = B = \mathsf{IA}(\sigma'_\exists)$, then $\mathsf{Out}(\sigma'_\exists, w) \not\models L$, and the implication holds. **2.** Otherwise $A \setminus B \neq \varnothing$ then let $w \in A \setminus B$. Since $w \in A$, $\mathsf{Out}(\sigma_\exists, w) \models L$ and since $w \notin B = \mathsf{IA}(\sigma'_\exists)$, $\mathsf{Out}(\sigma'_\exists, w) \not\models L$. Hence $\mathsf{Out}(\sigma_\exists, w) \models L \not\Rightarrow \mathsf{Out}(\sigma'_\exists, w) \models L$. ◀

References

1 Dietmar Berwanger. Admissibility in infinite games. In *STACS*, volume 4393 of *LNCS*, pages 188–199. Springer, February 2007.

2 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to handle assumptions in synthesis. In *SYNT*, pages 34–50, 2014. doi:10.4204/EPTCS.157.7.

3 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification*, pages 652–657. Springer, 2012.

4 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. In *CONCUR*, volume 42 of *LIPIcs*, pages 100–113. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.100.

5 Romain Brenguier, Jean-François Raskin, and Mathieu Sassolas. The complexity of admissibility in omega-regular games. In *CSL-LICS '14, 2014*. ACM, 2014. doi:10.1145/2603088.2603143.

6 Krishnendu Chatterjee and Thomas A Henzinger. Assume-guarantee synthesis. In *TACAS'07*, volume 4424 of *LNCS*. Springer, 2007.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In *CONCUR*, pages 147–161. Springer, 2008.

8 Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *FM 2011: Formal Methods*, pages 12–26. Springer, 2011.

9 E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377. IEEE, 1991.

10 Marco Faella. Admissible strategies in infinite games over graphs. In *MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009.

11 Rachel Faran and Orna Kupferman. Spanning the spectrum from safety to liveness. In *Automated Technology for Verification and Analysis*, pages 183–200. Springer, 2015.

12 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In *Computer Aided Verification*, pages 263–277. Springer, 2009.

13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag, New York, NY, USA, 2002.

14 Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNTCOMP 2014). *Int J Softw Tools Technol Transfer*, pages 1–24, 2016. doi:10.1007/s10009-016-0416-3.

15 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM, 1989.

Minimizing Expected Cost Under Hard Boolean Constraints, with Applications to Quantitative Synthesis*

Shaull Almagor[1], Orna Kupferman[2], and Yaron Velner[3]

1 **School of Computer Science and Engineering, The Hebrew University, Israel.**
2 **School of Computer Science and Engineering, The Hebrew University, Israel.**
3 **School of Computer Science and Engineering, The Hebrew University, Israel.**

Abstract

In Boolean synthesis, we are given an LTL specification, and the goal is to construct a transducer that realizes it against an adversarial environment. Often, a specification contains both Boolean requirements that should be satisfied against an adversarial environment, and multi-valued components that refer to the quality of the satisfaction and whose expected cost we would like to minimize with respect to a probabilistic environment.

In this work we study, for the first time, mean-payoff games in which the system aims at minimizing the expected cost against a probabilistic environment, while surely satisfying an ω-regular condition against an adversarial environment. We consider the case the ω-regular condition is given as a parity objective or by an LTL formula. We show that in general, optimal strategies need not exist, and moreover, the limit value cannot be approximated by finite-memory strategies. We thus focus on computing the limit-value, and give tight complexity bounds for synthesizing ϵ-optimal strategies for both finite-memory and infinite-memory strategies.

We show that our game naturally arises in various contexts of synthesis with Boolean and multi-valued objectives. Beyond direct applications, in synthesis with costs and rewards to certain behaviors, it allows us to compute the minimal sensing cost of ω-regular specifications – a measure of quality in which we look for a transducer that minimizes the expected number of signals that are read from the input.

1998 ACM Subject Classification F.4.3 : Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Stochastic and Quantitative Synthesis, Mean Payoff Games, Sensing.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.9

1 Introduction

Synthesis is the automated construction of a system from its specification: given a linear temporal logic (LTL) formula ψ over sets I and O of input and output signals, we synthesize a system that *realizes* ψ [11, 17]. At each moment in time, the system reads a truth assignment, generated by the environment, to the signals in I, and it generates a truth assignment to the signals in O. The system realizes ψ if all the computations that are generated by the above interaction satisfy ψ.

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410, from the Israel Science Foundation (grant no 1229/10), and from the US-Israel Binational Science Foundation (grant no 2010431).

One weakness of automated synthesis in practice is that it pays no attention to the quality of the synthesized system. Indeed, the classical setting is Boolean: a computation satisfies a specification or does not satisfy it. Accordingly, while the synthesized system is correct, there is no guarantee about its quality. In recent years, researchers have considered extensions of the classical Boolean setting to a quantitative one, which takes quality into account. Quality measures refer to the system itself, examining parameters like size or consumption of memory, sensors, voltage, bandwidth, etc., or refer to the way the system satisfies the specification. In the latter, we allow the designer to specify the quality of a behavior using quantitative specification formalisms [1, 6, 13]. For example, rather than the Boolean specification requiring all requests to be followed by a grant, a quantitative specification formalism gives a different satisfaction value to a computation in which requests are responded immediately and one in which requests are responded after long delays.[1]

Solving the synthesis problem in the Boolean setting amounts to solving a two-player zero-sum game between the system and the environment. The goal of the system is to satisfy the (Boolean) specification, and the environment is adversarial. Then, a winning strategy for the system corresponds to a transducer that realizes the specification. In the quantitative setting, the goal of the system is no longer Boolean, as every play is assigned a cost by the specification. In the classical quantitative approach, we measure the satisfaction value in the worst-case semantics. Thus, the value of a strategy for the system is the maximal cost of a play induced by this strategy, and the goal of the system is to minimize this value. Recently, there is a growing interest also in the expected cost of a play, under a probabilistic environment. The motivation behind this approach is that the quality of satisfaction is a "soft constraint", and should not be measured in a worst-case semantics. Then, the game above is replaced by a mean-payoff Markov Decision Process (MDP): a game in which each state has a cost, inducing also costs to infinite plays (essentially, the cost of an infinite play is the limit of the average cost of prefixes of increased lengths). The goal is to find a strategy that minimizes the expected cost [10, 12].

While quantitative satisfaction refines the Boolean one, often a specification contains both Boolean conditions that should be satisfied against all environments, and multi-valued components that refer to the quality of the satisfaction and whose expectation we would like to minimize with respect to a probabilistic environment. Accordingly, researchers have suggested the *beyond worst-case* approach, where a specification has both hard and soft constraints, and the goal is to realize the hard constraints, while maximizing the expected satisfaction value of the soft constraints.

In this work, we consider, for the first time, mean-payoff MDPs equipped with a parity winning condition (parity-MDPs, for short). The goal is to find a strategy that surely wins the parity game (that is, against an adversarial environment), while minimizing the expected cost of a play against a probabilistic environment. While the starting point in earlier related work is the MDP itself, possibly augmented by different objectives, our starting point depends on the application, and we view the construction of the MDP as an integral part of our contribution. We focus on two applications: synthesis with penalties to undesired scenarios and synthesis with minimal sensing.

Let us describe the two applications. We start with penalties to scenarios. Consider

[1] Note that the polarity of some quality measures is negative, as we want to minimize size, consumption, costs, etc., whereas the polarity of other measures is positive, as we want to maximize performance and satisfaction value. For simplicity, we assume that all measures are associated with costs, which we want to minimize.

an LTL specification ψ over I and O. Activating an output signal may have a cost; for example, when the activation involves a use of a resource. Taking these costs into account, the input to the synthesis problem includes, in addition to ψ, a cost function γ assigning cost to some assignments to output signals. The cost of a computation is then the mean cost of assignments in it. While the specification ψ is a hard constraint, as we only allow correct computations, minimizing the expected cost of computations with respect to γ is a soft constraint. More elaborated cost functions refer to on-going regular scenarios. Power consumption, for example, is an important consideration in modern chip design. As the chips become more complex, the cost of powering a server farm can easily outweigh the cost of the servers themselves, thus design teams go to great lengths in order to reduce power consumption in their designs. The most widely researched logical power saving techniques are *clock gating*, in which a clock is prevented from making a "tick" if it is redundant [5], and *power gating*, in which whole sections of the chip are powered off when not needed and then powered on again [14]. The goal of these techniques is to reduce power consumption and the number of changes in the values of signals, the main source of power consumption in chips. The input to the problem of synthesis with penalties to scenarios includes, in addition to ψ, a set of deterministic automata on finite words, each describing a undesired scenario and its cost. For example, it is easy to specify the scenario of "value flip" with a two-state deterministic automaton. We show how the setting can be translated into solving our parity-MDPs, thus generating systems that realize ψ with minimal expected cost.

Our primary application considers activation of sensors. The quality measure of sensing was introduced in [2, 3], as a measure for the detail with which a random input word needs to be read in order to realize the specification. In the context of synthesis, our goal is to construct a transducer that realizes the specification and minimizes the expected average number of sensors (of input signals) that are used along the interaction. Thus, the hard constraint is the LTL specification, and the soft one is the expected number of active sensors. Giving up sensing has a flavor of synthesis with incomplete information [15]: the transducer has to realize the specification no matter what the incomplete information is. Thus, as opposed to the examples above, the modeling of cost involves a careful construction of the MDP to be analyzed, and also involves an exponential blow-up, which we show to be unavoidable. In [3], the problem was solved for safety specifications. Our solution to the parity-MDP problem enables a solution for full LTL. We also study the complexity of the problem when the input is an LTL formula, rather than a deterministic automaton.

Back to parity-MDPs, we show that in general, optimal strategies need not exist. That is, there are parity-MDPs in which an infinite-state strategy can get arbitrarily close to some limit optimal value, but cannot attain it. Moreover, the limit value cannot be approximated by finite-memory strategies. Accordingly, our solution to parity-MDPs suggests two algorithms. The first, described in Section 3.1, finds the limit value of all possible strategies, which corresponds to infinite-state transducers. The second, described in Section 3.2, computes the limit value over all finite-memory strategies. The complexity of both algorithms is NP$\cap$coNP. Moreover, they are computable in polynomial time when an oracle to a two-player parity game is given. Hence, our complexity upper bounds match the trivial lower bounds that arise from the fact that every solution to a parity-MDP is also a solution to a parity game. For our applications, we show that the complexity of the synthesis problem for LTL specifications stays doubly-exponential, as in the Boolean setting, even when we minimize penalties to undesired scenarios or minimize sensing.

Related Work. The combination of worst-case synthesis with expected-cost synthesis, dubbed *beyond worst-case synthesis*, was studied in [7, 12] for models that are closely related to ours. In [7] the authors study mean-payoff MDPs, where both the hard constraints and the soft constraints are quantitative. Thus, a system needs to ensure a strict upper bound on the mean-payoff cost, while minimizing the expected cost. In [12], multidimensional mean-payoff MDPs are considered. Thus, the MDP is equipped with several mean-payoff costs, and the goal is to find a system that ensures the mean-payoff in some of the mean-payoffs is below an upper bound, while minimizing the expected mean-payoffs (or rather, approximating their Pareto-curve).

In comparison, our work is the first to consider a hard Boolean constraint (namely the parity condition). This poses both a conceptual and a technical difference. Conceptually, when quantitative synthesis is taken as a refinement of Boolean synthesis, it is typically meant as a ranking of different systems that satisfy a Boolean specification. Thus, it makes sense for the hard constraint to be Boolean as well. Technically, combining Boolean and quantitative constraints gives rise to some subtleties that do not exist in the pure-quantitative setting. Specifically, when both the hard and the soft constraints are quantitative, a strategy can intuitively "alternate" between satisfying them. Thus, if while trying to meet the soft constraint the hard constraint is violated, we can switch to a worst-case strategy until the hard constraint is satisfied, and go back to trying to minimize the soft constraint. This alternation can be done infinitely often. In the Boolean setting, however, this alternation can violate the Boolean constraint. We note that unlike classical parity games, where the parity winning condition can be translated to a richer mean-payoff objective, the parity winning condition in our parity-MDPs does not admit a similar translation.

Other works on MDPs and mean-payoff objectives tackle different aspects of quantitative analysis. In [18], a solution to the expected mean-payoff value over MDPs is presented. In [8] and [9], the authors study a combination of mean-payoff and parity objectives over MDPs and over stochastic two-player games. There, the goal of the system is to ensure with probability 1 that the parity condition holds and that the mean-payoff is below a threshold. This differs from our work in that the parity condition is not a hard constraint, as it is met only almost-surely, and in that the expected mean-payoff is not guaranteed to be minimized. As detailed in the paper, these differences make the technical challenges very different.

Due to lack of space, some of the proofs are omitted and can be found in the full version [4].

2 Parity-MDPs

A *parity Markov decision process* (Parity-MDP, for short) combines a parity game with a mean-payoff MDP. The game is played between Player 1, who models a system, and Player 2, who models the environment. The environment is adversarial with respect to the parity winning condition and is stochastic with respect to the mean-payoff objective. Formally, a parity-MDP is a tuple $\mathcal{M} = \langle S_1, S_2, s_0, A_1, A_2, \delta_1, \delta_2, \mathrm{P}, \mathit{cost}, \alpha \rangle$, with the following components. The sets S_1 and S_2 are finite set of states, for Players 1 and 2, respectively. Let $S = S_1 \cup S_2$. Then, $s_0 \in S$ is an initial state, and A_1 and A_2 are sets of actions for the players. Not all actions are available in all states: for every state $s \in S_i$, for $i \in \{1, 2\}$, we use $A_i(s)$ to denote the finite set of actions available to Player i in the state s. For $i \in \{1, 2\}$, the (partial) transition function $\delta_i : S_i \times A_i \nrightarrow S$ is such that $\delta_i(s, a)$ is defined iff $a \in A_i(s)$. Let $\delta = \delta_1 \cup \delta_2$. Note that δ_2 gets an action of Player 2 as a parameter. We distinguish between two approaches to the way an action is chosen in Player 2 states. In the

adversarial approach, it is Player 2 who chooses the action. In the stochastic approach, the choice depends on the (partial) function $\mathrm{P} : S_2 \times A_2 \nrightarrow [0,1]$, where for every state $s \in S_2$ and $a \in A_2$, we have that $\mathrm{P}(s,a) > 0$ only if $a \in A_2(s)$. Also, $\sum_{a \in A_2(s)} \mathrm{P}(s,a) = 1$. Finally, $cost : S \to \mathbb{N}$ is a cost function, and $\alpha : S \to \{0, ..., d\}$, for some $d \in \mathbb{N}$, is a parity winning condition.

The parity-MDP $\mathcal{M}$ induces a *parity game* $\mathcal{M}^{\mathtt{P}} = \langle S_1, S_2, s_0, A_1, A_2, \delta_1, \delta_2, \alpha \rangle$, obtained by omitting P and *cost*. In this game, we follow the adversarial approach to the environment. Thus, both players choose their actions. Formally, a *strategy* for Player i in $\mathcal{M}$, for $i \in \{1,2\}$ is a function $f_i : S^* \times S_i \to A_i$ such that for $s_0, \ldots, s_n$, we have $f(s_0, \ldots, s_n) \in A_i(s_n)$. Thus, a strategy suggests to Player i an available action given the history of the states traversed so far. Note that we do not consider randomized strategies, but rather deterministic ones. Our results in Section 3 show that this is sufficient, in the sense that the players cannot gain by using randomization.

Given strategies f_1 and f_2 for Players 1 and 2, the *play* induced f_1 and f_2 is is the infinite sequence of states $s_0, s_1, ...$ such that for every $j \geq 0$, if $s_j \in S_i$, for $i \in \{1,2\}$, then $s_{j+1} = \delta_i(s_j, f(s_0, ..., s_j))$. For an infinite play r, we denote by $\inf(r)$ the set of states that r visits infinitely often. The play $r = s_0, s_1, ...$ of $\mathcal{M}$ is *parity winning* if $\max\{\alpha(s) : s \in \inf(r)\}$ is even.

The parity-MDP $\mathcal{M}$ also induces an *MDP* $\mathcal{M}^{\mathtt{MDP}} = \langle S_1, S_2, s_0, A_1, A_2, \delta_1, \delta_2, \mathrm{P}, cost \rangle$, obtained by omitting α. In this game, we follow the stochastic approach to the environment and consider the distribution of plays when only a strategy for Player 1 is given. Formally, we first extend P to transitions as follows: For states $s \in S_2$ and $s' \in S$, we define $\mathrm{P}(s,s') = \sum_{a \in A(s) : \delta_2(s,a) = s'} \mathrm{P}(s,a)$. Then, a play of $\mathcal{M}$ with strategy f_1 for Player 1 is an infinite sequence of states $s_0, s_1, ...$ such that for every $j \geq 0$, if $s_j \in S_1$, then $s_{j+1} = \delta_1(s_j, f_1(s_0, ..., s_j))$, and if $s_j \in S_2$, then $\mathrm{P}(s_j, s_{j+1}) > 0$. The *cost* of a strategy f_1 is the expected average cost of a random walk in $\mathcal{M}$ in which Player 1 proceeds according to f_1. Formally, for $m \in \mathbb{N}$ and for a prefix $\tau = s_0, s_1, ...s_m$ of a play, let $I_2 = \{j : j < m \text{ and } s_j \in S_2\}$. Then, we define $\mathrm{P}_{f_1}(\tau) = \prod_{j \in I} \mathrm{P}(s_j, s_{j+1})$ and $cost_m(f_1, \tau) = \frac{1}{m+1} \sum_{j=0}^{m} cost(s_j)$. The cost of the strategy f_1 is then $cost(f_1) = \liminf_{m \to \infty} \sum_{\tau : |\tau| = m} cost_m(f_1\ , \tau) \cdot \mathrm{P}_{f_1}(\tau)$. We denote by $\inf(f_1)$ the random variable that associates $\inf(\rho)$ with a sequence of states $\rho = s_0, s_1, ...$, under the probability space induced by $\mathcal{M}$ with f_1.

A *finite memory* strategy for $\mathcal{M}$ is described by a finite set M called *memory*, an initial memory $init \in M$, a *memory update function* $next : S_1 \times M \to M$, and an *action function* $act : S_1 \times M \to A_1$ such that $act(s,m) \in A_1(s)$ for every $s \in S_1$ and $m \in M$.

A strategy is *memoryless* if it has finite memory M with $|M| = 1$. Note that a memoryless strategy depends only on the current state. Thus, we can describe a memoryless strategy by $f_1 : S_1 \to A_1$. Let $cost(\mathcal{M}) = \inf\{cost(f_1) : f_1 \text{ is a strategy for } \mathcal{M}\}$. That is, $cost(\mathcal{M})$ is the expected cost of a game played on $\mathcal{M}$ in which Player 1 uses an optimal strategy.

The following is a basic property of MDPs.

▶ **Theorem 1.** *Consider an MDP $\mathcal{M}$. Then, $cost(\mathcal{M})$ can be attained by a memoryless strategy, which can be computed in polynomial time.*

Recall that a strategy f_1 for player 1 is winning in $M^{\mathtt{P}}$ if every play of $\mathcal{M}$ with f_1 satisfies the parity condition α. Note that we require *sure* winning, in the sense that all plays must be winning, rather than winning with probability 1 (*almost-sure* winning). On the other hand, the definition of cost in $M^{\mathtt{MDP}}$ considered strategies for Player 1 and ignore the parity winning condition. We now define the *sure cost* of the parity-MDP, which does take them into account. For a strategy f_1 for Player 1, the sure cost of f_1, denoted $cost_{\text{sure}}(f_1)$, is

$cost(f_1)$, if f_1 is winning, and is ∞ otherwise. The sure cost of $\mathcal{M}$ is then $cost_{\text{sure}}(\mathcal{M}) = \inf\{cost_{\text{sure}}(f_1) : f_1 \text{ is a strategy for } \mathcal{M}\}$.

Consider a parity-MDP $\mathcal{M} = \langle S_1, S_2, s_0, A_1, A_2, \delta_1, \delta_2, \mathrm{P}, cost, \alpha\rangle$. An *end component* (EC, for short) is a set $U \subseteq S$ such that for every state $s \in U$, the following hold.

1. If $s \in S_1$, then there exists an action $a \in A_s$ such that $\delta_1(s,a) \in U$.
2. If $s \in S_2$, then for every $a \in A_2(s)$ such that $\mathrm{P}(s,a) > 0$, it holds that $\delta_2(s,a) \in U$.
3. For every $t, t' \in U$, there exist a path $t = t_0, t_1, ..., t_k = t'$ and actions $a_1, ..., a_t$ such that for every $0 \leq i < t$, it holds that $t_i \in U$, and there exists an action a such that $\delta(t_i, a) = t_{i+1}$.

Intuitively, the probabilistic player cannot force the play to leave U, and Player 1 has positive probability of reaching every state in U from every other state.

For an EC U and a state $s \in U$, we can consider the parity-MDP $\mathcal{M}|_U^s$, in which the states are U, the initial state is s, and all the components are naturally restricted to U. Since U is an EC, then this is indeed a parity-MDP. An EC U is *maximal* if for every nonempty $U' \subseteq S \setminus U$, we have that $U \cup U'$ is not an EC.

3 Solving Parity MDPs

In this section we study the problem of finding the sure cost for an MDP. Recall that for MDPs, there always exists an optimal memoryless strategy. We start by demonstrating that for the sure cost of parity-MDPs, the situation is much more complicated.

▶ **Theorem 2.** *There is a parity-MDP $\mathcal{M}$ in which Player 1 does not have an optimal strategy (in particular, not a memoryless one) for attaining the sure cost of $\mathcal{M}$. Moreover, for every $\epsilon > 0$, Player 1 may need infinite memory in order to ϵ-approximate $cost_{\text{sure}}(\mathcal{M})$.*

Proof. Consider the parity-MDP $\mathcal{M}$ appearing in Figure 1. Player 1 can decrease the cost of a play towards 1 by staying in the initial state s_0. However, in order to ensure an even parity rank, Player 1 must either play b and reach s_3 w.p. 0.5, or play c but incur cost 10. A finite memory strategy for Player 1 must eventually play c from s_0 in almost every play,[2] thus the cost of every winning finite-memory strategy is 10. On the other hand, for every $\epsilon > 0$, there exists an infinite memory strategy f that gets cost at most $1 + \epsilon$. Essentially (see Lemma 4 for a formal proof of the general case), the strategy f plays b for a long time. If s_3 is reached, it plays b for even longer, and otherwise plays c.

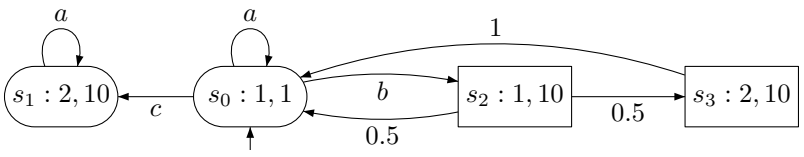


Figure 1 The Parity MDP $\mathcal{M}$. States of Player 1 are circles, these of Player 2 are squares, with outgoing edges marked by their probability. Each state is labeled by its parity rank (left) and cost (right). Player 1 has no optimal strategy and needs infinite memory for an ϵ approximation.

Finally, there is no optimal strategy for Player 1, as every strategy that plays c from s_0 eventually (i.e., as a response to some strategy of Player 2) gets cost 10 with some positive probability. However, a strategy that never plays c is not parity-winning. ◀

[2] Note that this also implies that randomized strategies could not be of help here.

Following Theorem 2, our solution to parity MDPs suggests two algorithms. The first, described in Section 3.1, finds the limit value of all possible strategies, which corresponds to infinite-state transducers. The second, described in Section 3.2, computes the limit value over all finite-memory strategies. The complexity of both algorithms is NP∩coNP. Moreover, they are computable in polynomial time when an oracle to a two-player parity game is given. Hence, our complexity upper bounds match the trivial lower bounds that arise from the fact that every solution to a parity-MDP is also a solution to a parity game.

3.1 Infinite-Memory Strategies

In this section we study the problem of finding the sure cost of a parity-MDP when infinite-memory strategies are allowed. We prove the upper bound in the following theorem. As stated above, the lower bound is trivial.

▶ **Theorem 3.** *Consider a parity-MDP $\mathcal{M}$. Then, $cost_{\text{sure}}(\mathcal{M})$ can be computed in NP∩co-NP, and is parity-games hard.*

Consider a parity-MDP $\mathcal{M} = \langle S_1, S_2, s_0, A_1, A_2, \delta_1, \delta_2, \mathrm{P}, cost, \alpha\rangle$. We first remove from $\mathcal{M}$ all states that are not sure-winning for Player 1 in $\mathcal{M}^{\mathtt{P}}$. Clearly, every strategy that attains $cost_{\text{sure}}(\mathcal{M})$ cannot visit a state that is losing in $\mathcal{M}^{\mathtt{P}}$. Thus, we henceforth assume that all states in $\mathcal{M}$ are winning for Player 1 in $\mathcal{M}^{\mathtt{P}}$. We say that an EC C of $\mathcal{M}$ is *good* (GEC, for short) if its maximal rank is even. That is, $\max_{s\in C}\{\alpha(s)\}$ is even.

The idea behind our algorithm is as follows. W.p. 1, each play in $\mathcal{M}$ eventually reaches and visits infinitely often all states of some EC. Hence, when restricting attention to plays that are winning for Player 1 in $\mathcal{M}^{\mathtt{P}}$, it must be the case that this EC is good. It follows that the sure cost of $\mathcal{M}$ is affected only by the properties of its GECs. Moreover, since the minimal expected mean-payoff value is the same in all the states of an EC, we can consider only maximal GECs and refer to the value of an EC, namely the minimal expected value that Player 1 can ensure while staying in the EC. Our algorithm constructs a new MDP (without ranks) $\mathcal{M}'$ in which the cost of a state is the value of the maximal GEC it belongs to. If a state does not belong to a GEC, then we assign it a very high cost in $\mathcal{M}'$, where the intuition is that Player 1 cannot benefit from visiting this state infinitely often. We claim that the sure cost in the parity-MDP $\mathcal{M}$ coincides with the cost of the MDP $\mathcal{M}'$.

Formally, for an EC C, let $C^{\max}$ be the set of the states of C with the maximal parity rank in C. By definition, this rank is even when C is a GEC. Note that if C and C' are GECs and $C\cap C' \neq \emptyset$, then $C\cup C'$ is also a GEC. Thus, we can restrict attention to maximal GEC. For a GEC C, there exists a memoryless strategy f^C that maximizes the probability of reaching $C^{\max}$ from every state $s \in C$ while staying in C. Moreover, since C is an EC, the probability of reaching $C^{\max}$ by playing f^C is strictly positive from every state $s \in C$. Let t be a state in C. Consider the MDP $\mathcal{M}^{\mathtt{MDP}}|^t_C$. Since C is EC, we have that $cost(\mathcal{M}^{\mathtt{MDP}}|^t_C)$ is independent of the initial state t. Thus, we can define $cost(\mathcal{M}^{\mathtt{MDP}}|_C)$ as $cost(\mathcal{M}^{\mathtt{MDP}}|^t_C)$ for some $t \in C$.

Recall that our algorithm starts by a preprocessing step that removes all states that are not sure-winning for Player 1 in $\mathcal{M}^{\mathtt{P}}$. It then finds the maximal GECs of $\mathcal{M}$ (using a polynomial-time procedure that we describe in the full version [4]), and obtain an MDP $\mathcal{M}'$ by assigning every state within a GEC C the cost $cost(\mathcal{M}^{\mathtt{MDP}}|_C)$, and assigning every state that is not inside a GEC cost $W+1$, where W is the maximal cost that appears in $\mathcal{M}$. We claim that $cost_{\text{sure}}(\mathcal{M}) = cost(\mathcal{M}')$.

Before proving the claim, note that all the steps of the algorithm except for the preprocessing step that involves a solution of parity game require polynomial time. In particular,

the strategies f^C above are computable in polynomial time by solving a reachability MDP, and, by Theorem 1, so does the final step of finding $cost(M')$.

Proving that $cost_{\text{sure}}(\mathcal{M}) = cost(\mathcal{M}')$ involves the following steps (see the full version [4] for the full proof). First, proving $cost_{\text{sure}}(\mathcal{M}) \geq cost(\mathcal{M}')$ is not hard, as a play with a winning strategy f for Player 1 in $\mathcal{M}$ reaches and stays in some GEC C w.p. 1, and within C, the best expected cost one can hope for is $cost(\mathcal{M}^{\texttt{MDP}}|_C)$, which is exactly what the strategy f attains when played in $\mathcal{M}'$.

Next, proving $cost_{\text{sure}}(\mathcal{M}) \leq cost(\mathcal{M}')$, we show how an optimal strategy f' in $\mathcal{M}'$ induces an ϵ-optimal strategy f in $\mathcal{M}$. We start with Lemma 4, which justifies the costs within a GEC.

▶ **Lemma 4.** *Consider a GEC C in $\mathcal{M}$, and $s \in C$. Let $v(s) = cost(\mathcal{M}^{\texttt{MDP}}|^s_C)$, then for every $\epsilon > 0$ there exists a strategy f of $\mathcal{M}^s$ with $cost_{\text{sure}}(f) \leq v(s) + \epsilon$.*

Intuitively, in a good EC, f minimizes the expected mean-payoff and *once in a while* it plays reachability, aiming to visit to a state with the maximal rank in the EC. Since the EC is good, this rank is even. If reachability is not obtained after N rounds, for a parameter N, then f gives up and aims at only surely satisfying the parity objective (our preprocessing step ensures that this is possible). Otherwise, after reaching the maximal rank, f switches again to minimizing mean-payoff. This process is repeated forever, increasing N in each iteration. Hence, the probability that Player 1 eventually gives up can be bounded from above by an arbitrarily small $\epsilon > 0$. Accordingly, Player 1 can achieve a value that is arbitrarily close to $cost(\mathcal{M}^{\texttt{MDP}}|_C)$.

Finally, we construct the ϵ-optimal strategy f in $\mathcal{M}$ as follows. The strategy f first mimics f' for a large number of steps k, or until an EC (in which f' stays forever) is reached. If a good EC is not reached, then f aims at only surely satisfying the parity objective. If a good EC is reached, then f behaves as prescribed above, per Lemma 4. Since the probability of f' reaching a good EC within k steps tends to 1, then Player 1 can achieve a value within ϵ of $cost(\mathcal{M}')$.

▶ Example 5. Recall the parity-MDP $\mathcal{M}$ in Figure 1. The GECs are $C_1 = \{s_0, s_2, s_3\}$ and $C_2 = \{s_1\}$, and their costs as MDPs are 1 and 10, respectively. Since Player 1 can force the play to stay in C_1, then $cost_{\text{sure}}(\mathcal{M}) = 1$, and an ϵ approximation is obtained as per Lemma 4, and as described in the proof of Theorem 2. ◀

3.2 Finite-Memory Strategies

In this section we study the problem of finding the sure cost of a parity-MDP, when restricted to finite memory strategies. For a parity-MDP $\mathcal{M}$, we define $cost_{\text{sure},<\infty}(\mathcal{M}) = \inf\{cost_{\text{sure}}(f) : f \text{ is a finite memory strategy for } \mathcal{M}\}$. We prove the upper bound in the following theorem. As stated above, the lower bound is trivial.

▶ **Theorem 6.** *Consider a parity-MDP $\mathcal{M}$. Then, $cost_{\text{sure},<\infty}(\mathcal{M})$ can be computed in NP∩co-NP, and is parity-games hard.*

The general approach is similar to the one we took in Section 3.1. That is, we remove from $\mathcal{M}$ all states that are not sure-winning for Player 1 in $\mathcal{M}^{\texttt{P}}$, and proceed by reasoning about a certain type of ECs. However, for finite-memory strategies, we need a more restricted class of ECs than the GECs that were used in Section 3.1. Indeed, a finite-memory strategy might not suffice to win the sure-parity condition in a GEC.

For a GEC C, let k be the maximal odd priority in C, with $k = -1$ if there are no odd priorities. We define $C^{\max}_{\text{even}} = \{q \in C : \alpha(q) > k \text{ and } \alpha(q) \text{ is even}\}$. We say that a GEC C in

$\mathcal{M}$ is *super good* (SGEC, for short) if from every state $s \in C$, there exists a finite-memory strategy f for $\mathcal{M}|_C^s$ such that the play of $\mathcal{M}$ under f reaches $C^{\max}_{\text{even}}$ w.p. 1, and if the play does not reach $C^{\max}_{\text{even}}$, then it is parity winning. We refer to f as a *witness* to C being a SGEC. If C is not a SGEC, we call its states that do satisfy the above *super-good states*.

We argue that SGECs are the proper notion for reasoning about finite-memory strategies. Specifically, we show that in a SGEC, Player 1 can achieve ϵ-optimal expected cost with a finite-memory strategy, and that every finite-memory winning strategy reaches a SGEC w.p. 1.

Our algorithm finds the maximal SGECs of $\mathcal{M}$ and obtain an MDP $\mathcal{M}'$ in the same manner we did in Section 3.1, namely by assigning high weights to states not in SGECs, and the optimal mean-payoff MDP value to states in SGECs. As there, we claim that $cost(\mathcal{M}') = cost_{\text{sure},<\infty}(\mathcal{M})$. The analysis of the algorithm as well as its concrete details, are, however, more intricate.

▶ Example 7. Recall again the parity-MDP $\mathcal{M}$ in Figure 1. The only SGEC is $\{s_1\}$, and thus we have that $cost_{\text{sure},<\infty}(\mathcal{M}) = 10$ (see the proof of Theorem 2). ◀

We start by proving that the notion of maximal SGECs is well defined. To this end, we present the following lemma, whose proof appears in the full version [4]. Note that in the case of GECs, the lemma was trivial.

▶ **Lemma 8.** *Consider SGEC C and D, such that $C \cap D \neq \emptyset$, then $C \cup D$ is also a SGEC.*

Intuitively, we prove this by considering witnesses f, g for C and D being SGECs. We then modify f such that from every state in C, it tries to reach D for N steps, for some parameter N. Once D is reached, g takes over. If D is not reached, f attempts to reach $C^{\max}_{\text{even}}$. Thus, w.p. 1, the strategy reaches $D^{\max}_{\text{even}}$, and if it does not, it either reaches $C^{\max}_{\text{even}}$ infinitely often, or wins the parity condition.

Next, we note that unlike the syntactic definition of GECs, the definition of SGECs is semantic, as it involves a strategy. Thus, finding the maximal SGECs adds another complication to the algorithm. In fact, it is not hard to see that even checking whether an EC is a SGEC is parity-games hard. Using techniques from [8], we show in the full version [4] that we can reduce the latter to the problem of solving a parity-Büchi game. We thus have the following lemma.

▶ **Lemma 9.** *Consider an EC C in a parity-MDP $\mathcal{M}$. We can decide whether C is a SGEC in NP∩ co-NP, as well as compute a witness strategy and, if C is not a SGEC, find the set of super-good states.*

Next, we show how to find the maximal SGECs of $\mathcal{M}$. Essentially, for every odd rank k, we can find the SGECs whose maximal odd rank is k by removing all states with higher odd ranks, and recursively refining ECs by keeping only super-good states, using Lemma 9. Thus, we have the following (see the full version [4] for complete details).

▶ **Theorem 10.** *Consider a parity-MDP $\mathcal{M}$. We can find the maximal SGECs of $\mathcal{M}$ in NP∩co-NP.*

Theorem 10 shows that our algorithm for computing $cost_{\text{sure},<\infty}(\mathcal{M})$ solves the problem in NP∩co-NP. It remains to prove its correctness. First, Lemma 11 justifies the assignment of costs within a SGEC.

▶ **Lemma 11.** *Consider a SGEC C in $\mathcal{M}$ and a state s in C. Let $v(s) = cost(\mathcal{M}^{\text{MDP}}|_C^s)$. Then, for every $\epsilon > 0$, there exists a finite-memory strategy f of $\mathcal{M}|_C^s$ with $cost_{\text{sure}}(f) \leq v(s) + \epsilon$.*

Proof. Let g be a memoryless strategy such that $cost(g) = cost(\mathcal{M}^{\texttt{MDP}}|^{s}_{C})$. By Theorem 1 such a strategy exists. Let h be a finite-memory strategy that witnesses C being a SGEC. For every $k \in \mathbb{N}$, consider the strategy f_k that repeatedly plays g for k steps and then plays h until $C^{\max}_{\text{even}}$ is reached. Since g and h are finite-memory, then f_k is finite memory. In addition, observe that h reaches $C^{\max}_{\text{even}}$ w.p. 1, and if $C^{\max}_{\text{even}}$ is not reached, then h is parity-winning. Thus f_k is parity-winning, and it reaches Step 1 infinitely often w.p. 1. Moreover, since h has finite memory, then for every $n \in \mathbb{N}$, there is a bounded probability $0 < p(n) \leq 1$ that f reaches $C^{\max}_{\text{even}}$ within n steps, with $\lim_{n\to\infty} p(n) = 1$. Thus, we get that $\lim_{k\to\infty} cost_{\text{sure}}(f_k) = cost_{\text{sure}}(g) = cost(\mathcal{M}^{\texttt{MDP}}|^{s}_{C})$, which concludes the proof. ◀

Lemma 11 implies that we can approximate the optimal value of SGECs with finite-memory strategies. It remains to show that it is indeed enough to consider SGECs. Consider a finite-memory strategy f. Then, w.p. 1, f reaches an EC. Let C be an EC with $\Pr_{\mathcal{M}}(\inf(f) = C) > 0$. The following lemma characterizes an assumption we can make on the behavior of f in such an EC.

▶ **Lemma 12.** *Consider a parity-MDP $\mathcal{M}$ and an EC C. For every finite-memory strategy f, if $\Pr_{\mathcal{M}}(inf(f) = C) > 0$, then there exists a finite-memory strategy g such that for every $s \in C$, we have that $\Pr_{\mathcal{M}^s}(inf(g) = C) = 1$ and every play of g from s stays in C. Moreover, if f is parity winning, then so is g.*

Intuitively, we show that there exists some finite history h such that the strategy f_h, which is f played after seeing the history h, has the following property: f_h reaches and stays in C, and w.p. 1 visits infinitely often all the states in C, and in particular $C^{\max}_{\text{even}}$. For the proof, we consider the set $F = \{f_h : h \text{ is a finite history}\}$. Since f has finite memory, it follows that this set is finite. Using this, we show that if $\Pr_{\mathcal{M}}(\inf(g) = C) < 1$ for every $g \in F$, then $\Pr_{\mathcal{M}}(\inf(f) = C) = 0$, which is a contradiction. Finally, since f is also parity winning, it follows that f_h above is also parity-winning, and is thus a witness for C being a SGEC. The full proof appears in the full version [4].

Finally, by Lemma 13, we can assume that once f reaches an EC C, it stays in C and visits all its states infinitely often w.p. 1. Since f is parity-winning, it follows that C has a maximal even rank, and that f reaches $C^{\max}_{\text{even}}$ w.p. 1. Moreover, in every play that does not reach $C^{\max}_{\text{even}}$, f wins the parity condition. We can thus conclude with the following Lemma, which completes the correctness proof of our algorithm for computing $cost_{\text{sure},<\infty}(\mathcal{M})$. See the full version [4] for the proof.

▶ **Lemma 13.** *Consider a parity-MDP $\mathcal{M}$ and an EC C. For every finite-memory strategy f, if f is parity winning and $\Pr_{\mathcal{M}}(inf(f) = C) > 0$, then C is a SGEC.*

3.3 Comparison with Related Work

Both our work and [7, 12] analyze ECs and reduce the problem to reasoning about an MDP that ignores the hard constraints. The main difference with [7] is that there, the hard and soft constraints have the same objective (i.e., worst-case mean-payoff value and expected-case mean-payoff value). In [7], the strategy played for N rounds to satisfy the soft objective and then at most M rounds to satisfy the hard objective, for some constants N and M. In our setting, we cannot bound M, and in fact it might be the case that Player 1 would play to satisfy the parity objective for the rest of the game (i.e., forever) even after reaching a super-good end component.

The difference with [12] is twofold. First, technically, the type of hard constraints in [12] is worst-case mean-payoff, whereas our setting uses the Boolean parity condition. In classical parity games, the parity condition can be reduced to a mean-payoff objective. Similar reductions, however, do not work in order to reduce our setting to the setting of [12]. Thus, our contribution is orthogonal to [12]. Secondly, Boolean constraints are conceptually different than quantitative constraints, and as we demonstrate in Section 4, they arise naturally in quantitative extensions of Boolean paradigms.

We note that [12] also study a relaxation in which almost-sure winning is allowed for the hard constraints. An analogue in our setting is to consider an almost-sure parity condition. We note that in such a setting, GECs are sufficient for reasoning both about finite-memory and infinite-memory strategies. Moreover, the preprocessing involves solving an almost-sure parity MDP (without mean-payoff constraints), which can be done in polynomial time. Thus, as is the case in [12], we can compute the cost of an MDP with almost-sure hard constraints in polynomial time.

4 Applications

In this section we study two applications of parity-MDPs. Both extend the Boolean synthesis problem. Due to lack of space, our description is only an overview. Full definitions and details can be found in the full version [4]. We start with some basic definitions.

For finite sets I and O of input and output signals, respectively, an I/O *transducer* is $\mathcal{T} = \langle I, O, Q, q_0, \delta, \rho \rangle$, where Q is a set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times 2^I \to Q$ is a total (deterministic) transition function, and $\rho : Q \to 2^O$ is a labeling function on the states. The run of $\mathcal{T}$ on a word $w = i_0 \cdot i_1 \cdots \in (2^I)^\omega$ is the sequence of states $q_0, q_1, \ldots$ such that $q_{k+1} = \delta(q_k, i_k)$ for all $k \geq 0$. The *output* of $\mathcal{T}$ on w is then $o_1, o_2, \ldots \in (2^O)^\omega$ where $o_k = \rho(q_k)$ for all $k \geq 1$. Note that the first output assignment is that of q_1, and we do not consider $\rho(q_0)$. This reflects the fact that the environment initiates the interaction. The *computation of* $\mathcal{T}$ *on* w is then $\mathcal{T}(w) = i_0 \cup o_1, i_1 \cup o_2, \ldots \in (2^{I \cup O})^\omega$. When Q is a finite set, we say that the transducer is finite.

The synthesis problem gets as input a specification $L \subseteq (2^{I \cup O})^\omega$ and generates a transducer $\mathcal{T}$ that realizes L; namely, all the computations of $\mathcal{T}$ are in L. The language L is typically given by an LTL formula [16] or by means of an automaton of infinite words.

4.1 Penalties on Undesired Scenarios

Recall that in Boolean synthesis, the goal is to generate a transducer that associates with each infinite sequence of inputs an infinite sequence of outputs so that the result computation satisfies a given specification. Typically, some behaviors generated by the transducers may be less desired than others. For example, as discussed in Section 1, designs that use fewer resources or minimize expensive activities are preferable. The input to the *synthesis with penalties* problem includes, in addition to the Boolean specification, languages of finite words that describe undesired behaviors, and their costs. The goal is to generate a transducer that realizes the specification and minimizes cost due to undesired behaviors.

Formally, the input to the problem includes languages $L_1, \ldots, L_m$ of finite words over the alphabet $2^{I \cup O}$ and a penalty function $\gamma : \{1, \ldots, m\} \to \mathbb{N}$ specifying for each $1 \leq i \leq m$ the penalty that should be applied for generating a behavior in L_i. As described in Section 1, the language L_i may be local (that is, include only words of length 1) and thus refer only to activation of output signals, may specify short scenarios like flips of output signals, and may also specify rich regular scenarios. Note that we allow penalties also for behaviors that

depend on the input signals. Intuitively, whenever a computation π includes a behavior in L_i, a penalty of $\gamma(i)$ is applied. Formally, if $\pi = \sigma_1, \sigma_2, \ldots$, then for every position $j \geq 1$, we define $\mathit{penalty}(j) = \{i : \text{ there is } k \leq i \text{ such that } \sigma_k \cdot \sigma_{k+1} \cdots \sigma_j \in L_i\}$. That is, $\mathit{penalty}(j)$ points to the subset of languages L_i such that a word in L_i ends in position j. Then, the cost of position j, denoted $\mathit{cost}(j)$, is $\sum_{i \in \mathit{penalty}(j)} \gamma(i)$. Finally, for a finite computation $\pi = \sigma_1, \sigma_2, \ldots$, we define its cost, denoted $\mathit{cost}(\pi)$, as $\limsup_{m \to \infty} \frac{1}{m} \sum_{j=1}^{m} \mathit{cost}(j)$.

Let $\mathcal{A}$ be a deterministic parity automaton (DPW, for short) over the alphabet $2^{I \cup O}$ that specifies the specification ψ. We describe a parity-MDP whose solution is a transducer that realizes $\mathcal{A}$ with the minimal cost for penalties. The idea is simple: on top of the parity game $\mathcal{G}$ described above, we compose monitors that detect undesired scenarios. We assume that the languages $L_1, \ldots, L_m$ and are given by means of deterministic automata on finite words (DFWs) $\mathcal{U}_1, \ldots, \mathcal{U}_m$ where for every $1 \leq i \leq m$, we have that $L(\mathcal{U}_i) = (2^{I \cup O})^* \cdot L_i$. That is, $\mathcal{U}_i$ accepts $\sigma_1 \cdots \sigma_n$ iff there exists $k \leq n$ such that $\sigma_k \cdots \sigma_n \in L_i$. Essentially, we turn $\mathcal{A}$ into a parity-MDP by composing it with the DFWs $\mathcal{U}_1, \ldots, \mathcal{U}_m$. Then, $\mathcal{U}_i$ reaching an accepting state indicates that the penalty for L_i should be applied, which induces the costs in the parity-MDP. The probabilities in the parity-MDP are induced form the distribution of the assignments to input signals. The full construction can be found in the full version [4]. We note that an alternative definition can replace the DFWs $\mathcal{U}_1, \ldots, \mathcal{U}_m$ and the cost function γ by a single weighted automaton that can be composed with $\mathcal{A}$.

4.2 Sensing

Consider a transducer $\mathcal{T} = \langle I, O, Q, q_0, \delta, \rho \rangle$. For a state $q \in Q$ and a signal $p \in I$, we say that p is *sensed in* q if there exists a set $S \subseteq I$ such that $\delta(q, S \setminus \{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is sensed in q if knowing its value may affect the destination of at least one transition from q. We use $\mathit{sensed}(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $\mathit{scost}(q) = |\mathit{sensed}(q)|$. For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{T}$, we define the sensing cost of r, denoted $\mathit{scost}(r)$, as $\frac{1}{m} \sum_{i=0}^{m-1} \mathit{scost}(q_i)$. That is, $\mathit{scost}(r)$ is the average number of sensors that $\mathcal{T}$ uses during r. For a finite input sequence $w \in (2^I)^*$, we define the sensing cost of w in $\mathcal{T}$, denoted $\mathit{scost}_{\mathcal{T}}(w)$, as the sensing cost of the run of $\mathcal{T}$ on w. Finally, the sensing cost of $\mathcal{T}$ is the expected sensing cost of input sequences of length that tends to infinity, which is parameterized by a distribution on $(2^I)^\omega$ given by a sequence of distributions $D_1, D_2, \ldots$ such that $D_t : 2^I \to [0, 1]$ describes the distribution over 2^I at time $t \in \mathbb{N}$. For simplicity, we assume that the distribution is uniform. Thus, $D_t(i) = 2^{-|I|}$ for every $t \in \mathbb{N}$. For the uniform distribution we have $\mathit{scost}(\mathcal{T}) = \lim_{m \to \infty} |(2^I)|^{-m} \sum_{w \in (2^I)^m} \mathit{scost}_{\mathcal{T}}(w)$.

Note that this definition also applies when the transducer is infinite. However, for infinite transducers, the limit in the definition of $\mathit{scost}(\mathcal{T})$ might not exist, and we therefore define $\mathit{scost}(\mathcal{T}) = \limsup_{m \to \infty} |2^I|^{-m} \sum_{w \in (2^I)^m} \mathit{scost}_{\mathcal{T}}(w)$. Finally, for a realizable specification $L \in 2^{I \cup O}$, we define $\mathit{scost}_{I/O}(L) = \inf\{\mathit{scost}(\mathcal{T}) : \mathcal{T} \text{ is an } I/O \text{ transducer that realizes } L\}$.

In [3], we study the sensing cost of *safety* properties. We show that there, a finite, minimally-sensing transducer, always exists (albeit of exponential size), and the problem of computing the sensing cost is EXPTIME-complete. In our current setting, however, a minimally-sensing transducer need not exist, and any approximation may require infinite memory. We demonstrate this with an example.

▶ Example 14. Let $I = \{a\}$ and $O = \{b\}$, and consider the specification $\psi = (\mathsf{GF}a \wedge \mathsf{G}b) \vee \mathsf{G}(\neg b \to \mathsf{XG}(a \leftrightarrow b))$. Thus, ψ states that either a holds infinitely often and b always holds, or, if b does not holds at a certain time, then henceforth, a holds iff b holds. Observe that once the system outputs $\neg b$, it has to always sense a in order to determine the output. The

system thus has an incentive to always output b. This, however, may render ψ false, as a need not hold infinitely often.

We start by claiming that every finite-memory transducer $\mathcal{T}$ that realizes ψ has sensing cost 1. Indeed, let n be the number of states in $\mathcal{T}$. A random input sequence contains the infix $(\neg a)^{n+1}$ w.p. 1. Upon reading such an infix, $\mathcal{T}$ has to output $\neg b$, as otherwise it would not realize ψ on a computation with suffix $(\neg a)^\omega$. Thus, from then on, $\mathcal{T}$ senses a in every state. So, $scost(\mathcal{T}) = 1$.

However, by using infinite-memory transducers, we can follow the construction in Section 3.1 and reduce the sensing cost arbitrarily close to 0. Let $M \in \mathbb{N}$. We construct a transducer $\mathcal{T}'$ as follows. After initializing i to 1, the transducer $\mathcal{T}'$ senses a and outputs b for iM steps. If a does not hold during this time, then $\mathcal{T}'$ outputs $\neg b$ and starts sensing a and outputting b accordingly. Otherwise, if a holds during this time, then $\mathcal{T}'$ stops sensing a for 2^i steps, while outputting b. It then increases i by 1 and repeats the process. Note that $\mathcal{T}'$ outputs $\neg b$ iff a does not hold for iM consecutive positions at the i-th round (which happens w.p. 2^{-iM}). Thus, the probability of $\mathcal{T}'$ outputting $\neg b$ in a random computation is bounded from above by $\sum_{i=1}^{\infty} 2^{-iM} = 2^{-M}$, which tends to 0 as M tends to ∞. Note that in the i-th round, $\mathcal{T}'$ senses a for only iM steps, and then does not sense anything for 2^{iM} steps, so if $\mathcal{T}'$ does not output $\neg b$, the sensing cost is 0. Thus, we have $\lim_{M\to\infty} scost(\mathcal{T}') = 0$. ◀

We proceed by describing the general solution to computing the sensing cost of a specification. Recall that synthesis of a DPW $\mathcal{A}$ is reduced to solving a parity game. When sensing is introduced, it is not enough for the system to win this game, as it now has to win while minimizing the sensing cost. Intuitively, not sensing some inputs introduces incomplete information to the game: once the system gives up sensing, it may not know the state in which the game is and knows instead only a set of states in which the game may be. Technically (see the full version for the detailed proof), we force the system to satisfy the specification with respect to all assignments to the un-sensed inputs by converting the DPW $\mathcal{A}$ into a *universal parity automaton* (UPW) – an automaton in which a the transition function maps each state and letter to a set of successor states, and a word is accepted if all the runs on it are accepting.

▶ **Theorem 15.** *Consider a* DPW *specification $\mathcal{A}$ over $2^{I\cup O}$. There exists a parity-MDP $\mathcal{M}$ such that $cost_{\text{sure}}(\mathcal{M}) = scost_{I/O}(L(\mathcal{A}))$. Moreover, the number of states of $\mathcal{M}$ is singly exponential in that of $\mathcal{A}$, and the number of parity ranks on $\mathcal{M}$ is polynomial in that of $\mathcal{A}$.*

▶ **Theorem 16.** *Consider a* DPW *specification $\mathcal{A}$ over $2^{I\cup O}$. We can compute $scost_{I/O}(L(\mathcal{A}))$ in singly-exponential time. Moreover, the problem of deciding whether $scost_{I/O}(L(\mathcal{A})) > 0$ is EXPTIME-complete.*

Proof. We obtain from $\mathcal{A}$ a parity-MDP $\mathcal{M}$ as per Theorem 15. Observe that the algorithm in the proof of Theorem 3 essentially runs in polynomial time, apart from solving a parity game, which is done in NP∩co-NP. However, deterministic algorithms for solving parity games run in time polynomial in the number of states, and singly-exponential in the number of parity ranks. Since the number of parity ranks in $\mathcal{M}$ is polynomial in that of $\mathcal{A}$, we can find $cost_{\text{sure}}(\mathcal{M})$ in time singly-exponential in the size of $\mathcal{A}$. Since $cost_{\text{sure}}(\mathcal{M}) = scost_{I/O}(L(\mathcal{A}))$, we are done.

For the lower bound, we note that the problem of deciding whether $scost_{I/O}(L(\mathcal{A})) > 0$ is EXPTIME-hard even for a restricted class of automata, namely looping automata [3]. ◀

The input to the synthesis problem is typically given as an LTL formula, rather than a DPW. Then, the translation from LTL to a DPW involves a doubly-exponential blowup.

Thus, a naive solution for computing the sensing cost of a specification given by an LTL formula is in 3EXPTIME. However, by translating the formula to a UPW, rather than a DPW, we show how we can avoid one exponent, thus matching the 2EXPTIME complexity of standard Boolean synthesis.

▶ **Theorem 17.** *Consider an* LTL *specification* ψ *over* $I \cup O$*. We can compute* $scost_{I/O}(L(\psi))$ *in doubly-exponential time.*

Proof. We start by translating ψ to a UPW $\mathcal{A}$ of size single-exponential in the size of ψ. This can be done, for example, by translating $\neg\psi$ to a nondeterministic Büchi automaton [19] and dualizing it. We then follow the proof of Theorem 15, by adding the universal transitions described there directly to the UPW $\mathcal{A}$. Thus, when we finally determinize the UPW to a DPW, the size of the DPW is doubly-exponential, so computing the sensing cost can also be done in doubly-exponential time. ◀

References

1 S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. *Journal of the ACM*, 63(3), 2016.

2 S. Almagor, D. Kuperberg, and O. Kupferman. Regular sensing. In *34th FSTTCS*, volume 29 of *LIPIcs*, pages 161–173, 2014.

3 S. Almagor, D. Kuperberg, and O. Kupferman. The sensing cost of monitoring and synthesis. In *35th FSTTCS*, volume 35 of *LIPIcs*, pages 380–393, 2015.

4 Shaull Almagor, Orna Kupferman, and Yaron Velner. Minimizing expected cost under hard boolean constraints, with applications to quantitative synthesis. *CoRR*, abs/1604.07064, 2016. URL: `http://arxiv.org/abs/1604.07064`.

5 E. Arbel, O. Rokhlenko, and K. Yorav. Sat-based synthesis of clock gating functions using 3-valued abstraction. In *FMCAD*, pages 198–204, 2009.

6 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *25th CONCUR*, pages 266–280, 2014.

7 V. Bruyère, E. Filiot, M. Randour, and J-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In *31st STACS*, volume 25 of *LIPIcs*, pages 199–213, 2014.

8 K. Chatterjee and L. Doyen. Energy and mean-payoff parity markov decision processes. In *36th MFCS*, pages 206–218, 2011.

9 K. Chatterjee, L. Doyen, H. Gimbert, and Y. Oualhadj. Perfect-information stochastic mean-payoff parity games. In *17th FOSSACS*, pages 210–225, 2014.

10 K. Chatterjee, Z. Komárková, and J. Kretínský. Unifying two views on multiple mean-payoff objectives in markov decision processes. In *30th LICS*, pages 244–256, 2015.

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

12 L. Clemente and J-F. Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *30th LICS*, pages 257–268, 2015.

13 L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, 2005.

14 M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi. *Low Power Methodology Manual*. Springer, 2007.

15 O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

16 A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

17 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th POPL*, pages 179–190, 1989.

18 M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

19 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I&C*, 115(1):1–37, 1994.

Stability in Graphs and Games*

Tomáš Brázdil[1], Vojtěch Forejt[2], Antonín Kučera[3], and Petr Novotný[4]

1 **Faculty of Informatics, Masaryk University, Czech Republic**
2 **Department of Computer Science, University of Oxford, UK**
3 **Faculty of Informatics, Masaryk University, Czech Republic**
4 **IST Austria, Klosterneuburg, Austria**

Abstract

We study graphs and two-player games in which rewards are assigned to states, and the goal of the players is to satisfy or dissatisfy certain property of the generated outcome, given as a mean payoff property. Since the notion of mean-payoff does not reflect possible fluctuations from the mean-payoff along a run, we propose definitions and algorithms for capturing the *stability* of the system, and give algorithms for deciding if a given mean payoff and stability objective can be ensured in the system.

1998 ACM Subject Classification F.1.1 Automata, D.2.4 Formal methods

Keywords and phrases Games, Stability, Mean-Payoff, Window Objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.10

1 Introduction

Finite-state graphs and games are used in formal verification as foundational models that capture behaviours of systems with controllable decisions and possibly with an adversarial environment. States correspond to possible configurations of a system, and edges describe how configurations can change. In a game, each state is owned by one of two players, and the player owning the state decides what edge will be taken. A graph is a game where only one of the players is present. When the choice of the edges is resolved, we obtain an *outcome* which is an infinite sequence of states and edges describing the execution of the system.

The long-run average performance of a run is measured by the associated *mean-payoff*, which is the limit average reward per visited state along the run. It is well known that memoryless deterministic strategies suffice to optimize the mean payoff, and the corresponding decision problem is in $\mathsf{NP} \cap \mathsf{coNP}$ for games and in P for graphs. If the rewards assigned to the states are multi-dimensional vectors of numbers, then the problem becomes coNP-hard for games [21].

Although the mean payoff provides an important metric for the average behaviour of the system, by definition it neglects all information about the fluctuations from the mean payoff along the run. For example, a "fully stable" run where the associated sequence of rewards is $1, 1, 1, 1, \ldots$ has the same mean payoff (equal to 1) as a run producing $n, 0, 0, \ldots, n, 0, 0, \ldots$ where a state with the reward n is visited once in n transitions. In many situations, the first run is much more desirable that the second one. Consider, e.g., a video streaming

* The work has been supported by the Czech Science Foundation, grant No. 15-17564S, by EPSRC grant EP/M023656/1, and by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no [291734].

application which needs to achieve a sufficiently high bit-rate (a long-run average number of bits delivered per second) but, in addition, a sufficient level of "stability" to prevent buffer underflows and overflows which would cause data loss and stuttering. Similar problems appear also in other contexts. For example, production lines should be not only efficient (i.e., produce the number of items per time unit as high as possible), but also "stable" so that the available stores are not overfull and there is no "periodic shortage" of the produced items. A food production system should not only produce a sufficiently large amount of food per day on average, but also a certain amount of food daily. These and similar problems motivate the search for a suitable formal notion capturing the intuitive understanding of "stability", and developing algorithms that can optimize the performance under given stability constraints. That is, we are still seeking for a strategy optimizing the mean payoff, but the search space is restricted to the subset of all strategies that achieve a given stability constraint.

Since the mean-payoff $mp(\lambda)$ of a given run λ can be seen as the average reward of a state visited along λ, a natural idea is to define the stability of λ as *sample variance* of the reward assigned to a state along λ. More precisely, let r_i be the reward of the i-th state visited by λ, and let $c_0, c_1, \ldots$ be an infinite sequence where $c_i = (mp(\lambda) - r_i)^2$. The *long-run variance* of the reward assigned to a state along λ, denoted by $va(\lambda)$, is the limit-average of $c_0, c_1, \ldots$ The notion of long-run variance has been introduced and studied for Markov decision processes in [5]. If $va(\lambda)$ is small, then large fluctuations from $mp(\lambda)$ are rare. Hence, if we require that a strategy should optimize mean-payoff while keeping the long-run variance below a given threshold, we in fact impose a *soft* stability constraint which guarantees that "bad things do not happen too often". This may or may not be sufficient.

In this paper, we are particularly interested in formalizing *hard* stability constraints which guarantee that "bad things never happen". We introduce a new type of objectives called *window-stability multi-objectives* that can express a rich set of hard stability constraints, and we show that the set of *all* strategies that achieve a given window-stability multi-objective can be characterized by an effectively constructible *finite-memory permissive strategy scheme*. From this we obtain a meta-theorem saying that if an objective (such as mean-payoff optimization) is solvable for finite-state games (or graphs), then the same objective is solvable also under a given window-stability multi-objective constraint. We also provide the associated upper and lower complexity bounds demonstrating that the time complexity of our algorithms is "essentially optimal".

More specifically, a single *window-stability objective* (inspired by [8], see Related work below) is specified by a window length $W \geq 1$, a checkpoint distance $D \geq 1$, and two bounds μ and ν. For technical reasons, we assume that D divides W. Every run $\lambda = s_0, s_1, s_2, \ldots$ then contains infinitely many *checkpoints* $s_0, s_D, s_{2D}, s_{3D}, \ldots$ The objective requires that the average reward assigned to the states $s_j, \ldots, s_{j+W-1}$, where s_j is a checkpoint, is between μ and ν. In other words, the "local mean-payoff" computed for the states fitting into a window of length W starting at a checkpoint must be within the "acceptable" bounds μ and ν. The role of W is clear, and the intuition behind D is the following. Since D divides W, there are two extreme cases: $D = 1$ and $D = W$. For $D = 1$, the objective closely resembles the standard "sliding window" model over data streams [13]; we require that the local mean-payoff stays within the acceptable bounds "continuously", like the "local bit-rate" in video-streaming. If $D = W$, then the windows do not overlap at all. This is useful in situations when we wish to guarantee some time-bounded periodic progress. For example, if we wish to say that the number of items produced per day stays within given bounds, we set W so that it represents the (discrete) time of one day and put $D = W$. However, there can be also scenarios when we wish to check the local mean-payoff more often then

once during W transitions, but not "completely continuously". In these cases, we set D to some other divisor of W. A *window-stability multi-objective* is a finite conjunction of single window-stability objectives, each with dedicated rewards and parameters. Hence, window-stability multi-objectives allow for capturing more delicate stability requirements such as "a factory should produce between 1500 and 1800 gadgets every week, and in addition, within every one-hour period at least 50 computer chips are produced, and in addition, the total amount of waste produced in 12 consecutive hours does not exceed 500 kg."

Our contribution. The results of this paper can be summarized as follows:

(A) We introduce the concept of window-stability multi-objectives.

(B) We show that there is an algorithm which inputs a game G and a window-stability multi-objective Δ, and outputs a *finite-state permissive strategy scheme* for Δ and G. A finite-state permissive strategy scheme for Δ and G is a finite-state automaton Γ which reads the history of a play and constraints the moves of Player $\Box$ (who aims at satisfying Δ) so that a strategy σ achieves Δ in G iff σ is admitted by Γ. Hence, we can also compute a synchronized product $\mathsf{G} \times \Gamma$ which is another game where the set of *all* strategies for Player $\Box$ precisely represents the set of all strategies for Player $\Box$ in G which achieve the objective Δ. Consequently, *any* objective of the form $\Delta \wedge \Psi$ can be solved for G by solving the objective Ψ for $\mathsf{G} \times \Gamma$. In particular, this is applicable to mean-payoff objectives, and thus we solve the problem of optimizing the mean-payoff under a given window-stability multi-objective constraint. We also analyze the time complexity of these algorithms, which reveals that the crucial parameter which negatively influences the time complexity is the number of checkpoints in a window (i.e., W/D).

(C) We complement the upper complexity bounds of the previous item by lower complexity bounds that indicate that the time complexity of our algorithms is "essentially optimal". Some of these results follow immediately from existing works [21, 8]. The main contribution is the result which says that solving a (single) window-stability objective is PSPACE-hard for games and NP-hard for graphs, even if all numerical parameters (W, D, μ, ν, and the rewards) are encoded in *unary*. The proof is based on novel techniques and reveals that the number of checkpoints in a window (i.e., W/D) is a crucial parameter which makes the problem computationally hard. The window stability objective constructed in the proof satisfies $D = 1$, and the tight window overlapping is used to enforce a certain consistency in Player $\Box$ strategies.

(D) For variance-stability, we argue that while it is natural in terms of using standard mathematical definitions, it does not prevent unstable behaviours. In particular, we show that the variance-stability objective may demand an infinite-memory strategy which switches between two completely different modes of behaviour with smaller and smaller frequency. We also show that the associated variance-stability problem with single-dimensional rewards is in NP for graphs. For this we use some of the results from [5] where the variance-stability is studied in the context of Markov decision processes. The main difficulty is a translation from randomized stochastic-update strategies used in [5] to deterministic strategies.

Related work. Multi-dimensional mean-payoff games were studied in [21], where it was shown that the lim-inf problem, relevant to our setting, is coNP-hard. Further, [11] studies memory requirements for the objectives, and [20] shows that for a "robust" extension (where Boolean combinations of bounds on the resulting vector of mean-payoffs are allowed) the

problem becomes undecidable. Games with quantitative objectives in which both lower and upper bound on the target value of mean-payoff is given were studied in [15]. We differ from these approaches by requiring the "interval" bounds to be satisfied within finite windows, making our techniques and results very different.

As discussed above, we rely on the concept of *windows*, which was in the synthesis setting studied in [8] (see also [14]), as a conservative approximation of the standard mean-payoff objective. More concretely, the objectives in [8] are specified by a maximal window length W and a threshold t. The task is to find a strategy that achieves the following property of runs: a run can be partitioned into contiguous windows of length *at most* W such that in each window, the reward accumulated inside the window divided by the window length is at least t. The objective ensures a local progress in accumulating the reward, and it was not motivated by capturing stability constraints. The fundamental difference between our window-stability approach and windows in [8] is that in the latter one can easily get rid of windows overlapping due to so called *inductive window property*, which does not hold under stricter stability constraints. This results in different computational problems, as witnessed by the fact that our PSPACE lower bound discussed in the point (C) above does not (most likely) carry over to the setting of [8], where a similar-looking decision problem is in P.

The notion of finite-state permissive strategy scheme is based on the concept of *permissive strategies* [1] and multi-strategies [4, 3].

The notion of long-run variance has been introduced and studied for Markov decision processes in [5]. Since we consider deterministic strategies, none of our results is a special case of [5], and we have to overcome new difficulties as it is explained in Section 4.

More generally, our paper fits into an active field of multi-objective strategy synthesis, where some objectives capture the "hard" constraints and the other "soft", often quantitative, objectives. Examples of recent results in this area include [2], where a 2-EXPTIME algorithm is given for the synthesis of combined LTL and mean-payoff objectives, [9], where a combination of parity and mean-payoff performance objectives is studied, or [10], where the controlling player must satisfy a given ω-regular objective while allowing the adversary to satisfy another "environmental" objective.

2 Preliminaries

We use $\mathbb{N}$, $\mathbb{N}_0$, and $\mathbb{Q}$ to denote the sets of positive integers, non-negative integers, and rationals, respectively. Given a set M, we use M^* to denote the set of all finite sequences (words) over M, including the empty sequence. For a vector $\vec{v} = (v_1, \ldots, v_k)$ of numbers and a non-zero number a, we use $\vec{v}[i]$ for v_i, and $\vec{v}/a$ for the vector given by $(\vec{v}/a)[i] = \vec{v}[i]/a$.

A *game* is a tuple $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ where S is a non-empty set of *states*, $(S_\Box, S_\Diamond)$ is a partition of S into two subsets controlled by Player $\Box$ and Player $\Diamond$, respectively, and $E \subseteq S \times S$ are the *edges* of the game such that for every $s \in S$ there is at least one edge $(s,t) \in E$. A *graph* is a game such that $S_\Diamond = \emptyset$. A *run* in G is an infinite path in the underlying directed graph of G. An *objective* Φ is a Borel property[1] of runs. Note that the class of all objectives is closed under conjunction.

A *strategy* for player $\odot$, where $\odot \in \{\Box, \Diamond\}$ is a function $\tau : S^* S_\odot \to S$ satisfying that $(s, \tau(hs)) \in E$ for all $s \in S_\odot$ and $h \in S^*$. The sets of all strategies of Player $\Box$ and Player $\Diamond$ are denoted by Σ_G and Π_G, respectively. When G is understood, we write just Σ and Π.

[1] Recall that the set of all runs can be given the standard Cantor topology. A property is Borel if the set of all runs satisfying the property belongs to the σ-algebra generated by all open sets in this topology.

A pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ together with an initial state s induce a unique run $outcome_s^{\sigma,\pi}$ in the standard way. We say that a strategy $\sigma \in \Sigma$ *achieves* an objective Φ in a state s if $outcome_s^{\sigma,\pi}$ satisfies Φ for every $\pi \in \Pi$. The set of all $\sigma \in \Sigma$ that achieve Φ in s is denoted by $\Sigma^{\Phi}(s)$. An objective Φ is *solvable* for a given subclass $\mathcal{G}$ of finite-state games if there is an algorithm which inputs $\mathsf{G} \in \mathcal{G}$ and its state s, and decides whether $\Sigma^{\Phi}(s) = \emptyset$. If $\Sigma^{\Phi}(s) \neq \emptyset$, then the algorithm also outputs a (finite description of) $\sigma \in \Sigma^{\Phi}(s)$.

We often consider strategies of Player $\Box$ tailored for a specific initial state. A finite sequence of states $s_0, \ldots, s_n$ is *consistent* with a given $\sigma \in \Sigma$ if $s_0, \ldots, s_n$ is a finite path in the graph of G, and $\sigma(s_0, \ldots, s_i) = s_{i+1}$ for every $0 \leq i < n$ where $s_i \in V_{\Box}$. Given $\sigma, \sigma' \in \Sigma$ and $s \in S$, we say that σ and σ' are *s-equivalent*, written $\sigma \equiv_s \sigma'$, if σ and σ' agree on all finite sequences of states initiated in s that are consistent with σ. Note that if $\sigma \equiv_s \sigma'$, then $outcome_s^{\sigma,\pi} = outcome_s^{\sigma',\pi}$ for every $\pi \in \Pi$.

A *reward function* $\varrho : S \to \mathbb{N}_0^k$, where $k \in \mathbb{N}$, assigns non-negative integer vectors to the states of G. We use dim_{ϱ} to denote the dimension k of ϱ, and max_{ϱ} to denote the maximal number employed by ϱ, i.e., $max_{\varrho} = \max\{\varrho(s)[i] \mid 1 \leq i \leq k, s \in S\}$. An objective is *reward-based* if its defining property depends just on the sequence of rewards assigned to the states visited by a run. For every run $\lambda = s_0, s_1, \ldots$ of G, let $mp_{\varrho}(\lambda) = \liminf_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} \varrho(s_i)$ be the *mean payoff* of λ, where the $\liminf_{n\to\infty}$ is taken component-wise. A *mean-payoff* objective is a pair (ϱ, b), where $\varrho : S \to \mathbb{N}_0^k$ is a reward function and $b \in \mathbb{Q}^k$. A run λ satisfies a mean-payoff objective (ϱ, b) if $mp_{\varrho}(\lambda) \geq b$.

Similarly, the *long-run variance of the reward* of a run λ is defined by $va_{\varrho}(\lambda) = \limsup_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} (\varrho(s_i) - mp(\lambda))^2$; intuitively, the long-run variance is a limit superior of sample variances where the samples represent longer and longer run prefixes. A *variance-stability* objective is a triple (ϱ, b, c), where $\varrho : S \to \mathbb{N}_0^k$ is a reward function and $b, c \in \mathbb{Q}^k$. A run λ satisfies a variance-stability objective (ϱ, b, c) if $mp_{\varrho}(\lambda) \geq b$ and $va_{\varrho}(\lambda) \leq c$.

Let $W \in \mathbb{N}$ be a *window size* and $D \in \mathbb{N}$ a *checkpoint distance* such that D divides W. For every $\ell \in \mathbb{N}_0$, the *local mean payoff at the ℓ^{th} checkpoint in a run λ* is defined by $lmp_{W,D,\varrho,\ell}(\lambda) = \frac{1}{W} \sum_{i=0}^{W-1} \varrho(s_{\ell \cdot D + i})$. Thus, every run λ determines the associated infinite sequence $lmp_{W,D,\varrho,0}(\lambda), lmp_{W,D,\varrho,1}(\lambda), lmp_{W,D,\varrho,2}(\lambda), \ldots$ of local mean payoffs. A *window-stability* objective is a tuple $\Phi = (W, D, \varrho, \mu, \nu)$, where $W, D \in \mathbb{N}$ such that D divides W, $\varrho : S \to \mathbb{N}_0^k$ is a reward function, and $\mu, \nu \in \mathbb{Q}^k$. A run λ satisfies Φ if, for all $\ell \in \mathbb{N}$, we have that $\mu \leq lmp_{W,D,\varrho,\ell}(\lambda) \leq \nu$. A *window-stability multi-objective* is a finite conjunction of window-stability objectives.

In this paper, we study the solvability of variance-stability objectives, window-stability multi-objectives, and objectives of the form $\Delta \wedge \Psi$ where Δ is a window-stability multi-objective and Ψ a mean-payoff objective.

3 The Window-Stability Multi-Objectives

This section is devoted to the window-stability multi-objectives and objectives of the form $\Delta \wedge \Psi$, where Δ is a window-stability multi-objective. In Section 3.1, we show how to solve these objectives for finite-state games, and we derive the corresponding upper complexity bounds. The crucial parameter which makes the problem computationally hard is the number of checkpoints in a window. In Section 3.2, we show that this blowup is unavoidable assuming the expected relationship among the basic complexity classes.

3.1 Solving Games with Window-Stability Multi-Objectives

We start by recalling the concept of *most permissive strategies* which was introduced in [1]. Technically, we define *permissive strategy schemes* which suit better our needs, but the underlying idea is the same.

▶ **Definition 1.** Let $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ be a game. A *(finite-memory) strategy scheme* for G is a tuple $\Gamma = (\mathit{Mem}, \mathit{Up}, \mathit{Const}, \mathit{Init})$, where $\mathit{Mem} \neq \emptyset$ is a finite set of *memory elements*, $\mathit{Up} : S \times \mathit{Mem} \to \mathit{Mem}$ is a *memory update* function, $\mathit{Const} : S_\Box \times \mathit{Mem} \to 2^S$ is a *constrainer* such that $\mathit{Const}(s,m) \subseteq \{s' \in S \mid (s,s') \in E\}$, and $\mathit{Init} : S \rightharpoonup M$ is a partial function assigning initial memory elements to some states of S.

We require[2] that $\mathit{Const}(s,m) \neq \emptyset$ for all $(s,m) \in \mathit{Reach}(\mathit{Init})$ such that $s \in S_\Box$. Here, $\mathit{Reach}(\mathit{Init})$ is the least fixed-point of $\mathcal{F} : 2^{S \times \mathit{Mem}} \to 2^{S \times \mathit{Mem}}$ where for all Ω the set $\mathcal{F}(\Omega)$ consists of all (s',m') such that either $(s',m') \in \mathit{Init}$, or there is some $(s'',m'') \in \Omega$ such that $(s'',s') \in E$ and $\mathit{Up}(s'',m'') = m'$; if $s'' \in S_\Box$, we further require $s' \in \mathit{Const}(s'',m'')$. ◀

We say that Γ is *memoryless* if the set Mem is a singleton. Every strategy scheme $\Gamma = (\mathit{Mem}, \mathit{Up}, \mathit{Const}, \mathit{Init})$ for a game $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ determines a game $\mathsf{G}_\Gamma = (S \times \mathit{Mem}, (S_\Box \times \mathit{Mem}, S_\Diamond \times \mathit{Mem}), F)$, where

- for every $(s,m) \in S_\Diamond \times \mathit{Mem}$, $((s,m),(s',m')) \in F$ iff $\mathit{Up}(s,m) = m'$ and $(s,s') \in E$;
- for every $(s,m) \in S_\Box \times \mathit{Mem}$ where $\mathit{Const}(s,m) \neq \emptyset$, we have that $((s,m),(s',m')) \in F$ iff $\mathit{Up}(s,m) = m'$ and $(s,s') \in \mathit{Const}(s,m)$;
- for every $(s,m) \in S_\Box \times \mathit{Mem}$ where $\mathit{Const}(s,m) = \emptyset$, we have that $((s,m),(s',m')) \in F$ iff $s = s'$ and $m = m'$.

A strategy $\sigma \in \Sigma_\mathsf{G}$ is *admitted* by Γ in a given $s \in S$ if $\mathit{Init}(s) \neq \bot$ and for every finite path $s_0, \ldots, s_n$ in G initiated in s which is consistent with σ there is a finite path $(s_0,m_0), \ldots, (s_n,m_n)$ in G_Γ such that $m_0 = \mathit{Init}(s_0)$ and $s_{i+1} \in \mathit{Const}(s_i,m_i)$ for all $0 \leq i < n$ where $s_i \in S_\Box$. Observe that if σ is admitted by Γ in s, then σ naturally induces a strategy $\tau[\sigma,s] \in \Sigma_{\mathsf{G}_\Gamma}$ which is unique up to $\equiv_{(s_0,m_0)}$. Conversely, every $\tau \in \Sigma_{\mathsf{G}_\Gamma}$ and every $s \in S$ where $\mathit{Init}(s) \neq \bot$ induce a strategy $\sigma[\tau,s] \in \Sigma_\mathsf{G}$ such that, for every finite path $(s_0,m_0), \ldots, (s_n,m_n)$ initiated in $(s, \mathit{Init}(s))$ which is consistent with τ, we have that $\sigma[\tau,s](s_0, \ldots, s_n) = s_{n+1}$ iff $\tau((s_0,m_0), \ldots, (s_n,m_n)) = (s_{n+1},m_{n+1})$. Note that $\sigma[\tau,s]$ is determined uniquely up to $\equiv_s$.

▶ **Definition 2.** Let G be a game, Γ a strategy scheme for G, $\Lambda_\mathsf{G} \subseteq \Sigma_\mathsf{G}$, $\Lambda_{\mathsf{G}_\Gamma} \subseteq \Sigma_{\mathsf{G}_\Gamma}$, and $s \in S$. We write $\Lambda_\mathsf{G} \approx_s \Lambda_{\mathsf{G}_\Gamma}$ if the following conditions are satisfied:

- Every $\sigma \in \Lambda_\mathsf{G}$ is admitted by Γ in s, and there is $\tau \in \Lambda_{\mathsf{G}_\Gamma}$ such that $\tau[\sigma,s] \equiv_{(s,\mathit{Init}(s))} \tau$.
- For every $\tau \in \Lambda_{\mathsf{G}_\Gamma}$ there is $\sigma \in \Lambda_\mathsf{G}$ such that $\sigma[\tau,s] \equiv_s \sigma$.

Further, we say that Γ is *permissive* for an objective Φ if $\Sigma^\Phi_\mathsf{G}(s) \approx_s \Sigma_{\mathsf{G}_\Gamma}(s)$ for all $s \in S$, where $\Sigma_{\mathsf{G}_\Gamma}(s)$ is either $\emptyset$ or $\Sigma_{\mathsf{G}_\Gamma}$, depending on whether $\mathit{Init}(s) = \bot$ or not, respectively.

The next proposition follows immediately.

▶ **Proposition 3.** Let G be a game, Φ, Ψ objectives, and Γ a strategy scheme permissive for Φ. Then, for every $s \in S$ we have that $\Sigma^{\Phi \wedge \Psi}_\mathsf{G}(s) \approx_s \Sigma^\Psi_{\mathsf{G}_\Gamma}(s)$.

Another simple but useful observation is that the class of objectives for which a permissive strategy scheme exists is closed under conjunction.

---

[2] Alternatively, we could stipulate $\mathit{Const}(s,m) \neq \emptyset$ for all $(s,m) \in S_\Box \times \mathit{Mem}$, but this would lead to technical complications in some proofs. The presented variant seems slightly more convenient.

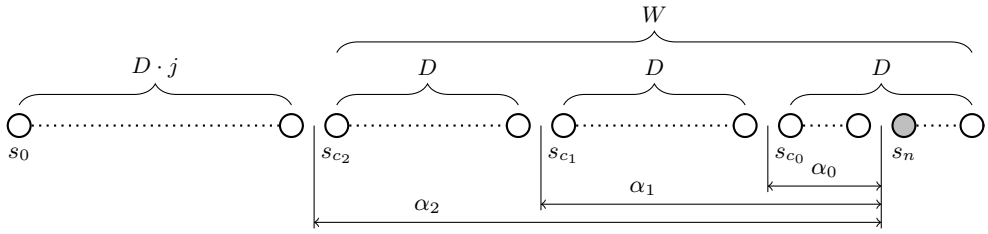


Figure 1 The information represented by the memory elements of Γ (for $\ell = 3$).

▶ **Proposition 4.** Let $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ be a finite-state game, and $n \in \mathbb{N}$. Further, for every $1 \leq i \leq n$, let $\Gamma_i = (Mem_i, Up_i, Const_i, Init_i)$ be a strategy scheme for G which is permissive for Φ_i. Then there is a strategy scheme for G with $\prod_{i=1}^{n} |Mem_i|$ memory elements computable in $\mathcal{O}(|S|^2 \cdot |E| \cdot \prod_{i=1}^{n} |Mem_i|^2)$ time which is permissive for $\Phi_1 \wedge \cdots \wedge \Phi_n$.

As it was noted in [1], permissive strategy schemes do not exist for objectives which admit non-winning infinite runs that do not leave the winning region of player $\Box$, such as reachability, Büchi, parity, mean payoff, etc. On the other hand, permissive strategy schemes exists for "time bounded" variants of these objectives. Now we show how to compute a permissive strategy scheme for a given window-stability objective.

▶ **Theorem 5.** *Let $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ be a finite-state game and $\Phi = (W, D, \varrho, \mu, \nu)$ a window-stability objective where $dim_\varrho = k$. Then there is a strategy scheme Γ with $W \cdot (max_\varrho \cdot W)^{k \cdot (W/D)}$ memory elements computable in $\mathcal{O}(|S|^2 \cdot |E| \cdot W^2 \cdot (max_\varrho \cdot W)^{2k \cdot (W/D)})$ time which is permissive for Φ.*

Proof. Let $\ell = W/D$ and $\mathcal{V} = \{0, \ldots, max_\varrho \cdot (W-1)\}^k$. We put

- $Mem = \{0, \ldots, D-1\} \times \{0, \ldots, \ell-1\} \times \mathcal{V}^\ell$.

Our aim is to construct Γ so that for every run $s_0, s_1, \ldots$ in G, the memory elements in the corresponding run $(s_0, m_0), (s_1, m_1), \ldots$ in G_Γ, where $(s_0, m_0) \in Init$, satisfy the following. Let $n \in \mathbb{N}_0$, and let $m_n = (i, j, \alpha_0, \ldots, \alpha_{\ell-1})$. Then

- $i = n \bmod D$ is the number of steps since the last checkpoint, and $j = \min\{\lfloor n/D \rfloor, \ell-1\}$ is a bounded counter which stores the number of checkpoint visited, up to $\ell - 1$ (this information is important for the initial W steps);
- for every $0 \leq r < \ell$, we put $c_r = n - r \cdot D - (n \bmod D)$ if $n - r \cdot D - (n \bmod D) \geq 0$, otherwise $c_r = n$. Intuitively, the state s_{c_r} is the r-th previous checkpoint visited along $s_0, s_1, \ldots$ before visiting the state s_n (see Figure 1). If the total number of checkpoints visited along the run up to s_n (including s_n) is less than r, we put $c_r = n$. The vector α_r stored in m_n is then equal to the total reward accumulated between s_{c_r} and s_n (not including s_n), i.e., $\alpha_r = \sum_{t=c_r}^{n-1} \varrho(s_t)$ where the empty sum is equal to $\vec{0}$. In particular $m_0 = (0, 0, \vec{0}, \ldots, \vec{0})$.

Note that by Definition 1, we are obliged to define $Up(s, m)$ for *all* pairs $(s, m) \in S \times Mem$, including those that will not be reachable in the end. Let '$\oplus$' be a bounded addition over $\mathbb{N}_0$ defined by $a \oplus b = \min\{a + b, max_\varrho \cdot (W-1)\}$. We extend '$\oplus$' to $\mathcal{V}$ in the natural (component-wise) way. The function Up is constructed as follows (consistently with the above intuition):

- For all $i, j \in \mathbb{N}_0$ such that $0 \leq i \leq D-2$ and $0 \leq j \leq \ell - 1$, we put $Up(s, (i, j, \alpha_0, \ldots, \alpha_{\ell-1})) = (i{+}1, j, \alpha_0 \oplus \varrho(s), \ldots, \alpha_j \oplus \varrho(s), \alpha_{j+1}, \ldots, \alpha_{\ell-1})$.

- For all $j \in \mathbb{N}_0$ such that $0 \leq j \leq \ell - 2$, we put $\mathit{Up}(s, (D{-}1, j, \alpha_0, \ldots, \alpha_{\ell-1})) = (0, j{+}1, \vec{0}, \alpha_0 \oplus \varrho(s), \ldots, \alpha_j \oplus \varrho(s), \alpha_{j+1}, \ldots, \alpha_{\ell-2})$.
- $\mathit{Up}(s, (D{-}1, \ell{-}1, \alpha_0, \ldots, \alpha_{\ell-1})) = (0, \ell{-}1, \vec{0}, \alpha_0 \oplus \varrho(s), \ldots, \alpha_{\ell-2} \oplus \varrho(s))$.

For every $(s, m) \in S \times \mathit{Mem}$, let $\mathit{succ}(s, m)$ be the set of all $(s', m') \in S \times \mathit{Mem}$ such that $(s, s') \in E$ and $\mathit{Up}(s, m) = m'$. Now we define a function $\mathcal{F} : 2^{S \times \mathit{Mem}} \to 2^{S \times \mathit{Mem}}$ such that, for a given $\Omega \subseteq S \times \mathit{Mem}$, the set $\mathcal{F}(\Omega)$ consists of all $(s, (i, j, \alpha_0, \ldots, \alpha_{\ell-1}))$ satisfying the following conditions:

- if $i = D{-}1$ and $j = \ell{-}1$, then $\mu \cdot W \leq \alpha_{\ell-1} + \varrho(s) \leq \nu \cdot W$.
- if $s \in S_\Diamond$, then $\mathit{succ}(s, (i, j, \alpha_0, \ldots, \alpha_{\ell-1})) \subseteq \Omega$.
- if $s \in S_\Box$, then $\mathit{succ}(s, (i, j, \alpha_0, \ldots, \alpha_{\ell-1})) \cap \Omega \neq \emptyset$.

Observe that $\mathcal{F}$ is monotone. Let $\mathit{gfix}(\mathcal{F})$ be the greatest fixed-point of $\mathcal{F}$. For every $(s, m) \in S_\Box \times \mathit{Mem}$, we put $\mathit{Const}(s, m) = \mathit{succ}(s, m) \cap \mathit{gfix}(\mathcal{F})$. Further, the set Init consists of all $(s, (0, 0, \vec{0}, \ldots \vec{0})) \in \mathit{gfix}(\mathcal{F})$. It follows directly from the definition of Γ that $\mathit{Const}(s, m) \neq \emptyset$ for all $(s, m) \in \mathit{Reach}(\mathit{Init})$ such that $s \in S_\Box$.

Since $\mathit{gfix}(\mathcal{F})$ can be computed in $\mathcal{O}(|S|^2 \cdot |E| \cdot W^2 \cdot (\mathit{max}_\varrho \cdot W)^{2k \cdot (W/D)})$ time by the standard iterative algorithm, the strategy scheme $\Gamma = (\mathit{Mem}, \mathit{Up}, \mathit{Const}, \mathit{Init})$ can also be computed in this time. Further, observe the following:

(A) Let $(s_0, m_0), (s_1, m_1), \ldots$ be a run in G_Γ such that $(s_0, m_0) \in \mathit{Init}$. Then $s_0, s_1, \ldots$ is a run in G that satisfies the window-stability objective Φ.

(B) Let $(s, m) \notin \mathit{gfix}(\mathcal{F})$, and let Γ^* be a strategy scheme which is the same as Γ except for its constrainer Const^* which is defined by $\mathit{Const}^*(s, m) = \mathit{succ}(s, m)$ for all $(s, m) \in S_\Box \times \mathit{Mem}$. Then there is a strategy $\pi^* \in \Pi_{\mathsf{G}_{\Gamma^*}}$ such that for every strategy $\sigma^* \in \Sigma_{\mathsf{G}_{\Gamma^*}}$ we have that $\mathit{outcome}_{(s,m)}^{\sigma^*, \pi^*}$ visits a configuration $(t, (D - 1, \ell - 1, \alpha_0, \ldots, \alpha_{\ell-1}))$ where $\alpha_{\ell-1} + \varrho(t) < \mu \cdot W$ or $\alpha_{\ell-1} + \varrho(t) > \nu \cdot W$.

Both (A) and (B) follow directly from the definition of $\mathcal{F}$. Now we can easily prove that Γ indeed encodes the window-stability objective Φ, i.e., $\Sigma_{\mathsf{G}}^{\Phi}(s) \approx_s \Sigma_{\mathsf{G}_\Gamma}(s)$ for all $s \in S$.

Let $\tau \in \Sigma_{\mathsf{G}_\Gamma}(s)$. We need to show that $\sigma[\tau, s]$ achieves the objective Φ in s. So, let $\pi \in \Pi_{\mathsf{G}}$, and let $s_0, s_1, \ldots$ be the run $\mathit{outcome}_s^{\sigma[\tau,s],\pi}$. Obviously, there is a corresponding run $(s_0, m_0), (s_1, m_1), \ldots$ in G_Γ initiated in $(s, \mathit{Init}(s))$, which means that $s_0, s_1, \ldots$ satisfies Φ by applying (A). Now let $\sigma \in \Sigma_{\mathsf{G}}^{\Phi}(s)$. We need to show that σ is admitted by Γ in s. Suppose it is not the case. If $\mathit{Init}(s) = \bot$, then $(s, (0, 0, \vec{0}, \ldots, \vec{0})) \notin \mathit{gfix}(\mathcal{F})$, and hence $\sigma \notin \Sigma_{\mathsf{G}}^{\Phi}(s)$ by applying (B). If $\mathit{Init}(s) \neq \bot$, there is a finite path $s_0, \ldots, s_n, s_{n+1}$ of *minimal length* such that $s_0 = s$, $s_n \in S_\Box$, and the corresponding finite path $(s_0, m_0), \ldots, (s_n, m_n), (s_{n+1}, m_{m+1})$ in G_{Γ^*}, where $m_0 = \mathit{Init}(s)$ and $m_{i+1} = \mathit{Up}(s_i, m_i)$ for all $0 \leq i \leq n$, satisfies that $s_{n+1} \notin \mathit{Const}(s_n, m_n)$. Note that for all $s_i \in S_\Box$ where $i < n$ we have that $s_{i+1} \in \mathit{Const}(s_i, m_i)$, because otherwise we obtain a contradiction with the minimality of $s_0, \ldots, s_n, s_{n+1}$. Since $(s_{n+1}, m_{n+1}) \notin \mathit{gfix}(\mathcal{F})$, by applying (B) we obtain a strategy $\pi^* \in \Pi_{\mathsf{G}_{\Gamma^*}}$ such that for every $\sigma^* \in \Sigma_{\mathsf{G}_{\Gamma^*}}$ we have that $\mathit{outcome}_{(s_{n+1}, m_{n+1})}^{\sigma^*, \pi^*}$ visits a configuration $(t, (D - 1, \ell - 1, \alpha_0, \ldots, \alpha_{\ell-1}))$ where $\alpha_{\ell-1} + \varrho(t) < \mu \cdot W$ or $\alpha_{\ell-1} + \varrho(t) > \nu \cdot W$. Let $\pi \in \Pi_{\mathsf{G}}$ be a strategy satisfying the following conditions:

- $\mathit{outcome}_s^{\sigma,\pi}$ starts with $s_0, \ldots, s_{n+1}$.
- For all finite paths of the form $s_0, \ldots, s_{n+1}, \ldots, s_t$ in G such that $s_t \in S_\Diamond$, let $(s_0, m_0), \ldots, (s_{n+1}, m_{n+1}), \ldots, (s_t, m_t)$ be the unique corresponding finite path in G_{Γ^*}. We put $\pi(s_0, \ldots, s_n, \ldots s_t) = s_{t+1}$, where $\pi^*((s_{n+1}, m_{n+1}), \ldots, (s_t, m_t)) = (s_{t+1}, m_{t+1})$.

Clearly, the run $\mathit{outcome}_s^{\sigma,\pi}$ does *not* satisfy the objective Φ, which contradicts the assumption $\sigma \in \Sigma_{\mathsf{G}}^{\Phi}(s)$. ◀

For every window-stability multi-objective $\Delta = \Phi_1 \wedge \cdots \wedge \Phi_n$ where we have $\Phi_i = (W_i, D_i, \varrho_i, \mu_i, \nu_i)$, we put $M_\Delta = \prod_{i=1}^{n} W_i \cdot (max_{\varrho_i} \cdot W_i)^{k_i \cdot (W_i/D_i)}$, where $k_i = dim_{\varrho_i}$. As a direct corollary to Theorem 5 and Proposition 4, we obtain the following:

▶ **Corollary 6.** *Let $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ be a finite-state game and Δ a window-stability multi-objective. Then there is a permissive strategy scheme for Δ with M_Δ memory elements constructible in time $\mathcal{O}(|S|^2 \cdot |E| \cdot M_\Delta^2)$.*

Now we can formulate a (meta)theorem about the solvability of objectives of the form $\Delta \wedge \psi$, where Δ is a window-stability multi-objective and ψ is a reward-based objective such that the time complexity of solving Ψ for a game $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ and a reward function ϱ can be asymptotically bounded by a function f in $|S|$, $|E|$, max_ϱ, and dim_ϱ.

▶ **Theorem 7.** *Let Ψ be a reward-based objective solvable in $\mathcal{O}(f(|S|, |E|, max_\varrho, dim_\varrho))$ time for every finite-state game $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ and every reward function ϱ for Ψ. Further, let Δ be a window-stability multi-objective. Then the objective $\Delta \wedge \Psi$ is solvable in time*

$$\mathcal{O}(\max\{f(|S| \cdot M_\Delta, |E| \cdot M_\Delta, max_\varrho, dim_\varrho), |S|^2 \cdot |E| \cdot M_\Delta^2\})$$

for every finite-state game $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ and every reward function ϱ for Ψ.

Note that Theorem 7 is a simple consequence of Corollary 6 and Proposition 3.

Since mean-payoff objectives are solvable in $\mathcal{O}(|S| \cdot |E| \cdot max_\varrho)$ time when $dim_\varrho = 1$ [7] and in $\mathcal{O}(|S|^2 \cdot |E| \cdot max_\varrho \cdot k \cdot (k \cdot |S| \cdot max_\varrho)^{k^2+2k+1})$ time when $dim_\varrho = k \geq 2$ [12], we finally obtain:

▶ **Theorem 8.** *Let $\mathsf{G} = (S, (S_\Box, S_\Diamond), E)$ be a finite-state game, Δ a window-stability multi-objective, and $\Psi = (\varrho, b)$ a mean-payoff objective. If $dim_\varrho = 1$, then the objective $\Delta \wedge \Psi$ is solvable in time $\mathcal{O}(|S|^2 \cdot |E| \cdot M_\Delta^2 \cdot max_\varrho)$. If $dim_\varrho = k \geq 2$, then the objective $\Delta \wedge \Psi$ is solvable in time $\mathcal{O}(|S|^2 \cdot |E| \cdot M_\Delta^3 \cdot max_\varrho \cdot k \cdot (k \cdot |S| \cdot M_\Delta \cdot max_\varrho)^{k^2+2k+1})$.*

Let us note that for a given window-stability multi-objective Δ and a given one-dimensional reward function ϱ, there exists the *maximal* bound b such that the objective $\Delta \wedge (\varrho, b)$ is achievable. Further, this bound b is rational and computable in time $\mathcal{O}(|S|^2 \cdot |E| \cdot M_\Delta^2 \cdot max_\varrho)$.

3.2 Lower Bounds for Window-Stability Objectives

We now focus on proving lower bounds for solving the window-stability objectives. More precisely, we establish lower complexity bounds for the problem whose instances are triples of the form (G, s, Φ), where G is a game (or a graph), s is a state of G, $\Phi = (W, D, \varrho, \mu, \nu)$ is a window-stability objective, and the question is whether there exists a strategy $\sigma \in \Sigma$ which achieves Φ in s. The components of Φ can be encoded in unary or binary, which is explicitly stated when presenting a given lower bound.

The main result of this section is Theorem 12 which implies that solving a window-stability objective is PSPACE-hard for games and NP-hard for graphs even if $dim_\varrho = 1$, $D = 1$, and W as well as the values $\varrho(s)$ for all $s \in S$ are encoded in *unary*. Note that an upper time complexity bound for solving these objectives is $\mathcal{O}(|S|^2 \cdot |E| \cdot W \cdot (max_\varrho \cdot W)^{W/D})$ by Corollary 6. Hence, the parameter which makes the problem hard is W/D.

As a warm-up, we first show that lower bounds for solving the window-stability objectives where the reward function is of higher dimension, or W, D, and the rewards are encoded in binary, follow rather straightforwardly from the literature. Then, we develop some new insights and use them to prove the main result.

▶ **Theorem 9.** *Solving the window-stability objectives (where $\dim_\varrho$ is not restricted) is* EXPTIME*-hard. The hardness result holds even if the problem is restricted to instances*

1. *where each component of each reward vector is in $\{-1, 0, 1\}$, or*
2. *where the reward vectors have dimension one (but the rewards are arbitrary binary-encoded numbers).*

Proof. The result can be proven by a straightforward adaptation of the proof of EXPTIME-hardness of multi-dimensional fixed-window mean-payoff problem [8, Lemma 23 and 24]. The reductions in [8] that we can mimic are from the acceptance problem for polynomial-space alternating Turing machines (item 1.) and *countdown games* [17] (item 2.). Although the fixed-window mean-payoff problem differs from ours (see Section 1), an examination of the proofs in [8] reveals that almost the same constructions work even in our setting. In particular, while the problem to which countdown games are reduced in [8] assumes two-dimensional rewards, in our setting we can restrict to single dimension due to window-stability objective imposing both a lower and an upper bound on local mean payoff. ◀

The reductions in the previous theorem require that the window size W is encoded in binary, as the windows need to be exponentially long in the size of the constructed graph. For the case when W is given in unary encoding, the following result can be adapted from [8].

▶ **Theorem 10.** *Solving the window-stability objectives (where $\dim_\varrho$ is not restricted) where the window size W is encoded in unary is* PSPACE*-hard, even if it is restricted to instances where the components of reward functions are in $\{-1, 0, 1\}$.*

A proof of Theorem 10 is obtained by adapting a proof from [8, Lemma 25], where a reduction from generalized reachability games is given.

The results of [8] do not yield lower bounds for window-stability objectives with one-dimensional reward functions in which either the windows size or the rewards are encoded in unary. In our setting, for the case of binary rewards/unary window size one can come up with NP-hardness for graphs and PSPACE-hardness for games via reductions from the Subset-Sum problem and its quantified variant [18], respectively. Similarly, for unary rewards/binary window size a PSPACE-hardness for games via reduction from emptiness of 1-letter alternating finite automata [16] seems plausible. We do not follow these directions, since we are able to prove an even stronger and somewhat surprising result: solving window-stability objectives with one-dimensional reward functions is PSPACE-hard for games and NP-hard for graphs even if *all* the numbers in the input instance are encoded in unary. The proof of this result requires a new proof technique sketched below.

We rely on reductions from special variants of the SAT and QBF problems. An instance of the Balanced-3-SAT problem is a propositional formula φ in a 3-conjunctive normal form which contains an even number of variables. Such an instance is positive if and only if φ admits a satisfying assignment which maps exactly half of φ's variables to 1 (*true*). We can also define a quantified variant, a Balanced-QBF problem: viewing a quantified Boolean formula $\psi = \exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n \, \varphi$ (where φ is quantifier-free), as a game between player controlling existentially quantified variables (who strives to satisfy φ) and player controlling universal variables (who aims for the opposite), we ask whether the existential player can enforce assignment mapping exactly half of the variables to 1 and satisfying φ (a formal definition of Balanced-QBF is given in [6]. The following lemma is easy.

▶ **Lemma 11.** *The* Balanced-QBF *problem is* PSPACE*-complete. The* Balanced-3-SAT *is* NP*-complete.*

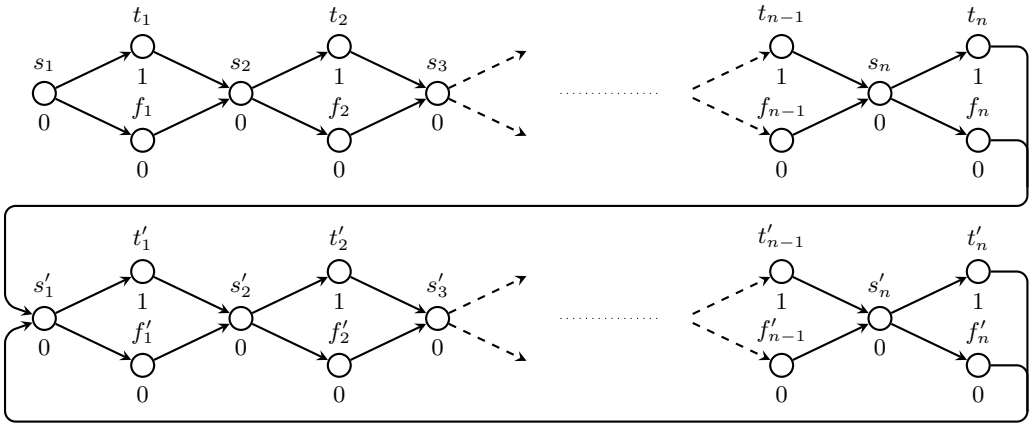


Figure 2 In the lower gadget, Player □ must mimic the assignment she chose in the upper one.

Let G be a finite-state game and $\Phi = (W, D, \varrho, \mu, \nu)$ a window-stability objective. An instance (G, s, Φ) is *small* if dim_ϱ=1, and W, D, max_ϱ, and the numerators and denominators of the fully reduced forms of μ and ν, are bounded by the number of states of G.

▶ **Theorem 12.** *Solving the window-stability objectives with one-dimensional reward functions is* PSPACE*-hard for games and* NP*-hard for graphs, even for small instances.*

Proof (sketch). We proceed by reductions from Balanced-3-SAT for graphs and from Balanced-QBF for games. As the reductions are somewhat technical, we explain just their core idea. The complete reduction can be found in [6].

Assume a formula φ in 3-CNF with variables $\{x_1, \ldots, x_n\}$, n being even. Consider the graph G in Figure 2. Both the "upper" gadget (consisting of non-primed states) and the "lower" gadget (with primed states) represent a standard "assignment choice" gadget, in which Player □ selects an assignment to variables in φ (e.g. choosing an edge going to t_1 from s_1 corresponds to setting variable x_1 to *true* etc.). With no additional constraints, □ can choose different assignments in the two gadgets, and she may change the assignment upon every new traversal of the lower gadget. Now assign reward 1 to states that correspond to setting some variable to *true* and 0 to all the other states, let window size $W = 2n$, checkpoint distance $D = 1$, $\mu = \frac{n}{2}$, and $\nu = \frac{n}{2} + \frac{1}{3n}$ (say). In order to satisfy the window-stability objective $(W, D, \varrho, \mu, \nu)$ from s_1, □ has to select a balanced assignment in the upper gadget and moreover, mimic this assignment in all future points in the lower gadgets. The necessity of the first requirement is easy. For the second, assume that there is some ℓ such that in the ℓ-th step of the run λ the player chooses to go from, say, s_i to t_i (or from s'_i to t'_i), while in the $(\ell + 2n)$-th step she goes from s'_i to f'_i. Then the rewards accumulated within windows starting in the ℓ-th and $(\ell + 1)$-th step, respectively, differ by exactly one. Thus, $|lmp_{W,D,\ell}(\lambda) - lmp_{W,D,\ell+1}(\lambda)| = 1/2n > 1/3n$, which means that the local mean payoffs at the ℓ-th and $(\ell + 1)$-th checkpoint cannot both fit into the interval $[\mu, \nu]$.

Note that we use the balanced variant of 3-SAT and QBF, as to set up μ and ν we need to know in advance the number of variables assigned to *true*.

Once we force the player to commit to some assignment using the above insight, we can add more copies of the "primed" gadget that are used to check that the assignment satisfies φ. Intuitively, we form a cycle consisting of several such gadgets, one gadget per clause of φ, the gadgets connected by paths of suitable length (not just by one edge as above). In each clause-gadget, satisfaction of the corresponding clause C by the chosen

assignment is checked by allowing the player to accrue a small additional reward whenever she visits a state representing satisfaction of some literal in C. This small amount is then subtracted and added again on a path that connects the current clause-gadget with the next one: subtracting forces the player to satisfy at least one literal in the previous clause-gadget (and thus accrue the amount needed to "survive" the subtraction) while adding ensures that this "test" does not propagate to the next clause-gadget. Rewards have to be chosen in a careful way to prevent the player from cheating. For PSPACE-hardness of the game version we simply let the adversary control states in the initial gadget (but not in clause-gadgets) corresponding to universally quantified variables. ◀

4 The variance-stability problem

In this section, we prove the results about variance-stability objectives promised in Section 1.

▶ **Theorem 13.** *The existence of a strategy achieving a given one-dimensional variance-stability objective for a given state of a given graph is in* NP. *Further, the strategy may require infinite memory.*

Let us now prove the above theorem. Consider a graph $\mathsf{G} = (S, (S, \emptyset), E)$ and an instance of the variance-stability problem determined by a reward function ϱ together with a mean-payoff bound $b \in \mathbb{Q}$ and a variance bound $c \in \mathbb{Q}$. We assume that all runs are initiated in a fixed initial state $\bar{s}$. A *frequency vector* is a tuple $(f_e)_{e \in E} \in [0,1]^{|E|}$ with $\sum_{e \in E} f_e = 1$ and

$$\sum_{s' : (s',s) \in E} f_{(s',s)} = \sum_{s' : (s,s') \in E} f_{(s,s')} \tag{1}$$

for all $s \in S$. Now consider the following constraints:

$$mp := \sum_{s \in S} f_s \cdot \varrho(s) \geq a \quad \text{and} \quad va := \sum_{s \in S} f_s \cdot (\varrho(s) - mp)^2 \leq b \tag{2}$$

Here $f_s = \sum_{(s',s) \in E} f_{(s',s)}$ for every $s \in S$. As every graph is a special case of a Markov decision process, we may invoke Proposition 5 of [5] and obtain the following proposition.

▶ **Proposition 14** ([5])**.** Assume that there is a solution to the given variance-stability problem. Then there is a frequency vector $(f_e)_{e \in E}$ satisfying the inequalities (2). All $e \in E$ satisfying $f_e > 0$ belong to the same strongly connected component of G reachable from $\bar{s}$.

The above inequalities (2) can be turned into a negative semi-definite program, using techniques of [5], and hence their satisfiability can be decided in non-deterministic polynomial time [19]. To finish our algorithm, we need to show that a solution to the above inequalities can also be turned into a strategy which visits each $e \in E$ with the frequency f_e.

Let $\lambda = s_0 s_1 \ldots$ be a run. Given $e \in E$ and $i \in \mathbb{N}$ we define a random variable $a_i^e(\lambda)$ to take value 1 if $(s_i, s_{i+1}) = e$, and 0 otherwise.

▶ **Lemma 15.** *Suppose $(f_e)_{e \in E}$ is a frequency vector such that all $e \in E$ satisfying $f_e > 0$ belong to the same strongly connected component reachable from the initial state $\bar{s}$. Then there is a strategy σ_f with $\lim_{i \to \infty} \frac{1}{i+1} \sum_{j=0}^{i} a_i^e(\lambda) = f_e$ for all $e \in E$, where λ is the run induced by σ_f (initiated in $\bar{s}$).*

Proof. Let us assume, w.l.o.g., that G itself is strongly connected. If $(f_e)_{e \in E}$ is rational and all edges e satisfying $f_e > 0$ induce a strongly connected graph, we may easily construct the strategy σ_f as follows. We multiply all numbers f_e with the least-common-multiple of

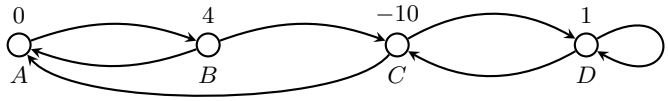


Figure 3 One player game in which there is an infinite-memory strategy σ such that $mp(outcome_s^{\sigma,\pi}) \geq 3/2$ and $va(outcome_s^{\sigma,\pi}) \leq 9/4$ (here π is the only "trivial" strategy of the environment). However, there is no finite-memory σ with this property.

their denominators and obtain a vector of natural numbers f'_e that still satisfy the above equation (1). Now we may imagine the game as a multi-digraph, where each edge e has the multiplicity f'_e. It is easy to show that the flow equations are exactly equivalent to existence of a directed Euler cycle. From this Euler cycle in the digraph we immediately get a cycle in our game which visits each edge exactly f'_e times. By repeating the cycle indefinitely we obtain a run with the desired frequencies f_e of edges.

For vectors with irrational frequencies we adapt the above approach and use a sequence of converging rational approximations. The proof is technical, and is given in [6]. ◀

This finishes the proof of Theorem 13.

We now show that variance-stability objectives may require strategies with infinite memory. Consider the graph in Figure 3, and the variance-stability objective which requires to achieve a mean payoff of at least 3/2 and long-run variance at most 9/4. Observe that there is an infinite-memory strategy achieving the above bounds. It works as follows: We start in the state A, the strategy proceeds in infinitely many phases. In the n-th phase it goes n times from A to B and back. Afterwards it goes to D, makes $2n$ steps on the loop on D, and then returns back to A. One can show that the mean payoff converges along this run. The limit is $4/4 + 0/4 + 1/2 = \frac{3}{2}$ since the -10 reward is obtained with zero frequency. The long-run variance is $\frac{1}{4}\left(-\frac{3}{2}\right)^2 + \frac{1}{4}\left(4-\frac{3}{2}\right)^2 + \frac{1}{2}\left(1-\frac{3}{2}\right)^2 = \frac{9}{4}$. Now we show that there is no finite-memory strategy achieving a mean payoff of 3/2 and a long-run variance of 9/4. Note that the maximal mean payoff achievable (without any constraints) in the graph is 2. Assume that there is a finite memory strategy σ yielding mean payoff x with $3/2 \leq x \leq 2$, and variance at most 9/4. We first argue that σ visits C with zero frequency. Denote by f_Y the frequency of state Y. Because $x = 0 \cdot f_A + 4 \cdot f_B + (-10) \cdot f_C + 1 \cdot f_D$ by the definition of mean payoff, and also $f_A = f_B$ and $f_D = 1 - f_A - f_B - f_C$ by the definition of our graph, we have $f_A = (x + 11 \cdot f_C - 1)/2$ and $f_D = 2 - x - 12 \cdot f_C$. Thus, the variance is

$$\begin{aligned} &f_A \cdot (0-x)^2 + f_B \cdot (4-x)^2 + f_C \cdot (-10-x)^2 + f_D (1-x)^2 \\ &\quad = \frac{x-1}{2} \cdot \left((0-x)^2 + (4-x)^2\right) + (2-x) \cdot (1-x)^2 \\ &\qquad + f_C \cdot \left(\frac{11}{2} \cdot \left((0-x)^2 + (4-x)^2\right) + (-10-x)^2 - 12 \cdot (1-x)^2\right). \end{aligned}$$

Using calculus techniques one can easily show that the first term is at least 9/4 for all $x \in [3/2, 2]$, while the parenthesized expression multiplied by f_C is positive for all such x. Hence $f_C = 0$. But any finite-memory strategy that stays in C with frequency 0 either eventually loops on D, in which case the mean payoff is only 1, or it eventually loops on A and B, in which case the variance is 4.

Even finite-memory strategies that approximate the desired variance-stability (up to some $\varepsilon > 0$) must behave in a peculiar way: Infinitely many times stay in $\{A, B\}$ for a large number of steps (depending on ε) and also stay in C for a large number of steps. Hence, in a real-life system, a user would observe two repeating phases, one with low mean payoff but high instability, and one with low stability and high mean payoff.

References

1 Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO – Theoretical Informatics and Applications*, 36(3):261–275, 2002.

2 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, and Jean-François Raskin. Synthesis from LTL specifications with mean-payoff objectives. In *Proc. of TACAS 2013*, volume 7795 of *LNCS*, pages 169–184. Springer, 2013.

3 Patricia Bouyer, Marie Duflot, Nicolas Markey, and Gabriel Renault. Measuring permissivity in finite games. In *Proc. of CONCUR 2009*, volume 5710 of *LNCS*, pages 196–210. Springer, 2009.

4 Patricia Bouyer, Nicolas Markey, Jörg Olschewski, and Michael Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In *Proc. of ATVA 2011*, volume 6996 of *LNCS*, pages 135–149. Springer, 2011.

5 Tomáš Brázdil, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Trading performance for stability in Markov decision processes. In *Proc. of LICS 2013*, pages 331–340. IEEE, 2013.

6 Tomáš Brázdil, Vojtech Forejt, Antonín Kučera, and Petr Novotný. Stability in graphs and games. *CoRR*, abs/1604.06386, 2016.

7 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2010.

8 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015.

9 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *Proc. of LICS 2005*, pages 178–187. IEEE, 2005.

10 Krishnendu Chatterjee, Florian Horn, and Christof Löding. Obliging games. In *Proc. of CONCUR 2010*, volume 6269 of *LNCS*, pages 284–296. Springer, 2010.

11 Krishnendu Chatterjee, Mickael Randour, and Jean-Franccois Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51:129–163, 2014.

12 Krishnendu Chatterjee and Yaron Velner. Hyperplane separation technique for multidimensional mean-payoff games. In *Proc. of CONCUR 2013*, volume 8052 of *LNCS*, pages 500–515. Springer, 2013.

13 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

14 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Looking at mean-payoff through foggy windows. In *Proc. of ATVA 2015*, volume 9364 of *LNCS*, pages 429–445. Springer, 2015.

15 Paul Hunter and Jean-François Raskin. Quantitative games with interval objectives. In *Proc. of FST&TCS 2014*, volume 29 of *LIPIcs*, pages 365–377, 2014.

16 Petr Jančar and Zdeněk Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.*, 104(5):164 – 167, 2007.

17 Marcin Jurdzinski, Jeremy Sproston, and François Laroussinie. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Comp. Sci.*, 4(3), 2008.

18 Stephen Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1–3):211 – 229, 2006.

19 Stephen A. Vavasis. Quadratic programming is in NP. *Inf. Process. Lett.*, 36(2):73–77, 1990.

20 Yaron Velner. Robust multidimensional mean-payoff games are undecidable. In *Proc. of FOSSACS 2015*, volume 9034 of *LNCS*, pages 312–327. Springer, 2015.

21 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Rabinovich, and Jean-Francois Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177 – 196, 2015.

On the Complexity of Heterogeneous Multidimensional Quantitative Games

Véronique Bruyère[1], Quentin Hautem[*2], and Jean-François Raskin[†3]

1 **Département d'informatique, Université de Mons (UMONS), Mons, Belgium**
2 **Département d'informatique, Université de Mons (UMONS), Mons, Belgium**
3 **Département d'informatique, Université Libre de Bruxelles (U.L.B.), Brussels, Belgium**

Abstract

We study two-player zero-sum turn-based games played on multidimensional weighted graphs with heterogeneous quantitative objectives. Our objectives are defined starting from the measures Inf, Sup, LimInf, and LimSup of the weights seen along the play, as well as on the window mean-payoff (WMP) measure recently introduced in [6]. Whereas multidimensional games with Boolean combinations of classical mean-payoff objectives are undecidable [19], we show that CNF/DNF Boolean combinations for heterogeneous measures taken among {WMP, Inf, Sup, LimInf, LimSup} lead to EXPTIME-completeness with exponential memory strategies for both players. We also identify several interesting fragments with better complexities and memory requirements, and show that some of them are solvable in PTIME.

1998 ACM Subject Classification B.6.3 [design aids]: automatic synthesis; F.1.2 [Modes of computation]: interactive and reactive computation

Keywords and phrases two-player zero-sum games played on graphs, quantitative objectives, multidimensional heterogeneous objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.11

1 Introduction

Two-player zero-sum turn-based games played on graphs are an adequate mathematical model to solve the *reactive synthesis problem* [18]. To model systems with resource constraints, like embedded systems, games with quantitative objectives have been studied, e.g. mean-payoff [22] and energy games [3]. In [5, 21, 6, 20], multidimensional games with conjunctions of several quantitative objectives have been investigated, such that all dimensions use the *same* measure. In this paper, we initiate the study of games played on multidimensional weighted graphs such that the objectives use *different* measures over the dimensions. As an example of conjunction of heterogeneous measures, you may want to design a system with (ϕ_1) a good window mean-response time (MP), that (ϕ_2) avoids too slow reaction after a finite prefix (LimInf), and that (ϕ_3) does not exceed some peak energy consumption in

* Author supported by FRIA fellowship
† Author supported by ERC Starting Grant (279499: inVEST) and partly by European project Cassting (FP7-ICT-601148).

the long run (LimSup). Now, assume that you want to ensure such a conjunction only under the hypothesis that (ψ) the frequency of requests from the environment is below some threshold (expressible as an MP). Such a property $\psi \to (\phi_1 \wedge \phi_2 \wedge \phi_3)$ is equivalent to the DNF Boolean combination of heterogeneous measures $\neg\psi \vee (\phi_1 \wedge \phi_2 \wedge \phi_3)$. We claim that such heterogeneous quantitative games provide a general, natural, and expressive model for the reactive synthesis problem and that the complexity of solving these games needs to be studied. Knowing that Boolean combination of MP objectives is undecidable [19], we have initiated this study with the omega-regular measures Inf, Sup, LimInf and LimSup, and the recent interesting window version WMP of MP introduced in [6].

While the MP measure considers the long-run average of the weights along the whole play, the WMP measure considers weights over a *local window of a given size* sliding along the play. A WMP objective asks now to ensure that the average weight satisfies a given constraint over every bounded window. This is a strengthening of the MP objective: winning for the WMP objective implies winning for the MP objective. Also, any finite-memory strategy that forces an MP measure larger than threshold $\nu + \epsilon$ (for any $\epsilon > 0$), also forces the WMP measure to be larger than ν provided that the window size is taken large enough. Aside from their naturalness, WMP objectives are algorithmically more tractable than classical MP objectives, see [6, 14]. First, unidimensional WMP games can be solved in polynomial time when working with polynomial windows [6] while only pseudo-polynomial time algorithms are known for MP games [22, 4]. Second, multidimensional games with Boolean combinations of MP objectives are undecidable [19], whereas we show here that games with Boolean combinations of WMP objectives and other classical objectives are decidable.

We show in this paper (see also Table 1) that the problem is EXPTIME-complete for CNF/DNF Boolean combinations of heterogeneous measures taken among {WMP, Inf, Sup, LimInf, LimSup}. We provide a detailed study of the complexity when the Boolean combination of the measures is replaced by an intersection, as it is often natural in practice to consider conjunction of constraints. EXPTIME-completeness of the problem still holds for the intersection of measures in {WMP, Inf, Sup, LimInf, LimSup}, and we get PSPACE-completeness when WMP measure is not considered. To avoid EXPTIME-hardness, we consider fragments where there is at most one occurrence of a WMP measure. In case of intersections of one WMP objective with any number of objectives of one kind among {Inf, Sup, LimInf, LimSup} (this number must be fixed in case of objectives Sup), we get P-completeness when dealing with polynomial windows, a reasonable hypothesis in practical applications. In case of no occurrence of WMP measure, we propose several refinements (on the number of occurrences of the other measures) for which we again get P-completeness. Some of our results are obtained by reductions to known qualitative games but most of them are obtained by new algorithms that require new ideas to handle in an optimal way one WMP objective together with qualitative objectives such as safety, reachability, Büchi and coBüchi objectives. In our results, we also provide a careful analysis of the memory requirements of winning strategies for both players.[1]

Let us mention some related work. Multidimensional mean-payoff games have been studied in [20]. Conjunction of lim inf mean-payoff ($\underline{\text{MP}}$) objectives are coNP-complete, conjunctions of lim sup MP ($\overline{\text{MP}}$) objectives are in $\text{NP} \cap \text{coNP}$. The general case of Boolean combinations of $\underline{\text{MP}}$ and $\overline{\text{MP}}$ is undecidable [19]. Multidimensional energy games with unfixed initial credit are coNP-complete [5, 8], and with fixed initial credit, they are 2EXPTIME-complete [16]. Generalization of these games with imperfect information have been studied

[1] All details of this paper can be found in the arXiv version arXiv:1511.08334v2.

Table 1 Overview - Our results are marked with (*).

<table>
<tr><th>Objectives</th><th>Complexity class</th><th>Player 1 memory</th><th>Player 2 memory</th></tr>
<tr><td>(CNF/DNF) Boolean combination of $\underline{\mathsf{MP}}$, $\overline{\mathsf{MP}}$ [19]</td><td>Undecidable</td><td>infinite</td><td>infinite</td></tr>
<tr><td>(CNF/DNF) Boolean combinaison of
WMP, Inf, Sup, LimInf, LimSup (*)</td><td rowspan="3">EXPTIME-complete</td><td colspan="2" rowspan="3">exponential</td></tr>
<tr><td>Intersection of WMP, Inf, Sup, LimInf, LimSup (*)</td></tr>
<tr><td>Intersection of WMP [6]</td></tr>
<tr><td rowspan="2">Intersection of Inf, Sup, LimInf, LimSup (*)
and refinements (*)</td><td>PSPACE-complete</td></tr>
<tr><td colspan="3">See Table 4</td></tr>
<tr><td>Intersection of $\underline{\mathsf{MP}}$ [20]</td><td>coNP-complete</td><td rowspan="2">infinite</td><td rowspan="3">memoryless</td></tr>
<tr><td>Intersection of $\overline{\mathsf{MP}}$ [20]</td><td rowspan="2">NP ∩ coNP</td></tr>
<tr><td>Unidimensional MP [22, 4]</td><td>memoryless</td></tr>
<tr><td>Unidimensional WMP [6]
WMP ∩ Ω with Ω ∈ {Inf, Sup, LimInf, LimSup} (*)</td><td>P-complete
(Polynomial windows)</td><td colspan="2">pseudo-polynomial</td></tr>
<tr><td>Unidimensional Inf, Sup, LimInf, LimSup [13]</td><td>P-complete</td><td colspan="2">memoryless</td></tr>
</table>

in [9] and shown undecidable. The WMP measure was first introduced in [6]. Unidimensional WMP games can be solved in polynomial time for polynomial windows, and multidimensional WMP games are EXPTIME-complete. In [6], the WMP measure is considered on all the dimensions, with no conjunction with other measures like Inf, Sup, LimInf, and LimSup, and the case of Boolean combinations of WMP objectives is not investigated. Games with objectives expressed in fragments of LTL have been studied in [1]. Our result that games with intersection of objectives in {Inf, Sup, LimInf, LimSup} are in PSPACE can be obtained by reduction to some of these fragments. But we here propose a simple proof adapted to our context, that allows to identify several polynomial fragments. Our other results cannot be obtained in this way and require new techniques and new algorithmic ideas.

2 Preliminaries

We consider turn-based two-player games on a finite multidimensional weighted directed graph. A *multi-weighted game structure* is a tuple $G = (V_1, V_2, E, w)$ where (i) (V, E) is a finite directed graph, with V 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), (ii) (V_1, V_2) forms a partition of V such that V_p is the set of vertices controlled by player $p \in \{1, 2\}$, and (iii) $w : E \to \mathbb{Z}^n$ is the n-dimensional weight function that associates a vector of n weights to each edge, for some $n \geq 1$. We also simply say that G is a (n-weighted) game structure.

The opponent of player $p \in \{1, 2\}$ is denoted by $\overline{p}$. A *play* of G is an infinite sequence $\rho = \rho_0\rho_1 \ldots \in V^\omega$ such that $(\rho_k, \rho_{k+1}) \in E$ for all $k \in \mathbb{N}$. *Histories* of G are finite sequences $\rho = \rho_0 \ldots \rho_i \in V^+$ defined in the same way. We denote by $\mathsf{Plays}(G)$ the set of plays in G and by $\mathsf{Hist}(G)$ the set of histories. Given a play $\rho = \rho_0\rho_1 \ldots$, the history $\rho_k \ldots \rho_{k+i}$ is denoted by $\rho_{[k,k+i]}$. We denote by w_m the projection of function w on the m^{th} dimension, and by W the maximum weight in absolute value on all dimensions.

Strategies, objectives and winning sets

A *strategy* σ for player $p \in \{1, 2\}$ is a function $\sigma : V^*V_p \to V$ assigning to each history $hv \in V^*V_p$ a vertex $v' = \sigma(hv)$ such that $(v, v') \in E$. It is *memoryless* if $\sigma(hv) = \sigma(h'v)$ for all histories $hv, h'v$ ending with the same vertex v, that is, σ is a function $\sigma : V_p \to V$. It

is *finite-memory* if $\sigma(hv)$ only needs finite memory of the history hv (recorded by a Moore machine). Given a strategy σ of player $p \in \{1,2\}$, we say that a play ρ of G is *consistent* with σ if $\rho_{k+1} = \sigma(\rho_0 \dots \rho_k)$ for all $k \in \mathbb{N}$ such that $\rho_k \in V_p$. A history consistent with a strategy is defined similarly. Given an initial vertex v_0, and a strategy σ_p of each player p, we have a unique play that is consistent with both strategies, called the *outcome* of (σ_1, σ_2) from v_0, and denoted by $\mathsf{Out}(v_0, \sigma_1, \sigma_2)$.

An *objective for player p* is a set of plays $\Omega \subseteq \mathsf{Plays}(G)$; it is *qualitative* (it only depends on the graph (V, E)), or *quantitative* (it also depends on the weight function w). A play ρ is *winning* for player p if $\rho \in \Omega$, and losing otherwise (i.e. winning for player $\overline{p}$). We thus consider zero-sum games such that the objective of player $\overline{p}$ is $\overline{\Omega} = \mathsf{Plays}(G) \setminus \Omega$, i.e., opposite to the objective Ω of player p. In the following, we always take the point of view of player 1 by supposing that Ω is his objective, and we denote by (G, Ω) the corresponding *game*. A strategy σ_p for player p is *winning* from an initial vertex v_0 if $\mathsf{Out}(v_0, \sigma_p, \sigma_{\overline{p}}) \in \Omega$ for all strategies $\sigma_{\overline{p}}$ of player $\overline{p}$. Vertex v_0 is also called *winning* for player p and the *winning set* Win_p^{Ω} is the set of all his winning vertices. Similarly the winning vertices of player $\overline{p}$ are those from which $\overline{p}$ can ensure his objective $\overline{\Omega}$ against all strategies of player p, and $\mathsf{Win}_{\overline{p}}^{\overline{\Omega}}$ is his winning set. The game is said *determined* when $\mathsf{Win}_p^{\Omega} \cup \mathsf{Win}_{\overline{p}}^{\overline{\Omega}} = V$. It is known that every game with Borel objectives is determined [17].

Qualitative objectives

Let $G = (V_1, V_2, E)$ be an *unweighted* game structure. Given a set $U \subseteq V$, classical qualitative objectives are the following ones. A *reachability* objective $\mathsf{Reach}(U)$ asks to visit a vertex of U at least once. A *safety* objective $\mathsf{Safe}(U)$ asks to visit no vertex of $V \setminus U$, that is, to avoid $V \setminus U$. A *Büchi* objective $\mathsf{Buchi}(U)$ asks to visit a vertex of U infinitely often. A *co-Büchi* objective $\mathsf{CoBuchi}(U)$ asks to visit no vertex of $V \setminus U$ infinitely often. Let $U_1, \dots, U_i$ be a family of subsets of V. A *generalized reachability* objective $\mathsf{GenReach}(U_1, \dots, U_i) = \cap_{k=1}^{i} \mathsf{Reach}(U_k)$ asks to visit a vertex of U_k at least once, for each $k \in \{1, \dots, i\}$. A *generalized Büchi* objective $\mathsf{GenBuchi}(U_1, \dots, U_i) = \cap_{k=1}^{i} \mathsf{Buchi}(U_k)$ asks to visit a vertex of U_k infinitely often, for each $k \in \{1, \dots, i\}$. All these objectives can be mixed by taking their intersection.

A game with an objective Ω is just called Ω *game*. As all the previous objectives Ω are ω-regular and thus Borel, the corresponding games (G, Ω) are determined.

Quantitative objectives

For a *1-weighted* game structure $G = (V_1, V_2, E, w)$ (with dimension $n = 1$), we now introduce quantitative objectives defined by some classical *measure functions* $f : \mathsf{Plays}(G) \to \mathbb{Q}$. Such a function f associates a rational number to each play $\rho = \rho_1 \rho_2 \dots$ according to the weights $w(\rho_k, \rho_{k+1})$, $k \geq 0$, and can be one among the next functions. The Inf measure $\mathsf{Inf}(\rho) = \inf_{k \geq 0}(w(\rho_k, \rho_{k+1}))$ (resp. the Sup measure $\mathsf{Sup}(\rho) = \sup_{k \geq 0}(w(\rho_k, \rho_{k+1}))$) defines the minimum (resp. maximum) weight seen along the play. The LimInf measure $\mathsf{LimInf}(\rho) = \liminf_{k \to \infty}(w(\rho_k, \rho_{k+1}))$ (resp. the LimSup measure $\mathsf{LimSup}(\rho) = \limsup_{k \to \infty}(w(\rho_k, \rho_{k+1}))$) defines the mininum (resp. maximum) weight seen infinitely often along the play.

Given such a measure function $f \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$, a bound $\nu \in \mathbb{Q}$, and a relation $\sim \in \{>, \geq, <, \leq\}$, we define the objective $\Omega = f(\sim \nu)$ such that

$$f(\sim \nu) \quad = \quad \{\rho \in \mathsf{Plays}(G) \mid f(\rho) \sim \nu\}. \tag{1}$$

We are also interested in the next two measure functions defined on histories instead of plays. Let $\rho = \rho_0 \dots \rho_i \in \mathsf{Hist}(G)$. The *total-payoff* ($\mathsf{TP}$) measure $\mathsf{TP}(\rho) = \Sigma_{k=0}^{i-1} w(\rho_k, \rho_{k+1})$

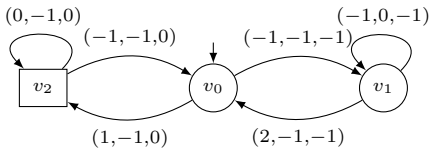


Figure 1 A multi-weighted two-player game.

defines the sum of the weights seen along ρ. The *mean-payoff* (MP) measure $\mathsf{MP}(\rho) = \frac{1}{i}\mathsf{TP}(\rho)$ defines the mean of the weights seen along ρ. The second measure can be extended to plays ρ as either $\underline{\mathsf{MP}}(\rho) = \liminf_{k\geq 0} \mathsf{MP}(\rho_{[0,k]})$ or $\overline{\mathsf{MP}}(\rho) = \limsup_{k\geq 0} \mathsf{MP}(\rho_{[0,k]})$. The MP measure on histories allows to define the *window mean-payoff objective*, a new ω-regular objective introduced in [6]: given a bound $\nu \in \mathbb{Q}$, a relation $\sim \in \{>, \geq, <, \leq\}$, and a window size $\lambda \in \mathbb{N}\setminus\{0\}$, the objective $\mathsf{WMP}(\lambda, \sim \nu)$[2] is equal to

$$\mathsf{WMP}(\lambda, \sim \nu) = \{\rho \in \mathsf{Plays}(G) \mid \forall k \geq 0, \exists l \in \{1, \ldots, \lambda\}, \mathsf{MP}(\rho_{[k,k+l]}) \sim \nu\}. \tag{2}$$

The window mean-payoff objective asks that the average weight becomes $\sim \nu$ inside a local bounded window for all positions of this window sliding along the play, instead of the classical mean-payoff objective asking that the long run-average $\underline{\mathsf{MP}}(\rho)$ (resp. $\overline{\mathsf{MP}}(\rho)$) is $\sim \nu$. This objective is a strengthening of the mean-payoff objective.

Given a n-weighted game structure G (with $n \geq 1$), we can mix objectives of (1) and (2) by fixing one such objective Ω_m for each dimension m, and taking the intersection $\cap_{m=1}^{n}\Omega_m$. More precisely, given a vector $(\sim_1 \nu_1, \ldots, \sim_n \nu_n)$, each objective Ω_m uses a measure function based on the weight function w_m; Ω_m is either of the form $f(\sim_m \nu_m)$ with $f \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$, or of the form $\mathsf{WMP}(\lambda, \sim_m \nu_m)$ for some window size λ (this size can change with m).

As for qualitative objectives, we use the shortcut Ω *game* for a game with quantitative objective Ω. For instance an $\mathsf{Inf}(\sim \nu)$ *game* is a 1-weighted game with objective $\mathsf{Inf}(\sim \nu)$, a $\mathsf{LimSup}(\sim_1 \nu_1)\cap\mathsf{WMP}(\lambda, \sim_2 \nu_2)\cap\mathsf{Inf}(\sim_3 \nu_3)$ *game* is a 3-weighted game with the intersection of a $\mathsf{LimSup}(\sim_1 \nu_1)$ objective on the first dimension, a $\mathsf{WMP}(\lambda, \sim_2 \nu_2)$ objective on the second one, and an $\mathsf{Inf}(\sim_3 \nu_3)$ objective on the third one. We sometimes abusively say that $\Omega = \cap_{m=1}^{n}\Omega_m$ with $\Omega_m \in \{\mathsf{WMP}, \mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$ without mentioning the used relations, bounds and window sizes. It is implicitly supposed that Ω_m deals with the m^{th} component of the weight function.

3 Problem

In this paper, we want to study the following problem.

▶ **Problem 1.** Let (G, Ω) be a multi-weighted game with dimension $n \geq 1$ and $\Omega = \cap_{m=1}^{n}\Omega_m$ such that each $\Omega_m \in \{\mathsf{WMP}, \mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$. Can we compute the winning sets Win_1^{Ω} and $\mathsf{Win}_2^{\overline{\Omega}}$? If yes what is the complexity of computing these sets and how (memoryless, finite-memory, general) are the winning strategies of both players? Given such a game (G, Ω) and an initial vertex v_0, the *synthesis problem* asks to decide whether player 1 has a winning strategy for Ω from v_0 and to build such a strategy when it exists.

[2] This objective is called "direct fixed window mean-payoff" in [6] among several other variants.

▶ **Remark 1.** (i) In this problem, we can assume that the bounds used in the vector $(\sim_1 \nu_1, \ldots, \sim_n \nu_n)$ are such that $(\nu_1, \ldots, \nu_n) = (0, \ldots, 0)$. Indeed, suppose that $\nu_m = \frac{a}{b}$ with $a \in \mathbb{Z}$ and $b \in \mathbb{N}\setminus\{0\}$, then replace the m^{th} component w_m of the weight function w by $b \cdot w_m - a$. (ii) Moreover notice that if $\nu_m = 0$ and $\Omega_m = \mathsf{WMP}$, then $\mathsf{MP}(\rho_{[k,k+l]})$ can be replaced by $\mathsf{TP}(\rho_{[k,k+l]})$ in (2). (iii) Finally, the vector $(\sim_1 0, \ldots, \sim_n 0)$ can be supposed to be equal to $(\geq 0, \ldots, \geq 0)$. Indeed strict inequality > 0 (resp. < 0) can be replaced by inequality ≥ 1 (resp. ≤ -1), and inequality ≤ 0 can be replaced by ≥ 0 by replacing the weight function by its negation and the measure Inf (resp. Sup, LimInf, LimSup, TP) by Sup (resp. Inf, LimSup, LimInf, TP).

From now on, we only work with vectors $(\geq 0, \ldots, \geq 0)$ and we no longer mention symbol $\geq$. Hence, as an example, $\mathsf{Inf}(\geq 0)$ and $\mathsf{WMP}(2, \geq 0)$ are replaced by $\mathsf{Inf}(0)$ and $\mathsf{WMP}(2, 0)$; and when the context is clear, we only mention Inf and WMP.

The games (G, Ω) of Problem 1 are determined since all the objectives Ω are ω-regular. Let us give an example where we mix quantitative objectives.

▶ **Example 1.** Consider the 3-weighted game structure of Figure 1. In all examples in this paper, we assume that circle (resp. square) vertices belong to player 1 (resp. player 2). Let $\Omega = \mathsf{WMP}(3, 0) \cap \mathsf{Sup}(0) \cap \mathsf{LimSup}(0)$ be the objective of player 1. We recall that by definition of Ω, we look at the WMP (resp. Sup, LimSup) objective on the first (resp. second, third) dimension. Let us show that v_0 is a winning vertex for player 1. Let σ_1 be the following strategy of player 1 from v_0: go to v_1, take the self loop once, go back to v_0 and then always go to v_2. Notice that $\rho \in v_0v_1v_1v_0\{v_2, v_0\}^\omega$. As player 1 forces ρ to begin with $v_0v_1v_1$, he ensures that $\mathsf{Sup}(\rho) \geq 0$ on the second component. Moreover as ρ visits infinitely often edge (v_2, v_2) or (v_2, v_0), player 1 also ensures to have $\mathsf{LimSup}(\rho) \geq 0$ on the third component. Finally, we have to check that $\rho \in \mathsf{WMP}(3, 0)$ with respect to the first component, that is (by Remark 1), for all k, there exists $l \in \{1, 2, 3\}$ such that $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$. For $k = 0$ (resp. $k = 1$, $k = 2$, $k = 3$) and $l = 3$ (resp. $l = 2$, $l = 1$, $l = 1$), we have $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$. Now, from position $k = 4$, the sum of weights is non-negative in at most 2 steps. Indeed, either player 2 takes the self loop (v_2, v_2) or he goes to v_0 where player 1 goes back to v_2. Therefore, for each $k \geq 4$, either $\rho_k = v_0$ and $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$ with $l = 1$, or $\rho_k = v_2$ and $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$ with $l = 1$ if $\rho_{k+1} = v_2$, and with $l = 2$ otherwise. It follows that $v_0 \in \mathsf{Win}_1^\Omega$. The strategy σ_1 needs memory: indeed, player 1 needs to remember if he has already visited the edge (v_1, v_1) as this is the only edge visiting a non-negative weight for the Sup objective. Finally, one can show that $\mathsf{Win}_1^\Omega = \{v_0, v_1\}$ and $\mathsf{Win}_2^{\overline{\Omega}} = \{v_2\}$.

▶ **Remark 2.** In Problem 1, the vector $(\sim_1 \nu_1, \ldots, \sim_n \nu_n)$ can be assumed equal to $(\geq 0, \ldots, \geq 0)$ by Remark 1. Thus an Inf (resp. Sup, LimInf, LimSup) objective is nothing else than a safety (resp. reachability, co-Büchi, Büchi) objective, and conversely. More precisely, every game $(G, \Omega = \cap_{m=1}^n \Omega_m)$ with each $\Omega_m \in \{\mathsf{WMP}, \mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$ can be polynomially reduced to a game $(G', \Omega' = \cap_{m=1}^n \Omega'_m)$ such that each Ω'_m belongs to $\{\mathsf{WMP}, \mathsf{Safe}, \mathsf{Reach}, \mathsf{CoBuchi}, \mathsf{Buchi}\}$, and with $|V| + |E|$ vertices and $2 \cdot |E|$ edges. This reduction is obtained by splitting each edge into two consecutive edges and decorating accordingly the new intermediate vertex to transfer the objectives. The WMP objectives in G are now WMP objectives in G' with a doubled window size. There also exists a polynomial reduction in the other direction, but without WMP objectives, and keeping the same game structure $G' = G$. These two game reductions will be used throughout this paper.

Table 2 Overview of some known results for qualitative objectives (i is the number of objectives in the intersection of reachability/Büchi objectives).

Objective	Complexity class	Algorithmic complexity	Player 1 memory	Player 2 memory
Reach/Safe [2, 13, 15]	P-complete	$O(\|V\| + \|E\|)$	memoryless	memoryless
Buchi/CoBuchi [7, 11, 15]	P-complete	$O(\|V\|^2)$	memoryless	memoryless
GenReach [12]	PSPACE-complete	$O(2^i \cdot (\|V\| + \|E\|))$	exponential memory	exponential memory
GenReach (i fixed) [12]	P-complete	$O(2^i \cdot (\|V\| + \|E\|))$	polynomial memory	polynomial memory
GenBuchi [10]	P-complete	$O(i \cdot \|V\| \cdot \|E\|)$	polynomial memory	memoryless
GenBuchi ∩ CoBuchi	P-complete	$O(i^2 \cdot \|V\| \cdot \|E\|)$	polynomial memory	memoryless

Some well-know properties

Table 2 gathers several well-known results about qualitative objectives in an unweighted game structure and Theorem 2 states the known results about the WMP objective. In this paper, the complexity of the algorithms is expressed in terms of the size $|V|$ and $|E|$ of the game structure G, the maximum weight W (in absolute value) and the dimension n of the weight function when G is weighted, the number i of objectives in an intersection of objectives[3], and the window size λ.

▶ **Theorem 2.** [6][4] *Let (G, Ω) be an n-weighted game such that $\Omega = \cap_{m=1}^{n} \Omega_m$ with $\Omega_m =$ WMP. The synthesis problem is* EXPTIME*-complete (with an algorithm in $O(\lambda^{4n} \cdot |V|^2 \cdot W^{2n})$ time), exponential memory strategies are sufficient and necessary for both players. This problem is already* EXPTIME*-hard when $n = 2$.*

When $n = 1$, the synthesis problem is decidable in $O(\lambda \cdot |V| \cdot (|V| + |E|) \cdot \lceil \log_2(\lambda \cdot W) \rceil)$ time, both players require finite-memory strategies, and memory in $O(\lambda^2 \cdot W)$ (resp. in $O(\lambda^2 \cdot W \cdot |V|)$) is sufficient for player 1 (resp. player 2). Moreover, if λ is polynomial in the size of the game, then the synthesis problem is P*-complete.*

Solution to Problem 1 and extensions

We here give the solution to Problem 1. We show that the synthesis problem is EXPTIME-complete and that exponential memory strategies are necessary and sufficient for both players:

▶ **Theorem 3.** *Let (G, Ω) be an n-weighted game such that $\Omega = \cap_{m=1}^{n} \Omega_m$ with $\Omega_m \in \{$*WMP*,* Inf*,* Sup*,* LimInf*,* LimSup*$\}$. The synthesis problem is* EXPTIME*-complete (with an algorithm in $O(|V| \cdot |E| \cdot (\lambda^2 \cdot W)^{2n})$ time), and exponential memory strategies are necessary and sufficient for both players.*

Proof. First, we show that the synthesis problem is in EXPTIME. To this end, we use an exponential reduction from [6] dealing with WMP objectives, that we adapt in a way to also deal with the Inf, Sup, LimInf and LimSup objectives. In this reduction, a game (G', Ω') with $\Omega' = \cap_{m=1}^{n} \Omega'_m$, is constructed such that for all m, $\Omega'_m \in \{$Buchi, CoBuchi$\}$, and the size of the game is exponential with $O(|V| \cdot (\lambda^2 \cdot W)^n)$ vertices and $O(|E| \cdot (\lambda^2 \cdot W)^n)$ edges. Recall that the intersection of co-Büchi objectives is a co-Büchi objective. It follows that G' is a generalized Büchi ∩ co-Büchi game. Then, by Table 2 (last item), we can compute the winning sets of both players in G in time $O(n^2 \cdot |V'| \cdot |E'|) = O(|V| \cdot |E| \cdot (\lambda^2 \cdot W)^{2n})$. Moreover,

[3] Notice that $i = n$ for the objectives considered in Problem 1.
[4] When $n = 1$, the time complexity and the memory requirements have been here correctly stated.

since polynomial memory strategies are sufficient in G', exponential memory strategies are sufficient in G. Finally, the EXPTIME-hardness and the necessity of exponential memory follow from Theorem 2. ◀

The previous theorem and proof can be generalized to Boolean combinations in DNF and CNF forms (instead of intersections) of objectives in {WMP, Inf, Sup, LimInf, LimSup}. One can assume w.l.o.g. that the dimension of the game is equal to the number of objectives that appear in the Boolean combination (by making copies of components of the weight function). The following theorem sums up the latter result, and then we discuss the general case of Boolean combinations.

▶ **Theorem 4.** *Let (G, Ω) be an nd-weighted game such that $\Omega = \cup_{k=1}^{d} \cap_{m=1}^{n} \Omega_{k,m}$ with $\Omega_{k,m} \in$ {WMP, Inf, Sup, LimInf, LimSup}. The synthesis problem is* EXPTIME*-complete (with an algorithm in $O(n^{d(d+2)} \cdot |V|^{d+1} \cdot |E| \cdot (\lambda^2 \cdot W)^{nd(d+2)})$ time), and exponential memory strategies are sufficient and necessary for both players. The same result holds when $\Omega = \cap_{k=1}^{d} \cup_{m=1}^{n} \Omega_{k,m}$.*

▶ **Remark 3.** (*i*) Whereas the synthesis problem for Boolean combinations of $\underline{\text{MP}}$ and $\overline{\text{MP}}$ is undecidable [19], it is here decidable for Boolean combinations of objectives in {WMP, Inf, Sup, LimInf, LimSup}. Furthermore, one can show that this problem is in EXPSPACE using a result of [1]. (*ii*) Note that when the number of dimensions is fixed (n in Theorem 3, nd in Theorem 4), the synthesis problem is still EXPTIME-hard by Theorem 2.

4 Efficient fragment with one WMP objective

In the previous section, we considered games (G, Ω) with $\Omega = \cap_{m=1}^{n} \Omega_m$ being any intersection of objectives in {WMP, Inf, Sup, LimInf, LimSup}. We here focus on a particular class of games in a way to achieve a lower complexity for the synthesis problem. We do not consider the case where at least two Ω_m are WMP objectives since the synthesis problem is already EXPTIME-hard in this case (by Theorem 2). We thus focus on the intersections of exactly one[5] objective WMP and any number of objectives of one kind in {Inf, Sup, LimInf, LimSup}. Note that this number must be fixed in the case of objectives Sup to avoid PSPACE-hardness in this case (see Table 2, third row). For the considered fragment, we show that the synthesis problem is P-complete for polynomial windows. The latter assumption is reasonable in practical applications where one expects a positive mean-payoff in any "short" window sliding along the play.

▶ **Theorem 5.** *Let (G, Ω) be an n-weighted game with objective $\Omega = \Omega_1 \cap \Gamma$ for player 1 such that $\Omega_1 =$ WMP and $\Gamma = \cap_{m=2}^{n} \Omega_m$ such that $\forall m\, \Omega_m =$ Inf (resp. $\forall m\, \Omega_m =$ LimInf, $\forall m\, \Omega_m =$ LimSup, $\{\forall m\, \Omega_m =$ Sup and n is fixed$\}$). Then the synthesis problem is decidable (in time polynomial in the size of the game, λ and $\lceil \log(W) \rceil$). In general, both players require finite-memory strategies, and pseudo-polynomial memory is sufficient for both players. Moreover, when λ is polynomial in the size of the game then the synthesis problem is* P*-complete.*

To prove this theorem, we use the first reduction of Remark 2 to obtain a game $(G', \Omega'_1 \cap \Gamma')$ (with $\Gamma' = \cap_{m=2}^{n} \Omega'_m$) such that $\Omega'_1 =$ WMP and $\forall m\, \Omega'_m \in$ {Reach, Safe, Buchi, CoBuchi}. Recall that the intersection of safety (resp. co-Büchi) objectives is a safety (resp. co-Büchi)

[5] The case with no WMP objective will be treated in the next section.

Table 3 Overview of the fragment (i is the number of objectives in the intersection of Reach/Büchi objectives).

Objective	Algorithmic complexity	Player 1 memory	Player 2 memory
WMP ∩ Safe	$O(\lambda \cdot \lvert V\rvert \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot W)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert)$
WMP ∩ Reach	$O(\lambda \cdot \lvert V\rvert \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert)$
WMP ∩ GenReach (i fixed)	$O(\lambda \cdot 2^{2^i} \cdot \lvert V\rvert \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot 2^i \cdot W \cdot \lvert V\rvert)$	$O(\lambda^2 \cdot 2^i \cdot W \cdot \lvert V\rvert)$
WMP ∩ Buchi	$O(\lambda \cdot \lvert V\rvert^2 \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert^2)$
WMP ∩ GenBuchi	$O(\lambda \cdot i^3 \cdot \lvert V\rvert^2 \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot W \cdot i \cdot \lvert V\rvert)$	$O(\lambda^2 \cdot W \cdot i^2 \cdot \lvert V\rvert^2)$
WMP ∩ CoBuchi	$O(\lambda \cdot \lvert V\rvert^2 \cdot (\lvert V\rvert + \lvert E\rvert) \cdot \lceil \log(\lambda \cdot W)\rceil)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert^2)$	$O(\lambda^2 \cdot W \cdot \lvert V\rvert)$

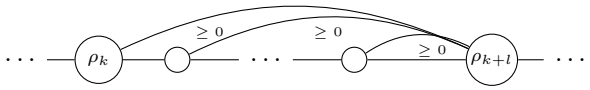


Figure 2 A λ-window at position k that is inductively-closed in $k+l$.

objective. We thus have to study WMP ∩ Safe (resp. WMP ∩ Reach, WMP ∩ GenReach (with n fixed), WMP ∩ Buchi, WMP ∩ GenBuchi, WMP ∩ CoBuchi) games. Table 3 gives an overview of the obtained properties; it indicates time polynomial in the size of the game, λ and $\lceil \log(W)\rceil$, and pseudo-polynomial[6] memories for both players. We then get Theorem 5 such that the algorithmic complexity and the memory requirements are those of Table 3 with $|V|$ replaced by $|V|+|E|$. Notice that finite memory is necessary for both players by Theorem 2. When λ is polynomial in size of the game, the complexity becomes polynomial, and the synthesis problem is P-hard (see Table 2).

Apart the objective $\Omega = \mathsf{WMP}(\lambda, 0) \cap \mathsf{Safe}(U)$, solving the games of Table 3 is difficult and requires to develop some new tools generalizing the classical concept of p-attractor while dealing with good windows (the p-attractor of a set U is the set of vertices from which player p has a strategy to reach U against any strategy of player $\overline{p}$). To this end, let us introduce some properties of windows.

Properties of windows

Let us focus on the $\mathsf{WMP}(\lambda, 0)$ objective (with TP in (2)) and introduce some terminology. Let $\rho = \rho_0\rho_1\dots$ be a play. A *λ-window at position k* is a window of size λ placed along ρ from k to $k+\lambda$. If there exists $l \in \{1, \dots, \lambda\}$ such that $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$, such a λ-window at position k is called *good* or *closed in $k+l$* (to specify index l), otherwise it is called *bad*. Moreover if l is the smallest index such that $\mathsf{TP}(\rho_{[k,k+l]}) \geq 0$, we say it is *first-closed in $k+l$*. An interesting property is the next one: a λ-window at position k is *inductively-closed in $k+l$* if it is closed in $k+l$ and for all $k' \in \{k+1, \dots, k+l-1\}$, the λ-window at position k' is also closed in $k+l$ (see Figure 2). A λ-window at position k that is first-closed in $k+l$ is inductively-closed in $k+l$. The next lemma will be useful:

▶ **Lemma 6.** *A play ρ is winning for $\mathsf{WMP}(\lambda, 0)$ iff there exists a sequence $(k_i)_{i\geq 0}$ with $k_0 = 0$ such that $\forall i, k_{i+1} - k_i \in \{1, \dots, \lambda\}$, and the λ-window at position k_i is inductively-closed in k_{i+1}.*

[6] Having pseudo-polynomial memories is not really a problem since the proofs show that the strategies can be efficiently encoded by programs using two counters, as in the case of Theorem 2.

Algorithm 1 ICWEnd

Require: 1-weighted game structure $G = (V_1, V_2, E, w)$, set $U \subseteq V$, window size $\lambda \in \mathbb{N}\setminus\{0\}$
Ensure: $\mathsf{Win}_1^{\mathsf{ICWEnd}^\lambda(U)}$

```
for all v ∈ V do
  if v ∈ U then
    C_0(v) ← 0
  else
    C_0(v) ← −∞
for all l ∈ {1, ..., λ} do
  for all v ∈ V_1 do
    C_l(v) ← max_{(v,v')∈E} {w(v, v') ⊕ max{C_0(v'), C_{l−1}(v')}}
  for all v ∈ V_2 do
    C_l(v) ← min_{(v,v')∈E} {w(v, v') ⊕ max{C_0(v'), C_{l−1}(v')}}
return {v ∈ V | C_λ(v) ≥ 0}
```

When such a sequence $(k_i)_{i\geq 0}$ exists for a play ρ, it is called a λ-*good decomposition* of ρ. We extend this notion to histories $\rho = \rho_0\rho_1\dots\rho_k$ as follows. A finite sequence $(k_i)_{i=0}^{j}$ is a λ-good decomposition of ρ if $k_0 = 0$, $k_j = k$, for each $i \in \{0,\dots j-1\}$ we have $k_{i+1} - k_i \in \{1,\dots,\lambda\}$, and the λ-window at position k_i is inductively-closed in k_{i+1}. From now on, a λ-window is simply called a window.

Two new objectives

Let us now introduce our new tools. They are based on the following new objectives.

▶ **Definition 7.** Let $G = (V_1, V_2, E, w)$ be a 1-weighted game structure, $U \subseteq V$ be a set of vertices, and $\lambda \in \mathbb{N} \setminus \{0\}$ be a window size. We consider the next two sets of plays:

- $\mathsf{ICWEnd}^\lambda(U) = \{\ \rho \in \mathsf{Plays}(G) \mid \exists l \in \{1,\dots,\lambda\}, \rho_l \in U$, and the window at position 0 is inductively-closed in $l\ \}$,
- $\mathsf{GDEnd}^\lambda(U) = \{\ \rho \in \mathsf{Plays}(G) \mid \exists l \geq 0, \rho_l \in U$ and $\rho_{[0,l]}$, has a λ-good decomposition $\}$.

Notice that the plays of $\mathsf{ICWEnd}^\lambda(U)$ are particular plays of $\mathsf{GDEnd}^\lambda(U)$. Hence we have $\mathsf{ICWEnd}^\lambda(U) \subseteq \mathsf{GDEnd}^\lambda(U)$. We propose two algorithms, Algorithms 1 and 2, one for computing the winning set of player 1 for the objective $\mathsf{ICWEnd}^\lambda(U)$, and the other for the objective $\mathsf{GDEnd}^\lambda(U)$.

Algorithm 1 uses the operator $\oplus$ defined as follows. Let $a, b \in \mathbb{Z}\cup\{-\infty\}$, then $a\oplus b = a+b$ if $a + b \geq 0$, and $-\infty$ otherwise. With this definition, either $a \oplus b \geq 0$ or $a \oplus b = -\infty$. Algorithm 1 intuitively works as follows. Given a vertex v and a number i of steps, the value $C_i(v)$ is computed iteratively (from $C_{i-1}(v)$) and represents the best total payoff that player 1 can ensure in *at most i steps* while closing the window from v in a vertex of U. Value $-\infty$ indicates that the window starting in v cannot be inductively-closed in U. The winning set of player 1 is thus the set of vertices v for which $C_\lambda(v) \geq 0$. This algorithm is inspired from Algorithm GoodWin in [6] computing $\mathsf{Win}_1^{\mathsf{ICWEnd}^\lambda(U)}$ with $U = V$.[7]

[7] We have detected a flaw in this algorithm that has been corrected in Algorithm 1. The algorithm in [6] wrongly computes the set of vertices from which player 1 can force to close the window in exactly l steps (instead of at most l steps) for some $l \in \{1,\dots,\lambda\}$.

Algorithm 2 GDEnd

Require: 1-weighted game structure $G = (V_1, V_2, E, w)$, subset $U \subseteq V$, window size $\lambda \in \mathbb{N} \setminus \{0\}$

Ensure: $\mathsf{Win}_1^{\mathsf{GDEnd}^\lambda(U)}$

1: $k \leftarrow 0$
2: $X_0 \leftarrow U$
3: **repeat**
4: $\quad X_{k+1} \leftarrow X_k \cup \mathsf{ICWEnd}(G, X_k, \lambda)$
5: $\quad k \leftarrow k + 1$
6: **until** $X_k = X_{k-1}$
7: **return** X_k

▶ **Lemma 8.** *Let G be a 1-weighted game structure, U be a subset of V, and $\lambda \in \mathbb{N} \setminus \{0\}$ be a window size. Then Algorithm* ICWEnd *computes the set $\mathsf{Win}_1^{\mathsf{ICWEnd}^\lambda(U)}$ in $O(\lambda \cdot (|V|+|E|) \cdot \lceil \log_2(\lambda \cdot W) \rceil)$ time, and finite-memory strategies with memory linear in λ are sufficient for both players.*

We now turn to Algorithm 2 for computing the winning set of player 1 when the objective is $\mathsf{GDEnd}^\lambda(U)$. It shares similarities with the classical algorithm computing the p-attractor of U while requiring to use previous Algorithm $\mathsf{ICWEnd}^\lambda(U)$.

▶ **Lemma 9.** *Let G be a 1-weighted game structure, U be a subset of V, and $\lambda \in \mathbb{N} \setminus \{0\}$ be a window size. Then Algorithm* GDEnd *computes the set $\mathsf{Win}_1^{\mathsf{GDEnd}^\lambda(U)}$ of winning vertices of player 1 for the objective $\mathsf{GDEnd}^\lambda(U)$ in $O(\lambda \cdot |V| \cdot (|V| + |E|) \cdot \lceil \log_2(\lambda \cdot W) \rceil)$ time, and finite-memory strategies with memory in $O(\lambda^2 \cdot W \cdot |V|)$ (resp. in $O(\lambda^2 \cdot W)$) are sufficient for player 1 (resp. player 2).*

The proof of Lemma 9 works as follows. Let $X^* = \cup_{k \geq 0} X_k$ be the set computed by Algorithm GDEnd. We explain why $X^* = \mathsf{Win}_1^{\mathsf{GDEnd}^\lambda(U)}$. From $v_0 \in X^*$, player 1 plays a winning strategy for the objective $\mathsf{ICWEnd}^\lambda(X_k)$, and as soon as this objective is realized he repeats such a strategy for decreasing values of k until reaching U. This strategy is winning for the objective $\mathsf{GDEnd}^\lambda(U)$ and it is finite-memory. Let us now consider $v_0 \in V \setminus X^*$. As player 2 has a winning strategy σ_2^* for the objective $\overline{\mathsf{ICWEnd}^\lambda(X^*)}$, then for $\rho = \mathsf{Out}(v_0, \sigma_1, \sigma_2^*)$ with σ_1 being any strategy of player 1, if the window at position 0 is inductively-closed in ρ_l for $l \in \{1, \ldots, \lambda\}$, then necessarily $\rho_l \in V \setminus X^*$. Therefore we propose the following strategy of player 2 from v_0: (1) If the current vertex v belongs to $V \setminus X^*$, play the winning strategy σ_2^*. (2) As soon as the window starting from v is first-closed in v' in l steps with $l \in \{1, \ldots, \lambda\}$, as $v' \in V \setminus X^*$, go back to step (1). (3) As soon as the window starting from v is bad, play whatever. This strategy is finite-memory and it is winning for player 2 since it ensures that the play cannot have a prefix with a λ-good decomposition ending in U.

▶ **Example 10.** Consider the game (G, Ω) depicted on Figure 3, where $\Omega = \mathsf{GDEnd}^2(U)$ with $U = \{v_1, v_3, v_4\}$. Let us execute Algorithm 2: $X_0 = U$, $X_1 = \mathsf{Win}_1^{\mathsf{ICWEnd}^2(X_0)} = \{v_1, v_2, v_3, v_4\}$ and then, $X_2 = \mathsf{Win}_1^{\mathsf{ICWEnd}^2(X_1)} = V$. Thus all vertices in G are winning for player 1 for the objective Ω. A winning strategy for player 1 consists in looping once in v_2 and then going to v_3. Indeed for any strategy of player 2, the outcome is either $v_0 v_1^\omega$ or $v_0 v_2 v_2 v_3 v_4^\omega$, and both outcomes admit a prefix which has a 2-good decomposition and ends with a vertex of U.

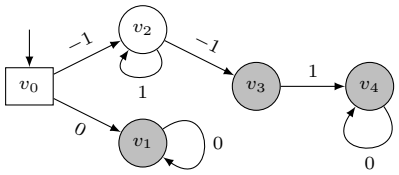


Figure 3 Objective $\mathsf{GDEnd}^2(U)$.

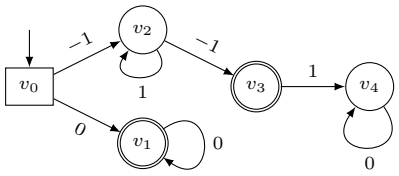


Figure 4 Objective $\mathsf{WMP}(2,0) \cap \mathsf{Reach}(U)$.

Objective WMP ∩ Reach

Now that we have introduced our new tools, let us come back to the games of Table 3, second row. Notice that the objectives $\mathsf{GDEnd}^\lambda(U)$ and $\mathsf{WMP}(\lambda,0) \cap \mathsf{Reach}(U)$ are close to each other: a play ρ belongs to $\mathsf{GDEnd}^\lambda(U)$ if it has a prefix which has a λ-good decomposition and ends with a vertex in U, while ρ belongs to $\mathsf{WMP}(\lambda,0) \cap \mathsf{Reach}(U)$ if it has a λ-good decomposition and one of its vertices belongs to U. Therefore solving games for the objective $\mathsf{WMP}(\lambda,0) \cap \mathsf{Reach}(U)$ requires to win for a modified objective $\mathsf{GDEnd}^\lambda(U)$ such that the λ-good decomposition visits U and ends in a winning vertex for the objective WMP. The following example illustrates this modified objective.

▶ **Example 11.** Consider the game (G,Ω) depicted on Figure 4, where $\Omega = \mathsf{WMP}(2,0) \cap \mathsf{Reach}(U)$ with $U = \{v_1, v_3\}$. To compute the winning set Win_1^Ω, we need to modify G in a new game structure G' where we add a bit to each vertex indicating whether U has been visited (bit equals 1) or not (bit equals 0). This game structure is the one of previous Figure 3, where the set $U' = \{v_1, v_3, v_4\}$ of gray nodes are those with bit 1. We already know that every vertex of G' is winning for objective $\mathsf{GDEnd}^2(U')$, and we note that they are also all winning for objective $\mathsf{WMP}(2,0)$. Therefore, in G', player 1 can ensure a finite 2-good decomposition ending in a vertex of U' from which he can ensure an infinite 2-good decomposition. Thanks to the added bit and coming back to G, we get that all vertices of G are winning for Ω.

Now, it is easy to solve games for the objective $\mathsf{WMP}(\lambda,0) \cap \mathsf{GenReach}(U_1,\dots,U_{i-1})$ when i is fixed (see Table 3, third row). From such a game, we construct a new game structure where we add i bits to each vertex: the j^{th} bit equals 1 if U_j has been visited, and 0 otherwise. Note that i has to be fixed to get a polynomial construction, otherwise the problem is already PSPACE-hard by Table 2 (third row). Finally, we solve the new game where the objective is $\mathsf{WMP}(\lambda,0) \cap \mathsf{GenReach}(U')$ where U' is the set of vertices with all bits equal to 1.

Objective WMP ∩ Buchi

Solving games for the objective $\mathsf{WMP} \cap \mathsf{Buchi}$ needs an algorithm that repeatedly uses the algorithm for the objective $\mathsf{WMP} \cap \mathsf{Reach}$.

▶ **Proposition 1.** *Let $G = (V_1, V_2, E, w)$ be a 1-weighted game structure, and Ω be the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Buchi}(U)$. Then Win_1^{Ω} can be computed in $O(\lambda \cdot |V|^2 \cdot (|V| + |E|) \cdot \lceil \log_2(\lambda \cdot W) \rceil)$ time and finite-memory strategies with memory in $O(\lambda^2 \cdot W \cdot |V|)$ (resp. in $O(\lambda^2 \cdot W \cdot |V|^2)$) are sufficient for player 1 (resp. player 2).*

The algorithm for the objective $\mathsf{WMP} \cap \mathsf{Buchi}$ works as follows: *(1)* Compute the winning set X for player 1 for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Reach}(U)$ in G; *(2)* compute the 2-attractor Y of the set $V \setminus X$ in G; *(3)* repeat step (1) in the game $G[V \setminus Y]$ and step (2) in the game G, until X is empty or X is the set of all vertices in $G[V \setminus Y]$. The final set X is the winning set of player 1 for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Buchi}(U)$.

We first explain why $X \subseteq \mathsf{Win}_1^{\mathsf{WMP}(\lambda,0) \cap \mathsf{Buchi}(U)}$. Notice that this algorithm exactly computes the set X of vertices such that player 1 wins for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Reach}(U)$ while staying in X (player 2 cannot leave this set). Therefore, from $v_0 \in X$, player 1 plays a winning strategy for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Reach}(U)$, and as soon as a vertex of U has been visited and the current history has a λ-good decomposition ending in a vertex $v \in X$, he repeats such a strategy, ad infinitum. This strategy is winning for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Buchi}(U)$ and it is finite-memory. Now, we show that $\mathsf{Win}_1^{\mathsf{WMP}(\lambda,0) \cap \mathsf{Buchi}(U)} \subseteq X$. If player 1 can win for the objective $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Buchi}(U)$, then a winning strategy ensures that he only visits vertices of $\mathsf{Win}_1^{\mathsf{WMP}(\lambda,0) \cap \mathsf{Reach}(U)}$. Then, player 1 has a strategy to win the $\mathsf{WMP}(\lambda, 0) \cap \mathsf{Reach}(U)$ objective using only vertices from which this property is ensured, which shows that $v_0 \in X$.

Concerning the two last rows of Table 3, one can derive an algorithm for the objective $\mathsf{WMP} \cap \mathsf{GenBuchi}$ from the algorithm for the objective $\mathsf{WMP} \cap \mathsf{Buchi}$; the objective $\mathsf{WMP} \cap \mathsf{CoBuchi}$ is the most difficult to solve and requires elaborated arguments to manage correctly two nested fixpoints together with the good windows.

5 Intersection of objectives in {Inf,Sup,LimInf,LimSup}

The aim of this section is to provide a refinement of Theorem 3 for games $(G, \Omega = \cap_{m=1}^{n} \Omega_m)$ when no objective Ω_m is a WMP objective. In this case, we get the better complexity of PSPACE-completeness (instead of EXPTIME-completeness) for the synthesis problem; nevertheless the two players still need exponential memory strategies to win (Theorem 12). We also study with precision (in Table 4) the complexity and the memory requirements in terms of the objectives of $\{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$ that appear in the intersection $\Omega = \cap_{m=1}^{n} \Omega_m$. When there is at most one Sup, we get a polynomial fragment and in certain cases, players can play memoryless. Notice that the membership to PSPACE in Theorem 12 could have been obtained from one result proved in [1] (in this paper, the objective is defined by an LTL formula, and it is proved that deciding the winner is in PSPACE for Boolean combinations of formulas of the form "eventually p" and "infinitely often p"). We here propose a simple proof adapted to our context, that allows an easy study of the winning strategies as well as the identification of polynomial fragments.

▶ **Theorem 12.** *Let (G, Ω) be an n-weighted game such that $\Omega = \cap_{m=1}^{n} \Omega_m$ with $\Omega_m \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$. The synthesis problem is PSPACE-complete (with an algorithm in $O(2^n \cdot (|V| + |E|))$ time) and exponential memory strategies are necessary and sufficient for both players.*

In Table 4, we recall Theorem 12 (first row) and exhibit several polynomial refinements (next rows). As shown by Example 13, these additional results are optimal with respect to the required memory (no/finite memory) for the winning strategies.

Table 4 Overview of properties for the intersection of objectives in {Inf, Sup, LimInf, LimSup}.

Inf	Sup	LimInf	LimSup	Complexity class	Algorithmic complexity	Player 1 memory	Player 2 memory
any	any	any	any	PSPACE-complete	$O(2^n \cdot (\|V\| + \|E\|))$	exponential memory	exponential memory
any	≤ 1	any	any	P-complete	$O(n^2 \cdot (\|V\| + \|E\|) \cdot \|E\|)$	polynomial memory	memoryless
any	0	any	≤ 1	P-complete	$O((\|V\| + \|E\|) \cdot \|E\|)$	memoryless	memoryless
any	1	0	0	P-complete	$O(\|V\| + \|E\|)$	memoryless	memoryless

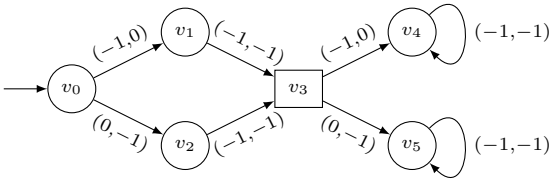


Figure 5 Example where player 2 needs memory.

▶ **Example 13.** First, we come back to the game structure G depicted on Figure 1, where we only keep the second and the third dimensions. Assume $\Omega = \mathsf{Sup}(0) \cap \mathsf{LimSup}(0)$. Then, v_0 is winning for player 1 but memory is required to remember if player 1 has visited the edge (v_1, v_1). The same argument holds for $\Omega = \mathsf{Sup}(0) \cap \mathsf{LimInf}(0)$ and $\Omega = \mathsf{Sup}(0) \cap \mathsf{Sup}(0)$. This example with objective $\Omega = \mathsf{Sup}(0) \cap \mathsf{LimSup}(0)$ indicates that player 1 cannot win memoryless in a game as in the second row of Table 4. This example with objective $\Omega = \mathsf{Sup}(0) \cap \mathsf{LimSup}(0)$ (resp. $\Omega = \mathsf{Sup}(0) \cap \mathsf{LimInf}(0)$, $\Omega = \mathsf{Sup}(0) \cap \mathsf{Sup}(0)$) also shows that player 1 needs memory to win if $[1, 0, 0]$ (referring to the second, third and fourth columns of Table 4) in the last row of Table 4 is replaced by $[1, 0, 1]$ (resp. $[1, 1, 0]$, $[2, 0, 0]$). Now, assume that $v_2 \in V_1$, that is, G is a one-player game and let $\Omega = \mathsf{LimSup}(0) \cap \mathsf{LimSup}(0)$. Again, v_0 is winning but player 1 needs memory since he has to alternate between v_1 (and take the self loop) and v_2. This shows that in the third row of Table 4, if $[\leq 1]$ is replaced by $[2]$ then player 1 needs memory to win. Finally, consider the game depicted on Figure 5. Let $\Omega = \mathsf{Sup}(0) \cap \mathsf{Sup}(0)$. Vertex v_0 is losing for player 1 (i.e. winning for player 2), but player 2 needs memory since he has to know which edge player 1 took from v_0 to counter him by taking the edge with the same vector of weights from v_3. This shows that in the second row of Table 4, if $[\leq 1]$ is replaced by $[2]$ then player 2 needs memory.

▶ **Remark 4.** When n is fixed, the synthesis problem becomes P-complete for games (G, Ω) such that $\Omega = \cap_{m=1}^{n} \Omega_m$ with $\Omega_m \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$. The P-easyness follows from Theorem 12 and the P-hardness follows from Table 2 and the second reduction of Remark 2.

Acknowledgments. We would like to thank Mickael Randour for his availability and help.

References

1 R. Alur, S. La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *CONCUR 2003*, volume 2761 of *LNCS*, pages 127–141. Springer, 2003.

2 C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Database Syst.*, 5(3):241–259, 1980.

3 P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS 2008*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008.

4 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

5 K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 505–516, 2010.

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

7 K. Chatterjee and M. Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014.

8 K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014.

9 A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Toruńczyk. Energy and mean-payoff games with imperfect information. In *CSL 2010*, volume 6247 of *LNCS*, pages 260–274. Springer, 2010.

10 S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How much memory is needed to win infinite games? In *LICS 1997*, pages 99–110. IEEE Computer Society, 1997.

11 E. Allen Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS 1991*, pages 368–377. IEEE Computer Society, 1991.

12 N. Fijalkow and F. Horn. Les jeux d'accessibilité généralisée. *Technique et Science Informatiques*, 32(9-10):931–949, 2013.

13 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

14 P. Hunter, G. A. Pérez, and J.-F. Raskin. Looking at mean-payoff through foggy windows. In *ATVA 2015*, volume 9364 of *LNCS*, pages 429–445. Springer, 2015.

15 N. Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981.

16 M. Jurdzinski, R. Lazic, and S. Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In *ICALP 2015*, volume 9135 of *LNCS*, pages 260–272. Springer, 2015.

17 D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

18 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM 1989*, pages 179–190. ACM Press, 1989.

19 Y. Velner. Robust multidimensional mean-payoff games are undecidable. In *FoSSaCS 2015*, volume 9034 of *LNCS*, pages 312–327. Springer, 2015.

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

21 Y. Velner and A. Rabinovich. Church synthesis problem for noisy input. In *FOSSACS 2011*, volume 6604 of *LNCS*, pages 275–289. Springer, 2011.

22 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

Soundness in Negotiations*

Javier Esparza[1], Denis Kuperberg[2], Anca Muscholl[3], and Igor Walukiewicz[4]

1 **Technical University of Munich**
2 **Technical University of Munich**
3 **Technical University of Munich, IAS & CNRS†**
4 **University of Bordeaux, CNRS, LaBRI**

Abstract

Negotiations are a formalism for describing multiparty distributed cooperation. Alternatively, they can be seen as a model of concurrency with synchronized choice as communication primitive. Well-designed negotiations must be sound, meaning that, whatever its current state, the negotiation can still be completed. In a former paper, Esparza and Desel have shown that deciding soundness of a negotiation is PSPACE-complete, and in PTIME if the negotiation is deterministic. They have also provided an algorithm for an intermediate class of acyclic, non-deterministic negotiations, but left the complexity of the soundness problem open.

In the first part of this paper we study two further analysis problems for sound acyclic deterministic negotiations, called the race and the omission problem, and give polynomial algorithms. We use these results to provide the first polynomial algorithm for some analysis problems of workflow nets with data previously studied by Trcka, van der Aalst, and Sidorova.

In the second part we solve the open question of Esparza and Desel's paper. We show that soundness of acyclic, weakly non-deterministic negotiations is in PTIME, and that checking soundness is already NP-complete for slightly more general classes.

1998 ACM Subject Classification D.1.3 Concurrent Programming, D.3.2 Language Classifications, D.2.2 Design Tools and Techniques, F.2.0 Analysis of Algorithms and Problem Commplexity – General, H.4.1 Office Automation

Keywords and phrases Negotiations, workflows, soundness, verification, concurrency.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.12

1 Introduction

A multiparty atomic negotiation is an event in which several processes (agents) synchronize in order to select one out of a number of possible outcomes. In [3] Esparza and Desel introduced *negotiations*, a model of concurrency with multiparty atomic negotiation as interaction primitive. The model describes a workflow of "atomic" negotiations. After an atomic negotiation concludes with the selection of an outcome, the workflow determines the set of atomic negotiations each agent is ready to engage next.

The negotiation model has been studied in [3, 4, 5], and in [6] the results have been applied to the analysis of industrial business processes modeled as workflow Petri nets, a very successful formal backend for graphical notations like BPMN (Business Process Modeling Notation), EPC (Event-driven Process Chain), or UML Activity Diagrams (see e.g. [15, 14]).

* This work was partially supported by the DFG Project "Negotiations: A Model for Tractable Concurrency"
† On leave from the University of Bordeaux.

As shown in [1], deterministic negotiations are very closely related to free-choice workflow nets, a class that is expressive enough to model many business processes (for example, 70% of the almost 2000 workflow nets from the suite of industrial models studied in [16, 7, 6] are free-choice).

The most prominent analysis problem for the negotiation model is *soundness*. Loosely speaking, a negotiation is sound if for every reachable configuration there is an execution leading to proper termination of the negotiation. In [3] it is shown that the soundness problem is PSPACE-complete for non-deterministic negotiations and coNP-complete for acyclic non-deterministic negotiations[1]. For this reason, and in search of a tractable class, [3] introduces the class of *deterministic negotiations*. In deterministic negotiations all agents are deterministic, meaning that they are never ready to engage in more than one atomic negotiation per outcome (in the same way that in a deterministic automaton, for each action the automaton is only ready to move to one state). The main results of [3] are a polynomial time *reduction algorithm* for checking soundness of acyclic deterministic negotiations, and an extension of the algorithm to the more expressive class of acyclic, weakly deterministic[2] negotiations. The runtime of this second algorithm was however left open, as well as the more general question of determining the complexity of checking soundness for other classes of acyclic negotiations. In [4] the polynomial result for acyclic deterministic negotiations is extended to the cyclic case.

While unsound negotiations are clearly faulty, sound negotiations are not automatically correct, they must satisfy other properties. In the first contribution of this paper, we study two other analysis problems for *sound* acyclic deterministic negotiations: the *race problem* and the *omission problem*. The race problem is to determine if there is an execution in which two given atomic negotiations are concurrently enabled. The omission problem asks for given sets of atomic negotiations P and B if there exists a run that visits all elements of P and omits all of B. We show that for sound negotiations the race problem is polynomial, as well as the omission problem for P of bounded size. We then apply these polynomial algorithms to analysis problems for negotiations with global data studied in [13, 11] in the context of workflow Petri nets. In this model atomic negotiations can manipulate global variables, so classical analysis questions are raised, for instance whether every value written into a variable is guaranteed to be read, or whether a variable can be allocated and deallocated by two atomic negotiations taking place in parallel. While the algorithms of [13, 11] are exponential, our solutions for acyclic sound deterministic negotiations take polynomial time.

Our second contribution is the study of the complexity of soundness for classes beyond deterministic negotiations. We propose to analyze this problem through properties of the graph of a negotiation. The first indication of the usefulness of this approach is a short argument giving an NLOGSPACE algorithm for deciding soundness of acyclic deterministic negotiations. Next, we settle the question left open in [3], and prove that the soundness problem can be solved in polynomial time for acyclic, weakly non-deterministic negotiations, a class even more general than the one defined in [3]. We then show that if we leave out one of the two assumptions, acyclicity or weak non-determinism, then the problem becomes coNP-complete[3]. These results set a limit to the class of negotiations with a polynomial soundness problems, but also admit a positive interpretation. Indeed, if all processes are

[1] In [3] the notion of soundness has one more requirement, which makes the soundness problem for acyclic negotiations coNP-hard and in DP.

[2] The class considered [3] was called "weakly deterministic". In this paper we refer to it as "very weakly non-deterministic".

[3] We show that coNP-hardness holds even for a very mild relaxation of acyclicity.

allowed to be cyclic and non-deterministic, then the soundness problem is PSPACE-complete, while for the class above it belongs to coNP.

Related formalisms and related work. The connection between negotiations and Petri nets is studied in detail in [1]. Every negotiation can be transformed into an exponentially larger 1-safe workflow Petri net with an isomorphic reachability graph. Every deterministic negotiation is equivalent to a 1-safe workflow free-choice net with a linear blow-up. Conversely, every sound workflow free-choice net can be transformed into a sound deterministic negotiation with a linear blow-up. Recent papers on free-choice workflow Petri nets are [8, 6]. In [8] soundness is characterized in terms of anti-patterns, which can be used to explain why a given workflow net is unsound. Our work provides an anti-pattern characterization for acyclic weakly non-deterministic negotiations, which goes beyond the free-choice case. In [6] a polynomial reduction algorithm for free-choice workflow Petri nets is presented. Our results show that soundness is also polynomial for workflow Petri nets coming from acyclic weakly deterministic negotiations.

As a process-based concurrent model, negotiations can be compared with another well-studied model for distributed computation, namely Zielonka automata [17, 2, 10]. Such an automaton is a parallel composition of finite transition systems with synchronization on common actions. The important point is that a synchronization involves exchange of information between states of agents: the result of the synchronization depends on the states of all the components taking part in it. Zielonka automata have the same expressive power as arbitrary, possibly nondeterministic negotiations. Deterministic negotiations correspond to a subclass that does not seem to have been studied yet, and for which verification becomes considerably easier. For example, the question whether some local state occurs in some execution is PSPACE-complete for "sound" Zielonka automata, while it can be answered in polynomial time for sound deterministic negotiations.

A somewhat similar graphical formalism are message sequence charts/graphs, used to describe asynchronous communication. Questions like non-emptiness of intersection are in general undecidable for this model, even assuming that communication buffers are bounded. Subclasses of message sequence graphs with decidable model-checking problem were proposed, but the complexity is PSPACE-complete [9].

Overview. Section 2 introduces definitions and notations, then Section 3 reconsiders soundness for acyclic, deterministic negotiations. In Section 4 we provide an NLOGSPACE algorithm for the race problem. Section 5 solves the omitting problem, that is used in Section 6 for analyzing properties of workflows described by acyclic, deterministic negotiations, and later in Section 7 to decide soundness for acyclic weakly non-deterministic negotiations in PTIME. Finally, Section 8 establishes the coNP complexity bounds.

2 Negotiations

A *negotiation* $\mathcal{N}$ is a tuple $\langle Proc, N, dom, R, \delta \rangle$, where $Proc$ is a finite set of *processes* (or agents) that can participate in negotiations, and N is a finite set of *nodes* (or *atomic negotiations*) where the processes can synchronize. The function $dom : N \to \mathcal{P}(Proc)$ associates to every atomic negotiation $n \in N$ the (non-empty) set $dom(n)$ of processes participating in it. Nodes are denoted as m or n, and processes as p or q; possibly with indices.

The set of possible outcomes of atomic negotiations is denoted R, and we use $a, b, \ldots$ to range over its elements. The control flow in a negotiation is determined by a partial transition function $\delta : N \times R \times Proc \dashrightarrow \mathcal{P}(N)$, telling that after the outcome a of an atomic

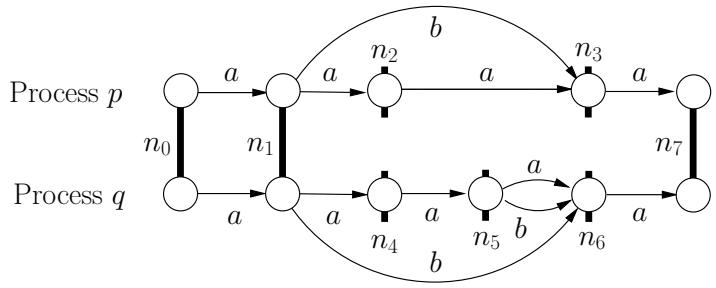


Figure 1 A negotiation. Atomic negotiation n_1 involves processes p, q, and has two possible outcomes a and b. The arrows show next negotiations in which respective processes are willing to engage.

negotiation n, process $p \in dom(n)$ is ready to participate in any of the negotiations from the set $\delta(n, a, p)$. So for every $n' \in \delta(n, a, p)$ we have $p \in dom(n') \cap dom(n)$. Every atomic negotiation $n \in N$ has its set of possible outcomes $out(n)$, and for every n, $a \in out(n)$ and $p \in dom(n)$ the result $\delta(n, a, p)$ has to be defined. So all processes involved in an atomic negotiation should be ready for all its possible outcomes. Observe that atomic negotiations may have one single participant process, and/or have one single outcome.

Negotiations admit a graphical representation. Figure 1 shows a negotiation with $Proc = \{p, q\}$, $N = \{n_0, \ldots, n_7\}$ and $R = \{a, b\}$. For example, we have $dom(n_1) = \{p, q\}$, $\delta(n_1, b, p) = \{n_3\}$ and $\delta(n_1, b, q) = \{n_6\}$. More details can be found in [3].

A *configuration* of a negotiation is a function $C : Proc \to \mathcal{P}(N)$ mapping each process p to the set of atomic negotiations in which p is ready to engage. An atomic negotiation n is *enabled* in a configuration C if $n \in C(p)$ for every $p \in dom(n)$, that is, if all processes that participate in n are ready to proceed with it. A configuration is a *deadlock* if no atomic negotiation is enabled in it. If an atomic negotiation n is enabled in C, and a is an outcome of n, then we say that (n, a) *can be executed*, and its execution produces a new configuration C' given by $C'(p) = \delta(n, a, p)$ for $p \in dom(n)$ and $C'(p) = C(p)$ for $p \notin dom(n)$. We denote this by $C \xrightarrow{(n,a)} C'$. For example, in Figure 1 we have $C \xrightarrow{(n_1,a)} C'$ for $C(p) = \{n_1\} = C(q)$ and $C'(p) = \{n_2\}, C(q) = \{n_4\}$.

A *run* of a negotiation $\mathcal{N}$ from a configuration C_1 is a finite or infinite sequence $w = (n_1, a_1)(n_2, a_2)\ldots$ such that there are configurations $C_2, C_3, \ldots$ with

$$C_1 \xrightarrow{(n_1,a_1)} C_2 \xrightarrow{(n_2,a_2)} C_3 \ldots$$

We denote this by $C_1 \xrightarrow{w}$, or $C_1 \xrightarrow{w} C_k$ if the sequence is finite and finishes with C_k. In the latter case we say that C_k is *reachable from* C_1 *on* w. We simply call it *reachable* if w is irrelevant, and write $C_1 \xrightarrow{*} C_k$.

Negotiations come equipped with two distinguished initial and final atomic negotiations n_{init} and n_{fin} in which *all* processes in $Proc$ participate. The *initial and final configurations* C_{init}, C_{fin} are given by $C_{init}(p) = \{n_{init}\}$ and $C_{fin}(p) = \{n_{fin}\}$ for all $p \in Proc$. A run is *successful* if it starts in C_{init} and ends in C_{fin}. We assume that every atomic negotiation (except possibly for n_{fin}) has at least one outcome. In Figure 1, $n_{init} = n_0$ and $n_{fin} = n_7$.

2.1 Main definitions

A negotiation $\mathcal{N}$ is *sound* if every partial run starting at C_{init} can be completed to a successful run. If a negotiation has no infinite runs, then it is sound iff it has no reachable deadlock configuration.

Process p is *deterministic* in a negotiation $\mathcal{N}$ if for every $n \in N$, and $a \in R$, the set of possible next negotiations, $\delta(n, a, p)$, is a singleton or the empty set. A negotiation is *deterministic* if every process $p \in Proc$ is deterministic. The negotiation of Figure 1 is deterministic.

A negotiation is *weakly non-deterministic* if for every $n \in N$ at least one of the processes in $dom(n)$ is deterministic. A negotiation is *very weakly non-deterministic*[4] if for every $n \in N$, $a \in R$, and $p \in Proc$, there is a deterministic process q such that $q \in dom(n')$ for all $n' \in \delta(n, a, p)$.

Examples of weakly non-deterministic negotiations can be found in [3]. In particular, weakly non-deterministic negotiations allow to model deterministic negotiations with global resources (see Section 6). The resource (say, a piece of data) can be modeled as an additional process, which participates in the atomic negotiations that use the resource. The outcome of a negotiation can change the state of the resource (say, from "confidential" to "public"), and at each state the resource may be ready to engage in a different set of atomic negotiations.

The *graph of a negotiation* has atomic negotiations, N, as set of nodes; the edges are $n \xrightarrow{p,a} n'$ if $n' \in \delta(n, a, p)$. Observe that $p \in dom(n) \cap dom(n')$.

A negotiation is *acyclic* if its graph is so. Acyclic negotiations cannot have infinite runs, so as mentioned above, soundness is equivalent to deadlock-freedom. For an acyclic negotiation $\mathcal{N}$ we fix a linear order $\preccurlyeq_{\mathcal{N}}$ on its nodes that is a topological order on the graph of $\mathcal{N}$. This means that if there is an edge from m to n in the graph of $\mathcal{N}$ then $m \preccurlyeq_{\mathcal{N}} n$.

The *restriction* of a negotiation $\mathcal{N}$ to a subset of its processes $Proc'$ is the negotiation $\langle Proc', N', dom', R, \delta' \rangle$ where N' is the set of those $n \in N$ for which $dom(n) \cap Proc' \neq \emptyset$, $dom'(n) = dom(n) \cap Proc'$, and $\delta'(n, r, p) = \delta(n, r, p) \cap N'$. The restriction of $\mathcal{N}$ to deterministic processes is denoted as $\mathcal{N}_D$ throughout the paper.

A negotiation $\mathcal{N}$ is *det-acyclic* if $\mathcal{N}_D$ is acyclic. It follows easily from the definitions that a weakly non-deterministic, det-acyclic negotiation does not have any infinite run.

3 Soundness of acyclic deterministic negotiations

The main objective of this section is to provide some tools that we will use later. We show how some properties of negotiations can be determined by patterns in their graphs. As an example of an application of our techniques we revisit the soundness problem for acyclic, deterministic negotiations. We provide an alternative polynomial-time algorithm that is actually in NLOGSPACE, in contrast with the algorithm of [3] that is based on rewriting.

Fix a negotiation $\mathcal{N}$. A *local path* is a path $n_0 \xrightarrow{p_0,a_0} n_1 \xrightarrow{p_1,a_1} \dots \xrightarrow{p_{k-1},a_{k-1}} n_k$ in the graph of $\mathcal{N}$. The path is *realizable* from some configuration C, if there is a run $C \xrightarrow{w}$ with w of the form $(n_0, a_0)w_1(n_1, a_1) \cdots w_{k-1}(n_{k-1}, a_{k-1})$, such that $p_i \notin dom(w_{i+1})$, for all i. Here we use $dom(v)$ to denote the set of all processes involved in some atomic negotiation appearing in sequence v: $dom(v) = \bigcup\{dom(n) : \text{for some } a, (n, a) \text{ appears in } v\}$.

For what follows Lemma 1 is particularly useful as it gives a simple criterion when an atomic negotiation is a part of some successful run.

▶ **Lemma 1.** *Let $n_0 \xrightarrow{p_0,a_0} n_1 \xrightarrow{p_1,a_1} \dots \xrightarrow{p_{k-1},a_{k-1}} n_k$ be a local path in the graph of a sound deterministic negotiation $\mathcal{N}$. If C is a reachable configuration of $\mathcal{N}$ and n_0 is enabled in C then the path is realizable from C.*

[4] This class was called *weakly deterministic* in [3].

Proof. Let C be such that $C(p) = n_0$ for every $p \in dom(n_0)$. By induction on i we show that there is a run $C \xrightarrow{*} C_i$ realizing $n_0 \xrightarrow{p_0,a_0} n_1 \xrightarrow{p_1,a_1} \dots \xrightarrow{p_{i-1},a_{i-1}} n_i$ and such that n_i is enabled in C_i.

For $i = 0$, we simply take $C_i = C$. For the induction step we assume the existence of C_i in which n_i is enabled. Let C'_{i+1} be the result of executing (n_i, a_i) from C_i. Observe that $C'_{i+1}(p_i) = n_{i+1}$ (recall that $\mathcal{N}$ is deterministic). Since $\mathcal{N}$ is sound, and C'_{i+1} is reachable, there is a run from C'_{i+1} to C_{fin}. We set then C_{i+1} to be the first configuration on this run when n_{i+1} is enabled. ◀

Lemma 1 says that there is a run containing the atomic negotiation m iff there is a local path from n_{init} to m. If $dom(m) \cap dom(n) \neq \emptyset$ then the lemma also provides an easy test for knowing whether there is a run containing both m, n: it suffices to check the existence of a local path $n_{init} \xrightarrow{*} m \xrightarrow{*} n$, or with m, n interchanged. The next lemma takes care of the opposite situation.

▶ **Lemma 2.** *Let m, n be two atomic negotiations in a sound deterministic negotiation $\mathcal{N}$, and assume that $dom(m) \cap dom(n) = \emptyset$.*

There exists some run of $\mathcal{N}$ containing both m, n iff there is an atomic negotiation m' such that

- *there is a local path from n_{init} to m',*
- *$\delta(m', p, a) = m_0$, $\delta(m', q, a) = n_0$ for some $p, q \in dom(m')$, $a \in out(m')$,*
- *there are two* disjoint *local paths in $\mathcal{N}$, one from m_0 to m, the other from n_0 to n.*

Soundness can be characterized by excluding a special variant of the pattern from the above lemma. Consider two processes $p \neq q$ of an acyclic negotiation $\mathcal{N}$. A (p, q)-*pair* is a pair of *disjoint* local paths of $\mathcal{N}$:

$$m_0 \xrightarrow{p,a_0} \dots \xrightarrow{p,a_{k-1}} m_k \qquad \text{and} \qquad n_0 \xrightarrow{q,b_0} \dots \xrightarrow{q,b_{l-1}} n_l$$

such that $m_k \preccurlyeq_{\mathcal{N}} n_l$ and $q \in dom(m_k)$.

▶ **Lemma 3.** *Let $\mathcal{N}$ be an acyclic deterministic negotiation. Then $\mathcal{N}$ is not sound if and only if there exist an atomic negotiation m' and two processes p, q such that:*

- *there is a local path from n_{init} to m',*
- *$\delta(m', p, a) = m_0$, $\delta(m', q, a) = n_0$ for some $a \in out(m')$,*
- *there is a (p, q)-pair as above.*

▶ **Theorem 4.** *Soundness of acyclic deterministic negotiations is* NLOGSPACE*-complete.*

Proof. Clearly the problem is NLOGSPACE-hard since graph reachability is a special instance of it. The NLOGSPACE algorithm for deciding soundness establishes the existence of the pattern from the previous lemma. Note that the topological order $\preccurlyeq_{\mathcal{N}}$ we use is arbitrary, so we can simply replace the condition $m_k \preccurlyeq_{\mathcal{N}} n_l$ by asking that there is no path from n_l to m_k.

◀

4 Races

For a given pair of atomic negotiations $m, n \in N$ of a deterministic negotiation $\mathcal{N} = \langle Proc, N, dom, R, \delta \rangle$, we want to determine if there is a reachable configuration at which m, n are concurrently enabled. In other words, we are asking whether a race between m and

n is possible. This is a standard question for concurrent systems, that is difficult to answer when working with linearizations. In this section we show a simple linear time algorithm answering the above question for acyclic, sound negotiations. Our algorithm reduces it to graph reachability questions, and can be implemented in logarithmic space. In the long version of our paper we also give a polynomial-time algorithm for possibly cyclic (and sound) negotiations.

We will write $m \parallel n$ when there is a reachable configuration C of $\mathcal{N}$ where both m and n are enabled. Our goal is to decide if $m \parallel n$ holds for given m, n.

We say below that a run $w \in (N \times R)^*$ can be reordered into another run w' if w' can be obtained from w by repeatedly exchanging adjacent $(m,a)(n,b)$ into $(n,b)(m,a)$ whenever $dom(m) \cap dom(n) = \emptyset$.

▶ **Lemma 5.** *Let $\mathcal{N}$ be an acyclic, deterministic, sound negotiation, and let m, n be two atomic negotiations in $\mathcal{N}$. Then $m \parallel n$ iff* every *run w from n_{init} containing both m and n can be reordered into a run w' such that $w' = C_{init} \xrightarrow{*} C \xrightarrow{*} C'$ for some configuration C where both m and n are enabled.*

Proof. It suffices to show the implication from left to right. So assume that there exists some reachable configuration C where both m and n are enabled. In particular, $dom(m) \cap dom(n) = \emptyset$. By way of contradiction, let us suppose that there exists some run containing both m and n, but this run cannot be reordered as claimed. We claim that there must be some local path from m to n in $\mathcal{N}$. To see this, assume the contrary and consider a run of the form $w = w_1(m,a)w_2(n,b)w_3$. The run w defines a partial order (actually a Mazurkiewicz trace) $tr(w)$ with nodes corresponding to positions in w, and edges from (m',c) to (n',d) if $dom(m') \cap dom(n') \neq \emptyset$ and (m',c) precedes (n',d) in w. Since there is no path from m to n in $\mathcal{N}$, nodes (m,a) and (n,b) are unordered in $tr(w)$. So we can choose a topological order w' of $tr(w)$ of the form $w' = w'_1(m,a)(n,b)w'_2$. This shows the claim.

So let π be a path in $\mathcal{N}$ from $m, n_1, \ldots, n_k, n$. Let p be some process such that $n_k \xrightarrow{p,a'} n$ for some outcome a'.

Let us go back to C. Since both m and n are enabled in C, we have a transition $C \xrightarrow{n,b} C_1$, for some $b \in out(n)$. Note that m is still enabled in C_1, since $dom(m) \cap dom(n) = \emptyset$. So we can apply Lemma 1 to C_1 and π (because $\mathcal{N}$ is sound), obtaining a configuration C_2 where $C_2(p) = n$. But since n was executed before C_1, this violates the acyclicity of $\mathcal{N}$. ◀

The next step is to convert the condition from Lemma 5 to a condition on the graph of a negotiation.

▶ **Proposition 6.** *Let $\mathcal{N}$ be an acyclic, deterministic, sound negotiation, and let m, n be two atomic negotiations in $\mathcal{N}$. Then $m \parallel n$ iff there exists a run containing both m, n, and there is neither a local path from m to n nor a local path from n to m.*

Observe that $dom(m) \cap dom(n) = \emptyset$ is a necessary condition for $m \parallel n$. Thus, from Proposition 6 and Lemma 2 we immediately obtain:

▶ **Theorem 7.** *For any acyclic, deterministic, sound negotiation $\mathcal{N}$ we can decide in linear time whether two atomic negotiations m, n of $\mathcal{N}$ satisfy $m \parallel n$. The above problem is* NLOGSPACE*-complete.*

5 Omitting problem

In this section we will be interested in determining the existence of some special successful runs of a deterministic negotiation $\mathcal{N}$. Let $B \subseteq N$ be a set of nodes of a negotiation $\mathcal{N}$. We

say that a run $(n_1, a_1)(n_2, a_2)\dots$ *omits* B if it does not contain any nodes from B, that is, $n_i \notin B$ for all i. Let $P \subseteq N \times R$ be a set of positive requirements. We say that a run as above *includes* P *and omits* B if it omits B and contains all the pairs from P.

We are interested in deciding if for a given $\mathcal{N}$ together with P and B there is a successful run of $\mathcal{N}$ including P and omitting B. We will consider only $\mathcal{N}$ that are sound, acyclic, and deterministic.

As a first step we define a *game* $G(\mathcal{N}, B)$ that is intended to produce runs that omit B (see e.g. [12] for an introduction to games):

- the positions of Eve are $N \setminus B$,
- the positions of Adam are $N \times R$,
- from n, Eve can go to any (n, a) with $a \in out(n)$,
- from (n, a), Adam can choose any process $p \in Proc$ and go to $n' = \delta(n, a, p)$,
- the initial position is n_{init},
- Adam wins if the play reaches a node in B, Eve wins if the play reaches n_{fin}.

Observe that since $\mathcal{N}$ is acyclic, the winning condition for Eve is actually a safety condition: every maximal play avoiding B is winning for Eve. So if Eve can win then she wins with a positional strategy. A *deterministic positional strategy for Eve* is a function $\sigma : N \to R$, it indicates that at position n Eve should go to position $(n, \sigma(n))$. Since $G(\mathcal{N}, B)$ is a safety game for Eve, there is a *biggest non-deterministic winning strategy* for Eve, i.e., a strategy of type $\sigma_{max} : N \to \mathcal{P}(R)$. The strategy σ_{max} is obtained by computing the set W_E of all winning positions for Eve in $G(\mathcal{N}, B)$, and then setting for every $n \in N$:

$$\sigma_{max}(n) = \{a \in out(n) : \forall p \in dom(n).\ \delta(n, a, p) \in W_E\}$$

▶ **Lemma 8.** *If $\mathcal{N}$ has a run omitting B then Eve has a winning strategy in $G(\mathcal{N}, B)$.*

▶ **Lemma 9.** *Suppose $\mathcal{N}$ is sound. Let $\sigma : N \to R$ be a winning strategy for Eve in $G(\mathcal{N}, B)$. Consider the set S of nodes that are reachable on a play from n_{init} respecting σ. There is a successful run of $\mathcal{N}$ containing precisely the nodes S.*

Proof. Consider an enumeration $n_1, n_2, ..., n_k$ of the nodes in $S \subseteq (N \setminus B)$ according to the topological order $\preccurlyeq_{\mathcal{N}}$. Let $w_i = (n_1, \sigma(n_1)) \dots (n_i, \sigma(n_i))$. By induction on $i \in \{1, \dots, k\}$ we prove that there is a configuration C_i such that $C_{init} \xrightarrow{w_i} C_i$ is a run of $\mathcal{N}$. This will show that w_k is a successful run containing precisely the nodes of S.

For $i = 1$, $n_1 = n_{init}$, in C_{init} all processes are ready to do n_1, so C_1 is the result of performing $(n_1, \sigma(n_1))$.

For the inductive step, we assume that we have a run $C_{init} \xrightarrow{w_i} C_i$, and we want to extend it by $C_i \xrightarrow{(n_{i+1}, \sigma(n_{i+1}))} C_{i+1}$. Consider a play respecting σ and reaching n_{i+1}. The last step in this play is $(n_j, \sigma(n_j)) \to n_{i+1}$, for some $j \leq i$ and n_j in S. This means that $\delta(n_j, \sigma(n_j), p) = n_{i+1}$ for some process p. Since $j \leq i$ and $(n_j, \sigma(n_j))$ occurred in w_i (but not n_{i+1}), we have $C_i(p) = n_{i+1}$. If we show that $C_i(q) = \{n_{i+1}\}$ for all $q \in dom(n_{i+1})$ then we obtain that n_{i+1} is enabled in C_i and we get the required C_{i+1}. Suppose by contradiction that $C_i(q) = \{n_l\}$ for some $l \neq i + 1$. We must have $l > i + 1$, since otherwise n_l already occurred in w_i. By definition of our indexing $n_{i+1} \prec_{\mathcal{N}} n_l$. But then no execution from C_i can bring process q to a state where it is ready to participate in negotiation n_{i+1}, and p will stay forever in n_{i+1}. This contradicts the fact that the negotiation is sound. ◀

▶ **Corollary 10.** *For a sound negotiation $\mathcal{N}$: Eve wins in $G(\mathcal{N}, B)$ iff $\mathcal{N}$ has a successful run omitting B.*

▶ **Theorem 11.** *Let K be a constant. It can be decided in* PTIME *if for a given deterministic, acyclic, and sound negotiation $\mathcal{N}$ and two sets $B \subseteq N$, and $P \subseteq N \times R$, with the size of P at most K, there is a successful run of $\mathcal{N}$ containing P and omitting B.*

Proof. If for some atomic negotiation m, we have $(m, a) \in P$ and $(m, b) \in P$ for $a \neq b$ then the answer is negative as $\mathcal{N}$ is acyclic. So let us suppose that it is not the case. By Lemmas 8 and 9 our problem is equivalent to determining the existence of a deterministic strategy σ for Eve in the game $G(\mathcal{N}, B)$ such that $\sigma(m) = a$ for all $(m, a) \in P$, and all these (m, a) are reachable on a play respecting σ.

To decide this we calculate σ_{max}, the biggest non-deterministic winning strategy for Eve in $G(\mathcal{N}, B)$. This can be done in PTIME as the size of $G(\mathcal{N}, B)$ is proportional to the size of the negotiation. Strategy σ_{max} defines a graph $G(\sigma_{max})$ whose nodes are atomic negotiations, and edges are (m, a, m') if $(m, a) \in \sigma_{max}$ and $m' = \delta(m, a, p)$ for some process p. The size of this graph is proportional to the size of the negotiation. In this graph we look for a subgraph H such that:

- for every node m in H there is at most one a such that (m, a, m') is an edge of H for some m';
- for every $(m, a) \in P$ there is an edge (m, a, m') in H for some m', and moreover m is reachable from n_{init} in H.

We show that such a graph H exists iff there is a strategy σ with the required properties.

Suppose there is a deterministic winning strategy σ such that $\sigma(m) = a$ for all $(m, a) \in P$, and all these (m, a) are reachable on a play respecting σ. We now define H by putting an edge (m, a, m') in H if $\sigma(m) = a$ and and $m' = \delta(m, a, p)$ for some process p. As σ is deterministic and winning, this definition guarantees that H satisfies the first item above. The second item is guaranteed by the reachability property that σ satisfies.

For the other direction, given such a graph H we define a deterministic strategy σ_H. We put $\sigma_H(m) = a$ if (m, a, m') is an edge of H. If m is not a node in H, or has no outgoing edges in H then we put $\sigma_H(m) = b$ for some arbitrary $b \in \sigma_{max}(m)$. It should be clear that σ_H is winning since every play respecting σ_H stays in winning nodes for Eve. By definition $\sigma_H(m) = a$ for all $(m, a) \in P$, and all these (m, a) are reachable on a play respecting σ_H.

So we have reduced the problem stated in the theorem to finding a subgraph H of $G(\sigma_{max})$ as described above. If there is such a subgraph H then there is one in form of a tree, where the edges leading to leaves are of the form (m, a, m') with $(m, a) \in P$. Moreover, there is such a tree with at most $|P|$ nodes with more than one child. So finding such a tree can be done by guessing the $|P|$ branching nodes and solving $|P| + 1$ reachability problems in $G(\sigma_{max})$. This can be done in PTIME since the size of P is bounded by K. ◀

6 Workflows and deterministic negotiations with data

We show how our algorithms from the previous sections can be used to check functional properties of deterministic negotiations, like those studied for workflow systems with data [15]. We take some of the functional properties of [15], and give polynomial algorithms for verifying them over deterministic, acyclic, sound negotiations.

In this section we consider *acyclic*, deterministic negotiations with shared variables over a finite domain, that can be updated by the outcomes of the negotiation. More precisely, each outcome $(n, a) \in N \times R$ comes with a set Σ of operations on these shared variables. In our examples this set Σ is composed of $alloc(x)$, $read(x)$, $write(x)$, and $dealloc(x)$.

Formally, a *negotiation with data* is a negotiation with one additional component: $\mathcal{N} = \langle Proc, N, dom, R, \delta, \ell \rangle$ where $\ell \colon (N \times R) \to \mathcal{P}(\Sigma \times X)$ maps every outcome to a (possibly

Table 1 Data for the negotiation of Figure 1 (adapted from [13]).

atom. neg.	n_0	n_1		n_2	n_3	n_4	n_5		n_6	n_7
outcome	a	a	b	a	a	a	a	b	a	a
alloc	$1,\dots,10$									
read		1	1		1, 8		5		2, 7, 9	6, 8, 9
write		3, 5, 6	3	1, 4, 8	9, 10	2, 7, 10	7	9	6, 8	
dealloc				4		2			5, 6, 7	

empty) set of data operations on variables from X. We assume that for each $(n, a) \in N \times R$ and for each variable $x \in X$ the label $\ell(n, a)$ contains at most one operation on x, that is, at most one element of $\Sigma \times \{x\}$.

As an example, we enrich the negotiation of Figure 1 with data, as shown in Table 1. (This example is taken from [13]). The variables are $X = \{x_1, \dots, x_{10}\}$. The table gives for each outcome and for each operation the set of (indices of the) variables to which the outcome applies this operation.

In [13] some examples of data specifications for workflows are considered. They are presented in the form of anti-patterns, that is, patterns that the negotiation should avoid.

1. *Inconsistent data*: an atomic negotiation reads or writes a variable x while another atomic negotiation is writing, allocating, or deallocating it in parallel.
   In our example there is an execution in which (n_2, a) and (n_6, a) write to x_8 in parallel.
2. *Weakly redundant data*: there is an execution in which a variable is written and never read before it is deallocated or the execution ends.
   In the example, there is an execution in which x_{10} is written by (n_4, a), and never read again.
3. *Never destroyed*: there is an execution in which a variable is allocated and then never deallocated before the execution ends.
   In the example, the execution taking (n_5, b) never deallocates x_2.

It is easy to give algorithms for these properties that are polynomial *in the size of the reachability graph*. We give the first algorithms that check these properties in polynomial time *in the size of the negotiation*, which can be exponentially smaller than its reachability graph.

For the first property we can directly use the algorithm of the previous section: It suffices to check if the negotiation has two outcomes $(m, a), (n, b)$ such that m and n can be concurrently enabled, and there is variable x such that $\ell(a) \cap \{read(x), write(x)\} \neq \emptyset$ and $\ell(b) \cap \{write(x), alloc(x), dealloc(x)\} \neq \emptyset$.

In the rest of the section we present a polynomial algorithm for the following abstract problem, which has the problems (2) and (3) above as special instances. Given sets $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O} \subseteq N \times R$ such that $\mathcal{O}_1 \cup \mathcal{O}_2 \subseteq \mathcal{O}$, we say that the negotiation $\mathcal{N}$ *violates the specification* $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$ if there is a run $w = (n_1, a_1) \cdots (n_k, a_k)$ with indices $0 \leq i < j \leq k$ such that $(n_i, a_i) \in \mathcal{O}_1$, $(n_j, a_j) \in \mathcal{O}_2$, and $(n_l, a_l) \notin \mathcal{O}$ for all $i < l < j$. In this case we also say that $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$ is violated at $(n_i, a_i), (n_j, a_j)$. Otherwise $\mathcal{N}$ *complies with* $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$.

▶ **Example 12.** Observe that variable x is weakly redundant (specification of type (2)) iff $\mathcal{N}$ violates $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$, where $\mathcal{O}_1 = \{(n, a) \in N \times R : write(x) \in \ell(n, a)\}$, $\mathcal{O}_2 = \{(n, b) \in N \times R : n = n_{fin} \vee dealloc(x) \in \ell(n, b)\}$ and $\mathcal{O} = \{(n, c) : \ell(n, c) \cap (\Sigma \times \{x\}) \neq \emptyset\}$.

Variable x is never destroyed (specification of type (3)) iff $\mathcal{N}$ violates $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$, where $\mathcal{O}_1 = \{(n, a) \in N \times R : alloc(x) \in \ell(n, a)\}$, $\mathcal{O}_2 = \{n_{fin}\}$, $\mathcal{O} = \{(n, c) : n = n_{fin} \vee \ell(n, c) \cap \{alloc(x), dealloc(x)\} \neq \emptyset\}$.

For the next proposition it is convenient to use the notation $m \xrightarrow{+} n$, whenever there is a (non-empty) local path in $\mathcal{N}$ from the atomic negotiation m to the atomic negotiation n.

▶ **Proposition 13.** *Let $\mathcal{N}$ be an acyclic, deterministic, sound negotiation with data, and $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$ a specification. Let $(m, a) \in \mathcal{O}_1$, $(n, b) \in \mathcal{O}_2$. Then $\mathcal{N}$ violates $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$ at $(m, a), (n, b)$ iff either $m \parallel n$, or $m \xrightarrow{+} n$ and $\mathcal{N}$ has a run from n_{init} containing $P = \{(m, a), (n, b)\}$, and omitting the set $B = \{m' \in \mathcal{O} : m \xrightarrow{+} m' \xrightarrow{+} n\}$.*

Putting together Proposition 13 and Theorem 11 we obtain:

▶ **Corollary 14.** *Given an acyclic, deterministic, sound negotiation with data $\mathcal{N}$, and a specification $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$, it can be checked in polynomial time whether $\mathcal{N}$ complies with $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O})$.*

7 Soundness of acyclic weakly non-deterministic negotiations is in PTIME

In previous sections we have presented algorithms for analysis of sound negotiations. Here we show that our techniques also allow to find a bigger class of negotiations for which we can decide soundness in PTIME. The class we consider is that of acyclic, weakly non-deterministic negotiations, c.f. page 5. That is, we allow some processes to be non-deterministic, but every atomic negotiation should involve at least one deterministic process.

Recall that $\mathcal{N}_D$ is the restriction of $\mathcal{N}$ to deterministic processes. Since $\mathcal{N}$ is weakly non-deterministic, every atomic negotiation involves a deterministic process, so $N_D = N$. Recall also that for an acyclic negotiation $\mathcal{N}$ we fixed some linear order $\preccurlyeq_{\mathcal{N}}$ that is a topological order of the graph of $\mathcal{N}$.

The first lemma gives a necessary condition for the soundness of $\mathcal{N}$ that is easy to check. It is proved by showing that $\mathcal{N}_D$ cannot have much more behaviours than $\mathcal{N}$.

▶ **Lemma 15.** *If $\mathcal{N}$ is a sound, acyclic, weakly non-deterministic negotiation then $\mathcal{N}_D$ is sound.*

We then first consider the case of a negotiation with only one non-deterministic process. The next lemma reduces (un)soundness of $\mathcal{N}$ to some pattern in $\mathcal{N}_D$.

▶ **Lemma 16.** *Let $\mathcal{N}$ be an acyclic, weakly non-deterministic negotiation with only one non-deterministic process p. Then $\mathcal{N}$ is not sound, if and only if, either:*

- *$\mathcal{N}_D$ is not sound, or*
- *$\mathcal{N}_D$ is sound, and it has two nodes $m \preccurlyeq_{\mathcal{N}} n$ with outcomes $a \in out(m)$, $b \in out(n)$ such that:*
  - *$p \in dom(m) \cap dom(n)$, $n \notin S_p = \delta(m, a, p)$, and*
  - *there is a successful run of $\mathcal{N}_D$ containing $P = \{(m, a), (n, b)\}$ and omitting $B = \{n' \in S_p : m \prec_{\mathcal{N}} n' \prec_{\mathcal{N}} n\}$.*

The next lemma deals with the case when there is more than one non-deterministic process.

▶ **Lemma 17.** *An acyclic weak non-deterministic negotiation $\mathcal{N}$ is not sound if and only if:*

1. *either its restriction $\mathcal{N}_D$ to deterministic processes is not sound,*
2. *or, for some non-deterministic process p, its restriction $\mathcal{N}^p$ to p and the deterministic processes is not sound.*

▶ **Theorem 18.** *Soundness can be decided in* PTIME *for acyclic, weakly non-deterministic negotiations.*

Proof. By Lemma 17 we can restrict to negotiations $\mathcal{N}$ with one non-deterministic process. For every $m \preccurlyeq_{\mathcal{N}} n$, a and b we check the conditions described in Lemma 16. The existence of a run of $\mathcal{N}_D$ can be checked in PTIME thanks to Theorem 11 and the fact that the size of P is constant. ◀

8 Beyond acyclic weakly non-deterministic negotiations

In this section we show that the polynomial-time result for acyclic, weakly non-deterministic negotiations from Section 7 requires both acyclicity and weak non-determinism. We prove that if we remove one of the two assumptions then the problem becomes coNP-complete. Indeed, even a very mild extension of acyclicity makes the soundness problem coNP-complete.

It is not very surprising that deciding soundness for acyclic, non-deterministic negotiations is coNP-complete. The problem is in coNP since all runs are of polynomial size, so it suffices to guess a run and check if the reached configuration is a deadlock. The hardness is by a simple reduction of SAT to the complement of the soundness problem. It strongly relies on non-determinism.

▶ **Proposition 19.** *Soundness of acyclic non-deterministic negotiations is* coNP*-complete.*

In view of the above proposition, the other possibility is to keep weak non-determinism and relax the notion of acyclicity. We consider a very mild relaxation: deterministic processes still need to be acyclic. This condition implies that all the runs are of polynomial size. We show that even for *very* weakly non-deterministic negotiations (c.f. page 5) the problem is already coNP-complete.

▶ **Theorem 20.** *Non-soundness of det-acyclic, very weakly non-deterministic negotiations is* NP*-complete.*

9 Conclusions

Analysis of concurrent systems is very often PSPACE-hard because of the state explosion problem. One way to address this problem is to look for restricted classes of concurrent systems which are non-trivial, and yet are algorithmically easier to analyze. We argue in this paper that negotiations are well adapted for this task. Processes in a negotiation are stateless, at every moment their state is the set of negotiations they are willing to engage. When processes are non-deterministic this mechanism can simulate states, so that the interesting cases occur when non-determinism is limited. These limitations are still relevant as show examples from workflow nets. In short, the negotiation model offers a simple way to formulate restrictions that are sufficiently expressive and algorithmically relevant.

We have shown that a number of verification problems for sound deterministic acyclic negotiations can be solved in PTIME or even in NLOGSPACE. In our application to workflow Petri nets, acyclicity and determinism (equivalent to free-choiceness) are quite common: about 70% of the industrial workflow nets of [16, 7, 6] are free-choice, and about 60% are both acyclic and free-choice (see e.g. the table at the end of [6]).

Open problems. It would be interesting to have a better understanding what verification problems for deterministic, acyclic, sound negotiations can be solved in PTIME. The coNP

result for weakly-deterministic negotiations shows that one should proceed carefully here: allowing arbitrary products with finite automata would not work.

References

1 Jörg Desel and Javier Esparza. Negotiations and Petri nets. In *Int. Workshop on Petri Nets and Software Engineering (PNSE'15)*, volume 1372 of *CEUR Workshop Proceedings*, pages 41–57. CEUR-WS.org, 2015.

2 Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

3 Javier Esparza and Jörg Desel. On negotiation as concurrency primitive. In *CONCUR*, pages 440–454, 2013. Extended version in CoRR abs/1307.2145.

4 Javier Esparza and Jörg Desel. On negotiation as concurrency primitive II: Deterministic cyclic negotiations. In *FoSSaCS*, pages 258–273, 2014. Extended version in CoRR abs/1403.4958.

5 Javier Esparza and Jörg Desel. Negotiation programs. In *Application and Theory of Petri Nets and Concurrency*, volume 9115 of *LNCS*, pages 157–178. Springer, 2015.

6 Javier Esparza and Philipp Hoffmann. Reduction rules for colored workflow nets. In *FASE*, volume 9633 of *LNCS*, pages 342–358. Springer, 2016.

7 Dirk Fahland, Cédric Favre, Barbara Jobstmann, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Instantaneous soundness checking of industrial business process models. In *Business Process Management*, pages 278–293. Springer, 2009.

8 Cédric Favre, Hagen Völzer, and Peter Müller. Diagnostic information for control-flow analysis of workflow graphs (a.k.a. free-choice workflow nets). In *TACAS 2016*, volume 9636 of *LNCS*, pages 463–479. Springer, 2016.

9 Blaise Genest, Dietrich Kuske, and Anca Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf.& Comput.*, 204(6):920–956, 2006.

10 Anca Muscholl. Automated synthesis of distributed controllers. In *ICALP 2015*, volume 9135 of *LNCS*, pages 11–27. Springer, 2015.

11 Natalia Sidorova, Christian Stahl, and Nikola Trcka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.*, 36(7):1026–1043, 2011.

12 Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *LNCS*, pages 635–655. Springer, 2008.

13 Nikola Trcka, Wil M. P. van der Aalst, and Natalia Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In *Advanced Information Systems Engineering (CAiSE)*, volume 5565 of *LNCS*, pages 425–439. Springer, 2009.

14 Wil M. P. van der Aalst. Business process management as the "Killer App" for Petri nets. *Software & Systems Modeling*, 14(2):685–691, 2015.

15 Wil M.P. van der Aalst. The application of Petri nets to workflow management. *J. Circuits, Syst. and Comput.*, 08(01):21–66, 1998.

16 B. van Dongen, M. Jansen-Vullers, H.M.W. Verbeek, and Wil M. P. van der Aalst. Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. *Computers in Industry*, 58(6):578–601, 2007.

17 W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.

Deciding Hyperproperties*

Bernd Finkbeiner[1] and Christopher Hahn[2]

1 **Saarland University**
Saarbrücken, Germany
finkbeiner@react.uni-saarland.de
2 **Saarland University**
Saarbrücken, Germany
hahn@react.uni-saarland.de

Abstract

Hyperproperties, like observational determinism or symmetry, cannot be expressed as properties of individual computation traces, because they describe a relation between multiple computation traces. HyperLTL is a temporal logic that captures such relations through trace variables, which are introduced through existential and universal trace quantifiers and can be used to refer to multiple computations at the same time. In this paper, we study the satisfiability problem of HyperLTL. We show that the problem is PSPACE-complete for alternation-free formulas (and, hence, no more expensive than LTL satisfiability), EXPSPACE-complete for $\exists^*\forall^*$ formulas, and undecidable for $\forall\exists$ formulas. Many practical hyperproperties can be expressed as alternation-free formulas. Our results show that both satisfiability and implication are decidable for such properties.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases temporal logics, satisfiability, hyperproperties, complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.13

1 Introduction

Hyperproperties [4] are system properties that relate multiple computation traces. For example, in the design of a system that handles sensitive information, we might specify that a certain secret is kept confidential by requiring that the system is *deterministic* in its legitimately observable inputs, i.e., that all computations with the same observable inputs must have the same observable outputs, independently of the secret [13, 16]. In the design of an access protocol for a shared resource, we might specify that the access to the resource is *symmetric* between multiple clients by requiring that for every computation and every permutation of the clients, there exists a computation where the access is granted in the permuted order [7].

To express hyperproperties in a temporal logic, linear-time temporal logic (LTL) has recently been extended with trace variables and trace quantifiers. In HyperLTL [3], observational determinism can, for example, be expressed as the formula $\forall\pi.\forall\pi'.\ \square(I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$, where I is the set of observable inputs and O is the set of observable outputs. The universal quantification of the trace variables π and π' indicates that the property must hold for all pairs of computation traces. It has been shown that many hyperproperties of interest can be expressed in HyperLTL [12].

* This work was partially supported by the German Research Foundation (DFG) in the Collaborative Research Center 1223 and by the Graduate School of Computer Science at Saarland University.

In this paper, we study the *satisfiability problem* of HyperLTL. Unlike the model checking problem, for which algorithms and tools exist [3, 7], the decidability and complexity of the satisfiability problem was, so far, left open. The practical demand for a decision procedure is strong. Often, one considers multiple formalizations of similar, but not necessarily equivalent hyperproperties. An alternative (and slightly stronger) version of observational determinism requires, for example, that differences in the observable output may only occur *after* differences in the observable input have occurred: $\forall\pi.\forall\pi'.\ (O_\pi = O_{\pi'})\ \mathcal{W}\ (I_\pi \neq I_{\pi'})$. A decision procedure for HyperLTL would allow us to automatically check whether such formalizations imply each other. Another important application is to check whether the functionality of a system, i.e., a standard trace property, is compatible with the desired hyperproperties, such as confidentiality. Since both types of properties can be expressed in HyperLTL, a decision procedure for HyperLTL would make it possible to identify inconsistent system requirements early on, before an attempt is made to implement the requirements.

The fundamental challenge in deciding hyperproperties is that hyperproperties are usually not ω-regular [1]. HyperLTL formulas thus cannot be translated into equivalent automata [6]. Intuitively, since hyperproperties relate multiple infinite traces, an automaton, which only considers one trace at a time, would have to memorize an infinite amount of information from one trace to the next. This means that the standard recipe for checking the satisfiability of a temporal logic, which is to translate the given formula into an equivalent Büchi automaton and then check if the language of the automaton is empty [15], cannot be applied to HyperLTL.

In model checking, this problem is sidestepped by verifying the *self-composition* [2] of the given system: instead of verifying a hyperproperty that refers to n traces, we verify a trace property that refers to a *single* trace of a new system that contains n copies of the original system. Since the satisfiability problem does not refer to a system, this idea cannot immediately be applied to obtain a decision procedure for HyperLTL. However, it would seem natural to define a similar self-composition, on the formula rather than the system, in order to determine satisfiability.

We organize our investigation according to the quantifier structure of the HyperLTL formulas. LTL, for which the satisfiablity problem is already solved [14], is the sublogic of HyperLTL where the formulas have a single universally quantified trace variable, which is usually left implicit. The next larger fragment consists of the alternation-free formulas, i.e., formulas with an arbitrary number of trace variables and a quantifier prefix that either consists of only universal or only existential quantifiers. Many hyperproperties of practical interest, such as observational determinism, belong to this fragment. It turns out that the satisfiability of alternation-free formulas can indeed be reduced to the satisfiability of LTL formulas by replicating the atomic propositions such that there is a separate copy for each trace variable. This construction is sound, because in an alternation-free formula, the values for the quantifiers can be chosen independently of each other. The size of the resulting LTL formula is the same as the given HyperLTL formula; as a result, the satisfiability problem of the alternation-free fragment has the same complexity, PSPACE-complete, as LTL satisfiability.

If the formula contains a quantifier alternation, the values of the quantifiers can no longer be chosen independently of each other. However, if the quantifier structure is of the form $\exists^*\forall^*$, i.e., the formula begins with an existential quantifier and then has a single quantifier alternation, then it is still possible to reduce HyperLTL satisfiability to LTL satisfiability by explicitly considering all possible interactions between the existential and universal quantifiers. For example, $\exists\pi_0\exists\pi_1\forall\pi_2.\ (\bigcirc p_{\pi_0}) \wedge (\square p_{\pi_1}) \wedge (\diamondsuit p_{\pi_2})$ is equisatisfiable to $\exists\pi_0\exists\pi_1.\ (\bigcirc p_{\pi_0}) \wedge (\square p_{\pi_1}) \wedge (\diamondsuit p_{\pi_0}) \wedge (\diamondsuit p_{\pi_1})$, which is in turn equisatisfiable to the

Table 1 Complexity results for the satisfiability problem of HyperLTL.

$\exists^*$	$\forall^*$	$\exists^*\forall^*$	bounded $\exists^*\forall^*$	$\forall\exists$
PSPACE-complete	PSPACE-complete	EXPSPACE-complete	PSPACE-complete	undecidable

LTL formula $(\bigcirc p_0) \wedge (\Box p_1) \wedge (\Diamond p_0) \wedge (\Diamond p_1)$. In general, enumerating all combinations of existential and universal quantifiers causes an exponential blow-up and we show that the satisfiability problem for the $\exists^*\forall^*$-fragment is indeed EXPSPACE-complete. This high complexity is, however, relativized by the fact that practical hyperproperties rarely need a large number of quantifiers. If we bound the number of universal quantifiers by a constant, the complexity becomes PSPACE again.

Formulas where an existential quantifier occurs in the scope of a universal quantifier make the logic dramatically more powerful, because they can be used to enforce, inductively, models with an infinite number of traces. We show that a single pair of quantifers of the form $\forall\exists$ suffices to encode Post's correspondence problem. The complete picture is thus as summarized in Table 1: The largest decidable fragment of HyperLTL is the EXPSPACE-complete $\exists^*\forall^*$ fragment. Bounding the number of universal quantifiers and in particular restricting to alternation-free formulas reduces the complexity to PSPACE. Any fragment that contains the $\forall\exists$ formulas is undecidable.

From a theoretical point of view, the undecidability of the $\forall\exists$ fragment is a noteworthy result, because it confirms the intuition that hyperproperties are truly more powerful than trace properties. In practice, already the alternation-free fragment suffices for many important applications (cf. [7]). From a practical point of view, the key result of the paper is therefore that both satisfiability of alternation-free formulas and implication between alternation-free formulas, which can be expressed as unsatisfiability of an $\exists^*\forall^*$ formula, are decidable.

2 HyperLTL

Let AP be a set of *atomic propositions*. A *trace* t is an infinite sequence over subsets of the atomic propositions. We define the set of traces $TR := (2^{AP})^\omega$. A subset $T \subseteq TR$ is called a *trace property*. We use the following notation to manipulate traces: let $t \in TR$ be a trace and $i \in \mathbb{N}$ be a natural number. $t[i]$ denotes the i-th element of t. Therefore, $t[0]$ represents the starting element of the trace. Let $j \in \mathbb{N}$ and $j \geq i$. $t[i,j]$ denotes the sequence $t[i]\ t[i+1] \dots t[j-1]\ t[j]$. $t[i,\infty]$ denotes the infinite suffix of t starting at position i.

LTL Syntax. Linear-time temporal logic (LTL) [9] combines the usual boolean connectives with temporal modalities such as the *Next* operator $\bigcirc$ and the *Until* operator $\mathcal{U}$. The syntax of LTL is given by the following grammar:

$$\varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \bigcirc\varphi \ \mid \ \varphi \,\mathcal{U}\, \varphi$$

where $p \in AP$ is an atomic proposition. $\bigcirc\varphi$ means that φ holds in the *next* position of a trace; $\varphi_1 \,\mathcal{U}\, \varphi_2$ means that φ_1 holds *until* φ_2 holds. There are several derived operators, such as $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$, and $\varphi_1\ W\varphi_2 \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \Box\varphi_1$. $\Diamond\varphi$ states that φ will *eventually* hold in the future and $\Box\varphi$ states that φ holds *globally*; $\mathcal{W}$ is the *weak* version of the *until* operator.

LTL Semantics. Let $p \in AP$ and $t \in TR$. The semantics of an LTL formula is defined as the smallest relation $\models$ that satisfies the following conditions:

$$
\begin{array}{lll}
t \models p & \text{iff} & p \in t[0] \\
t \models \neg\psi & \text{iff} & t \not\models \psi \\
t \models \psi_1 \vee \psi_2 & \text{iff} & t \models \psi_1 \text{ or } t \models \psi_2 \\
t \models \bigcirc \psi & \text{iff} & t[1,\infty] \models \psi \\
t \models \psi_1 \,\mathcal{U}\, \psi_2 & \text{iff} & \text{there exists } i \geq 0 : t[i,\infty] \models \psi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } t[j,\infty] \models \psi_1
\end{array}
$$

LTL-SAT is the problem of deciding whether there exists a trace $t \in TR$ such that $t \models \psi$.

▶ **Theorem 1.** *LTL-SAT is PSPACE-complete [14].*

HyperLTL Syntax. HyperLTL [3] extends LTL with trace variables and trace quantifiers. Let $\mathcal{V}$ be an infinite supply of trace variables. The syntax of HyperLTL is given by the following grammar:

$$
\begin{array}{lll}
\psi & ::= & \exists \pi.\ \psi \ \mid\ \forall \pi.\ \psi \ \mid\ \varphi \\
\varphi & ::= & a_\pi \ \mid\ \neg\varphi \ \mid\ \varphi \vee \varphi \ \mid\ \bigcirc \varphi \ \mid\ \varphi \,\mathcal{U}\, \varphi
\end{array}
$$

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The quantification over traces makes it possible to express properties like "on all traces ψ must hold", which is expressed by $\forall \pi.\ \psi$. Dually, one can express that "there exists a trace such that ψ holds", which is denoted by $\exists \pi.\ \psi$. The derived operators $\Diamond$, $\Box$, and $\mathcal{W}$ are defined as for LTL.

HyperLTL Semantics. A HyperLTL formula defines a *hyperproperty*, i.e., a set of sets of traces. A set T of traces satisfies the hyperproperty if it is an element of this set of sets. Formally, the semantics of HyperLTL formulas is given with respect to a *trace assignment* Π from $\mathcal{V}$ to TR, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped according to Π. $\Pi[i,\infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i,\infty]$ for all π.

$$
\begin{array}{lll}
\Pi \models_T \exists \pi.\psi & \text{iff} & \text{there exists } t \in T \ :\ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T \forall \pi.\psi & \text{iff} & \text{for all } t \in T \ :\ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T a_\pi & \text{iff} & a \in \Pi(\pi)[0] \\
\Pi \models_T \neg\psi & \text{iff} & \Pi \not\models_T \psi \\
\Pi \models_T \psi_1 \vee \psi_2 & \text{iff} & \Pi \models_T \psi_1 \text{ or } \Pi \models_T \psi_2 \\
\Pi \models_T \bigcirc \psi & \text{iff} & \Pi[1,\infty] \models_T \psi \\
\Pi \models_T \psi_1 \,\mathcal{U}\, \psi_2 & \text{iff} & \text{there exists } i \geq 0 : \Pi[i,\infty] \models_T \psi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } \Pi[j,\infty] \models_T \psi_1
\end{array}
$$

HyperLTL-SAT is the problem of deciding whether there exists a *non-empty* set of traces T such that $\Pi \models_T \psi$, where Π is the empty trace assignment and $\models_T$ is the smallest relation satisfying the conditions above. If it is clear from the context, we omit Π and simply write $T \models \psi$. If $\models_T \psi$, we call T a model of ψ.

3 Alternation-free HyperLTL

We begin with the satisfiability problem for the alternation-free fragments of HyperLTL. We call a HyperLTL formula ψ (quantifier) *alternation-free* iff the quantifier prefix only consists of either only universal or only existential quantifiers. We denote the corresponding fragments as the $\forall^*$ and $\exists^*$ fragments, respectively. For both fragments, we show that every formula can be reduced, as discussed in the introduction, to an equisatisfiable LTL formula of the same size. As a result, we obtain that the satisfiability problem of alternation-free HyperLTL is PSPACE-complete, like the satisfiability problem of LTL. In the following, some proofs are omitted due to space constraints. The proofs can be found in the full version of this paper [5].

3.1 The $\forall^*$ Fragment

The $\forall^*$ fragment is particularly easy to decide, because we can restrict the models, without loss of generality, to singleton sets of traces: since all quantifiers are universal, every model with more than one trace could immediately be translated into another one where every trace except one is omitted. Hence, we can ignore the trace variables and interpret the HyperLTL formula as a plain LTL formula.

▶ **Example 2.** Consider the following HyperLTL formula with atomic propositions $\{a, b\}$:

$$\forall \pi_1 \forall \pi_2.\ \square b_{\pi_1} \wedge \square \neg b_{\pi_2}$$

Since the trace variables are universally quantified, we are reasoning about *every* pair of traces, and thus in particular about the pairs where both variables refer to the same trace. It is, therefore, sufficient to check the satisfiability of the LTL formula $\square b \wedge \square \neg b$, which turns out to be unsatisfiable.

The satisfiability of hyperproperties that can be expressed in the $\forall^*$ fragment, such as observational determinism, thus immediately reduces to LTL satisfiability.

▶ **Lemma 3.** *For every $\forall^*$ HyperLTL formula there exists an equisatisfiable LTL formula of the same size.*

3.2 The $\exists^*$ Fragment

A model of a formula in the $\exists^*$ fragment may, in general, have more than one trace. For example the models of $\exists \pi_1 \exists \pi_2.\ a_{\pi_1} \wedge \neg a_{\pi_2}$ have (at least) two traces. In order to reduce HyperLTL satisfiability again to LTL satisfiability, we *zip* such traces together. For this purpose, we introduce a fresh atomic proposition for every atomic proposition a and every path variable π that occur as an indexed proposition a_π in the formula. We obtain an equisatisfiable LTL formula by removing the quantifier prefix and replacing every occurrence of a_π with the new proposition.

▶ **Example 4.** Consider the following HyperLTL formula over the atomic propositions $\{a, b\}$:

$$\exists \pi_1 \exists \pi_2.\ a_{\pi_1} \wedge \square \neg b_{\pi_1} \wedge \square b_{\pi_2}$$

By discarding the quantifier prefix and replacing the indexed propositions with fresh propositions, we obtain the equisatisfiable LTL formula over the atomic propositions $\{a_1, b_1, b_2\}$:

$$a_1 \wedge \square \neg b_1 \wedge \square b_2$$

The LTL formula is satisfied by the trace $\tilde{p}$: $(\{a_1, b_2\})^\omega$. We can map the fresh propositions back to the original indexed propositions. In this way, we obtain witnesses for π_1 and π_2 by splitting $\tilde{p}$ into two traces $\{a\}^\omega$ and $\{b\}^\omega$, where for every position in these traces only those atomic propositions that were labelled with π_1 or π_2, respectively, hold. Hence, the trace set satisfying the HyperLTL formula is $\{\{a\}^\omega, \{b\}^\omega\}$.

▶ **Lemma 5.** *For every $\exists^*$ HyperLTL formula there exists an equisatisfiable LTL formula of the same size.*

Combining Lemma 3 and Lemma 5, we conclude that HyperLTL-SAT inherits the complexity of LTL-SAT for the alternation-free fragment.

▶ **Theorem 6.** *HyperLTL-SAT is PSPACE-complete for the alternation-free fragment.*

4 The $\exists^*\forall^*$ Fragment

Allowing quantifier alternation makes the satisfiability problem significantly more difficult, and even leads to undecidability, as we will see in the next section. In this section, we show that deciding formulas with a single quantifier alternation is still possible if the quantifiers start with an existential quantifier. A HyperLTL formula is in the $\exists^*\forall^*$ fragment iff it is of the form $\exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m.\ \psi$. This fragment is especially interesting, because it includes implications between alternation-free formulas. The idea of the decision procedure is to eliminate the universal quantifiers by explicitly enumerating all possible interactions between the universal and existential quantifiers. This leads to an exponentially larger, but equisatisfiable $\exists^*$ formula.

▶ **Lemma 7.** *For every formula in the $\exists^*\forall^*$ fragment, there is an equisatisfiable formula in the $\exists^*$ fragment with exponential size.*

Proof. We define a function *sp* that takes a formula of the form $\exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m.\ \psi$ and yields an $\exists^*$ HyperLTL formula ψ' of size $\mathcal{O}(n^m)$ of the following shape, where $\psi[\pi'_i \backslash \pi_i]$ denotes that the trace variable π'_i in ψ is replaced by π_i:

$$\exists \pi_1 \dots \exists \pi_n.\ \bigwedge_{j_1=1}^{n} \dots \bigwedge_{j_m=1}^{n}.\ \psi[\pi'_1 \backslash \pi_{j_1}] \dots \psi[\pi'_m \backslash \pi_{j_m}]$$

Let φ be an $\exists^*\forall^*$ HyperLTL formula satisfied by some model T. Hence, there exist traces $t_1, \dots, t_n \in T$ such that $\{t_1, \dots, t_n\}$ satisfies $sp(\varphi)$. Assume $sp(\varphi)$ is satisfied by some model T'. Since *sp* covers every possible combination of trace assignments for the universally quantified trace variables, $T' \models \varphi$. ◀

▶ **Example 8.** Consider the $\exists^*\forall^*$ formula $\exists \pi_1 \exists \pi_2 \forall \pi'_1 \forall \pi'_2.\ (\square a_{\pi'_1} \wedge \square b_{\pi'_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})$. Applying the construction from Lemma 7, we obtain the following $\exists^*$ formula:

$$\begin{aligned} &sp(\exists \pi_1 \exists \pi_2 \forall \pi'_1 \forall \pi'_2.\ (\square a_{\pi'_1} \wedge \square b_{\pi'_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \text{ yields :} \\ &\quad \exists \pi_1 \exists \pi_2.\ ((\square a_{\pi_1} \wedge \square b_{\pi_1}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_2} \wedge \square b_{\pi_1}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_1} \wedge \square b_{\pi_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \\ &\qquad \wedge ((\square a_{\pi_2} \wedge \square b_{\pi_2}) \wedge (\square c_{\pi_1} \wedge \square d_{\pi_2})) \end{aligned}$$

Combining the construction from Lemma 7 with the satisfiability check for $\exists^*$ formulas from Section 3, we obtain an exponential-space decision procedure for the $\exists^*\forall^*$ fragment.

▶ **Theorem 9.** $\exists^*\forall^*$ *HyperLTL-SAT is EXPSPACE-complete.*

Proof. Membership in EXPSPACE follows from Lemma 7 and Lemma 5. We show EXPSPACE-hardness via a reduction from the problem whether an exponential-space bounded deterministic Turing machine T accepts an input word x. Given T and x, we construct an $\exists^*\forall^*$ HyperLTL formula φ such that T accepts x iff φ is satisfiable.

Let $T = (\Sigma, Q, q_0, F, \rightarrow)$, where Σ is the alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\rightarrow \subseteq Q \times \Sigma \times Q \times \Sigma \times \{L, R\}$ is the transition relation. We use $(q, \sigma) \rightarrow (q', \sigma', \Delta)$ to indicate that when T is in state q and it reads the input σ in the current tape cell, it changes its state to q', writes σ' in the current tape cell, and moves its head one cell to the left if $\Delta = L$ and one cell to the right if $\Delta = R$. Let $n \in \mathcal{O}(|x|)$ be such that the working tape of T has 2^n cells. We encode each letter of Σ as a valuation of a set $\vec{s} = \{s_1, \ldots, s_{k_\Sigma}\}$ of atomic propositions and each state Q as a valuation of another set $\vec{q} = \{q_1, \ldots, q_{k_Q}\}$ of atomic propositions, where k_Σ is logarithmic in $|\Sigma|$ and k_Q is logarithmic in $|Q|$. We furthermore use the valuations of a set $\vec{a} = \{a_1, \ldots, a_n\}$ to encode the position of a tape cell in a configuration of T, and the valuations of a set $\vec{h} = \{h_1, \ldots, h_n\}$ to encode the position of the head of the Turing machine. With these atomic propositions, we can represent configurations of the Turing machine as sequences of valuations of the atomic propositions. The state of the Turing machine is encoded as the valuation of $\vec{q}$ at the position indicated by $\vec{h}$. Computations of a Turing machine are sequences of configurations; we thus represent computations as traces.

We begin our encoding into HyperLTL with four quantifier-free formulas over a free trace variable π: $\varphi_{init}(\pi)$ encodes that the initial configuration represents x and q_0, and places the head in the first position of the sequence. $\varphi_{head}(\pi)$ ensures that the position of the head may only change when a new configuration begins and that the change of the position as well as the change of the state is as defined by $\rightarrow$. $\varphi_{count}(\pi)$ expresses that the addresses in $\vec{a}$ continuously count from 1 to 2^n. $\varphi_{halt}(\pi)$ expresses that the Turing machine halts eventually, i.e., the trace eventually visits a final state at the position of the head.

The more difficult part of the encoding now concerns the comparison of the tape content from one configuration to the next. We need to enforce that the tape content at the position represented by $\vec{h}$ changes as defined by $\rightarrow$, and that the content of all tape cells except for the position represented by $\vec{h}$ stays the same. For this purpose, we need to be able to memorize a position from one configuration to the next. We accomplish the "memorization" with the following trick: we introduce two existentially quantified trace variables π_{zero} and π_{one}. Let v be a new atomic proposition. We use a quantifier-free formula $\varphi_{zero/one}(\pi_{zero}, \pi_{one})$ to ensure that v is always *false* on π_{zero} and always *true* on π_{one}. We now introduce another set of n universally quantified trace variables $\pi_1, \pi_2, \ldots, \pi_n$ that will serve as memory: if one of these trace variables is bound to π_{zero} its "memory content" is 0, if it is bound to π_{one} its memory content is 1. We add a sufficient number of universally quantified variables to memorize the position of some cell and its content. Our complete encoding of the Turing machine as a HyperLTL formula then looks, so far, as follows:

$$\begin{aligned}&\exists \pi, \pi_{zero}, \pi_{one}.\ \forall \pi_1, \pi_2, \ldots, \pi_n, \pi'_1, \pi'_2, \ldots, \pi'_{k_\Sigma}.\\ &\quad \varphi_{init}(\pi) \wedge \varphi_{head}(\pi) \wedge \varphi_{count}(\pi) \wedge \varphi_{halt}(\pi) \wedge\ \varphi_{zero/one}(\pi_{zero}, \pi_{one})\\ &\quad \wedge \psi(\pi, \pi_1, \pi_2, \ldots, \pi_n, \pi'_1, \pi'_2, \ldots, \pi'_{k_\Sigma})\end{aligned}$$

The missing requirement about the correct contents of the tape cells is encoded in the last conjunct ψ. We first ensure that all the universally quantified traces have constant values in v, i.e., v is either always *true* or always *false*. To enforce that the tape content changes at the head position, we specify in ψ that whenever we are at the head position, i.e., whenever

$a_{i,\pi} = h_{i,\pi}$ for all $i = 1, \dots, n$, then when we visit the same position in the next configuration, the tape content must be as specified by $\rightarrow$: i.e., if $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, then when $a_{i,\pi} = v_{\pi_i}$ holds again for all $i = 1, \dots, n$ during the next configuration, the tape content as represented in $\vec{s}$ must be the one defined by $\rightarrow$. To enforce that the tape content is the same at every position except that encoded in $\vec{h}$, we specify that for all positions except the head position, i.e., whenever $a_{i,\pi} \neq h_{i,\pi}$ for some $i = 1, \dots, n$, then if $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, and $s_{i,\pi} = v_{\pi'_i}$ for all $i = 1, \dots, k_\Sigma$, then the following must hold: when, during the next configuration, we visit the same position again, i.e., when again $a_{i,\pi} = v_{\pi_i}$ for all $i = 1, \dots, n$, we must also find the same tape content again, i.e., $s_{i,\pi} = v_{\pi'_i}$ for all $i = 1, \dots, k_\Sigma$.

By induction on the length of the computation prefix, we obtain that any model of the HyperLTL formula represents in π a correct computation of the Turing machine T. Since this computation must reach a final state, the model exists iff T accepts the input word x. ◀

In practice, the number of quantifiers is usually small. Often it is sufficient to reason about pairs of traces, which can be done with just two quantifiers. To reflect this observation, we define a *bounded* version of the $\exists^*\forall^*$ fragment, where the number of universal quantifiers that may occur in the HyperLTL formula is bounded by some constant $b \in \mathbb{N}$. A bounded $\exists^*\forall^*$ formula of length n with bound b can be translated to an equisatisfiable LTL formulas of size $\mathcal{O}(n^b)$. The satisfiablility problem can thus be solved in polynomial space.

▶ **Corollary 10.** *Bounded $\exists^*\forall^*$ HyperLTL-SAT is PSPACE-complete.*

Another observation that is important for the practical application of our results is that implication between alternation-free formulas is decidable. As discussed in the introduction, it frequently occurs that multiple formalizations are proposed for the same hyperproperty, and one would like to determine whether the proposals are equivalent, or whether one version is stronger than the other. A HyperLTL formula ψ *implies* a HyperLTL formula φ iff every set T of traces that satisfies ψ also satisfies φ.

To determine whether ψ implies φ, we check the satisfiability of the negation $\neg(\psi \rightarrow \varphi)$. If one formula is in the $\forall^*$ fragment and the other in the $\exists^*$ fragment, implication checking is especially easy, because the formula we obtain is alternation-free.

▶ **Example 11.** To determine if the $\forall^*$ formula $\forall\pi_1 \dots \forall\pi_n.\ \psi$ implies the $\forall^*$ formula $\forall\pi'_1 \dots \forall\pi'_m.\ \varphi$, we check the $\exists^*\forall^*$ formula $\exists\pi_1 \dots \pi_n \forall\pi'_1 \dots \pi'_m.\ \psi \wedge \neg\varphi$ for unsatisfiability.

Analogously to Theorem 9, we obtain that checking implication between two alternation-free HyperLTL formulas is EXPSPACE-complete.

▶ **Theorem 12.** *Checking implication between alternation-free HyperLTL formulas is EXPSPACE-complete.*

Proof. The upper bound of Theorem 9 applies here as well. For the lower bound, we note that the encoding in the proof of Theorem 9 is of the form

$$\exists\pi, \pi_{zero}, \pi_{one}.\ \forall\vec{\pi}'.\ \varphi_1(\pi) \wedge \varphi_2(\pi_{zero}, \pi_{one}) \wedge \psi(\pi, \vec{\pi}'),$$

which is not an implication of alternation-free formulas. We can, however, transform this formula into an equisatisfiable formula by quantifying π universally:

$$\exists\pi_{zero}, \pi_{one}.\ \forall\pi, \vec{\pi}'.\ \varphi_1(\pi) \wedge \varphi_2(\pi_{zero}, \pi_{one}) \wedge \psi(\pi, \vec{\pi}')$$

In the models of the new formula, the accepting computation of the Turing machine is simply represented on *all* traces instead of on *some* trace. The formula is satisfiable iff the following

$$\forall\pi\exists\pi_s\exists\pi'.\ \Big(\big((\dot{a},\dot{a})_{\pi_s} \vee (\dot{b},\dot{b})_{\pi_s}\big) \tag{1}$$

$$\wedge\ ((\tilde{a},\tilde{a})_{\pi_s} \vee (\tilde{b},\tilde{b})_{\pi_s})\,\mathcal{U}\,\square(\#,\#)_{\pi_s}\Big) \tag{2}$$

$$\wedge\ \diamondsuit\square(\#,\#)_{\pi} \tag{3}$$

$$\wedge\ \Big(\bigvee_{i\in\{1,2,3\}} \mathit{StoneEncoding}_i \tag{4}$$

$$\vee\ \square(\#,\#)_{\pi}\Big) \tag{5}$$

Figure 1 Reduction to HyperLTL for the PCP instance from Example 13.

implication between $\exists^*$ formulas does *not* hold:

$$\exists\pi_{zero},\pi_{one}.\ \varphi_2(\pi_{zero},\pi_{one}) \quad \text{implies} \quad \exists\pi,\vec{\pi}'.\ \neg(\varphi_1(\pi)\wedge\psi(\pi,\vec{\pi}'))$$

Hence, we have reduced the problem whether an exponential-space bounded deterministic Turing machine accepts a certain input word to the implication problem between two $\exists^*$ HyperLTL formulas. ◀

With the results of this section, we have reached the borderline of the decidable HyperLTL fragments. We will see in the next section that HyperLTL-SAT immediately becomes undecidable if the formulas contain a quantifier alternation that starts with a universal quantifier.

5 The Full Logic

We now show that any extension beyond the already considered fragments makes the satisfiability problem undecidable. We prove this with a many-one-reduction from *Post's correspondence problem* (PCP) [11] to the satisfiability of a $\forall\exists$ HyperLTL formula. In PCP, we are given two lists α and β consisting of finite words from some alphabet Σ. For example, α, with $\alpha_1 = a$, $\alpha_2 = ab$ and $\alpha_3 = bba$ and β, with $\beta_1 = baa$, $\beta_2 = aa$ and $\beta_3 = bb$, where α_i denotes the ith element of the list, and α_{i_j} denotes the jth symbol of the ith element. In this example, α_{3_1} corresponds to b. PCP is the problem to find an index sequence $(i_k)_{1\leq k\leq K}$ with $K \geq 1$ and $1 \leq i_k \leq n$ for all k, such that $\alpha_{i_1}\dots\alpha_{i_K} = \beta_{i_1}\dots\beta_{i_K}$. We denote the finite words of a PCP solution with i_α and i_β, respectively.

It is a useful intuition to think of the PCP instance as a set of n domino stones. The first stone of our example is $\frac{a}{baa}$, the second is $\frac{ab}{aa}$ and the third, and last, is $\frac{bba}{bb}$. Those stones must be arranged (where copying is allowed) to construct the same word with the α- and β-concatenations. A possible solution for this PCP instance would be $(3,2,3,1)$, since the stone sequence $\frac{bba}{bb}\ \frac{ab}{aa}\ \frac{bba}{bb}\ \frac{a}{baa}$ produces the same word, i.e., $bbaabbbaa = i_\alpha = i_\beta$. For modelling the necessary correspondence between the α and β components, we will use pairs of the PCP instance alphabet as atomic propositions, e.g., (a,b). We represent a stone as a sequence of such pairs, where the first position of the pair contains a symbol of the α component and the second position a symbol of the β component. For example, the first stone $\frac{a}{baa}$ will be represented as $(a,b),(\#,b)(\#,a)$. We will use $\#$ as a termination symbol. Since

$$\mathit{StoneEncoding}_3 = \tag{6}$$

$$\Big(\big((\dot{b},\dot{b})_\pi \wedge \bigcirc(b,b)_\pi \wedge \bigcirc\bigcirc(a,\dot{*})_\pi \wedge \bigcirc\bigcirc\bigcirc(\dot{*},\tilde{*})_\pi\big) \tag{7}$$

$$\vee\, ((\dot{b},\dot{b})_\pi \wedge \bigcirc(b,b)_\pi \wedge \bigcirc\bigcirc(a,\#)_\pi \wedge \bigcirc\bigcirc\bigcirc(\#,\#)_\pi)\Big) \tag{8}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\tilde{a},*)_\pi \rightarrow (\tilde{a},*)_{\pi'}) \tag{9}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\tilde{b},*)_\pi \rightarrow (\tilde{b},*)_{\pi'}) \tag{10}$$

$$\wedge\, \square(\bigcirc\bigcirc\bigcirc(\#,*)_\pi \rightarrow (\#,*)_{\pi'}) \tag{11}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\tilde{a})_\pi \rightarrow (*,\tilde{a})_{\pi'}) \tag{12}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\tilde{b})_\pi \rightarrow (*,\tilde{b})_{\pi'}) \tag{13}$$

$$\wedge\, \square(\bigcirc\bigcirc(*,\#)_\pi \rightarrow (*,\#)_{\pi'})\Big) \tag{14}$$

Figure 2 Formula in the reduction of the PCP instance from Example 13, encoding that a trace may start with a valid stone 3 and that there must also exist a trace where stone 3 is deleted.

the α and β component of a stone may differ in its length, a sequence of stone representations might "overlap". Therefore, we indicate the start of a new stone with a dotted symbol. For example, we can string the first stone two times together: $(\dot{a},\dot{b}),(\dot{a},b)(\#,a)(\#,\dot{b})(\#,b)(\#,a)$. In the following, we write $\tilde{a}$ if we do not care if this symbol is an a or $\dot{a}$ and use $*$ as syntactic sugar for an arbitrary symbol of the alphabet. We assume that only singletons are allowed as elements of the trace, which could be achieved by adding for every atomic proposition (y_1,y_2) the conjunction $\bigwedge_{(y_1,y_2)\neq(y,y')} \square(\neg((y_1,y_2)\wedge(y,y')))$, for all (y,y').

▶ **Example 13.** Consider, again, the following PCP instance with $\Sigma = \{a,b\}$ and two lists α, with $\alpha_1 = a$, $\alpha_2 = ab$ and $\alpha_3 = bba$ and β, with $\beta_1 = baa$, $\beta_2 = aa$ and $\beta_3 = bb$. We can reduce this PCP instance to the question whether the HyperLTL formula shown in Figure 1 is satisfiable. Let $AP \coloneqq (\{a,b,\dot{a},\dot{b}\}\cup\{\#\})^2$. The stone encoding is sketched with the example of stone 3 in Figure 2.

The subformula (1) expresses that there exists a trace that starts with $(\dot{a},\dot{a})$ or $(\dot{b},\dot{b})$. Intuitively, this means that there must exist a stone whose α and β component start with the same symbol. Subformula (2) requires that there exists a "solution" trace π_s. It ensures that the trace ends synchronously with $(\#,\#)^\omega$. Combined, this guarantees that the word constructed from the α components is equal to the word constructed from the β components, i.e., $i_\alpha = i_\beta$ for a PCP solution $i(k)$. Subformula (3) ensures that every trace eventually ends with the termination symbol $\#$. It is important to notice here that all traces besides π_s are allowed to end asynchronously.

It remains to ensure that trace π_s only consists of valid stones. This is where the $\forall\exists$ structure of the quantifier prefix comes into play. The key idea is to use a $\forall\exists$ formula to specify that for every trace with at least one stone there is another trace *with the first stone removed*. Since we check that *every* trace begins with a valid stone, this implies that all stones are valid. The encoding of stone 3 is exemplarily shown in Figure 2. The first *three* α components and the first *two* β components of the new trace are deleted. The example set shown in Figure 3 shows this behavior for π_s, which starts with stone 3. By deleting stone 3 from π_s and shifting every position accordingly, we obtain π_s'. Since π_s' starts with a valid stone, namely stone 2, it satisfies subformula (4) for $i = 2$. This requires that there exists

another trace where stone 2 is deleted analogously. This argument is repeated until the trace is reduced to $(\#,\#)^\omega$, which is the only possibility for "termination" in the sense that π_s ends synchronously with $(\#,\#)^\omega$.

Corresponding to this example, we can give a generalized reduction, establishing the undecidability of $\forall\exists$ formulas.

▶ **Theorem 14.** $\forall\exists$ *HyperLTL-SAT is undecidable.*

Proof. Let a PCP instance with $\Sigma = \{a_1, a_2, ..., a_n\}$ and two lists α and β be given. We choose our alphabet as follows: $\Sigma' = (\Sigma \cup \{\dot{a}_1, \dot{a}_2, ..., \dot{a}_n\} \cup \#)^2$, where we use the dot symbol to encode that a stone starts at this position of the trace. Again, we write $\tilde{a}$ if we do not care if this symbol is an a or $\dot{a}$ and use $*$ as syntactic sugar for an arbitrary symbol of the alphabet. We encode the idea from Example 13 in the following formula.

$$\varphi_{\text{reduc}} \coloneqq \forall\pi\exists\pi_s\exists\pi'.\ \varphi_{\text{sol}}(\pi_s) \wedge \varphi_{\text{validStone}}(\pi) \wedge \varphi_{\text{delete}}(\pi,\pi') \wedge \Diamond\Box(\#,\#)_\pi$$

- $\varphi_{\text{sol}}(\pi_s) \coloneqq (\bigvee_{i=1}^{n}(\dot{a}_i,\dot{a}_i)_{\pi_s}) \wedge (\bigvee_{i=1}^{n}(\tilde{a}_i,\tilde{a}_i)_{\pi_s})\,\mathcal{U}\,\Box(\#,\#)_{\pi_s}$
  We ensure that there exists a "solution" trace π_s, which starts pointed, i.e., where the α and β components are the same. Accordingly to PCP, we require *synchronous* "termination".
- $\varphi_{validStone}(\pi)$. This is ensured by a generalization of lines (7) and (8) of the stone encoding sketched in Figure 2.
  Every trace in the trace set starts with a valid stone. Note that we do *not* require synchronous termination in any other trace than the "solution" trace.
- $\varphi_{\text{delete}}(\pi,\pi')$. This is ensured by a generalization of lines (9) to (14) of the stone encoding sketched in Figure 2.
  By exploiting the $\forall\exists$ structure of the formula, we encode that for every trace π there exists another trace π' which is *nearly* an exact copy of π but with its first stone *removed*.

Correctness. We prove correctness of the reduction by showing that if there exists a solution, namely an index sequence $i(l)$ with $l \in \mathbb{N}$, for a PCP instance, then there exists a trace set T satisfying the resulting formula φ_{reduc} and vice versa. For the sake of readability, again, we omit the set braces around atomic propositions, since we can assume that only singletons occur.

- Assume there exists a solution i to the given PCP instance with $|i_\alpha| = |i_\beta| = k$. We can construct a trace set T by building the trace $(i_\alpha[0], i_\beta[0]) \dots (i_\alpha[k], i_\beta[k])(\#,\#)^\omega$, denoted by t_0 and adding a dot to the symbol corresponding to the new stones start. We can infer the correct placement of the dots from the solution. We distinguish two cases. If the solution is only of length 1, we add $(\#,\#)^\omega$ to T and successfully constructed a trace set satisfying the formula. Otherwise, let t_0 start with stone j. We also add one of the following traces t_1 based on t_0 to T:

  $$\text{if } |\alpha_j| = |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k], i[k])(\#,\#)^\omega$$
  $$\text{if } |\alpha_j| < |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k], i[k - |\beta_j| + |\alpha_j|]) \dots (\#, i[k])(\#,\#)^\omega$$
  $$\text{if } |\alpha_j| > |\beta_j| : (i[|\alpha_j|], i[|\beta_j|]) \dots (i[k - |\alpha_j| + |\beta_j|], i[k]) \dots (i[k], \#)(\#,\#)^\omega$$

  We repeat adding traces t_n based on the starting stone of every newly added trace t_{n-1} until we terminate with $(\#,\#)^\omega$. Note that t_{n-1} might already end asynchronously. By construction this is exactly a trace set T satisfying φ_{reduc}.

Start

π_s : $(\dot{b},\dot{b})(b,b)(a,\dot{a})(\dot{a},a)(b,\dot{b})(\dot{b},b)(b,\dot{b})(a,a)(\dot{a},a)(\#,\#)(\#,\#)\dots$

Delete stone 3

π'_s : $(\dot{a},\dot{a})(b,a)(\dot{b},\dot{b})(b,b)(a,\dot{b})(\dot{a},a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 2

π''_s : $(\dot{b},\dot{b})(b,b)(a,\dot{b})(\dot{a},a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 3

π'''_s : $(\dot{a},\dot{b})(\#,a)(\#,a)(\#,\#)(\#,\#)\dots$

Delete stone 1

π''''_s : $(\#,\#)(\#,\#)\dots$

End

Figure 3 Trace set satisfying the formula from Example 13, with omitted set braces around the atomic propositions.

Let the formula φ_{reduc} be satisfiable by a trace set T. Therefore, there exists a witness t_0 for π_s, which starts with a dot, whose α and β components are the same at all positions, and which ends *synchronously* with $(\#,\#)^\omega$. t_0 also needs to start with a valid stone, which is ensured by the stone encoding, since otherwise $t_0 \notin T$. By construction there exists a subset $T_{\text{min}} \subseteq T$ that satisfies φ_{reduc}, which contains t_0 and every trace constructed by deleting one stone after another, with the last trace being $(\#,\#)^\omega$. Because t_0 eventually terminates synchronously with $(\#,\#)$, the solution remains finite. We define a total order for the traces in T_{min} according to the number of dots or, equivalently, the number of stones. We also define a function s that maps traces to the index of their starting stone. Let $A = [t_0, t_1, \dots, t_n]$ be the list of traces in T_{min} sorted in descending order. A possible solution for the PCP instance is the index sequence $s(t_0)\ s(t_1)\dots s(t_n)$.

Since we can use the construction from Subsection 3.2, the minimal undecidable fragment of HyperLTL is, in fact, $\forall\exists$. ◀

6 Conclusion

We have analyzed the decidability and complexity of the satisfiability problem for various fragments of HyperLTL. The largest decidable fragment of HyperLTL is the EXPSPACE-complete $\exists^*\forall^*$ fragment; the alternation-free $\exists^*$ and $\forall^*$ formulas are PSPACE-complete; any fragment that contains the $\forall\exists$ formulas is undecidable. Despite the general undecidability, our results provide a strong motivation to develop a practical SAT checker for HyperLTL. The key result is the PSPACE-completeness for the alternation-free fragment and the bounded $\exists^*\forall^*$ fragment, which means that for the important class of hyperproperties that can be expressed as a HyperLTL formula with a bounded number of exclusively universal or exclusively existential quantifiers, satisfiability and implication can be decided within the same complexity class as LTL.

There are several directions for future work. An important open question concerns the extension to branching time. HyperLTL is a sublogic of the branching-time temporal logic HyperCTL* [3]. While the undecidability of HyperLTL implies that HyperCTL* is also, in

general, undecidable (this was already established in [3]), the obvious question is whether it is possible to establish decidable fragments in a similar fashion as for HyperLTL.

Another intriguing, and still unexplored, direction is the synthesis problem for HyperLTL (and HyperCTL*) specifications. In synthesis, we ask for the existence of an implementation, which is usually understood as an infinite tree that branches according to the possible inputs to a system and whose nodes are labeled with the outputs of the system. Since HyperLTL can express partial observability, the synthesis problem for HyperLTL naturally generalizes the well-studied synthesis under incomplete information [8] and the synthesis of distributed systems [10].

Finally, it will be interesting to develop a practical implementation of the constructions presented in this paper and to use this implementation to analyze the relationships between various hyperproperties studied in the literature.

References

1 Rajeev Alur, Pavol Cerný, and Steve Zdancewic. Preserving secrecy under refinement. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006. doi:10.1007/11787006_10.

2 Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, CSFW-17 2004, 28-30 June 2004, Pacific Grove, CA, USA*, pages 100–114. IEEE Computer Society, 2004. doi:10.1109/CSFW.2004.17.

3 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

4 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

5 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. *CoRR*, abs/1606.07047, 2016. URL: http://arxiv.org/abs/1606.07047.

6 Bernd Finkbeiner and Markus N. Rabe. The linear-hyper-branching spectrum of temporal logics. *it - Information Technology*, 56(6):273–279, 2014. URL: http://www.degruyter.com/view/j/itit.2014.56.issue-6/itit-2014-1067/itit-2014-1067.xml.

7 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.

8 Orna Kupferman and Moshe Vardi. Synthesis with incomplete informatio. In *Advances in Temporal Logic*, pages 109–127. Springer, 2000.

9 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

10 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA,*

October 22-24, 1990, Volume II, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

11 Emil L Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

12 Markus N. Rabe. *A Temporal Logic Approach to Information-flow Control*. PhD thesis, Saarland University, 2016.

13 A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 114–127. IEEE Computer Society, 1995. doi:10.1109/SECPRI.1995.398927.

14 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

15 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

16 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop, CSFW-16 2003, 30 June - 2 July 2003, Pacific Grove, CA, USA*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

Homogeneous Equations of Algebraic Petri Nets*

Marvin Triebel[1] and Jan Sürmeli[2]

1 **Humboldt-Universität zu Berlin**
`triebel@hu-berlin.de`
2 **Humboldt-Universität zu Berlin**
`suermeli@hu-berlin.de`

Abstract

Algebraic Petri nets are a formalism for modeling distributed systems and algorithms, describing control and data flow by combining Petri nets and algebraic specification. One way to specify correctness of an algebraic Petri net model N is to specify a *linear equation* E over the places of N based on term substitution, and coefficients from an abelian group $\mathbb{G}$. Then, E is *valid* in N iff E is valid in each reachable marking of N. Due to the expressive power of Algebraic Petri nets, validity is generally undecidable. *Stable* linear equations form a class of linear equations for which validity is decidable. *Place invariants* yield a well-understood but incomplete characterization of all stable linear equations. In this paper, we provide a complete characterization of stability for the subclass of *homogeneous* linear equations, by restricting ourselves to the interpretation of terms over the Herbrand structure without considering further equality axioms. Based thereon, we show that stability is decidable for homogeneous linear equations if $\mathbb{G}$ is a cyclic group.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, D.3.1 Formal Definitions and Theory

Keywords and phrases Algebraic Petri Nets, Invariants, Linear Equations, Validity, Stability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.14

1 Introduction

The formalism of *algebraic Petri nets* (APNs) permits to formally model both control flow and data flow of distributed systems and algorithms, extending Petri nets with concepts from algebraic specification, namely a signature together with equality axioms. Thus, APNs combine the benefits of Petri nets, such as explicit modeling of concurrency and options for structural analysis, with the ability to describe data objects on a freely chosen level of abstraction. The price to pay for this expressive power is that many important behavioral properties, such as reachability of a certain marking, are undecidable. However, there are behavioral properties that can be proven based on *structural* properties, such as invariants.

In this paper, we study a particular class of behavioral properties, namely *linear equations*. Intuitively, a linear equation E formalizes a linear correlation between the tokens on different places, requiring that each reachable marking satisfies E. More formally, an APN N is defined over a signature Σ, and the tokens are ground terms over Σ. A linear equation E has the form $\sum_{p \in P} \gamma_p \kappa_p = b_1 \mu_1 + \ldots + b_n \mu_n$, where P is the set of places, each γ_p and b_i are coefficients stemming from an abelian group, each κ_p is a term over Σ, and each μ_i is a ground term over Σ. A marking satisfies E if substituting each variable in each κ_p with the tokens on p yields an equality. *Validity* of E in N requires that each reachable marking of N satisfies E. Case

* This work was partially funded by the DFG Graduiertenkolleg 1651 (SOAMED).

studies have shown that this class of properties permits to formalize important behavioral properties of distributed systems and algorithms. Unfortunately, verifying the validity of E in N is generally infeasible. However, if E is *stable* then validity of E becomes decidable. Stability requires the preservation of E along all—not necessarily reachable—steps, that is, if a marking satisfies E, then firing a transition yields a marking satisfying E. Now, if E is stable, validity of E coincides with the initial marking satisfying E.

Place invariants yield a subclass of stable linear equations. Intuitively, a place invariant is a solution of a homogeneous system of linear equations given by the structure of N, providing the coefficients γ_p and terms κ_p—the right hand side can be chosen arbitrarily. This characterization is known to be decidable but incomplete, that is, there are stable linear equations, such that the left hand side is not given by a place invariant. A decidable, complete characterization of stability—or an undecidability proof—is still an open problem.

In this paper, we contribute to this field of study as follows:

1. We show the undecidability of validity of homogeneous equations.
2. We provide a complete characterization of stability, restricting ourselves to
   - *homogeneous* linear equations, that is, $n = 1$ and $b_1 = 0$, and
   - the interpretation of terms in the Herbrand structure, that is, assuming coincidence of syntax and semantics of a term, without considering further equality axioms for terms.
3. We show that our characterization is decidable if the coefficients stem from a cyclic group.

Section 2 recalls required notions for equations of algebraic Petri nets. We summarize our main theorems in Section 3, and prove these theorems in Section 4 and Section 5. We discuss related work in Section 6, and conclude in Section 7. Due to lack of space, the reader is referred to the companion technical report [16] for the missing proofs.

2 Formalization

2.1 Preliminaries

We write $\mathbb{Z}$ for the set of all integers, and $\mathbb{N}$ denotes the set $\{0, 1, 2, \ldots\}$ of natural numbers including 0. Let $z \in \mathbb{Z}$. Then, $|z|$ denote the absolute value.

2.1.1 Polynomials over Abelian Groups

Polynomials over abelian groups serve as a common algebraic base to formalize APNs and linear equations of APNs.

▸ **Definition 1** (Abelian Group, Scalar Product)**.** An *abelian group* $(\mathbb{G}, \oplus)$ consists of a set $\mathbb{G}$, and an associative, commutative, binary operation $\oplus$ on $\mathbb{G}$ with an identity $0_{\mathbb{G}}$, and inverses $\ominus g$ for each $g \in \mathbb{G}$. Let $z \in \mathbb{Z}$ and $a \in \mathbb{G}$. We define the *scalar product* $za \in \mathbb{G}$ by

$$za := \begin{cases} \bigoplus_{i=0}^{z} a & \text{if } z \geq 0 \\ \ominus(-za) & \text{otherwise.} \end{cases}$$

$(\mathbb{G}, \oplus)$ is *cyclic* iff there exists $a \in \mathbb{G}$, such that $\mathbb{G} = \{za \mid z \in \mathbb{Z}\}$.

Whenever clear from context, we simply write $\mathbb{G}$ for $(\mathbb{G}, \oplus)$. Examples for abelian groups are the real numbers, rational numbers, integers, and the additive group $\mathbb{Z}/n\mathbb{Z}$ of integers modulo some $n \in \mathbb{N}$. The group $\mathbb{Z}$ is infinite and cyclic, the group $\mathbb{Z}/n\mathbb{Z}$ is finite and cyclic. In contrast to that, the group of rational numbers is not cyclic.

▸ **Definition 2** (Series, Polynomial, Monomial, Empty Polynomial)**.** Let M be a set, $\mathbb{G}$ be an abelian group, and $f : M \to \mathbb{G}$ be a function. Then, f is a (linear) *series* over M and $\mathbb{G}$ with *support* $\operatorname{supp}(f) := \{m \in M \mid f(m) \neq 0_{\mathbb{G}}\}$. If $\operatorname{supp}(f)$ is finite, then f is a *polynomial*. We write $\mathbb{G}\langle M\rangle$ for the set of all polynomials over M and $\mathbb{G}$. If $\operatorname{supp}(f)$ is singleton, f is a *monomial*, and we denote f by (m, a) where $\operatorname{supp}(f) = \{m\}$ and $f(m) = a$. If $\operatorname{supp}(f) = \varnothing$, then f is *empty*, and we denote f by $0_{\mathbb{G}}$.

We lift $\oplus$ and the scalar product to $\mathbb{G}\langle M\rangle$ by pointwise application:

▸ **Definition 3** (Addition of Polynomials)**.** Let M be a set and $\mathbb{G}$ be an abelian group. For $p_1, p_2 \in \mathbb{G}\langle M\rangle$, $m \in M$, and $z \in \mathbb{Z}$, we define the polynomials $p_1 \oplus p_2$ and zp_1 over M and $\mathbb{G}$ by

$$(p_1 \oplus p_2)(m) := p_1(m) \oplus p_2(m) \ ,$$
$$(zp_1)(m) := zp_1(m) \ .$$

We lift associative binary operations from M to $\mathbb{G}\langle M\rangle \times \mathbb{Z}\langle M\rangle$ by applying the Cauchy product:

▸ **Definition 4** (Cauchy Product)**.** Let $\odot$ be an associative binary operation on a set M, $\mathbb{G}$ be an abelian group, $p_1 \in \mathbb{G}\langle M\rangle$, and $p_2 \in \mathbb{Z}\langle M\rangle$. We define the series $p_1 \odot p_2$ over M and G by

$$(p_1 \odot p_2)(m) := \bigoplus_{m = m_1 \odot m_2} \underbrace{p_2(m_2)}_{\in \mathbb{Z}} \underbrace{p_1(m_1)}_{\in \mathbb{G}} .$$

Because p_1 and p_2 are polynomials, the set $\operatorname{supp}(p_1 \odot p_2) = \{m_1 \odot m_2 \mid m_1, m_2 \in \mathbb{G}, p_1(m_1) \neq 0_{\mathbb{G}}, p_2(m_2) \neq 0\}$ is finite, and thus $p_1 \odot p_2$ is again a polynomial over M and $\mathbb{G}$.

2.1.2 Terms

For this paper, we fix a set of variables VAR, a non-empty, finite index set $\mathbb{I}$, and a signature $\Sigma = (\dot{f}_i/\mathrm{a}_i)_{i \in \mathbb{I}}$ consisting of $|\mathbb{I}|$ distinct function symbols $\dot{f}_i$ with respective arity a_i.

▸ **Definition 5** (Term)**.** For a set $V \subseteq \mathrm{VAR}$, the set Θ_V of *terms* over variables V is the smallest set satisfying the following conditions:

1. $V \subset \Theta_V$.
2. Let $i \in \mathbb{I}$, and $\theta_1, \ldots, \theta_{\mathrm{a}_i} \in \Theta_V$. Then, $\dot{f}_i(\theta_1, \ldots, \theta_{\mathrm{a}_i}) \in \Theta_V$.

The elements of $\Theta_\varnothing$ are called *ground terms*.

As usual, if $\mathrm{a}_i = 0$, we abbreviate $\dot{f}_i()$ as $\dot{f}_i$. We abbreviate the set Θ_{VAR} of all terms as Θ.

A *substitution* maps each variable to a term. A substitution is an *assignment* if it maps each variable to a ground term.

▸ **Definition 6** (Substitution, Assignment)**.** Every function $\sigma : \mathrm{VAR} \to \Theta$ is a substitution. Let $\theta \in \Theta$. The term $\theta\sigma$ is defined by:

$$\theta\sigma := \begin{cases} \sigma(\theta) & \text{if } \theta \in \mathrm{VAR} \\ \dot{f}_i(\theta_1\sigma, \ldots, \theta_{\mathrm{a}_i}\sigma) & \text{if } \theta = \dot{f}_i(\theta_1, \ldots, \theta_{\mathrm{a}_i}), i \in \mathbb{I}. \end{cases}$$

If $\sigma(x) \in \Theta_\varnothing$ for each $x \in \mathrm{VAR}$, then σ is an *assignment*, and we also write $[\![\theta]\!]_\sigma$ instead of $\theta\sigma$.

Obviously, if σ is an assignment, then $[\![\theta]\!]_\sigma \in \Theta_\varnothing$ for all $\theta \in \Theta$.

Unification is the problem of applying a substitution to terms, such that the resulting terms become identical.

▸ **Definition 7** (Unification problem, unifier, solvable)**.** A *unification problem* U is a finite subset $\{(\theta_1, \theta_1'), \ldots, (\theta_n, \theta_n')\}$ of $\Theta \times \Theta$, also denoted by $\{\theta_1 \doteq \theta_1', \ldots, \theta_n \doteq \theta_n'\}$. A substitution σ is a *unifier* for U iff for all $1 \leq i \leq n$: $\theta_i\sigma = \theta_i'\sigma$. If there exists a unifier for U, then U is *solvable*.

It is known that every solvable unification problem has a *most general unifier* (up to variants) that subsumes all other unifiers:

▸ **Lemma 8.** *Let U be a solvable unification problem. Then, there exists a unifier $\hat{\sigma}$ for U, such that: For each unifier σ for U, there exists a substitution σ' with $\sigma(x) = \hat{\sigma}(x)\sigma'$ for all $x \in \mathrm{VAR}$.*

We define a *product* on terms by means of term substitution: The product of ϱ and θ is defined by substituting every occurrence of any variable in ϱ by θ.

▸ **Definition 9** (Term Product)**.** Let $\varrho, \theta \in \Theta$ be terms, and σ be the substitution with $\sigma(x) = \theta$ for all $x \in \mathrm{VAR}$. Then, $\varrho \odot \theta := \varrho\sigma$ is the *product* of ϱ and θ.

We observe that $\odot$ is associative. If $\varrho \in \Theta_\varnothing$, then $\varrho \odot \theta = \varrho$.

We lift substitutions from terms to polynomials over terms and abelian groups by pointwise substitution and subsequent "simplification" of the polynomial:

▸ **Definition 10** (Substitutions in Polynomials over Terms)**.** Let $\mathbb{G}$ be an abelian group, and $p \in \mathbb{G}\langle\Theta\rangle$. Let σ be a substitution. We define $p\sigma \in \mathbb{G}\langle\Theta\rangle$ by

$$p\sigma(\theta) := \bigoplus_{\theta = [\![\theta']\!]_\sigma} p(\theta').$$

If σ is an assignment, we also write $[\![p]\!]_\sigma$ instead of $p\sigma$.

We observe $(\varrho \odot \theta)\sigma = \varrho \odot \theta\sigma$ for all $\varrho, \theta \in \Theta$, and $(p_1 \odot p_2)\sigma = p_1 \odot p_2\sigma$ for all $p_1, p_2 \in \mathbb{G}\langle\Theta\rangle$. Moreover, if σ is an assignment then $\mathrm{supp}([\![p]\!]_\sigma) \subseteq \Theta_\varnothing$.

2.1.3 Vectors

In this paper, a *P-vector* is a mapping from a set P into polynomials over terms and an abelian group.

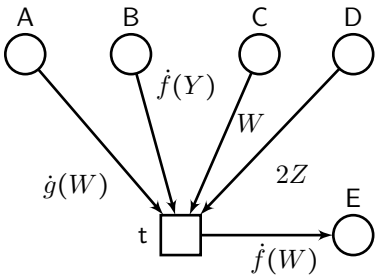
▸ **Definition 11** (P-vector)**.** Let P be a set, $(\mathbb{G}, \oplus)$ be an abelian group, and $\vec{k} : P \to \mathbb{G}\langle\Theta\rangle$. Then, $\vec{k}$ is a *P-vector* over $\mathbb{G}$. We write $\mathbb{G}\langle\Theta\rangle^P$ for the set of all P-vectors over $\mathbb{G}$. If $\vec{k}(p)$ is a monomial for each $p \in P$, then $\vec{k}$ is *simple*. If $\mathbb{G} = \mathbb{Z}$, and $\vec{k} \geq 0$ ($\vec{k} \leq 0$), then $\vec{k}$ is *semi-positive* (*semi-negative*).

In order to simplify notation, we lift the basis notions from polynomials to P-vectors:

▸ **Definition 12** (P-vectors: Support, emptiness, addition, Cauchy product, and assignments)**.** Let P be a set, $(\mathbb{G}, \oplus)$ be an abelian group, $\vec{k}, \vec{k}_1, \vec{k}_2 \in \mathbb{G}\langle\Theta\rangle^P$, and $\vec{k}' \in \mathbb{Z}\langle\Theta\rangle^P$.

- $\mathrm{supp}(\vec{k}) := \bigcup_{p \in P} \mathrm{supp}(\vec{k}(p))$ is the *support* of $\vec{k}$.
- If $\vec{k}(p) = 0_\mathbb{G}$ for all $p \in P$, then $\vec{k}$ is the empty P-vector, also denoted by $0_\mathbb{G}$.
- We define $(\vec{k}_1 \oplus \vec{k}_2)(p) := \vec{k}_1(p) \oplus \vec{k}_2(p)$ for all $p \in P$,
- We extend $\odot$ from $\mathbb{G}\langle\Theta\rangle \times \mathbb{Z}\langle\Theta\rangle \to \mathbb{G}\langle\Theta\rangle$ to $\mathbb{G}\langle\Theta\rangle^P \times \mathbb{Z}\langle\Theta\rangle^P \to \mathbb{G}\langle\Theta\rangle$ by defining $(\vec{k} \odot \vec{k}')(\theta) := \bigoplus_{p \in P} \vec{k}(p) \odot \vec{k}'(p)$ for all $\theta \in \Theta$,
- If σ is an assignment, we define $[\![\vec{k}]\!]_\sigma \in \mathbb{G}\langle\Theta\rangle^P$ by $[\![\vec{k}]\!]_\sigma(p) := [\![\vec{k}(p)]\!]_\sigma$ for all $p \in P$.

Let $\vec{k}_1 \in \mathbb{G}\langle\Theta\rangle^P$, $\vec{k}_2 \in \mathbb{Z}\langle\Theta\rangle^P$ and δ be a substitution. We observe: $\vec{k}_1 \odot (\vec{k}_2\delta) = \sum_{p \in P} \vec{k}_1(p) \odot (\vec{k}_2\delta)(p) = \sum_{p \in P} \vec{k}_1(p) \odot (\vec{k}_2(p)\delta) = \sum_{p \in P} (\vec{k}_1(p) \odot \vec{k}_2(p))\delta = (\vec{k}_1 \odot \vec{k}_2)\delta$.



(a) An APNS S_1.

$\Sigma = \{\dot{f}/1, \dot{g}/1, \dot{c}/0\}$

(b) The Signature of S_1.

	A	B	C	D	E	$\mathbb{G}$
E_1	$4\dot{f}(\mathsf{A})$	$3\dot{g}(\mathsf{B})$	$-5\dot{f}(\dot{g}(\mathsf{C}))$	$-\mathsf{D}$	0	$\mathbb{Z}$
E_2	$3\dot{c}$	0	0	2D	0	$\mathbb{Z}/7\mathbb{Z}$

(c) Homogeneous equations over $\mathbb{Z}$ and $\mathbb{Z}/7\mathbb{Z}$.

Figure 1 An APNS S_1 with equations E_1 and E_2.

2.1.4 Algebraic Petri Nets

An *algebraic Petri net structure* consists of *places* P and *transitions* T. A place $p \in P$ describes a token store, and a transition t is given by two semi-positive P-vectors $\vec{t}^-$ and $\vec{t}^+$, describing token consumption and production, respectively.

▸ **Definition 13** (Transition, algebraic Petri net structure)**.** Let $P \neq \varnothing$ be a set. A *transition* $t = (\vec{t}^-, \vec{t}^+)$ over P consists of two semi-positive simple P-vectors $\vec{t}^-, \vec{t}^+$ over $\mathbb{Z}$. We define the *effect* $\vec{t}^\Delta \in \mathbb{Z}\langle\Theta\rangle^P$ of t by $\vec{t}^\Delta := -\vec{t}^- + \vec{t}^+$. Let T be a set of transitions over P. Then, (P,T) is an *algebraic Petri net structure* (APNS). We write $\mathrm{pre}(t)$ for $\{p \in P \mid \vec{t}^-(p) > 0\}$.

Figure 1 shows an example of an APNS S_1 with transition t, places A, B, C, D and E and signature Σ using two unary function symbols $\dot{f}$ and $\dot{g}$ and the constant $\dot{c}$. Transition t consists of $\mathsf{t}^- = (\dot{g}(W) \quad \dot{f}(Y) \quad W \quad 2Z \quad 0)$ and $\mathsf{t}^+ = (0 \quad 0 \quad 0 \quad 0 \quad \dot{f}(W))$.

A *token* is a ground term, a *marking* maps each place to a multiset of tokens:

▸ **Definition 14** (Marking)**.** Let (P,T) be an APNS. Let $\vec{m} \in \mathbb{Z}\langle\Theta\rangle^P$ be a semi-positive P-vector over $\mathbb{Z}$ with $\mathrm{supp}(\vec{m}) \subseteq \Theta_\varnothing$. Then, $\vec{m}$ is a *marking* of (P,T). We write $\mathbb{Z}\langle\Theta_\varnothing\rangle^P_{\geq 0}$ for the set of all markings of (P,T).

Algebraic Petri net semantics are defined by the notion of a *step* based on the effect of a transition, and the notion of a *firing mode*:

▸ **Definition 15** (Step)**.** Let (P,T) be an APNS, $\vec{m}, \vec{m}' \in \mathbb{Z}\langle\Theta_\varnothing\rangle^P_{\geq 0}$, $t \in T$, and σ be an assignment, such that $\vec{m} \geq [\![\vec{t}^-]\!]_\sigma$ and $\vec{m}' = \vec{m} + [\![\vec{t}^\Delta]\!]_\sigma$. Then, $\vec{m}$ *enables* transition t in *firing mode* σ, denoted by $\vec{m}\,[t\sigma\rangle$, and $(\vec{m}, t, \sigma, \vec{m}')$ is a step of (P,T), denoted by $\vec{m}\,[t\sigma\rangle\,\vec{m}'$.

We remark that our definition of *enabling* does not consider additional equality axioms; permitting such axioms is left for future work.

An *algebraic Petri net* APN is an APNS together with an *initial marking*. Subsequent steps from the initial marking are *runs*, the resulting markings are *reachable*:

▸ **Definition 16** (Algebraic Petri net, run, reachable)**.** Let (P,T) be an APNS, and $\vec{m}_0 \in \mathbb{Z}\langle\Theta_\varnothing\rangle^P_{\geq 0}$. Then, $(P,T,\vec{m}_0)$ is an *algebraic Petri net* (APN). Let $\vec{m}_0\,[t_1\sigma_1\rangle\,\vec{m}_1 \ldots \vec{m}_{n-1}\,[t_n\sigma_n\rangle\,\vec{m}_n$ be a sequence of steps. Then, $(t_1,\sigma_1)\ldots(t_n,\sigma_n)$ is a *run* of $(P,T,\vec{m}_0)$ and $\vec{m}_n$ is *reachable* in $(P,T,\vec{m}_0)$.

2.2 Homogeneous Linear Equations of APNs

A homogeneous *(linear) P-equation* over a set P of places has the form $\sum_{p\in P} \gamma_p \kappa_p = 0_\mathbb{G}$, where $\gamma_p \in \mathbb{G}$ $(p \in P)$ are elements of an abelian group $\mathbb{G}$ with $0_\mathbb{G}$ as neutral element and each κ_p $(p \in P)$ is a term. Formally, a homogeneous P-equation is given by a simple P-vector.

▸ **Definition 17** (Homogeneous P-equation)**.** Let P be a set, $\mathbb{G}$ be an abelian group and $\vec{k} \in \mathbb{G}\langle\Theta\rangle^P$ be simple. Then, $\vec{k}$ induces a *homogeneous P-equation* over $\mathbb{G}$.

Figure 1 shows two equations E_1 and E_2. E_1 is over the group of integer $\mathbb{Z}$ and E_2 is over the group of integers modulo 7, $\mathbb{Z}/7\mathbb{Z}$. The table shows the simple P-vectors. For instance, $\vec{k}_1(\mathsf{A}) \odot \mathcal{X}_\mathsf{A}$ is the monomial $(\dot{f}(\mathsf{A}), 4)$.

A marking $\vec{m}$ satisfies E if replacing P by $\vec{m}$ yields an identity. A homogeneous P-equation is *valid* in an APN if it is satisfied by each reachable marking.

▸ **Definition 18** (Satisfaction, validity)**.** Let (P,T) be an APNS, $\vec{m}$ be a marking, $\mathbb{G}$ be an abelian group, and E be a homogeneous P-equation over $\mathbb{G}$ given by the simple P-vector $\vec{k} \in \mathbb{G}\langle\Theta\rangle^P$. If $\vec{k} \odot \vec{m} = 0_\mathbb{G}$, then $\vec{m}$ *satisfies* E. If each reachable marking of $(P,T,\vec{m})$ satisfies E, then E is *valid* in $(P,T,\vec{m})$.

A homogeneous P-equation is *stable* if satisfaction is preserved by all steps:

▸ **Definition 19** (Stability)**.** Let (P,T) be an APNS, $t \in T$, $\mathbb{G}$ be an abelian group, and E be a homogeneous P-equation over $\mathbb{G}$. Then, E is *t-stable* in (P,T) iff for each step $\vec{m}\,[t\sigma\rangle\,\vec{m}'$ of (P,T): If $\vec{m}$ satisfies E, then $\vec{m}'$ satisfies E.

Stability together with satisfaction in the initial marking yields validity:

▸ **Lemma 20.** *Let $(P,T,\vec{m})$ be an APN, $\mathbb{G}$ be an abelian group, and E be a homogeneous P-equation over $\mathbb{G}$ given by a simple P-vector $\vec{k} \in \mathbb{G}\langle\Theta\rangle^P$. If E is t-stable for each $t \in T$, and $\vec{m}$ satisfies E, then E is valid in $(P,T,\vec{m})$.*

A *place invariant* $\vec{k}$ is a simple P-vector such that for each $t \in T$, we have $\vec{k} \odot \vec{t}^{\,\Delta} = 0_\mathbb{G}$. Then, the homogeneous equation induced by $\vec{k}$ is stable:

▸ **Lemma 21.** *Let (P,T) be an APN, $\mathbb{G}$ be an abelian group, and E be a homogeneous P-equation over $\mathbb{G}$ given by a simple P-vector $\vec{k} \in \mathbb{G}\langle\Theta\rangle^P$. Let $t \in T$ and $\vec{k} \odot \vec{t}^{\,\Delta} = 0_\mathbb{G}$. Then, E is t-stable in (P,T).*

3 Contributions

We summarize our contributions in the form of two main theorems which we prove in the subsequent sections. Our first contribution is a proof that validity of a given P-equation in an APN is undecidable. The proof can be found in Section 4 and bases on a reduction of the halting problem of Minsky machines.

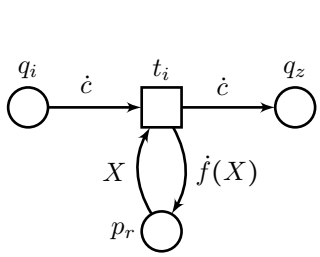
▸ **Theorem 22.** *Let $(P,T,\vec{m})$ be an APN and E a homogeneous P-equation. Then, validity of E in $(P,T,\vec{m})$ is undecidable.*

Proof. Follows from Lemma 25 and Lemma 29. ◂

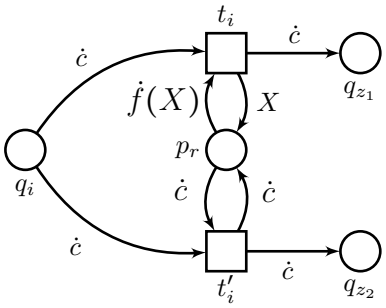
Our second contribution is a decidability proof for the stability of a homogeneous P-equation in an APNS under the assumption that the coefficients stem from a cyclic group. Here, we develop a decidable, necessary and sufficient criterion, generalizing the invariant theorem (cf. Lemma 21), in Section 5.

▸ **Theorem 23.** *Let (P,T) be an APNS and E be a homogeneous P-equation over a cyclic group, then stability of E in (P,T) is decidable.*

Proof. Follows from Lemma 44 and Lemma 46. ◂



(a) Encoding an instruction $I_i = INC(r, z)$.



(b) Encoding an instruction $I_i = JZ(r, z_1, z_2)$.

Figure 2 Encoding Minsky Machines into APNs.

4 Undecidability of Validity of Homogeneous Equations

In this section, we give short description how to encode a *Minsky Machine* [10] M into an APN N_M using the Herbrand structure. Then, the halting problem in the Minsky Machine reduces to validity of an equation. This proof technique has been used before for Petri Nets, for example in [12]. First, we recall the required notions of a Minsky machine, its states and its steps:

▸ **Definition 24** (Minsky machine)**.** A *Minsky Machine* $M = (\mathcal{I}, \mathcal{R})$ consists of number of registers $\mathcal{R} \in \mathbb{N}$ and a sequence $\mathcal{I} = I_1, \ldots, I_n$ of instructions, where each instruction $I_i \in \{INC(r, z) \mid 1 \le r \le \mathcal{R}, 1 \le z \le n\} \cup \{JZ(r, z_1, z_2) \mid 1 \le r \le \mathcal{R}, 1 \le z_1 \le n-1, 1 \le z_2 \le n-1\}$ and $I_n = HALT$.

Every tuple $(\rho, \ell) \in \mathbb{N}^{\mathcal{R}} \times \{1, \ldots, n\}$ is a *state* of M. If $I_\ell = INC(r, z)$, then $(\rho, \ell) \to (\rho', z)$ is a step in M with $\rho'(r) = \rho(r) + 1$ and $\rho'(q) = \rho(q)$ for all $q \neq r$. If $I_\ell = JZ(r, z_1, z_2)$ and $\rho(r) > 0$, then $(\rho, \ell) \to (\rho', z_1)$ is a step in M with $\rho'(r) = \rho(r) - 1$ and $\rho'(q) = \rho(q)$ for all $q \neq r$. If $I_\ell = JZ(r, z_1, z_2)$ $\rho(r) = 0$, then $(\rho, \ell) \to (\rho, z_2)$ is a step. We denote the reflexive transitive closure of $\to$ with $\to^*$.

We recall that the halting problem for Minsky machines is undecidable:

▸ **Lemma 25** ([10])**.** *Let M be a Minsky Machine. It is undecidable, whether M halts, i.e. the following problem is undecidable: $\exists \rho \in \mathbb{N}^{\mathcal{R}}$ such that $(0, 1) \to^* (\rho, n)$.*

To reduce the halting problem, we encode a Minsky Machine into an APNS.

▸ **Definition 26** (Encoding of Minsky Machine)**.** Let M be a Minsky Machine M, then the APNS N_M encodes M, if:

- The signature is $\Sigma_M = \{\dot{f}/1, \dot{c}/0\}$,
- the set of places is $P = \{p_r \mid 1 \le r \le \mathcal{R}\} \cup \{q_i \mid 1 \le i \le n\}$,
- for every INC-instruction I_i, let t_i be the transition with the pattern shown in Figure 2a,
- and for every JZ-instruction I_i let t_i and t_i' be the transitions following the pattern shown in Figure 2b.

▸ **Definition 27.** Let $(\rho, \ell) \in \mathbb{N}^{\mathcal{R}}$ be a state of M. For $x \in \mathbb{N}$, we define $\theta_x \in \Theta$ by

$$\theta_x := \begin{cases} \dot{c} & \text{if } x = 0 \\ \dot{f}(\theta_{x-1}) & \text{otherwise.} \end{cases}$$

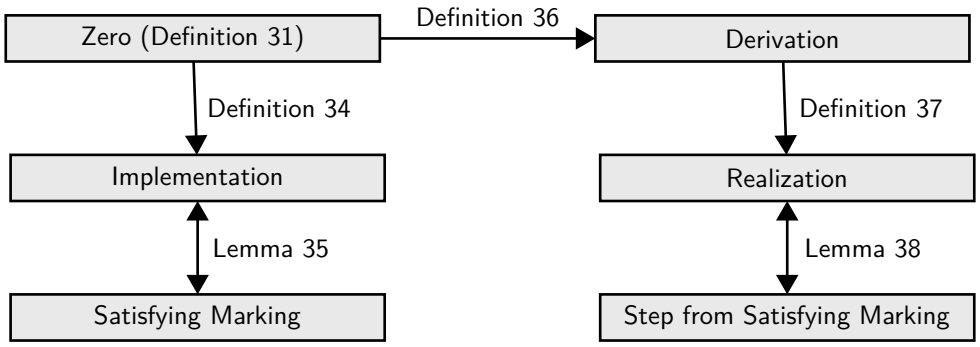


Figure 3 Overview of the proof of Theorem 23.

Then, we define the marking $\vec{m}_\ell^\rho \in \mathbb{Z}\langle\Theta_\varnothing\rangle_{\geq 0}^P$ of N_M as follows for $p \in P$ and $\theta \in \Theta$:

$$\vec{m}(p) := \begin{cases} (\dot{c}, 1) & \text{if } p = q_\ell \\ (\theta_{\rho(r)}, 1) & \text{if } p = p_r \\ 0 & \text{otherwise} \end{cases}$$

Now, we can relate the steps of a Minsky Machine M to the steps of the encoding N_M.

▸ **Lemma 28.** *Let $(\rho, \ell), (\rho', \ell')$ be states of M with $(\rho, \ell) \to (\rho', \ell')$. Then:*
1. *There exists a step $\vec{m}_\ell^\rho \,[t\sigma\rangle\, \vec{m}'$ of N_M.*
2. *If $\vec{m}_\ell^\rho \,[t\sigma\rangle\, \vec{m}'$ is a step of N_M, then $\vec{m}' = \vec{m}_{\ell'}^{\rho'}$.*

Finally, we reduce the halting problem for M to the validity of the homogeneous P-equation $q_n = 0$ in $(N_M, \vec{m}_0^1)$. The P-vector over $\mathbb{Z}$ that induces the P-equation is zero for all places $p \in P \setminus \{q_n\}$ and 1 for q_n. Inductively applying Lemma 28 reduces reachability of the $HALT$ state in M to non-emptiness of the place q_n in $(N_M, \vec{m}_0^1)$ and thus to validity of $q_n = 0$.

▸ **Lemma 29.** *The equation $q_n = 0$ is valid in $(N_M, \vec{m}_0^1)$ if and only if the Minsky Machine M does not halt.*

5 Deciding Stability of Homogeneous Equations over Cyclic Groups

In this section, we show that stability of a homogeneous P-equation E given by a simple P-vector $\vec{k}$ in an APNS $N = (P, T)$ is decidable, if $\mathbb{G}$ is a cyclic group. To this end, we identify a decidable, necessary and sufficient condition for stability, which generalizes the necessary but not sufficient condition given by the classical invariant theorem (cf. Lemma 21). We develop our condition based on the following lemma, which directly follows from applying additivity arguments to the definition of stability:

▸ **Lemma 30.** *Let $t \in T$ be a transition. Then, the following statements are equivalent:*
1. *E is t-stable.*
2. *For all steps $\vec{m} \,[t\sigma\rangle\, \vec{m}'$: If $\vec{k} \odot \vec{m} = 0_\mathbb{G}$, then $\vec{k} \odot [\![\vec{t}^\Delta]\!]_\sigma = 0_\mathbb{G}$.*

Lemma 30 generalizes Lemma 21 in the sense that we can derive Lemma 21 from Lemma 30, but not vice versa. However, the condition stated in Lemma 30 does not directly infer a decision procedure, because the set of steps $\vec{m} \,[t\sigma\rangle\, \vec{m}'$ with $\vec{k} \odot [\![\vec{t}^\Delta]\!]_\sigma = 0_\mathbb{G}$ is infinite, that

	A	B	C	D	Zero of E_1?	Zero of E_2?	$\varrho(\nu_i)$
ν_1	0	1	0	3	yes	no	$\dot{g}(\mathsf{B})$
ν_2	5	0	4	0	yes	no	$\dot{f}(\dot{g}(\mathsf{C}))$
ν_3	0	2	0	6	yes	no	$\dot{g}(\mathsf{B})$
ν_4	1	1	1	2	no	yes	$\dot{c}$
ν_5	2	0	0	4	no	yes	$\dot{c}$

(a) Zeros $\nu_1, \dots, \nu_5 \in \mathbb{N}^P$.

	A	B	C	D	impl. ν_1 for E_1?	impl. ν_2 for E_1?	impl. ν_5 for E_2?
$\vec{m}_1$	0	$\dot{c}$	0	$3\dot{g}(\dot{c})$	yes	no	no
$\vec{m}_2$	0	$2\dot{f}(\dot{c})$	0	$6\dot{g}(\dot{f}(\dot{c}))$	yes	no	no
$\vec{m}_3$	$5\dot{g}(\dot{c})$	0	$4\dot{c}$	0	no	yes	no
$\vec{m}_4$	$2\dot{g}(\dot{c})$	0	0	$4\dot{c}$	no	no	yes

(b) Implementations of zeros ν_1 (w.r.t. E_1), ν_2 (w.r.t. E_1) and ν_5 (w.r.t. E_2).

Figure 4 Examples for zeros, realizations, and implementations.

is, one has to reason about infinitely many markings $\vec{m}$ and firing modes σ. Our approach copes with this challenge by applying symbolic techniques, that is, we finitely characterize the infinite set of all such $\vec{m}$ and σ conveniently for computation. Figure 3 summarizes the notions applied in our proof: We first symbolically describe the set of E-satisfying markings by means of *zeros* and their *implementations*. Then, we *derive* symbolically described firing modes from zeros, and characterize stability by means of *realizability*.

In order to simplify notation, we fix for this section an APNS (P, T), an abelian group $\mathbb{G}$, and a homogeneous P-equation E given by a simple vector $\vec{k} \in \mathbb{G}\langle\Theta\rangle^P$. Moreover, we assume that for each $p \in P$, $\vec{k}(p)$ is the monomial (κ_p, γ_p), that is, $\gamma_p = \vec{k}(p)(\kappa_p) \in \mathbb{G}$ is the coefficient of the only term κ_p in $\mathrm{supp}(\vec{k}(p))$.

Our first goal is to abstractly characterize infinite sets of E-satisfying markings by means of a *zero*. Intuitively, an E-satisfying marking assigns "right number" of a "right kind of tokens" to each place.

▸ **Definition 31** (Zero). Let $\nu : P \to \mathbb{N}$ such that $\sum_{p\in P} \nu(p)\gamma_p = 0$. If the unification problem $U = \{\kappa_p \doteq \kappa_{p'} \mid p, p' \in P, \gamma_p, \gamma_{p'}, \nu(p), \nu(p') \neq 0\}$ is solvable, ν is a *zero* of E, and we write $\underline{\nu}$ for the most general unification of U.

We observe that 0 is always a zero. Furthermore, the sum of two zeros ν_1, ν_2 yield again $\sum_{p\in P} (\nu_1(p) + \nu_2(p)) = 0$, but the unification problem is not necessarily solvable. However, a zero may be the sum of other zeros.

Figure 4a shows some examples for zeros using the net structure and equations shown in Figure 1. In this section, we ignore the place E, as it is irrelevant for enabling t. ν_1 is a zero of E_1 as $3 - 3 = 0$, and $\dot{g}(\mathsf{B}) \doteq \mathsf{D}$ can be unified with $D \mapsto \dot{g}(\mathsf{B})$. ν_2 is a zero of E_1 as $20 - 20 = 0$ and $\mathsf{A} \mapsto \dot{g}(\mathsf{C})$ unifies $\dot{f}(\mathsf{A}) \doteq \dot{f}(\dot{g}(\mathsf{C}))$. For ν_4 and E_1 we have $4 + 3 - 5 - 2 = 0$, but it is not a zero of E_1 as $\dot{f}(\mathsf{A}) \doteq \dot{g}(\mathsf{B})$ cannot be unified. ν_5 is not a zero for E_1 as $8 - 4 \neq 0$. Regarding E_2, ν_1 and ν_2 aren't zeros as $6 \not\equiv_7 0$ and $15 \not\equiv_7 0$. ν_4 is a zero for E_2 as $3 + 4 \equiv_7 0$ and $\mathsf{D} \mapsto \dot{c}$ unifies $\dot{c} \doteq \mathsf{D}$. Finally, ν_5 is also a zero of E_2, as $6 + 8 \equiv_7 = 0$ and as for ν_4 the unification problem is solvable as for ν_4.

Because $\underline{\nu}$ is a unifier, applying $\underline{\nu}$ to κ_p yields the same result for every $p \in P$ satisfying $\gamma_p \neq 0_{\mathbb{G}}$ and $\nu(p) \neq 0$.

▸ **Lemma 32.** *Let ν be a zero. The set $\{\kappa_p\underline{\nu} \mid p \in P, \gamma_p \neq 0_{\mathbb{G}}, \nu(p) \neq 0\}$ is singleton.*

▸ **Definition 33** (Result of the unification)**.** We define $\varrho(\nu)$ by $\{\varrho(\nu)\} = \{\kappa_p\underline{\nu} \mid p \in P, \gamma_p \neq 0_{\mathbb{G}}, \nu(p) \neq 0\}$

Intuitively, an *implementation* of a zero ν is a marking which satisfies E "in the same way" as ν. Formally, we define this based on an assignment transforming the result of the unification to a marking.

▸ **Definition 34** (Implementation of a zero)**.** Let $\vec{m} \in \mathbb{Z}\langle\Theta_\varnothing\rangle^P_{\geq 0}$ be a marking and ν be a zero for E. Then, $\vec{m}$ *implements* ν, or: $\vec{m}$ is an *implementation* of ν, if for all $p \in P$ with $\nu(p) \neq 0$ and $\gamma_p \neq 0_{\mathbb{G}}$:

1. $\nu(p) = \sum_{\theta \in \mathrm{supp}(\vec{m}(p))} \vec{m}(p)(\theta)$, and
2. there exists an assignment σ, such that $\{[\![\varrho(\nu)]\!]_\sigma\} = \mathrm{supp}(\vec{k}(p) \odot \vec{m}(p))$.

As an example, in Figure 4b, the marking $\vec{m}_1$ implements ν_1 for E_1 as for assignment σ_1 with $\sigma_1(\mathsf{B}) = \dot{c}$ we have $[\![B]\!]_{\sigma_1} = \dot{c} = \mathsf{D} \odot \dot{g}(\dot{c})$. $\vec{m}_2$ implements ν_1 for E_1, because for assignment σ_2 with $\sigma_2(\mathsf{B}) = \dot{f}(\dot{c})$, we have $[\![\mathsf{B}]\!]_{\sigma_2} = \mathsf{D} \odot \dot{f}(\dot{g}(\dot{c}))$. $\vec{m}_3$ implements ν_2 for E_1, because for assignment σ_3 with $\sigma_3(\mathsf{C}) = \dot{c}$, we have $[\![\dot{f}(\dot{g}(\mathsf{C}))]\!]_{\sigma_2} = \dot{f}(\dot{g}(\mathsf{C})) = \dot{f}(A) \odot \dot{g}(\dot{c}) = \dot{f}(\dot{g}(\mathsf{C})) \odot \dot{c}$. Moreover, $\vec{m}_4$ implements ν_5 for E_2 as for assignment σ_4 with $\sigma_4(\mathsf{D}) = \dot{c}$ we have $[\![\dot{c}]\!]_{\sigma_4} = \dot{c} \odot \dot{g}(\dot{c}) = \mathsf{D} \odot \dot{c}$.

Next, we show that the set of all zeros exactly characterizes the set of all E-satisfying markings: For every term ω used by an E-satisfying marking $\vec{m}$ we can identify an implementation $\vec{m}_\omega$ of a zero. Because the set of E-satisfying markings is closed under addition, the converse also holds.

▸ **Lemma 35.** *Let $\vec{m}$ be a marking, the following are equivalent:*

1. *$\vec{k} \odot \vec{m} = 0_{\mathbb{G}}$.*
2. *There exist zeros $\nu_1, \ldots, \nu_n$ of E, and markings $\vec{m}_1, \ldots, \vec{m}_n$, such that: $\vec{m} = \sum_{1 \leq i \leq n} \vec{m}_i$ and $\vec{m}_i$ implements ν_i for all $i = 1, \ldots, n$.*

Our next goal is to abstractly describe sets of firing modes *derivable* from a set of zeros. Formally, we describe such a set of derived firing modes by a substitution, abstractly describing a way of enabling a transition.

▸ **Definition 36** (Derivable)**.** Let $t \in T$. Let $\mathcal{S}$ be a set of zeros. For every $q \in \mathrm{pre}(t)$ let $X_q \in \mathrm{VAR}$ be a fresh variable, such that X_q does not occur in E or t and $X_q = X_{q'}$ implies $q = q'$. Let $\nu_q \in S$ be a zero with $\nu_q(q) \geq 1$. Let $U = \{\varrho(\nu_q) \odot X_q \doteq \kappa_q \odot \theta_{q,t} \mid q \in \mathrm{pre}(t)\}$, where $\{\theta_{q,t}\} = \mathrm{supp}(\vec{t}^-(q))$. Let U be solvable by most general unification δ. Then, δ is *derivable* from $\mathcal{S}$.

In the example of Figure 5, we can derive δ_1 for E_1 with $\nu_\mathsf{A} = \nu_\mathsf{C} = \nu_4$ and $\nu_\mathsf{B} = \nu_\mathsf{D} = \nu_1$. For E_2, we can derive δ_2 with $\nu_\mathsf{A} = \nu_\mathsf{B} = \nu_\mathsf{C} = \nu_\mathsf{D} = \nu_5$.

A *realization* is an assignment which refines a derivable substitution:

▸ **Definition 37** (Realization)**.** Let $\mathcal{S}$ be a set of zeros and δ be derivable from $\mathcal{S}$. Then, σ is a *realization* of δ, if there exists an assignment σ' with $\sigma(X) = [\![\delta(X)]\!]_{\sigma'}$ for all $X \in \mathrm{VAR}$.

The assignment σ_1 shown in Figure 5 is a realization of δ_1. The assignment σ with $\sigma(X_\mathsf{C}) = \sigma(X_\mathsf{B}) = \dot{c}$ gives $\sigma_1(\mathsf{A}) = [\![X_\mathsf{A}]\!]_\sigma = \dot{c}$, $\sigma_1(\mathsf{B}) = [\![X_\mathsf{B}]\!]_\sigma = \dot{c}$ and $\sigma_1(\mathsf{C}) = [\![\dot{g}(X_\mathsf{B})]\!]_\sigma = \dot{g}(\dot{c})$.

	W	Y	Z	Derivable from some E_j?	$\vec{k}_j \odot (\vec{t}^{\Delta}\delta_i)$	
δ_1	X_C	X_B	$\dot{g}(X_B)$	yes, for $j = 1$	$-\dot{f}(\dot{g}(X_C)) + \dot{g}(X_B)$	$(j = 1)$
δ_2	X_C	X_B	$\dot{c}$	yes, for $j = 2$	0	$(j = 2)$

	W	Y	Z	Realization of	$\vec{k}_1 \odot [\![\vec{t}^{\Delta}]\!]_{\sigma_1}$
σ_1	$\dot{c}$	$\dot{c}$	$\dot{g}(\dot{c})$	δ_1	$-\dot{f}(\dot{g}(\dot{c})) + \dot{g}(\dot{c})$

Figure 5 Derivable substitutions δ_1 and δ_2, and a realization σ_1 of δ_1.

Next, we show that the derived substitutions from the set of all zeros exactly characterize the set of E-satisfying, t-enabling markings: If an E-satisfying marking $\vec{m}$ enables t in firing mode σ, then σ is a realization of some derivable substitution, and vice versa:

▸ **Lemma 38.** *Let $\mathcal{S}$ be the set of all zeros and σ be an assignment. Then, the following two statements are equivalent:*

1. *There exists a marking $\vec{m}$ with: $\vec{m} \geq [\![\vec{t}^{-}]\!]_{\sigma}$ and $\vec{k} \odot \vec{m} = 0_{\mathbb{G}}$.*
2. *There exists a δ that is derivable from $\mathcal{S}$ and σ is a realization of δ.*

A derivable substitution δ generally has infinitely many realizations. We show that the choice of the realization does not matter for deciding stability.

▸ **Lemma 39.** *Let $\mathcal{S}$ be a set of zeros and δ be derivable from $\mathcal{S}$. Then, the following two statements are equivalent:*

1. $\vec{k} \odot (\vec{t}^{\Delta}\delta) = 0$
2. $\vec{k} \odot [\![\vec{t}^{\Delta}]\!]_{\sigma} = 0$ *for all σ that are realizations of δ.*

Our proof of "2.⇒1." utilizes the existence of a realization σ preserving the distinctness of terms in $\vec{k} \odot \vec{t}^{\Delta}$, that is, if two terms θ_1, θ_2 occur in $\vec{k} \odot \vec{t}^{\Delta}$ with $\theta_1\delta \neq \theta_2\delta$, then $[\![\theta_1]\!]_{\sigma} \not\equiv [\![\theta_2]\!]_{\sigma}$.

Now, we prove that t-stability can be characterized by the set of all derivable substitutions:

▸ **Lemma 40.** *Let $\mathcal{S}$ be the set of all zeros. The following are equivalent:*

1. *E is t-stable.*
2. *For all δ derivable from $\mathcal{S}$ holds: $\vec{k} \odot (\vec{t}^{\Delta}\delta) = 0$.*

In the example shown in Figure 1, E_1 is not stable. Consider the marking $\vec{m}_5 := \vec{m}_1 + \vec{m}_2 + \vec{m}_3$. There, t is enabled. But, for the firing mode σ_1, we have $\vec{k}_1 \odot \sigma_1 \not\equiv 0$. On the other hand, E_2 is stable, although we have $\vec{k}_2 \odot \vec{t}^{\Delta} \not\equiv 0$.

The following lemma proves a closure property for the derived substitutions: If one combines zeros from a set $\mathcal{S}$ to a new zero ν, then for every realizable substitution derivable from $\mathcal{S} \cup \{\nu\}$, there exists a realizable substitution derivable from $\mathcal{S}$.

▸ **Lemma 41.** *Let $\mathcal{S}$ be a set of zeros and $\nu \notin \mathcal{S}$ with $\nu = \sum_{i=1}^{n} \nu_i$ where $\nu_i \in \mathcal{S}$. Let δ be derivable from $\mathcal{S} \cup \{\nu\}$ and σ be assignments that realizes δ. Then, there exists δ' such that: δ' is derivable from $\mathcal{S}$ and σ realizes δ'.*

We observe that we can only derive finite sets of substitutions from finite sets of zeros.

▸ **Lemma 42.** *Let $\mathcal{S}$ be a finite set of zeros. The set $\{\delta : \mathrm{VAR} \to \Theta \mid \delta$ is derivable from $S\}$ is finite and computable.*

Our next goal is to combine Lemma 41 and Lemma 42. To this end, we first define the notion of a *spanning set* of zeros: A set capable of generating all zeros by means of addition.

▸ **Definition 43** (Spanning Set)**.** Let S be a set of zeros of E, such that for each zero ν of E, there exist $\nu_1, \dots, \nu_n \in S$, with $\nu(p) = \sum_{i=1}^{n} \nu_i(p)$ for all $p \in P$. Then, S is a *spanning set* (of zeros) of E.

Now, we show that given a *finite* spanning set of zeros, we can decide t-stability.

▸ **Lemma 44.** *Given a finite spanning set $\mathcal{S}$ of zeros, t-stability of E is decidable.*

Proof. By Lemma 41, for every δ that is derivable from the set of zeros, there exists a δ' derivable from $\mathcal{S}$. By Lemma 42, the set of all these δ' is finite and computable. By Lemma 40, E is stable if and only if for every δ' we have $\vec{k} \odot \vec{t}^{\Delta} \delta' = 0$, which is computable. ◂

The last step in our proof of Theorem 23 is showing that a finite spanning set of zeros can be computed if $\mathbb{G}$ is cyclic. For infinite cyclic groups, we apply that there exists a computable isomorphism into the integers. As a prerequisite, we observe that every spanning set contains every *indecomposable zero*, i.e., a zero which cannot be written as a sum of other zeros. For example, consider the zeros ν_1, ν_2 and ν_3 from Figure 4a: ν_1 and ν_2 are indecomposable, but $\nu_3 = \nu_1 + \nu_1$ is not. Thus, we show that there exists an upper bound for the coefficients of indecomposable zeros. To this end, we first show an auxiliary lemma, based on the maximum coefficient $\overline{\gamma}$, and the absolute value $\underline{\gamma}$ of the minimal coefficient in γ. In the example equation E_1 from Figure 1, we have $\overline{\gamma} = 4$ and $\underline{\gamma} = 5$. Intuitively, if the maximum constituent in a zero ν over places with negative (resp. positive) coefficients is less than $\overline{\gamma}$ (resp. $\underline{\gamma}$), then the sum of the constituents in ν is bounded by $2|P|\overline{\gamma}\underline{\gamma}$. For E_1, the upper bound is $2 \cdot 5 \cdot 4 \cdot 5 = 200$.

▸ **Lemma 45.** *Let $\nu \in \mathbb{N}^P$. Let $\eta \in \mathbb{Z}^P$ be mixed with $\sum_{p \in P} \nu(p) \cdot \eta(p) = 0$. Let $\overline{\eta} := \max\{\eta(p) \mid p \in P\}$ and $\underline{\eta} := \max\{|\eta(p)| \mid \eta(p) < 0, p \in P\}$ with:*
1. *$\max\{\nu(p) \mid \eta(p) < 0, p \in P\} < \overline{\eta}$*
2. *or $\max\{\nu(p) \mid \eta(p) > 0, p \in P\} < \underline{\eta}$.*
Then, $\sum_{p \in P} \nu(p) < 2|P|\overline{\eta}\underline{\eta}$

Finally, we show the computability of a finite spanning set of zeros. To this end, we utilize Lemma 45 to show that the sum of constituents of each indecomposable zero is bounded by $2|P|\overline{\gamma}\underline{\gamma}$: We assume a zero ν with $\sum_{p \in P} \nu(p) \geq 2|P|\overline{\gamma}\underline{\gamma}$, and show that ν decomposes into two zeros $\hat{\nu}$ and $\nu - \hat{\nu}$. Thus, extracting all zeros from the finite set of all $\nu \in \mathbb{N}^P$ with $\sum_{p \in P} \nu(p) < 2|P|\overline{\gamma}\underline{\gamma}$ yields a set of zeros containing all indecomposable zeros, and hence a finite spanning set.

▸ **Lemma 46.** *If $\mathbb{G}$ is cyclic, a finite spanning set $\mathcal{S}$ of zeros is computable.*

Proof. Assume $\vec{k}$ is semi-positive or semi-negative, then 0 is the only zero. In the following, we assume $\vec{k}$ to have mixed coefficients. We distinguish the cases whether $\mathbb{G}$ is finite or infinite.

- First case: $\mathbb{G}$ is infinite. As $\mathbb{G}$ is cyclic, there exists a computable isomorphism to $\mathbb{Z}$ (see for instance [17]). Thus, we assume w.l.o.g that $\mathbb{G} = \mathbb{Z}$. Let $\overline{\gamma} := \max\{\gamma(p) \mid p \in P\}$ and $\underline{\gamma} := \max\{|\gamma(p)| \mid \gamma(p) < 0, p \in P\}$. Let ν be a zero with $\sum_{p \in P} \nu(p) > 2|P|\overline{\gamma}\underline{\gamma}$ (*). We show that that then, there exist $\underline{p}, \overline{p} \in P$ with: $\gamma_{\overline{p}} > 0 \wedge \gamma_{\underline{p}} < 0 \wedge \nu(\overline{p}) \geq |\gamma_{\underline{p}}| \wedge \nu(\underline{p}) \geq \gamma_{\overline{p}}$. Assume the opposite: Then, $\max\{\nu(p) \mid \gamma_p < 0, p \in P\} < \overline{\gamma}$ or $\max\{\nu(p) \mid \gamma_p > 0, p \in P\} < \underline{\gamma}$. By Lemma 45, then $\sum_{p \in P} \nu(p) < 2|P|\overline{\gamma}\underline{\gamma}$, which contradicts (*).
  Now, let $\hat{\nu} : P \to \mathbb{N}$ with:

$$\hat{\nu}(p) = \begin{cases} |\gamma_{\underline{p}}| & \text{if } p = \overline{p} \\ \gamma_{\overline{p}} & \text{if } p = \underline{p} \\ 0 & \text{otherwise} \end{cases}$$

By definition, we have $\hat{\nu} \leq \nu$, moreover as $\sum_{p \in P} \hat{\nu}(p) \leq \underline{\gamma}\overline{\gamma} < \sum_{p \in P} \nu(p)$, we have $\hat{\nu} < \nu$. Let $\nu' = \nu - \hat{\nu}$. Then, $\nu' : P \to \mathbb{N}$ and $\nu' > 0$.

Now we show that $\hat{\nu}$ and ν' are zeros. For $\hat{\nu}$ we have $\sum_{p \in P} \nu(p) = |\gamma_{\underline{p}}|\gamma_{\overline{p}} + \gamma_{\underline{p}}\gamma_{\overline{p}} = -\gamma_{\underline{p}}\gamma_{\overline{p}} + \gamma_{\underline{p}}\gamma_{\overline{p}} = 0$ and accordingly $0 = \sum p \in P\nu(p) = \sum p_{\epsilon} P\hat{\nu}(p) + \sum_{p \in P} \nu'(p) = 0 + \sum_{p \in P} \nu'(p)$. It remains to show that the unification problems of $\hat{\nu}$ and ν' are solvable. We observe $\hat{\nu} \leq \nu$ ($\nu' \leq \nu$) implies that unification problem of $\hat{\nu}$ (ν') is a subset of the unification problem of ν. Thus, ν is a sum of the zeros ν' and $\hat{\nu}$.

Now, we see that $\sum_{p \in P} \hat{\nu}(p) < 2|P|\overline{\gamma}\underline{\gamma}$. Assume additionally $\sum_{p \in P} \nu'(p) \leq 2|P|\overline{\gamma}\underline{\gamma}$, then we can continue. Otherwise, if $\sum_{p \in P} \nu'(p) > 2|P|\overline{\gamma}\underline{\gamma}$, we can apply induction, as $\nu' < \nu$, Hence, ν is the sum of other zeros $\nu_1, \ldots, \nu_n$, where for each $1 \leq i \leq n$: $\sum_{p \in P} \nu_i(p) \leq 2|P|\underline{\gamma}\overline{\gamma}$. Finally, $\{\nu \in \mathbb{N}^p \mid \sum_{p \in P} |\nu(p)| \leq |2P|\overline{\gamma}\underline{\gamma}$ and ν is zero$\}$ is finite, spanning and computable.

Second Case: Let $\mathbb{G}$ be finite with order $o \in \mathbb{N} \setminus \{0\}$. As $\mathbb{G}$ is cyclic, there exists the generator $e \in \mathbb{G}$. Let $g \in \mathbb{G}$. Then, it holds that $g + oe = g$. Thus, for every $\nu : P \to \mathbb{N}$, and $p \in P$ with $\nu(p) > o$, we have $\nu(p)\gamma_p = (\nu(p) - o)\gamma_p$. Hence, for every zero ν we can find a zero ν' with $\nu'(p) \leq o$ and $\sum_{p \in P} \gamma_p \nu(p) = \sum p \in P\gamma_p \nu'(p)$. Therefore, $\{\nu \in \mathbb{N}^p \mid \nu(p) \leq o$ and ν is zero$\}$ is finite, spanning and computable. ◂

6 Related Work

APNs or similar "high level net"-formalisms are an established, expressive modeling language for distributed systems[11, 2]. Moreover, tools for Colored Petri Nets support simulation and (partial) verification [7, 8]. The idea to prove stable properties in Petri nets that use distinguishable tokens has been pursued at least since the early 80s [5]. Ever since, the class of invariants became a substantial part of Petri Net analysis [9, 2, 11]. Other stable properties for Algebraic Petri Nets have been studied in the context of Traps/Co-Traps [15]. In elementary Petri Nets (P/T-Nets), stable properties such as traps and co-traps have been studied [11] and been shown as useful for verification [11, 4]. Compared to this, the number of publications regarding stable properties in APNs is comparatively small. In the last years, Petri Net variants with distinguishable tokens gained more attention to model data in distributed systems and applying analytic methods such as [3, 6, 13].

The concept of stability has been used in other areas of research; the most similar maybe being abstract interpretation as a technique for verification of iterative programs [1]. In the context of data-aware business processes, stability has been used in a similar context, following a graph-oriented approach focusing on data modeling [14].

7 Concluding Remarks

Throughout this paper, we applied three restrictions: First, we only considered the interpretation of terms in the Herbrand structure, second, we only considered homogeneous P-equations, and third, we required for the decidability proof that the group of coefficients is cyclic.

If one chooses another structure for the interpretation of terms than the Herbrand structure, one can observe that validity and stability are preserved in one direction: If a P-equation is valid (stable) w.r.t. the Herbrand structure, then it is valid (stable) w.r.t. every generated structure. Because the Herbrand structure is a specific structure, the undecidability result (Theorem 22) could be generalized by allowing an arbitrary, but not fixed, structure. For the decidability result (Theorem 23), we observe that we can use our decision procedure as a sufficient but not necessary criterion for an arbitrary fixed structure.

The restriction to homogeneous P-equations yields that satisfying markings are closed under addition, which is not the case if one allowed arbitrary constants on the right hand side. Here, our approach of finding a finite spanning set symbolically describing all satisfying markings does not work. The main challenge for generalizing our approach is that markings have natural numbers as coefficients (in contrast to integers).

For our decidability result, we require that the coefficients stem from a cyclic group. Here, we explicitly exploit in the proofs that there exist a distinct generator element, and an isomorphism to the integers, or the integers modulo some natural number n.

References

1 Ahmed Bouajjani, Cezara Dragoi, Constantin Enea, Ahmed Rezine, and Mihaela Sighireanu. Invariant synthesis for programs manipulating lists with unbounded data. In *CAV'10*, volume 6174 of *LNCS*, pages 72–88. Springer, 2010.

2 Hartmut Ehrig and Wolfgang Reisig. An algebraic view on petri nets. *Bulletin of the EATCS*, 61, 1997.

3 Javier Esparza and Philipp Hoffmann. Reduction rules for colored workflow nets. In *FASE'16*, volume 9633 of *LNCS*, pages 342–358. Springer, 2016.

4 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. An smt-based approach to coverability analysis. In *CAV'14*, volume 8559 of *LNCS*, pages 603–619. Springer, 2014.

5 Hartmann J. Genrich and Kurt Lautenbach. Special issue semantics of concurrent computation system modelling with high-level petri nets. *Theoretical Computer Science*, 13(1):109 – 135, 1981.

6 Piotr Hofman, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for petri nets with unordered data. In *FOSSACS'16*, pages 445–461, 2016.

7 Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

8 Kurt Jensen and Lars Michael Kristensen. Colored petri nets: a graphical language for formal modeling and validation of concurrent systems. *Commun. ACM*, 58(6):61–70, 2015.

9 Gérard Memmi and Jacques Vautherin. Analysing nets by the invariant method. In *Petri Nets: Central Models and Their Properties*, pages 300–336, 1986.

10 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

11 Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

12 Pierre-Alain Reynier and Arnaud Sangnier. Weak time petri nets strike back! In *CONCUR'09*, pages 557–571, 2009.

13 Fernando Rosa-Velardo, María Martos-Salgado, and David de Frutos-Escrig. Accelerations for the coverability set of petri nets with names. *Fundam. Inform.*, 113(3-4):313–341, 2011.

14 Ognjen Savkovic, Elisa Marengo, and Werner Nutt. Query stability in monotonic data-aware business processes. In *ICDT'16*, volume 48 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

15 Karsten Schmidt. Verification of siphons and traps for algebraic petri nets. In *ICATPN'97*, pages 427–446, 1997.

16 M. Triebel and J. Sürmeli. Homogeneous Equations of Algebraic Petri Nets. *ArXiv e-prints*, June 2016. `arXiv:1606.05490`.

17 T.A. Whitelaw. *An introduction to abstract algebra*. Blackie, 1978.

Bounded Petri Net Synthesis from Modal Transition Systems is Undecidable*

Uli Schlachter

Department of Computing Science, Carl von Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany
uli.schlachter@informatik.uni-oldenburg.de

Abstract

In this paper, the synthesis of bounded Petri nets from deterministic modal transition systems is shown to be undecidable. The proof is built from three components. First, it is shown that the problem of synthesising bounded Petri nets satisfying a given formula of the conjunctive nu-calculus (a suitable fragment of the mu-calculus) is undecidable. Then, an equivalence between deterministic modal transition systems and a language-based formalism called modal specifications is developed. Finally, the claim follows from a known equivalence between the conjunctive nu-calculus and modal specifications.

1998 ACM Subject Classification F.4.1 Mathematical Logic and Formal Languages

Keywords and phrases Petri net synthesis, conjunctive nu-Calculus, modal transition systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.15

1 Introduction

Modal transition systems are a well-known and useful method for specifying systems [18, 1, 8, 17]. Petri net synthesis, or more precisely, the problem of finding an unlabelled Petri net implementing a given labelled transition system, has also been investigated since many years [3]. Petri net synthesis not only yields implementations which are correct by design, but it also allows to extract concurrency and distributability information from a sequential specification [4, 7, 21]. Since modal transition systems are extensions of labelled transition systems, it has been suggested to extend Petri net synthesis to cover modal transition systems, e.g., in [9]. However, some questions that are settled in the basic net synthesis theory are still open for such an extension. For example, the decidability status of checking whether a bounded Petri net implementing a given modal transition system exists, has been stated as unknown in [3].

In this paper, we give a negative answer to this question, by proving that it is undecidable whether a given deterministic modal transition system can be implemented by a bounded Petri net. This is done in several steps, two reductions and an equivalence. First, a counter machine, also known as Minsky machine [20], is encoded in a specification that is interpreted on a special class of Petri nets. The technique used in this step resembles constructions known from other papers, for instance [15] (but in the context of labelled Petri nets). In this first step, a variation of the model checking problem is shown to be undecidable. In a second step, the class of Petri nets used in the first step is encoded in a second specification, allowing the Petri net synthesis problem to be shown undecidable. In principle, the two steps

* This work was supported by the German Research Foundation (DFG) project ARS (Algorithms for Reengineering and Synthesis), reference number Be 1267/15-1.

would suffice for our proof, because both specifications used in them can be represented as deterministic modal transition systems. However, these representations are complex and hard to understand or reason about. Therefore, in order to simplify the representations, we add a third step using the conjunctive nu-calculus [11, 12, 13], a fragment of the modal mu-calculus [16, 2]. The first two steps are done with the conjunctive nu-calculus and in this third step we show the equivalence of the nu-calculus, modal specifications and deterministic modal transition systems to derive our main result.

Section 2 introduces the bounded execution problem for two-counter machines. In section 3, we define a class of Petri nets $N_{\text{sim}}(b_0, b_1)$ which can simulate two-counter machines whose counters stay below the bounds $(b_0, b_1) \in \mathbb{N}^2$. Section 4 characterises the reachability graphs of $N_{\text{sim}}(b_0, b_1)$ in a specification that is independent of b_0 and b_1 and derives the undecidability of the bounded Petri net synthesis problem for the conjunctive nu-calculus. We then obtain the main result, the undecidability of the bounded Petri net synthesis problem for deterministic modal transition systems, in section 5 by introducing an effective translation between deterministic modal transition systems and modal specifications, which are known to be equivalent to the conjunctive nu-calculus [11, 13].

The nu-calculus, used in all but the last section, does not allow to express, e.g., nondeterminism or unimplementable specifications. It turns out to be well-suited for research into Petri net synthesis from modal transition systems and was (to the author's knowledge) invented in [12] for the investigation of its Petri net synthesis problem. In fact, the pure and unbounded Petri net synthesis problem for the conjunctive nu-calculus was shown to be undecidable in [12]. Unfortunately, the approach described in [12] cannot be lifted to impure or bounded Petri nets because it encodes the effect of transitions on individual places via auxiliary transitions that have to be able to surpass any bounds. The effect of a transition only characterises the transition in pure Petri nets, but not in impure nets. Instead, a new approach for encoding Petri nets in the nu-calculus has been developed for the purpose of our proof. In fact, the results of [12] and the present paper are incomparable: neither implies the other.

The limitation of the expressivity of the nu-calculus tends to make it incomparable to other fragments of the mu-calculus. For example, the mu-calculus fragment considered in [10] has an undecidable model checking problem, while the nu-calculus is weaker and has a decidable model checking problem, likely even in polynomial time, because of its equivalence with deterministic modal transition systems, where model checking is possible in polynomial time [6].

2 Preliminaries

▶ **Definition 1.** A (finite, initial) *labelled transition system (lts)* is a structure $A = (Q, \Sigma, \to, q_0)$ where Q is a finite set of *states*, Σ is an alphabet, $\to \subseteq Q \times \Sigma \times Q$ is the *transition relation* and q_0 is the *initial state*. An *arc* $(q, a, q') \in \to$ is written as $q \xrightarrow{a} q'$. This is extended to words $w \in \Sigma^*$ via $q \xrightarrow{\varepsilon} q$ and $q \xrightarrow{w} q' \xrightarrow{a} q'' \Rightarrow q \xrightarrow{wa} q''$. If some $q' \in Q$ with $q \xrightarrow{w} q'$ exists, we write $q \xrightarrow{w}$. If no such q' exists, write $q \not\xrightarrow{w}$. The language of A is $L(A) = \{w \in \Sigma^* \mid q_0 \xrightarrow{w}\}$. A is *deterministic* if $\forall q, q', q'', a \colon q \xrightarrow{a} q' \wedge q \xrightarrow{a} q'' \Rightarrow q' = q''$. Two lts A_1, A_2 with $A_i = (Q_i, \Sigma, \to_i, q_{0i})$ are *isomorphic* if there exists a bijection $\xi \colon Q_1 \to Q_2$ so that $\xi(q_{01}) = q_{02}$ and for all $q, q' \in Q_1, a \in \Sigma \colon q \xrightarrow{a}_1 q' \iff \xi(q) \xrightarrow{a}_2 \xi(q')$.

▶ **Definition 2.** A (place-transition, initially marked, injectively labelled, arc-weighted) *Petri net* is a tuple $N = (P, \Sigma, F, M_0)$ where P, Σ are disjoint and finite sets of *places*, respectively *transitions*, $F \colon ((P \times \Sigma) \cup (\Sigma \times P)) \to \mathbb{N}$ is the *flow relation* and M_0 is the

initial marking where a *marking* M is a mapping $M\colon P \to \mathbb{N}$. We call $M(p)$ the number of *tokens* on place p in M. A Petri net is *pure* if it satisfies $\forall p \in P, t \in \Sigma\colon F(p,t) = 0 \vee F(t,p) = 0$, otherwise it is *impure*. The *effect* $E(t) \in \mathbb{Z}^p$ of a transition $t \in \Sigma$ is defined by $E(t)(p) = F(t,p) - F(p,t)$. A transition $t \in \Sigma$ *has no effect* if $E(t)(p) = 0$ for all p. A transition $t \in \Sigma$ is *enabled* in marking M, written $M[t\rangle$, if $\forall p \in P\colon M(p) \geq F(p,t)$. In this situation t can *fire* and leads to marking M', written $M[t\rangle M'$, if $M' = M + E(t)$. This is extended to sequences via $M[\varepsilon\rangle M$, and if $M[u\rangle M'[v\rangle M''$ then $M[uv\rangle M''$. The set of all markings reachable from M_0 is $\mathcal{E}(N) = \{M \mid \exists w \in \Sigma^*\colon M_0[w\rangle M\}$. The *reachability graph* of M is the lts $\mathrm{RG}(N) = (\mathcal{E}(N), \Sigma, \to, M_0)$ where $\to = \{(M,t,M') \mid M[t\rangle M'\}$. We call N *k-bounded* if $\forall M \in \mathcal{E}(N), p \in P\colon M(p) \leq k$ and *bounded* if such a k exists. The *language* of a Petri net is $L(N) = L(\mathrm{RG}(N))$.

The conjunctive nu-calculus is a syntactic fragment of the modal mu-calculus [16, 2]. For example the disjunction, the negation and the least fixed point $\mu X.\beta$ from mu-calculus are missing. Usually, the semantics of the mu-calculus is defined as sets of states. The language-based version used here simplifies some of the proofs in the rest of the paper. A deterministic lts satisfies a formula $\beta \in L_\nu$ in the language-based semantics if and only if it does in state-based semantics [12]. The nu-calculus has operations similar to the modal process logic [19] except that the nu-calculus cannot express non-determinism.

▶ **Definition 3** ([11, 12, 13])**.** Given a set of variables $\mathsf{Var} = \{X_1, X_2, \dots\}$ and an alphabet Σ, the set of all formulas of the *conjunctive nu-calculus* is called L_ν and is defined recursively as the least set containing $\mathtt{true}, X, \to^a, \not\to^a$ and if $\beta_1, \beta_2 \in L_\nu$, then $[a]\beta_1, \beta_1 \wedge \beta_2, \nu X.\beta_1 \in L_\nu$, where $a \in \Sigma$ and $X \in \mathsf{Var}$. A variable $X \in \mathsf{Var}$ is *free in* $\beta \in L_\nu$ if it is not under the scope of any νX, which is the *greatest fixed point operator*. The interpretation $[\![\beta]\!]^{\mathsf{val}}_L \subseteq L$ of a formula $\beta \in L_\nu$ is based on a prefix-closed language $L \subseteq \Sigma^*$ and a valuation $\mathsf{val}\colon \mathsf{Var} \to 2^L$ which assigns subsets of L to variables. $[\![\beta]\!]^{\mathsf{val}}_L$ is defined inductively over the structure of β:

$$
\begin{array}{lcllcl}
[\![\mathtt{true}]\!]^{\mathsf{val}}_L & = & L & [\![X]\!]^{\mathsf{val}}_L & = & \mathsf{val}(X) \\
[\![\to^a]\!]^{\mathsf{val}}_L & = & \{w \in L \mid wa \in L\} & [\![\not\to^a]\!]^{\mathsf{val}}_L & = & \{w \in L \mid wa \notin L\} \\
[\![[a]\beta]\!]^{\mathsf{val}}_L & = & \{w \in L \mid wa \in [\![\beta]\!]^{\mathsf{val}}_L \vee wa \notin L\} & [\![\beta_1 \wedge \beta_2]\!]^{\mathsf{val}}_L & = & [\![\beta_1]\!]^{\mathsf{val}}_L \cap [\![\beta_2]\!]^{\mathsf{val}}_L \\
[\![\nu X.\beta]\!]^{\mathsf{val}}_L & = & \bigcup\{V \subseteq L \mid [\![\beta]\!]^{\mathsf{val}(V/X)}_L \supseteq V\} & & &
\end{array}
$$

The valuation $\mathsf{val}(V/X)$ is defined by $\mathsf{val}(V/X)(X_1) = V$ for $X = X_1$ and $\mathsf{val}(V/X)(X_1) = \mathsf{val}(X_1)$ otherwise. We write $\langle a\rangle\beta$ for the formula $\to^a \wedge [a]\beta$ and by definition this means $[\![\langle a\rangle\beta]\!]^{\mathsf{val}}_L = \{w \in L \mid wa \in [\![\beta]\!]^{\mathsf{val}}_L\}$. For a word $w = a_1 \dots a_n \in \Sigma^+$ we define both $\langle w\rangle\beta = \langle a_1\rangle \dots \langle a_n\rangle\beta$ and $[w]\beta = [a_1] \dots [a_n]\beta$. Via this we can define $[V]\beta = [v_1]\beta \wedge \dots \wedge [v_n]\beta$ for a *finite* set $V \subseteq \Sigma^+$. Because the semantics of a formula β without any free variables does not depend on the valuation val, we simply write $[\![\beta]\!]_L$. A language L satisfies such a formula, written $L \models \beta$, if and only if $\varepsilon \in [\![\beta]\!]_L$ (understand this as the state reached via ε, i.e. the initial state, satisfying the formula). For a Petri net N we define $N \models \beta$ as $L(N) \models \beta$.

The operator $[a]\beta$ can be interpreted as a kind of disjunction as its meaning is $\langle a\rangle\beta \vee \not\to^a$, so it allows a continuation of a word, but does not require it. This is the analogue of a may arc in modal transition systems [18].

▶ **Example 4.** We consider the formula $[a]\to^a$ for the languages $L_1 = \{\varepsilon\}$, $L_2 = \{\varepsilon, a\}$ and $L_3 = \{\varepsilon, a, aa\}$. The interpretation is $[\![[a]\to^a]\!]_{L_i} = \{w \in L_i \mid wa \in [\![\to^a]\!]_{L_i} \vee wa \notin L_i\}$. First, we evaluate $[\![\to^a]\!]_{L_1} = \emptyset$, $[\![\to^a]\!]_{L_2} = \{\varepsilon\}$, and $[\![\to^a]\!]_{L_3} = \{\varepsilon, a\}$. Then, we derive $[\![[a]\to^a]\!]_{L_1} = \emptyset \cup \{\varepsilon\} = \{\varepsilon\}$, $[\![[a]\to^a]\!]_{L_2} = \emptyset \cup \{a\} = \{a\}$, and $[\![[a]\to^a]\!]_{L_3} = \{\varepsilon\} \cup \{aa\} = \{\varepsilon, aa\}$. The results are $L_1 \models [a]\to^a$, $L_2 \not\models [a]\to^a$ and $L_3 \models [a]\to^a$. An intuitive understanding of

$[a]{\rightarrow^a}$ is: If $a \in L$, then $aa \in L$. However, this formula only checks the beginning of words and is not influenced by a elsewhere, since, for example, $\{\varepsilon, b, ba\} \models [a]{\rightarrow^a}$ for the same reasons as $L_1 \models [a]{\rightarrow^a}$.

For an example exhibiting a fixed point operator, consider $\nu X_1.{\rightarrow^a} \wedge [a]X_1$. We begin evaluating the subformula ${\rightarrow^a} \wedge [a]X_1$. The first part requires $a \in L$. The second part states that if $a \in L$ (which is true by the first part), then $a \in \mathsf{val}(X_1)$ where X_1 is bound by the fixed point operator. For an intuitive understanding, we can substitute ${\rightarrow^a} \wedge [a]X_1$ for X_1 and get the formula ${\rightarrow^a} \wedge [a]({\rightarrow^a} \wedge [a]X_1)$. Now we see that also $aa \in L$ is required. Substituting X_1 many times results in the 'infinite formula' ${\rightarrow^a} \wedge [a]({\rightarrow^a} \wedge [a]({\rightarrow^a} \wedge [a](\dots)))$. An example for a language which satisfies this formula is produced by the regular expression a^*. Our formula is equivalent to $\nu X_1.\langle a\rangle X_1$ by the definition of $\langle a\rangle\beta$.

As another example, consider $\beta = \nu X_1.{\rightarrow^b} \wedge \langle a\rangle X_1$ and the regular language $L = L(a^*(b+\varepsilon))$. In this setting, $L \models \beta$. This language is prefix-closed, as required for $[\![\beta]\!]_L$ to be defined.

▶ **Definition 5.** A *two-counter machine* [20] is a tuple $\mathcal{C} = (\ell, \gamma)$ where $\ell \in \mathbb{N}$ is the number of states and $\gamma\colon \{1, \dots, \ell\} \to \Gamma_\ell$ maps states to instructions. Every instruction $\gamma(k) \in \Gamma_\ell$ has one of three possible forms; $\gamma(k) = \texttt{INC}_i\ k'$, $\gamma(k) = \texttt{DEC}_i\ k'\ \texttt{ELSE}\ k''$ or $\gamma(k) = \texttt{HALT}$ where $i \in \{0, 1\}$ and $k', k'' \in \{1, \dots, \ell\}$. The first instruction increments counter i and switches to state k'. The second instruction decrements counter i if possible and then switches to state k'. Otherwise, it switches to state k''. A *configuration* is a tuple $c = (k, (j_0, j_1))$ where $k \in \{1, \dots, \ell\}$ is the current state and $j_0, j_1 \in \mathbb{N}$ are the values of the counters. An *execution of* $\mathcal{C}$ is a sequence of configurations which begins with $(1, (0, 0))$ and a configuration $c = (k, (j_0, j_1))$ is followed by $c' = (k', (j_0', j_1'))$ if $\gamma(k) \neq \texttt{HALT}$ and:

- If $\gamma(k) = \texttt{INC}_i\ k''$, then $k' = k''$, $j_i' = j_i + 1$ and $j_{1-i}' = j_{1-i}$.
- If $\gamma(k) = \texttt{DEC}_i\ k''\ \texttt{ELSE}\ k'''$, then either $j_i = 0$ and $c' = (k''', (j_0, j_1))$ or $j_i \neq 0$, $k' = k''$, $j_i' = j_i - 1$ and $j_{1-i}' = j_{1-i}$.

The execution of $\mathcal{C}$ is unique, hence two-counter machines are deterministic. We say that $\mathcal{C}$ *halts* if a configuration with an instruction $\texttt{HALT}$ is reached. An execution is called *bounded by* (b_0, b_1) if all of its configurations $(k, (j_0, j_1))$ satisfy $j_0 \leq b_0$ and $j_1 \leq b_1$. It is called *bounded* if such (b_0, b_1) exist.

The *bounded execution problem for* $\mathcal{C}$ is to decide if the execution of the two-counter machine $\mathcal{C}$ is bounded. This problem is undecidable, because two-counter machines have the same computational power as Turing machines and solving this problem would allow to decide the halting problem.

3 Simulating Two-Counter Machines with Petri nets

This section introduces a family of bounded Petri nets $N_{\text{sim}}(b_0, b_1)$ with transitions $\Sigma = \{\oplus, \ominus, \odot, \boxplus, \boxminus, \boxdot\}$ and parameters $b_0, b_1 \in \mathbb{N}$. They are defined such that a two-counter machine $\mathcal{C}$ can be simulated on the reachability graph of such a net via a formula $\Phi_\mathcal{C}$ of the conjunctive nu-calculus if and only if its execution is bounded (lemma 10).

A prototypical member of the class $N_{\text{sim}}(b_0, b_1)$ is depicted in Figure 1. The Petri nets from this class can simulate two bounded counters. The values of the counters are the number of tokens on p_0, respectively p_1. Each counter has an initial value of zero, a capacity of b_0, resp. b_1 (this is because the complement places $\overline{p_i}$ have b_i tokens initially), and can be incremented, decremented and tested for zero via transitions $\oplus$, $\ominus$ and $\odot$ (resp. $\boxplus$, $\boxminus$ and $\boxdot$). Every reachable marking M satisfies, by the structure of the net, $M(p_0) + M(\overline{p_0}) = b_0$ and $M(p_1) + M(\overline{p_1}) = b_1$. As an example of the behaviour of these Petri nets, the reachability graph $\mathrm{RG}(N_{\text{sim}}(2, 3))$ is shown in Figure 1.

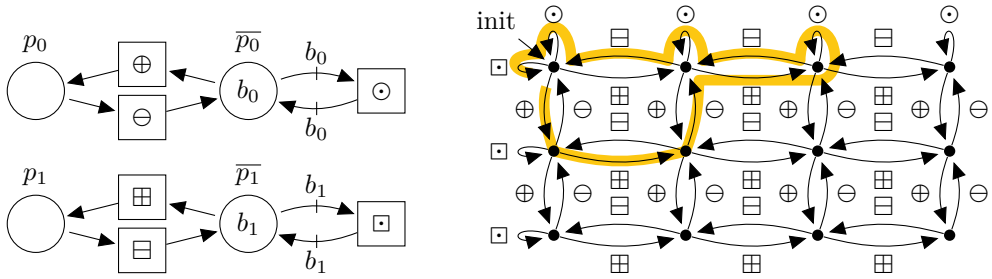


Figure 1 The family of nets $N_{\text{sim}}(b_0, b_1)$ with $b_0, b_1 \in \mathbb{N}$ is depicted on the left. On the right, the reachability graph of $N_{\text{sim}}(2, 3)$ is shown and the firing sequence $\oplus\boxplus\ominus\boxplus\odot\boxminus\odot\boxdot$ is highlighted.

▶ **Lemma 6.** *For $t \in \Sigma = \{\oplus, \ominus, \odot, \boxplus, \boxminus, \boxdot\}$ and $M \in \mathcal{E}(N_{sim}(b_0, b_1))$: $M[t\rangle M'$ iff:*

- *For $t = \oplus$ we have $M'(p_0) = M(p_0) + 1 \leq b_0$ and $M'(p_1) = M(p_1)$.*
- *For $t = \ominus$ we have $M'(p_0) = M(p_0) - 1 \geq 0$ and $M'(p_1) = M(p_1)$.*
- *For $t = \odot$ we have $M' = M$ and $M(p_0) = 0$.*
- *Analogously for $t \in \{\boxplus, \boxminus, \boxdot\}$ (the second counter).*

Proof. This lemma follows from the structure and behaviour of each net $N_{\text{sim}}(b_0, b_1)$. ◀

We define a formula $\Phi_{\mathcal{C}}$ describing a two-counter machine $\mathcal{C}$. By exploiting the structure of $N_{\text{sim}}(b_0, b_1)$, such a formula is satisfied on $N_{\text{sim}}(b_0, b_1)$ iff the execution of $\mathcal{C}$ is bounded.

▶ **Definition 7.** Given a two-counter machine $\mathcal{C} = (\ell, \gamma)$, the formula Φ_k for a state $k \in \{1, \ldots, \ell\}$ in the variables X_1 to X_ℓ is defined by:

$$\Phi_k = \begin{cases} \langle\oplus\rangle X_{k'} & \text{if } \gamma(k) = \texttt{INC}_0\ k' \\ \langle\boxplus\rangle X_{k'} & \text{if } \gamma(k) = \texttt{INC}_1\ k' \\ [\ominus]X_{k'} \wedge [\odot]X_{k''} & \text{if } \gamma(k) = \texttt{DEC}_0\ k'\ \texttt{ELSE}\ k'' \\ [\boxminus]X_{k'} \wedge [\boxdot]X_{k''} & \text{if } \gamma(k) = \texttt{DEC}_1\ k'\ \texttt{ELSE}\ k'' \\ \texttt{true} & \text{if } \gamma(k) = \texttt{HALT} \end{cases}$$

Consider the equation system $X_i = \Phi_i(X_1, \ldots, X_\ell)$ for $i \in \{1, \ldots, \ell\}$ where the $\Phi_k = \Phi_k(X_1, \ldots, X_\ell)$ are as defined above and the syntax $\beta(X)$ is used to clarify free variables. The Gaussian elimination principle (see, e.g., [2]), constructs for each k a formula Ψ_k representing a greatest fixed point solution of X_k in this system. For example, a variable X_k currently defined as $X_k = \beta(X_k)$ is eliminated by replacing $X_k = \beta(X_k)$ with $\Psi_k = \nu X_k.\beta(X_k)$ and substituting X_k in all other formulas with $\nu X_k.\beta(X_k)$. In this way, all variables are eliminated. Since the starting state of $\mathcal{C}$ is state 1, define $\Phi_{\mathcal{C}} := \Psi_1$.

Intuitively we can understand that Φ_k is fulfilled if the behaviour of state k can be simulated. The free variables are used to connect the formulas with each other. For the increment operation, the corresponding event has to be possible and afterwards the following state should be simulated. The decrement operation is more complicated, because there are two possibilities for the following state. To implement this, the structure of $N_{\text{sim}}(b_0, b_1)$ is exploited. In every reachable marking, exactly one of the transitions $\ominus$ and $\odot$ (resp. $\boxminus$ and $\boxdot$) is enabled. Thus, the $[a]$-operator can be used to express this choice. The increment operation uses the $\langle a\rangle$-operator instead, which will make the simulation fail if a counter needs to be incremented beyond the bound b_0 or b_1 of $N_{\text{sim}}(b_0, b_1)$ (because this transition is disabled).

The following examples show the construction and the Gaussian elimination principle:

▶ **Example 8.** We will construct $\Phi_{\mathcal{C}}$ for the two-counter machine $\mathcal{C} = (5, \gamma)$ with:

$\gamma(1) = \texttt{INC}_0\ 2$ $\quad$ $\gamma(3) = \texttt{DEC}_0\ 2\ \texttt{ELSE}\ 4$ $\quad$ $\gamma(5) = \texttt{HALT}$
$\gamma(2) = \texttt{INC}_1\ 3$ $\quad$ $\gamma(4) = \texttt{DEC}_1\ 3\ \texttt{ELSE}\ 5$

This machine begins by incrementing its first counter once (instruction 1). In a loop, it then increments the second counter and decrements the first (instructions 2–3). When the first counter reaches zero, another loop is done decrementing the second counter (instructions 3–4). Its execution and the corresponding operations are:

$$(1,(0,0)) \xrightarrow{\oplus} (2,(1,0)) \xrightarrow{\boxplus} (3,(1,1)) \xrightarrow{\ominus} (2,(0,1)) \xrightarrow{\boxplus} (3,(0,2)) \xrightarrow{\odot} (4,(0,2)) \xrightarrow{\boxminus}$$
$$(3,(0,1)) \xrightarrow{\odot} (4,(0,1)) \xrightarrow{\boxminus} (3,(0,0)) \xrightarrow{\odot} (4,(0,0)) \xrightarrow{\boxdot} (5,(0,0))$$

This execution is bounded by $(1,2)$, but also by $(2,3)$. The formulas for each state are shown in the following system, where a solution for X_1 is needed for $\Phi_{\mathcal{C}}$:

$$X_1 = \langle\oplus\rangle X_2 \quad X_2 = \langle\boxplus\rangle X_3 \quad X_3 = [\ominus]X_2 \wedge [\odot]X_4 \quad X_4 = [\boxminus]X_3 \wedge [\boxdot]X_5 \quad X_5 = \texttt{true}$$

The Gaussian elimination principle now eliminates variables. We begin by substituting the only uses of X_5 and X_4 with their definitions. This produces $X_4 = [\boxminus]X_3 \wedge [\boxdot]\texttt{true}$ and $X_3 = [\ominus]X_2 \wedge [\odot]([\boxminus]X_3 \wedge [\boxdot]\texttt{true})$. The variable X_3 is eliminated by substituting $\nu X_3.[\ominus]X_2 \wedge [\odot]([\boxminus]X_3 \wedge [\boxdot]\texttt{true})$ (note the added νX_3 in front of the value of X_3). This yields $X_2 = \langle\boxplus\rangle(\nu X_3.[\ominus]X_2 \wedge [\odot]([\boxminus]X_3 \wedge [\boxdot]\texttt{true}))$. Continuing by substituting X_2 and eliminating X_1 produces the result:

$$\Phi_{\mathcal{C}} = \Psi_1 = \langle\oplus\rangle(\nu X_2.\langle\boxplus\rangle(\nu X_3.[\ominus]X_2 \wedge [\odot]([\boxminus]X_3 \wedge [\boxdot]\texttt{true})))$$

As we saw above, the execution of $\mathcal{C}$ is bounded by $(2,3)$. The reader may verify that $N_{\text{sim}}(2,3) \models \Phi_{\mathcal{C}}$ due to $\oplus\boxplus\ominus\boxplus\odot\boxminus\odot\boxminus\odot\boxdot$ being in $L(N_{\text{sim}}(2,3))$ (compare Figure 1), which is the word representing the correct execution of $\mathcal{C}$.

▶ **Example 9.** An example for a machine which has a bounded execution, but does not halt, is the machine $\mathcal{C}' = (1, \gamma')$ with $\gamma'(1) = \texttt{DEC}_0\ 1\ \texttt{ELSE}\ 1$. This machine loops in its initial configuration. Its formula is $\Phi_{\mathcal{C}'} = \nu X_1.[\ominus]X_1 \wedge [\odot]X_1$. Here we have $N_{\text{sim}}(0,0) \models \Phi_{\mathcal{C}'}$ and the 'infinite word' representing a correct simulation is $\odot^\omega$. If we modify this machine by setting $\gamma'(1) = \texttt{INC}_0\ 1$, we get $\Phi_{\mathcal{C}'} = \nu X_1.\langle\oplus\rangle X_1$. This machine's unbounded execution is represented by $\oplus^\omega$. No instance of $N_{\text{sim}}(b_0, b_1)$ allows an infinite sequence of increments.

▶ **Lemma 10.** *The execution of a two-counter machine $\mathcal{C}$ is bounded by $(b_0, b_1) \in \mathbb{N} \times \mathbb{N}$ if and only if $N_{sim}(b_0, b_1) \models \Phi_{\mathcal{C}}$.*

Proof sketch. An analogous result was shown in [12]. The main difference is in the definitions of $N_{\text{sim}}(b_0, b_1)$ and Φ_k, so that we can incorporate impure Petri nets.

We can define words $w_i \in \Sigma^*$ that contain the operations done to reach the i-th configurations in $\mathcal{C}$'s execution. Assuming $\mathcal{C}$ is bounded, by induction it follows from lemma 6 that all $w_i \in L := L(N_{\text{sim}}(b_0, b_1))$. These words can be grouped into sets V_k containing for a state k all words w_i where the i-th configuration is in state k. Show that $[\![\Phi_k]\!]_L^{\mathsf{val}} \supseteq V_k$ holds for val defined by $\mathsf{val}(X_i) = V_i$. By standard fixed point theory it follows that these values are contained in the unique greatest fixed point and so $w_1 = \varepsilon \in V_1 \subseteq [\![\Phi_1]\!]_L^{\mathsf{val}} \subseteq [\![\Phi_{\mathcal{C}}]\!]_L$, which was to show.

Reachable markings of $N_{\text{sim}}(b_0, b_1)$ are related to counter values in an obvious way. Assuming $\varepsilon \in [\![\Phi_{\mathcal{C}}]\!]_L$, it can be shown via lemma 6 that the words w_i as defined above reach markings corresponding to the i-th configuration. Thus, $\Phi_{\mathcal{C}}$ follows the operations done by $\mathcal{C}$ and since in $N_{\text{sim}}(b_0, b_1)$ only bounded markings are reachable, the execution of $\mathcal{C}$ is bounded by the same numbers. ◀

Because the bounded execution problem is undecidable, we obtain:

▶ **Corollary 11.** *Given $\mathcal{C}$, it is undecidable if $\exists b_0, b_1 \in \mathbb{N}$ such that $N_{sim}(b_0, b_1) \models \Phi_{\mathcal{C}}$.*

4 Undecidability of Petri Net Synthesis from the Nu-calculus

This section characterises the reachability graph of Petri nets $N_{\text{sim}}(b_0, b_1)$ via a formula $\Phi_{N_{\text{sim}}}$ independent of b_0 and b_1. Our goal is to show that, given a formula $\beta \in L_\nu$, the existence of a bounded Petri net satisfying β is undecidable. For this, we first want to show that if a bounded Petri net satisfies $\Phi_{N_{\text{sim}}} \wedge \Phi_{\mathcal{C}}$, then there are numbers b_0, b_1 so that $N_{\text{sim}}(b_0, b_1) \models \Phi_{\mathcal{C}}$. The undecidability then follows via corollary 11. In the following sections, various parts of $\Phi_{N_{\text{sim}}}$ are introduced. Section 4.3 then shows the result sketched above.

4.1 Auxiliary Formulas

We begin with a formula that signifies that a word $w \in \Sigma^*$ can be fired infinitely often:

$$\text{NoEffect}(w) = \nu X_1.(\langle w \rangle X_1)$$

▶ **Lemma 12.** *$N \models \text{NoEffect}(w)$ for a bounded Petri net N if and only if $M_0[w\rangle M_0$.*

Proof. By definition $[\![\text{NoEffect}(w)]\!]_L = \bigcup\{V \subseteq L \mid \{v \in L \mid vw \in V\} \supseteq V\}$. This means it is the largest subset W of L that satisfies $v \in W \Rightarrow vw \in W$. Thus, the premise $\varepsilon \in [\![\text{NoEffect}(w)]\!]_L$ is equivalent to $\{w\}^* \subseteq [\![\text{NoEffect}(w)]\!]_L$. Since always $[\![\beta]\!]_L \subseteq L$, we derive $\{w\}^* \subseteq L$. In any *bounded* Petri net it holds that $\{w\}^* \subseteq L$ iff $M_0[w\rangle M_0$. ◀

By $M_0[aa'\rangle M_0 \Rightarrow M_0 = M_0 + E(a) + E(a')$, we get that $\text{Inv}(a, a')$ expresses that transitions $a, a' \in \Sigma$ have opposite effects:

$$\text{Inv}(a, a') = \text{NoEffect}(aa')$$

▶ **Corollary 13.** *$N \models \text{Inv}(a, a')$ for a bounded Petri net N if and only if $M_0[aa'\rangle$ and $E(a) = -E(a')$.*

Next, we define a formula that requires some formula β to hold globally. Here, X_1 is a fresh variable which does not appear in β.

$$\text{Global}(\beta) = \nu X_1.([\Sigma]X_1 \wedge \beta)$$

▶ **Lemma 14.** *$\varepsilon \in [\![\text{Global}(\beta)]\!]_L^{\mathsf{val}}$ if and only if $L = [\![\beta]\!]_L^{\mathsf{val}}$.*

Proof. We begin with $[\![[\Sigma]X_1 \wedge \beta]\!]_L^{\mathsf{val}} = \{w \in [\![\beta]\!]_L^{\mathsf{val}} \mid \forall a \in \Sigma\colon wa \in \mathsf{val}(X_1) \vee wa \notin L\}$. The fixed point produces $[\![\text{Global}(\beta)]\!]_L^{\mathsf{val}} = \{w \in [\![\beta]\!]_L^{\mathsf{val}} \mid \forall a \in \Sigma\colon wa \in L \Rightarrow wa \in [\![\beta]\!]_L^{\mathsf{val}}\} = \{w \in [\![\beta]\!]_L^{\mathsf{val}} \mid \forall u \in \Sigma^*\colon wu \in L \Rightarrow wu \in [\![\beta]\!]_L^{\mathsf{val}}\}$. The following implications follow: $\varepsilon \in [\![\text{Global}(\beta)]\!]_L^{\mathsf{val}} \Rightarrow L = [\![\beta]\!]_L^{\mathsf{val}} \Rightarrow L = [\![\text{Global}(\beta)]\!]_L^{\mathsf{val}} \Rightarrow \varepsilon \in [\![\text{Global}(\beta)]\!]_L^{\mathsf{val}}$. ◀

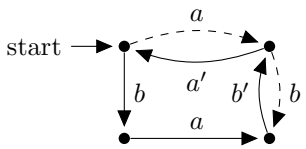


Figure 2 Illustration for $\mathrm{Swp}_h(a, b, a', b')$. Whenever the dashed arcs are present, the solid arcs are required. The interpretation begins in the upper left corner.

Next we define formulas which require two events to be swappable in the language, using events with opposite effects to 'undo' the firing of a transition. Swapping ab into ba is expressed by Swp_h and the opposite direction is added by Swp. This is extended to all combinations of the four events a, b, a' and b' by Swap. Figure 2 illustrates $\mathrm{Swp}_h(a, b, a', b')$.

$$\mathrm{Swp}_h(a, b, a', b') = \mathrm{Global}([a][b]\langle b'\rangle\langle a'\rangle\langle b\rangle\langle a\rangle \mathtt{true})$$
$$\mathrm{Swp}(a, b, a', b') = \mathrm{Swp}_h(a, b, a', b') \wedge \mathrm{Swp}_h(b, a, b', a')$$
$$\mathrm{Swap}(a, b, a', b') = \mathrm{Swp}(a, b, a', b') \wedge \mathrm{Swp}(a, b', a', b) \wedge \mathrm{Swp}(a', b, a, b') \wedge \mathrm{Swp}(a', b', a, b)$$

▶ **Lemma 15.** *Let N be a Petri net with $E(a) = -E(a')$ and $E(b) = -E(b')$. Then $N \models \mathrm{Swp}(a, b, a', b')$ if and only if $\forall w \in L(N)\colon wab \in L(N) \iff wba \in L(N)$.*

Proof. First, we have $\varepsilon \in [\![\mathrm{Swp}_h(a, b, a', b')]\!]_L$ iff for all $wab \in L$ also $wabb'a'ba \in L$ by the definition of the semantics and by lemma 14. Since a and a' (and b and b') have opposite effects, this can be restated as: For all reachable markings M, we have $M[ab\rangle \Rightarrow M[ba\rangle$. This shows both directions of the lemma. ◀

▶ **Corollary 16.** *For a net N with $E(a) = -E(a')$ and $E(b) = -E(b')$, $N \models \mathrm{Swap}(a, b, a', b')$ if and only if $\forall c \in \{a, a'\}, d \in \{b, b'\}, w \in L(N)\colon wcd \in L(N) \iff wdc \in L(N)$.*

4.2 Characterisation of our Class of Nets via the Nu-Calculus

We will characterise $N_{\mathrm{sim}}(b_0, b_1)$ by a formula $\Phi_{N_{\mathrm{sim}}} = \Phi_{\mathrm{E}} \wedge \Phi_{\mathrm{swap}} \wedge \Phi^1_{\mathrm{dec}} \wedge \Phi^2_{\mathrm{dec}} \wedge \Phi_{\mathrm{dep}} \wedge \Phi_{\mathrm{zero}}$. With Φ_{E}, we will restrict the effects of transitions, Φ_{swap} requires swaps to be possible, and Φ^1_{dec} forbids the decrement of a counter with value zero. The formula Φ^2_{dec} requires a counter decrement to be possible after every increment and the formula Φ_{dep} ensures that counters are independent. The correct behaviour of the zero-test event is required by Φ_{zero}.

$$\Phi_{\mathrm{E}} = \mathrm{Inv}(\oplus, \ominus) \wedge \mathrm{NoEffect}(\odot) \wedge \mathrm{Inv}(\boxplus, \boxminus) \wedge \mathrm{NoEffect}(\boxdot)$$

By lemma 12 and corollary 13, Φ_{E} expresses that the increment and decrement transitions have opposite effects and that the test transitions have no effect on the marking. It can easily be seen that all Petri nets in the class $N_{\mathrm{sim}}(b_0, b_1)$ have these properties.

Another property of $N_{\mathrm{sim}}(b_0, b_1)$ is that it contains two independent subnets. The order in which transitions from different subnets fire is arbitrary. By Φ_{E}, transitions $\odot$ and $\boxdot$ have no effect and do not need to be considered. For the other events, Swap is used. Additionally, in the initial marking, the decrement operations are disabled by ϕ^1_{dec}:

$$\Phi_{\mathrm{swap}} = \mathrm{Swap}(\oplus, \boxplus, \ominus, \boxminus) \qquad \Phi^1_{\mathrm{dec}} = \not\to^{\ominus} \wedge \not\to^{\boxminus}$$

▶ **Lemma 17.** *For a bounded Petri net N over Σ such that $N \models \Phi_E \wedge \Phi_{swap} \wedge \Phi^1_{dec}$, for every reachable marking M there are words $w \in \{\oplus\}^*$ and $v \in \{\boxplus\}^*$ so that $M_0[vw\rangle M$ and $M_0[wv\rangle M$.*

Proof. Since M is a reachable marking, there is a word $u \in \Sigma^*$ so that $M_0[u\rangle M$. By $N \models \Phi_{\mathrm{E}}$ and lemma 12, events $\odot$ and $\boxdot$ have no effect and can be removed from u without problems. By corollary 16 and $N \models \Phi_{\mathrm{swap}}$, we can freely swap remaining events belonging to different counters. Every subword ba of u with $a \in \{\oplus, \ominus\}$ and $b \in \{\boxplus, \boxminus\}$ is now swapped into ab. This results in a word from $\{\oplus, \ominus\}^* \cdot \{\boxplus, \boxminus\}^*$. Again by $N \models \Phi_{\mathrm{E}}$ and by corollary 13, we know that $\oplus$ and $\ominus$ (respectively $\boxplus$ and $\boxminus$) have opposite effects. Thus all subwords $\oplus\ominus$, $\ominus\oplus$, $\boxplus\boxminus$ and $\boxminus\boxplus$ can be removed. A firing sequence $u = vw$ of the required form reaching M remains and induction over corollary 16 swaps this into wv. Note that by Φ^1_{dec}, neither v nor w begin with a decrement operation and thus both only contain increments. ◀

Whenever a counter is incremented, it can be decremented afterwards:

$$\Phi^2_{\mathrm{dec}} = \mathrm{Global}([\oplus]{\rightarrow}^{\ominus}) \wedge \mathrm{Global}([\boxplus]{\rightarrow}^{\boxminus})$$

▶ **Lemma 18.** *If $N \models \Phi^2_{dec}$ then for all $w \in \Sigma^*$ it holds that $w\oplus \in L(N) \Rightarrow w\oplus\ominus \in L(N)$ and $w\boxplus \in L(N) \Rightarrow w\boxplus\boxminus \in L(N)$.*

Proof. We only show the first part. Let $w\oplus \in L(N)$. Since $L(N)$ is prefix closed and by lemma 14, we have $w \in L(N) = [\![[\oplus]{\rightarrow}^{\ominus}]\!]_{L(N)} = \{u \in L(N) \mid u\oplus \notin L(N) \vee u\oplus\ominus \in L(N)\}$. By $w\oplus \in L(N)$, w must satisfy the second part of the disjunction. Thus, $w\oplus\ominus \in L(N)$. ◀

The formulas defined so far allow the simulation of counters. However, it is still possible that the counters are interdependent and can, for example, reach the values $(2, 3)$ and $(3, 2)$, but not $(3, 3)$. Such dependencies are not present in $N_{\mathrm{sim}}(b_0, b_1)$ and we exclude them via the formula Φ_{dep}:

$$\Phi_{\mathrm{dep}} = \mathrm{Global}([\ominus][\boxplus]{\rightarrow}^{\oplus})$$

We use this formula in the next lemma to express that, if both counters can be incremented, then they can also be incremented one after another. Since L_ν does not allow implications, the situation that both counters can be incremented is captured as $[\ominus][\boxplus]$. Φ_{dep} only makes a requirement on the first counter, because the second counter follows via Φ_{swap}.

▶ **Lemma 19.** *If $N \models \Phi_E \wedge \Phi_{swap} \wedge \Phi^2_{dec} \wedge \Phi_{dep}$, then for every $w \in \Sigma^*$, we have $w\oplus \in L(N) \wedge w\boxplus \in L(N) \Rightarrow w\boxplus\oplus \in L$*

Proof. First, we can apply lemma 18 and get $w\oplus \in L(N) \Rightarrow w\oplus\ominus \in L(N)$. By $N \models \Phi_{\mathrm{E}}$, we know that $\oplus\ominus$ has no effect, so w and $w\oplus\ominus$ reach the same marking. Because of $w\boxplus \in L(N)$, this means that we also have $w\oplus\ominus\boxplus \in L(N)$. Thus, after $w\oplus$, the sequence $\ominus\boxplus$ is enabled and Φ_{dep} yields that $\oplus$ is enabled afterwards, so $w\oplus\ominus\boxplus\oplus \in L(N)$. Since $\oplus\ominus$ has no effect, we can remove this part and get $w\boxplus\oplus \in L(N)$. ◀

▶ **Definition 20.** For a Petri net N define the associated numbers $b_0(N), b_1(N) \in \mathbb{N} \cup \{\infty\}$ to be the supremum of possible values so that $M_0[\oplus^{b_0(N)}\rangle$ and $M_0[\boxplus^{b_1(N)}\rangle$.

We can now characterise the state space of a Petri net satisfying the formulas defined so far:

▶ **Lemma 21.** *For a bounded Petri net N over Σ with $N \models \Phi_E \wedge \Phi_{swap} \wedge \Phi^1_{dec} \wedge \Phi^2_{dec} \wedge \Phi_{dep}$, a marking M is reachable if and only if it is reachable via $M_0[\oplus^{j_0}\boxplus^{j_1}\rangle M$ with $j_i \leq b_i(N)$.*

Proof. First we show that all sequences $\oplus^{j_0}\boxplus^{j_1}$ are enabled. By definition we have $\oplus^{j_0} \in L(N)$ and $\boxplus^{j_1} \in L(N)$. Applying lemma 19 and corollary 16 (to swap the firing sequences into the needed form) inductively shows that $\oplus^{j_0}\boxplus^{j_1} \in L(N)$. It remains to show that no

more markings are reachable. According to lemma 17, we only have to consider markings M reachable through words of our form. Thus, it only remains to show that $j_0 \leq b_0(N)$ and $j_1 \leq b_1(N)$. However, corollary 16 can be used to bring either the part $\oplus^{j_0}$ or $\boxplus^{j_1}$ to the beginning of the word. Now, the definitions of $b_0(N)$ and $b_1(N)$ guarantee that these bounds are not exceeded. ◀

$\Phi_{\text{zero}} = \Phi^0_{\text{zero}} \wedge \Phi^1_{\text{zero}}$ requires that the zero test is enabled when a counter has the value zero, no matter which value the other counter has, and is disabled otherwise:

$$\Phi^0_{\text{zero}} = (\nu X_1.[\boxplus]X_1 \wedge \rightarrow^{\odot}) \wedge \text{Global}([\oplus]\not\rightarrow^{\odot})$$
$$\Phi^1_{\text{zero}} = (\nu X_1.[\oplus]X_1 \wedge \rightarrow^{\boxdot}) \wedge \text{Global}([\boxplus]\not\rightarrow^{\boxdot})$$

For a language satisfying this formula, the first part of Φ^0_{zero} requires that for any $\boxplus^i \in L$, also $\boxplus^i\odot \in L$. The second part states that no word may end in $\oplus\odot$ and thus the zero test is not possible after an increment.

▶ **Lemma 22.** *Let N be a bounded Petri net with $N \models \Phi_E \wedge \Phi_{swap} \wedge \Phi^1_{dec} \wedge \Phi_{zero}$ and $j_0, j_1 \in \mathbb{N}$ so that $w = \oplus^{j_0}\boxplus^{j_1} \in L(N)$. Then $w\odot \in L(N) \iff j_0 = 0$ and $w\boxdot \in L(N) \iff j_1 = 0$.*

Proof. If $j_1 = 0$, then the first part of Φ^1_{zero} requires $w\boxdot = \oplus^{j_0}\boxdot \in L(N)$ for any value of j_0. Similarly, if $j_1 > 0$, the second part requires $w\boxdot = \oplus^{j_0}\boxplus^{j_1}\boxdot \notin L(N)$. For the analogous statement for the first counter, we apply lemma 17 to get $\boxplus^{j_1}\oplus^{j_0} \in L(N)$ and make an analogous argument afterwards. ◀

▶ **Lemma 23.** *Let N be a bounded Petri net such that $N \models \Phi_E \wedge \Phi_{zero}$. Then the transition $\oplus$ has a negative effect on some place p and $\boxplus$ has a negative effect on some place q, where $p = q$ is allowed, and both $b_0(N) \neq 0 \neq b_1(N)$ and $b_0(N) \neq \infty \neq b_1(N)$ hold.*

Proof. Observe that by Φ^0_{zero}, transition $\odot$ is initially enabled, but disabled after $\oplus$ (which has to be enabled by Φ_{E}). Thus, $\oplus$ consumes tokens required by $\odot$ from a place p.

For the second part, by corollary 13, $\text{Inv}(\oplus, \ominus)$ (part of Φ_{E}) requires $\oplus$ to be enabled in the initial marking. Thus, $b_0(N) > 0$. For $b_0(N) \neq \infty$, we observe that the initial token count $M_0(p)$ on the place p from which $\oplus$ consumes tokens is finite and $\oplus$ becomes disabled eventually ($b_0(N) < \infty$). A similar argument can be made for q and $0 < b_1 < \infty$. ◀

4.3 Undecidability of Petri Net Synthesis from the Nu-calculus

With the definitions and results of the previous sections, we now prove that $\Phi_{N_{\text{sim}}} := \Phi_{\text{E}} \wedge \Phi_{\text{swap}} \wedge \Phi^1_{\text{dec}} \wedge \Phi_{\text{dep}} \wedge \Phi^2_{\text{dec}} \wedge \Phi_{\text{zero}}$ characterises the reachability graph of $N_{\text{sim}}(b_0, b_1)$:

▶ **Lemma 24.** *Let N be a bounded Petri net over the alphabet Σ with $N \models \Phi_{N_{sim}}$, then $\text{RG}(N)$ and $\text{RG}(N_{sim}(b_0(N), b_1(N)))$ are isomorphic.*

Proof. Lemma 23 shows that $b_0(N) \neq \infty \neq b_1(N)$. Thus, $N_{\text{sim}}(b_0(N), b_1(N))$ is well-defined. By lemma 6 and lemma 21, all reachable markings in either net are reached via firing sequences $\oplus^{j_0}\boxplus^{j_1}$ with $j_0 \leq b_0(N)$ and $j_1 \leq b_1(N)$. By lemma 23 and by the structure of $N_{\text{sim}}(b_0(N), b_1(N))$, different such firing sequences reach different markings. Thus, we can uniquely identify markings of either net with firing sequences $\oplus^{j_0}\boxplus^{j_1}$ and use this relation as a bijection between markings.

It remains to show that this bijection preserves the firing of transitions. By the structure of $N_{\text{sim}}(b_0(N), b_1(N))$ and by lemma 22, the zero test is enabled in a marking M of either net exactly if the corresponding counter was not increased to reach M. By Φ_{E} and the structure

of $N_{\text{sim}}(b_0(N), b_1(N))$, the zero test has no effect and reaches the marking M again. By a similar argument, this time including lemma 18, a decrement is possible if the corresponding counter has a non-zero value and reaches the marking where the corresponding counter was incremented one time less often. Finally, by the structure of $N_{\text{sim}}(b_0(N), b_1(N))$ and lemma 21, a counter can be incremented as long as it is below its maximal value.

Altogether this shows that we have an isomorphism between $\text{RG}(N_{\text{sim}}(b_0(N), b_1(N)))$ and $\text{RG}(N)$ that preserves the firing of transitions. ◀

▶ **Theorem 25.** *It is undecidable whether there exists a bounded Petri net N with transitions $\Sigma = \{\oplus, \ominus, \odot, \boxplus, \boxminus, \boxdot\}$ so that $N \models \beta$ for a given formula $\beta \in L_\nu$ without free variables.*

Proof. We use a reduction from the problem of finding $b_0, b_1 \in \mathbb{N}$ so that $N_{\text{sim}}(b_0, b_1) \models \Phi_{\mathcal{C}}$, which is undecidable according to corollary 11. Define $\beta_{\mathcal{C}} = \Phi_{N_{\text{sim}}} \wedge \Phi_{\mathcal{C}}$. We show that $\exists N \colon N \models \beta_{\mathcal{C}} \iff \exists b_0, b_1 \in \mathbb{N} \colon N_{\text{sim}}(b_0, b_1) \models \Phi_{\mathcal{C}}$.

Assume a bounded Petri net N with $N \models \beta_{\mathcal{C}}$. By lemma 24, $\text{RG}(N)$ and $A := \text{RG}(N_{\text{sim}}(b_0(N), b_1(N)))$ are isomorphic. From this, we derive $L(\text{RG}(N)) = L(A)$ and because the semantics of L_ν are defined on languages, we arrive at $N_{\text{sim}}(b_0(N), b_1(N)) \models \beta_{\mathcal{C}}$. By definition of the conjunction, this gives us $N_{\text{sim}}(b_0(N), b_1(N)) \models \Phi_{\mathcal{C}}$. We can easily compute $b_0(N), b_1(N) \in \mathbb{N}$ via their definition.

For the other direction, assume that there are numbers $b_0, b_1 \in \mathbb{N}$ so that $N_{\text{sim}}(b_0, b_1) \models \Phi_{\mathcal{C}}$. Without loss of generality we can assume that $b_0 \neq 0 \neq b_1$, because if the execution of $\mathcal{C}$ is bounded by (b_0, b_1), it is also bounded by $(b_0 + 1, b_1 + 1)$ (apply lemma 10). It remains to show that $\forall x, y > 0 \colon N_{\text{sim}}(x, y) \models \Phi_{N_{\text{sim}}}$, because we can combine this to $\Phi_{N_{\text{sim}}} \wedge \Phi_{\mathcal{C}} = \beta_{\mathcal{C}}$ and arrive at $N_{\text{sim}}(b_0, b_1) \models \beta_{\mathcal{C}}$. To satisfy $\Phi_{N_{\text{sim}}}$, all its parts have to hold. The formula Φ_{E} only makes requirements about the effect of transitions by requiring some infinite sequences to be initially enabled (see lemma 12 and corollary 13). These sequences are possible in $N_{\text{sim}}(b_0, b_1)$, but only if $b_0 > 0$ and $b_1 > 0$, which we can assume. The rest follows directly from the semantics of $\Phi_{N_{\text{sim}}}$ and the structure of the Petri net together with corollary 16. ◀

5 Undecidability of Synthesis from Modal Transition Systems

We will show our main result in theorem 30 via an equivalence between the conjunctive nu-calculus and deterministic modal transition systems. This equivalence is established via another formalism, modal specifications, which is known to be equivalent to the conjunctive nu-calculus. Similar equivalences are known for more expressive models [5], but depend on a $\vee$-operator, which L_ν does not have, for transforming a formula to an automaton. A different approach follows.

Remember that a finite automaton $A = (Q, \Sigma, \to, F, q_0)$ is a lts equipped with a set $F \subseteq Q$ of final states. The language of A is $L_f(A) = \{w \in \Sigma^* \mid \exists q \in F \colon q_0 \xrightarrow{w} q\}$. Every regular language is the language of a unique minimal deterministic finite automaton A_L [14].

▶ **Definition 26** ([18])**.** A *modal transition system* is a tuple $\mathcal{M} = (S, \Sigma, \to_\Box, \to_\Diamond, s_0)$ where S is a finite set of *states*, Σ an alphabet, s_0 the *initial state* and $\to_\Box, \to_\Diamond \subseteq S \times \Sigma \times S$ with $\to_\Box \subseteq \to_\Diamond$ are its *must* and *may arcs*, respectively. The *maximal implementation of* $\mathcal{M}$ is the lts $\overline{\mathcal{M}} = (S, \Sigma, \to_\Diamond, s_0)$. $\mathcal{M}$ is *deterministic* if $\overline{\mathcal{M}}$ is.

A lts $A = (Q, \Sigma, \to, q_0)$ is an *implementation of* $\mathcal{M}$, written $A \models \mathcal{M}$, if a relation $R \subseteq Q \times S$ exists so that $(q_0, s_0) \in R$ and for all $(q, s) \in R$ and all $a \in \Sigma$ the following holds:

1. If $s \xrightarrow{a}_\Box s'$, then $\exists q' \in Q$ with $q \xrightarrow{a} q'$ and $(q', s') \in R$.
2. If $q \xrightarrow{a} q'$, then $\exists s' \in S$ with $s \xrightarrow{a}_\Diamond s'$ and $(q', s') \in R$.

Also, $L \models \mathcal{M}$ for a prefix-closed language L, iff there is a det. lts $A \models \mathcal{M}$ with $L = L(A)$.

▶ **Lemma 27.** *For any modal transition system $\mathcal{M} = (S, \Sigma, \to_\Box, \to_\Diamond, s_0)$ and any prefix-closed language L, both $\overline{\mathcal{M}} \models \mathcal{M}$ and $L \models \mathcal{M} \Rightarrow L \subseteq L(\overline{\mathcal{M}})$ hold. If $A \models \mathcal{M}$ for a lts $A = (Q, \Sigma, \to, q_0)$ and $\mathcal{M}$ and A are both deterministic, then $R = \{(q, s) \mid \exists w \in \Sigma^* \colon q_0 \xrightarrow{w} q \wedge s_0 \xrightarrow{w}_\Diamond s\}$ is a relation witnessing $A \models \mathcal{M}$.*

▶ **Definition 28** ([11])**.** A *modal specification* is a tuple $S_m = (\{C_a\}_{a\in\Sigma}, I)$ describing a class of languages where $C_a \subseteq \Sigma^*$ contains all words that must be extendable by an additional a, while words from $I \subseteq \Sigma^*$ are forbidden to occur at all. All C_a and I are regular languages. The *completion operator* associated to S_m is the function $C_{S_m} \colon 2^{\Sigma^*} \to 2^{\Sigma^*}$ defined by $C_{S_m}(L) = \bigcup_{a\in\Sigma}(L \cap C_a) \cdot \{a\}$. A prefix-closed language satisfies S_m, written $L \models S_m$, if and only if $C_{S_m}(L) \subseteq L$ and $L \cap I = \emptyset$. A modal specification is *incoherent* if $I \cap C_{S_m}(\Sigma^*) \neq \emptyset$.

▶ **Lemma 29** ([13])**.** *For every modal specification S_m there is an equivalent coherent modal specification S'_m.*

Proof sketch. For S'_m, modify the C_a according to $C'_a = C_a \setminus \{w \in \Sigma^* \mid wa \in I\}$. ◀

We can now prove our main result:

▶ **Theorem 30.** *Given a deterministic modal transition system $\mathcal{M}$, it is undecidable if a bounded Petri net N with $L(N) \models \mathcal{M}$ exists.*

Proof. It is known that formulas of L_ν without free variables and modal specification are equally expressive [11, 13]. Therefore, invoking theorem 25 shows that it is undecidable if for a given modal specification S_m there is a bounded Petri net N with $L(N) \models S_m$. We prove this theorem by showing that modal specification and deterministic modal transition systems are equally expressive.

Given a deterministic modal transition system $\mathcal{M} = (S, \Sigma, \to_\Box, \to_\Diamond, s_0)$, we construct a modal specification $S_m(\mathcal{M})$ so that for all prefix-closed languages $L \subseteq \Sigma^* \colon L \models \mathcal{M} \iff L \models S_m(\mathcal{M})$ as follows. The set $S(a) := \{s \in S \mid s \xrightarrow{a}_\Box\}$ is the set of all states having an outgoing must arc with label a. Via this, define $A_{\mathcal{M}}^{S(a)}$ as the finite automaton based on the lts $\overline{\mathcal{M}}$ with $S(a)$ as its set of final states. Now $C_a := L_f(A_{\mathcal{M}}^{S(a)})$ is the regular language containing words after which an a is required. Finally, define the modal specification $S_m(\mathcal{M}) := (\{C_a\}_{a\in\Sigma}, I)$ where $I := \Sigma^* \setminus L(\overline{\mathcal{M}})$ is the set of forbidden words.

Analogously, given a coherent modal specification S_m, there is a deterministic modal transition system $\mathcal{M}(S_m)$ so that for all prefix-closed languages $L \subseteq \Sigma^* \colon L \models S_m \iff L \models \mathcal{M}(S_m)$. For a regular language L, let A_L be the deterministic automaton accepting it, let $S_m = (\{C_a\}_{a\in\Sigma}, I)$ be given and let A be the synchronous product of all A_{C_a} and A_I. The synchronous product of two automata is a lts constructed by taking the Cartesian product of the sets of states and defining the transition relation element-wise, i.e. if $q \xrightarrow{a} q'$ and $s \xrightarrow{a} s'$, then $(q, s) \xrightarrow{a} (q', s')$. This can be generalized to multiple automata.

The states of $\mathcal{M}(S_m)$ are the states of A and its initial state and alphabet are kept. The may arcs $\to_\Diamond$ are based on the arcs of A. However, all arcs that reach states which belong to final states of A_I are removed. This ensures that the allowed behaviour between S_m and $\mathcal{M}(S_m)$ is the same. $\mathcal{M}(S_m)$ is deterministic because A is deterministic. For $\to_\Box$, for each $a \in \Sigma$ look at each state s containing a final state q of A_{C_a}. Because S_m is coherent, the arc $s \xrightarrow{a}_\Diamond$ was not removed in the previous step. The (unique) arc $s \xrightarrow{a} s'$ is added to $\to_\Box$.

The correctness of both constructions can be shown. For this, lemma 27 is useful. ◀

▶ **Example 31.** Figure 3 shows a deterministic modal transition system $\mathcal{M}$. We will now construct $S_m(\mathcal{M})$. We have $S(a) = S(b) = \{1\}$, since for both letters this is the only state

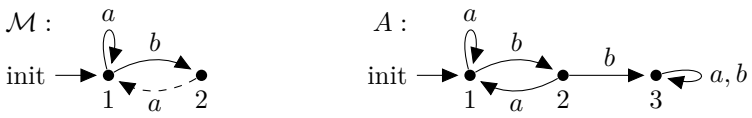


Figure 3 On the left, a modal transition system $\mathcal{M}$ is shown. May arcs are dashed and solid lines represent may and must arcs.On the right, the product automaton A being the synchronous product of A_{C_a}, A_{C_b} and A_I is depicted. For A_{C_a} and A_{C_b}, state 1 is the only final state. For A_I, this is state 3.

with an outgoing must arc, so $C_a = C_b = L_f(A_{\mathcal{M}}^{S(a)})$. The reader may easily verify that $L_f(A_{\mathcal{M}}^{S(a)}) = L((a+ba)^*)$. For I, derive $L(\overline{\mathcal{M}}) = L((a+ba)^*(\varepsilon+b))$ and $I = \Sigma^* \setminus L(\overline{\mathcal{M}}) = L((a+ba)^*bb(a+b)^*)$. This finishes the construction of $S_m(\mathcal{M}) = (\{C_e\}_{e\in\{a,b\}}, I)$

For the other direction, we begin with the modal specification $S_m := S_m(\mathcal{M})$ that was just derived and construct $\mathcal{M}(S_m)$. The synchronous product of A_{C_a}, A_{C_b} and A_I is shown in Figure 3. The may arcs of $\mathcal{M}(S_m)$ are all arcs of A except those reaching a final state of A_I. Thus, the arc from state 2 to state 3 and the two loops around state 3 are removed. For the must arcs, the arcs leaving a final state of A_{C_a} (resp. A_{C_b}) labelled with the corresponding event are considered. These are both arcs leaving state 1. The only may arc which is not also a must arc is the arc from state 2 to state 1. The constructed modal transition system $\mathcal{M}(S_m)$ is the same as $\mathcal{M}$ from Figure 3, except that it also contains an isolated state 3.

6 Conclusion

In this paper, the problem of finding a bounded Petri net implementing a given deterministic modal transition system was shown to be undecidable. This was done by first showing this problem to be undecidable for the conjunctive nu-calculus. The main result followed via an equivalence between the nu-calculus and deterministic modal transition systems. This result also settles the undecidability for the more powerful class of non-deterministic modal transition systems.

Several interesting questions are still open. The author believes that the presented approach can also be applied to the pure and bounded Petri net synthesis problem. The main difficulty is the zero-test transition, which would have to be split into two parts.

Another direction is to find decidable subcases of the Petri net synthesis problem. This could be done via structural restrictions on the modal transition systems or via limiting the class of synthesised Petri nets. For example, for a given k, the k-bounded Petri net synthesis problem from modal transition systems is decidable, because k-bounded Petri nets can be restricted to arc weights $\leq k$ without loss of generality. Only a finite number of Petri nets remain and each can be model-checked against the specification.

Possible structural restrictions on the modal transition systems might forbid must-arcs to form loops or might require each specification to be reachable via must-arcs.

Acknowledgements. I am grateful for the thoughtful remarks and suggestions by Eric Badouel, Eike Best, Valentin Spreckels, and Harro Wimmel. Also, I'd like to thank the anonymous reviewers for their helpful comments.

References

1 Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS*, 95:94–129, 2008.

2 André Arnold and Damian Niwiński. *Rudiments of μ-calculus*. North Holland, 2001.

3 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri Net Synthesis*. Springer, 2015. doi:10.1007/978-3-662-47967-4.

4 Eric Badouel, Benoît Caillaud, and Philippe Darondeau. Distributing finite automata through petri net synthesis. *Formal Aspects of Computing*, 13(6):447–470, 2002. doi:10.1007/s001650200022.

5 Nikola Benes, Benoît Delahaye, Uli Fahrenberg, Jan Kretínský, and Axel Legay. Hennessy-milner logic with greatest fixed points as a complete behavioural specification theory. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013*, volume 8052 of *LNCS*, pages 76–90. Springer, 2013. doi:10.1007/978-3-642-40184-8_7.

6 Nikola Benes, Jan Kretínský, Kim Larsen, and Jirí Srba. On determinism in modal transition systems. *Theoretical Computer Science*, 410(41):4026–4043, 2009. doi:10.1016/j.tcs.2009.06.009.

7 Eike Best and Philippe Darondeau. Petri net distributability. In Edmund M. Clarke, Irina Virbitskaite, and Andrei Voronkov, editors, *PSI 2011, Revised Selected Papers*, volume 7162 of *LNCS*, pages 1–18. Springer, 2011. doi:10.1007/978-3-642-29709-0_1.

8 Glenn Bruns. An industrial application of modal process logic. *Science of Computer Programming*, 29(1-2):3–22, 1997. doi:10.1016/S0167-6423(96)00027-5.

9 Philippe Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with Petri nets. In Eduardo Camacho, editor, *Proceedings of the 44th IEEE CDC-ECC 2005*, pages 2107–2112. IEEE, 2005. doi:10.1109/CDC.2005.1582472.

10 Javier Esparza. On the decidability of model checking for several μ-calculi and petri nets. In Sophie Tison, editor, *CAAP 1994*, volume 787 of *LNCS*, pages 115–129. Springer, 1994. doi:10.1007/BFb0017477.

11 Guillaume Feuillade. Modal specifications are a syntactic fragment of the Mu-calculus. Research Report RR-5612, INRIA, 2005. URL: https://hal.inria.fr/inria-00070396.

12 Guillaume Feuillade. *Spécification logique de réseaux de Petri*. PhD thesis, Université de Rennes I, 2005.

13 Guillaume Feuillade and Sophie Pinchinat. Modal specifications for the control theory of discrete event systems. *DEDS*, 17(2):211–232, 2007. doi:10.1007/s10626-006-0008-6.

14 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass, 1st edition, 1979.

15 Petr Jancar. Nonprimitive recursive complexity and undecidability for petri net equivalences. *Theoretical Computer Science*, 256(1-2):23–30, 2001. doi:10.1016/S0304-3975(00)00100-6.

16 Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

17 Jan Kretínský and Salomon Sickert. MoTraS: A tool for modal transition systems and their extensions. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA 2013*, volume 8172 of *LNCS*, pages 487–491. Springer, 2013. doi:10.1007/978-3-319-02444-8_41.

18 Kim Larsen. Modal specifications. In Joseph Sifakis, editor, *AVMFSS*, volume 407 of *LNCS*, pages 232–246. Springer, 1989. doi:10.1007/3-540-52148-8_19.

19 Kim Larsen and Bent Thomsen. A modal process logic. In *LICS 1988*, pages 203–210. IEEE, 1988. doi:10.1109/LICS.1988.5119.

20 Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

21 Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann. On characterising distributability. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:17)2013.

Decentralized Asynchronous Crash-Resilient Runtime Verification*

Borzoo Bonakdarpour[1], Pierre Fraigniaud[2], Sergio Rajsbaum[3], David A. Rosenblueth[4], and Corentin Travers[5]

1 **McMaster University, Canada,** borzoo@mcmaster.ca
2 **CNRS and University Paris Diderot, France,** pierref@irif.fr
3 **UNAM, México,** rajsbaum@unam.mx
4 **UNAM, México,** drosenbl@unam.mx
5 **University of Bordeaux, France,** travers@labri.fr

Abstract

Runtime Verification (RV) is a lightweight method for monitoring the formal specification of a system during its execution. It has recently been shown that a given state predicate can be monitored consistently by a set of crash-prone asynchronous *distributed* monitors, only if sufficiently many different verdicts can be emitted by each monitor. We revisit this impossibility result in the context of LTL semantics for RV. We show that employing the four-valued logic RV-LTL will result in inconsistent distributed monitoring for some formulas. Our first main contribution is a family of logics, called LTL_{2k+4}, that refines RV-LTL incorporating $2k+4$ truth values, for each $k \geq 0$. The truth values of LTL_{2k+4} can be effectively used by each monitor to reach a consistent global set of verdicts for each given formula, provided k is sufficiently large. Our second main contribution is an algorithm for monitor construction enabling fault-tolerant distributed monitoring based on the aggregation of the individual verdicts by each monitor.

1998 ACM Subject Classification C.2.4 Distributed Systems, D.2.5 Testing and Debugging; Monitors, D.2.4 Software/Program Verification

Keywords and phrases Runtime monitoring, Distributed algorithms, Fault-tolerance

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.16

1 Introduction

Runtime Verification (RV) is a technique where a *monitor* process determines whether or not the current execution of a system under inspection complies with its formal specification. The state-of-the-art RV methods for distributed systems exhibit the following shortcomings. They (1) employ a central monitor, (2) employ several monitors but lack a systematic way to monitor formally specified properties of a system (e.g., [12, 10, 11]), or (3) assume a fault-free setting, where each individual monitor is resilient to failures [16, 7, 15, 17, 19, 5, 8]. Relaxing the latter assumption, that is, handling monitors subject to failures, poses significant challenges as individual monitors would become unable to agree on the same perspective of the execution, due to the impossibility of consensus [9]. Thus, it is unavoidable

* This work was partially sponsored by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grants 430575-2012 and 463324-2014, UNAM-PAPIIT Grant IN107714, PASPA-DGAPA-UNAM and Conacyt grants 221341 and 261225, as well as the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02).

that individual monitors emit different *local* verdicts about the current execution, so that a consistent *global* verdict with respect to a correctness property can be constructed from these verdicts.

The necessity of using more than just the two truth values of Boolean logic is a known fact in the context of RV with a single monitor. For instance, RV-LTL [3] has four truth values $\mathbb{B}_4 = \{\top, \bot, \top_p, \bot_p\}$. These values identify cases where a finite execution (1) permanently satisfies, (2) permanently violates, (3) presumably satisfies, or (4) presumably violates an LTL formula. For example, consider a request/acknowledge property, where a request r_1 is eventually responded by acknowledgement a_1, and a_1 should not occur before r_1; i.e., LTL formula $\varphi = \mathbf{G}(\neg a_1 \wedge \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]$. In RV-LTL, a finite execution containing r_1 and ending in a_1 (i.e., the request has been acknowledged) yields the truth value 'permanently satisfied', whereas an execution containing only r_1 (i.e., the request has not yet been acknowledged) yields 'presumably violated'.

Although RV-LTL can monitor φ (see Figure 1 for its monitor automaton) in a centralized setting, we show $\mathbb{B}_4$ is not sufficient to *consistently* monitor a conjunction of two such formulas in a framework of several asynchronous unreliable monitors. Namely, the set of verdicts emitted by the monitors may not be sufficient to distinguish executions that satisfy the formula from those that violate it. Intuitively, this is because each monitor has only a partial view of the system under scrutiny, and after a finite number of rounds of communication among monitors, still too many different perspectives about the global system state remain. In fact, it was proved in [10] using algebraic topology techniques [13] that fault-tolerant distributed monitoring requires that the individual verdicts are taken from a set whose size depends on the formula being monitored.

Our results. In this paper, we propose a framework for distributed fault-tolerant RV. To this end, we make a novel connection between RV and consensus in a failure-prone distributed environment by proposing a *multi-valued temporal logic*. This new logic is a refinement of RV-LTL. More specifically, we propose a family of $(2k + 4)$-valued logics, denoted LTL_{2k+4}, for $k \geq 0$. In particular, LTL_{2k+4} coincides with RV-LTL when $k = 0$. The syntax of LTL_{2k+4} is identical to that of LTL. Its semantics is based on FLTL [14] and LTL_3 [4], two LTL-based finite trace semantics for RV. For each $k \geq 0$, the kth instance of the family has $2k + 4$ truth values, that intuitively represent a *degree of certainty* that the formula is satisfied. We characterize the formulas that when verified at run time with LTL_{2k+4}, no additional information is gained if they are verified with $\text{LTL}_{2k'+4}$, for a larger value k'. We present a monitor construction algorithm that generates a finite-state Moore machine for any given LTL formula and $k \geq 0$.

For example, for formula $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_t$, where each φ_i is an independent request/acknowledgement formula, LTL_{2k+4} can be used to consistently monitor φ, whenever $k \geq t$. In particular, when $t = 2$, the set of truth values is $\mathbb{B}_8 = \{\top_0, \bot_0, \top_1, \bot_1, \top_2, \bot_2, \top, \bot\}$. Moreover, formula φ evaluates to: $\top_0$ (presumably true with the lowest degree of certainty) in a finite execution that does not contain neither r_1 nor a_1, then to $\bot_1$ in an extension where r_1 appears (presumably true with a higher degree of certainty), to $\top_1$ in an extension that includes both r_1 and a_1, to $\bot_2$ if r_2 appears, and finally to $\top$ (permanently true) in an execution that contains r_1, a_1, r_2, and a_2.

Our second contribution is an algorithm for fault-tolerant distributed RV, where the monitors are asynchronous *wait-free* processes that communicate with each other via a read/write shared-memory, and any of them can fail by crashing. (For simplicity we use this abstract model, which is well-understood [2, 13], and is known to be equivalent, with respect to task

computability, to a message-passing model where less than half the processes can crash.) Each monitor gets a partial view of the system's global state, communicates with the other monitors a fixed number of rounds, and then emits a verdict from $\mathbb{B}_{2k+4}$. We show how, given any LTL formula and a large enough k, the truth values of LTL_{2k+4} can be effectively used such that a set of verdicts collectively provided by the monitors can be mapped to the verdict computed by a centralized monitor that has full view of the system under inspection. It follows from the general lower bound result in [10] that our algorithm is optimal, meaning that for any $k \geq 0$, there exists an LTL formula that cannot be monitored consistently in LTL_{2k+4}, if k is not sufficiently large. Finally, we prove that the value of k is solely a function of the structure of the LTL formula.

Related Work. While there has been significant progress in sequential monitoring in the past decade, there has been less work devoted to distributed monitoring. Lattice-theoretic centralized and decentralized online predicate detection in distributed systems has been studied in [7, 15]. This line of work does not address monitoring properties with temporal requirements. This shortcoming is partially addressed in [17], but for offline monitoring. In [19], the authors design a method for monitoring safety properties in distributed systems using the past-time linear temporal logic (PLTL). In such a work, however, the valuation of some predicates and properties may be overlooked. This is because monitors gain knowledge about the state of the system by piggybacking on the existing communication among processes. That is, if processes rarely communicate, then monitors exchange little information and, hence, some violations of properties may remain undetected. Runtime verification of LTL for synchronous distributed systems where processes share a single global clock has been studied in [5, 8]. In [6], the authors introduce parallel algorithms for runtime verification of sequential programs. As already mentioned, our work is inspired by the research line of [10, 12, 11], the first one to study the effects of monitor failures in distributed RV. Distributed applications that can be runtime monitored with three *opinions* were studied in [12], and the number of opinions needed to runtime monitor set agreement was analyzed in [11]. More generally, [10] proves a tight lower bound on the number of opinions needed to monitor a property based on its alternation number. The goal of this paper is to give a formal semantics to the opinions studied in [10, 12, 11], and derive a framework in the actual formal context of runtime verification.

2 Background: Linear Temporal Logics for RV

Let AP be a set of *atomic propositions* and $\Sigma = 2^{AP}$ be the set of all possible *states*. A *trace* is a sequence $s_0 s_1 \cdots$, where $s_i \in \Sigma$ for every $i \geq 0$. We denote by Σ^* (resp., Σ^ω) the set of all finite (resp., infinite) traces. Throughout the paper, we denote infinite traces by the letter σ, and finite traces by the letter α. We denote the empty trace by ϵ. For a finite trace $\alpha = s_0 s_1 \cdots s_n$, $|\alpha|$ denotes its *length*, i.e., its number of states $n+1$. Finally, by α^i, we mean trace $s_i s_{i+1} \cdots s_n$ of α. We assume that the syntax and semantics of standard LTL is common knowledge.

Example. We use the following *request/acknowledgement* LTL formula throughout the paper to explain the concepts:

$$\varphi_{ra} = \mathbf{G}(\neg a \wedge \neg r) \vee [(\neg a \,\mathbf{U}\, r) \wedge \mathbf{F} a]$$

That is (1) if a request is emitted (i.e., $r = \mathit{true}$), then it should eventually be acknowledged (i.e., $a = \mathit{true}$), and (2) an acknowledgement happens only in response to a request.

Finite LTL (FLTL). In the context of runtime verification, the semantics of LTL is not fully appropriate as it is defined over infinite traces. Finite LTL (FLTL, see [14]) allows us to reason about finite traces for verifying properties at run time. The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $\mathbb{B}_2 = \{\top, \bot\}$. The semantics of FLTL for atomic propositions and Boolean operators are identical to those of LTL. We now recall the semantics of FLTL for the temporal operators. Let φ, φ_1, and φ_2 be LTL formulas, $\alpha = s_0 s_1 \cdots s_n$ be a non-empty finite trace, and $\models_F$ denote satisfaction in FLTL. We have

$$[\alpha \models_F \mathbf{X}\,\varphi] = \begin{cases} [\alpha^1 \models_F \varphi] & \text{if } \alpha^1 \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

and

$$[\alpha \models_F \varphi_1\,\mathbf{U}\,\varphi_2] = \begin{cases} \top & \text{if } \exists k \in [0,n] : ([\alpha^k \models_F \varphi_2] = \top) \ \wedge \ (\forall \ell \in [0,k), [\alpha^\ell \models_F \varphi_1] = \top) \\ \bot & \text{otherwise} \end{cases}$$

To illustrate the difference between LTL and FLTL, let $\varphi = \mathbf{F}p$ and $\alpha = s_0 s_1 \cdots s_n$. If $p \in s_i$ for some $i \in [0,n]$, then we have $[\alpha \models_F \varphi] = \top$. Otherwise, $[\alpha \models_F \varphi] = \bot$, and this holds even if the program under inspection extends α in the future to a state where p becomes true.

Multi-valued LTLs. As illustrated above, for a finite trace α, FLTL ignores the possible future extensions of α, when evaluating a formula. 3-valued LTL (LTL_3, see [4]) evaluates LTL formulas for finite traces with an eye on possible future extensions. In LTL_3, the set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where '$\top$' (resp., '$\bot$') denotes that the formula is permanently satisfied (resp., violated), no matter how the current execution extends, and '?' denotes an unknown verdict; i.e., there exist an extension that can falsify the formula, and another extension that can truthify the formula.

Now, let $\alpha \in \Sigma^*$ be a non-empty finite trace. The truth value of an LTL_3 formula φ with respect to α, denoted by $[\alpha \models_3 \varphi]$, is defined as follows:

$$[\alpha \models_3 \varphi] = \begin{cases} \top & \text{if} \quad \forall \sigma \in \Sigma^\omega : \alpha\sigma \models \varphi \\ \bot & \text{if} \quad \forall \sigma \in \Sigma^\omega : \alpha\sigma \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

RV-LTL [3], which we will denote in this paper LTL_4, refines the truth value ? into $\bot_p$ and $\top_p$. That is, $\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$. More specifically, evaluation of a formula in LTL_4 agrees with LTL_3 if the verdict is $\bot$ or $\top$. Otherwise, (i.e., when the verdict in LTL_3 is ?), LTL_4 utilizes FLTL to compute a more refined truth value.

Now, let $\alpha \in \Sigma^*$ be a finite trace. The truth value of an LTL_4 formula φ with respect to α, denoted by $[\alpha \models_4 \varphi]$, is defined as follows:

$$[\alpha \models_4 \varphi] = \begin{cases} \top & \text{if} \quad [\alpha \models_3 \varphi] = \top \\ \bot & \text{if} \quad [\alpha \models_3 \varphi] = \bot \\ \top_p & \text{if} \quad [\alpha \models_3 \varphi] = ? \ \wedge \ [\alpha \models_F \varphi] = \top \\ \bot_p & \text{if} \quad [\alpha \models_3 \varphi] = ? \ \wedge \ [\alpha \models_F \varphi] = \bot \end{cases}$$

The LTL_4 *monitor* of a formula φ is the unique deterministic finite state machine $\mathcal{M}_4^\varphi = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is the

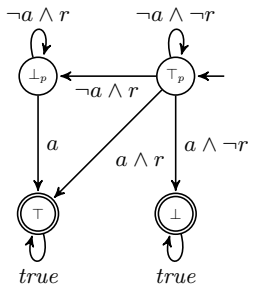


Figure 1 LTL$_4$ monitor of φ_{ra}.

transition function, and $\lambda : Q \to \mathbb{B}_4$, is a function such that:

$$\lambda(\delta(q_0, \alpha)) = [\alpha \models_4 \varphi]$$

for every finite trace $\alpha \in \Sigma^*$. In [4], the authors introduce an algorithm that takes as input an LTL formula and constructs as output an LTL$_4$ monitor. For example, Figure 1 shows the LTL$_4$ monitor for the request/acknowledgement formula $\varphi_{ra} = \mathbf{G}(\neg a \wedge \neg r) \vee [(\neg a \,\mathbf{U}\, r) \wedge \mathbf{F}a]$.

3 Distributed Runtime Monitoring and Insufficiency of LTL$_4$

In this section, we present a general computation model for asynchronous distributed wait-free monitoring. Throughout the rest of the paper, the system under inspection produces a finite trace $\alpha = s_0 s_1 \cdots s_k$, and is inspected with respect to an LTL formula φ by a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of asynchronous distributed wait-free monitors.

Algorithm sketch: For every $j \in [0, k-1]$, between each s_j and s_{j+1}, each monitor, in a wait-free manner:

1. reads the value of propositions in s_j, which may result in a *partial* observation of s_j;
2. repeatedly communicates its partial observation with other monitors through a single-writer/multi-reader shared memory;
3. updates its knowledge resulting from the aforementioned communication, and
4. evaluates φ and emits a verdict from $\mathbb{B}_4$.

Since each monitor observes and maintains only a partial view of s_j, and since the monitors run asynchronously, different read/write interleavings are possible, where each interleaving may lead to a different collective set of verdicts emitted by the monitors in $\mathcal{M}$ for s_j. In Subsection 3.1, we formally introduce our notion of *wait-free distributed monitoring*.

To ensure *consistent* distributed monitoring, one has to be able to map a collective set of verdicts of monitors (for any execution interleaving) to one and only one verdict of a centralized monitor that has the full view s_j. A necessary condition for this mapping is that, for every two finite traces $\alpha, \alpha' \in \Sigma^*$, if $[\alpha \models_F \varphi] \neq [\alpha' \models_F \varphi]$, then the monitors in $\mathcal{M}$ should compute different collective sets of verdicts for α and α', no matter what their initial partial observation and subsequent read/write interleavings are. We call this condition *global consistency*, described in detail in Subsection 3.2.

3.1 Wait-Free Distributed Monitoring

We consider a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of *monitors*, each observing a system under inspection. We assume that each monitor in $\mathcal{M}$ has only a *partial view* of the system under inspection.

▶ **Definition 1.** A *partial state* is a mapping $\mathcal{S}$ from the set AP of atomic propositions to the set $\{true, false, \natural\}$, where $\natural$ denotes an *unknown* value.

When a state s is reached in a finite trace, each monitor $M_i \in \mathcal{M}$, for $1 \leq i \leq n$, takes a *sample* from s, which results in obtaining a partial state. More formally:

▶ **Definition 2.** A *sample* of a state $s \in \Sigma$ by monitor M_i is a partial state $\mathcal{S}_i^s$ such that, for all $ap \in AP$, we have: $(\mathcal{S}_i^s(ap) = true \rightarrow ap \in s) \wedge (\mathcal{S}_i^s(ap) = false \rightarrow ap \notin s)$.

Definition 2 entails that, in a sample, if the value of an atomic proposition is not unknown, then the sampled value is consistent with state s. Thus, two monitors M_i and M_j cannot take inconsistent samples. That is, for any state s and samples $\mathcal{S}_i^s$, $\mathcal{S}_j^s$, and for every $ap \in AP$, we have: $(\mathcal{S}_i^s(ap) \neq \mathcal{S}_j^s(ap)) \rightarrow (\mathcal{S}_i^s(ap) = \natural \vee \mathcal{S}_j^s(ap) = \natural)$.

We say that a set of monitors *cover* a state if the collection of partial views of these monitors covers the value of the all atomic propositions. Formally:

▶ **Definition 3.** A set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ satisfies *state coverage* for a state s if and only if for every $ap \in AP$, there exists $M_i \in \mathcal{M}$ such that $\mathcal{S}_i^s(ap) \neq \natural$.

Each monitor M_i in $\mathcal{M}$ is a process, and the monitors run in the standard *asynchronous wait-free read/write shared memory* model [2]. Each monitor (1) runs at its own speed, that may vary along with time and (2) may fail by *crashing* (i.e., halt and never recover). We assume that up to $n-1$ monitors can crash, and thus a monitor never "waits" for another monitor (since this may cause a livelock). Every monitor that does not fail is required to output; i.e., to emit a verdict. Hence, a distributed algorithm in this settings consists for each monitor in a bounded sequence of read/write accesses to the shared memory at the end of which a verdict is emitted. If the number of possible inputs is bounded, the lengths of such sequences are globally bounded. We thus assume without loss of generality that each monitor accesses the shared memory a *fixed* number of times before emitting a verdict [13].

More specifically, for every state s_j in $\alpha = s_0 s_1 \cdots s_k$, each monitor M_i maintains a so-called *local snapshot* $LS_i[j]$ consisting of n *registers*, one per monitor in $\mathcal{M}$ (i.e., the local snapshot is organized as an array of registers). We denote by $LS_i^l[j]$ the local register of monitor M_i associated with monitor M_l for state s_j. Each register has $|AP|$ elements, one for each atomic proposition in AP. The monitors in $\mathcal{M}$ communicate by means of *shared memory*. The structure of the shared memory SM is similar to monitor local snapshots: for each state s_j, $SM[j]$ consists of n atomic registers, one per monitor, and each register has $|AP|$ elements one for each atomic proposition (i.e, single-writer/multiple-reader (SWMR) registers). Thus, for state s_j, each monitor M_i can read the entire content of $SM[j]$, but can only write into register $SM_i[j]$[1].

The distributed monitoring algorithm. Each monitor $M_i \in \mathcal{M}$, $i \in [1, n]$, runs Algorithm 1 that we shall now describe in detail. For any given new state s_j, Monitor M_i first initializes all registers of its local snapshot to $\natural$ (cf. Line 1). Then, M_i takes a sample from state s_j (cf. Line 2). Recall from Def. 2 that the value of an atomic proposition in a sample is either true, false, or $\natural$. The set of values in the sample is copied in local register $LS_i^i[j]$.

[1] We assume that each monitor is aware of the change of state of the system under inspection. Thus, for a state s_j, a monitor M_i reads and writes in the associated local and shared memory locations, i.e., $LS_i[j]$ and $SM[j]$.

Algorithm 1: Behavior of Monitor M_i, for $i \in [1, n]$

Data: LTL formula φ and state s_j
Result: a verdict from $\mathbb{B}_4$
1 initialize all elements of $LS_i[j]$ with $\natural$;
2 $LS_i^i[j] \leftarrow \mathcal{S}_i^{s_j}$; /* *take sample from state* s_j */
3 **for** some fixed number of rounds **do**
4 $\quad SM_i[j] \leftarrow \mathbf{p}(LS_i[j])$; /* *write (i.e., project) current knowledge in shared memory* */
5 $\quad LS_i[j] \leftarrow SM[j]$; /* *take a snapshot of the shared memory* */
6 emit $[\mathbf{x}(LS_i[0]) \dots \mathbf{x}(LS_i[j]) \models_4 \varphi]$; /* *evaluate* φ *using extrapolation function* */

After sampling, each monitor M_i executes a sequence of write/snapshot actions (cf. Lines 4 and 5) for some a priori known number of times, that we detail next[2].

In Line 4, M_i computes its knowledge about each proposition ap, given its content of $LS_i[j]$, and atomically *writes* it into its associated register $SM_i[j]$ in the shared memory. Function $\mathbf{p} = (\mathbf{p}_{ap})_{ap \in AP}$ where $\mathbf{p}_{ap} : \{true, false, \natural\}^n \to \{true, false, \natural\}$ is the *projection function* defined by

$$\mathbf{p}_{ap}(v_1, \dots, v_n) = \begin{cases} true & \text{if } \exists i \in [1, n] : v_i = true \\ false & \text{if } \exists i \in [1, n] : v_i = false \\ \natural & \text{otherwise} \end{cases}$$

Given a local snapshot LS_i, $\mathbf{p}(LS_i)$ denotes the partial state obtained by applying $\mathbf{p}_{ap}$ to n values of each atomic proposition ap in LS_i. Notice that, based on Definition 2, $\mathbf{p}$ cannot receive contradicting values for an atomic proposition.

In Line 5, M_i reads of all the registers in $SM[j]$, and copies them into $LS_i[j]$, in a single atomic step. Finally, after a certain number of iterations, the for-loop ends, and M_i evaluates φ and emits a verdict based on the content of its local snapshots $LS_i[0] \cdots LS_i[j]$ (cf. Line 6). To evaluate φ on $s_0 s_1 \cdots s_j$, monitor M_i needs to compute one and only one Boolean value for each atomic proposition. To this end, we assume that for each atomic proposition $ap \in AP$, all monitors are provided with the same *extrapolation function* $\mathbf{x}_{ap}$ allowing them to associate a Boolean value to each atomic proposition, even if its truth value is unknown at some monitors. Such an extrapolation function must satisfy the following consistency condition.

▶ **Definition 4.** Given $ap \in AP$, a function $\mathbf{x}_{ap} : \{true, false, \natural\}^n \to \{true, false\}$ is an *extrapolation function* if and only if $\mathbf{p}_{ap}(v_1, \dots, v_n) \neq \natural \;\to\; \mathbf{x}_{ap}(v_1, \dots, v_n) = \mathbf{p}_{ap}(v_1, \dots, v_n)$.

Given a local snapshot array LS, $\mathbf{x}(LS)$ denotes the state obtained by applying $\mathbf{x}_{ap}$ to n values of each atomic proposition ap in LS. Also given a state s_j, by $[\![LS_i[j]]\!]$, we mean the local snapshot of monitor M_i obtained after termination of the for loop in Algorithm 1.

Example. Let $\mathcal{M} = \{M_1, M_2\}$ and consider the formula for two requests and acknowledgements:

$$\varphi_{ra_2} = \Big(\mathbf{G}(\neg a_1 \wedge \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]\Big) \wedge \Big(\mathbf{G}(\neg a_2 \wedge \neg r_2) \vee [(\neg a_2 \,\mathbf{U}\, r_2) \wedge \mathbf{F} a_2]\Big)$$

[2] Algorithm 1 uses snapshot operations for the sake of simplifying the presentation. We emphasize that atomic snapshots can be implemented using atomic read/write operations in a wait-free manner [1].

Figure 2 shows different execution interleavings of monitors M_1 and M_2 when running Algorithm 1 from states $s_0 = \{r_1, a_1\}$ and $s_0' = \{r_1, a_1, r_2\}$. Based on the order of monitor write-snapshot actions: M_1, M_2 (resp., M_2, M_1) denotes the case where monitor M_1 (resp., M_2) executes a write-snapshot before monitor M_2 (resp., M_1) does, and $M_1||M_2$ denotes the case where monitors M_1 and M_2 execute their write-snapshot actions concurrently. In case of s_0, after executing Line 2 of Algorithm 1, monitor M_1's sample, i.e., the local snapshot $LS_1^1[0]$, consists of $\mathcal{S}_1^{s_0}(r_1) = \mathit{true}$, $\mathcal{S}_1^{s_0}(a_1) = \natural$, and $\mathcal{S}_1^{s_0}(r_2) = \mathcal{S}_1^{s_0}(a_2) = \mathit{false}$. Moreover, initially, M_1 has no knowledge of M_2's sample. Monitor M_2's sample from s_0, i.e., the local snapshot $LS_2^2[0]$, consists of $\mathcal{S}_2^{s_0}(r_1) = \mathcal{S}_2^{s_0}(a_1) = \mathit{true}$, $\mathcal{S}_2^{s_0}(r_2) = \natural$, and $\mathcal{S}_2^{s_0}(a_2) = \mathit{false}$ while it initially has no knowledge of M_1's sample. Likewise, for state s_0', Figure 2 shows different local snapshots by M_1 and M_2. Given two values v_1 and v_2, we define (an arbitrary) extrapolation function as follows:

$$\mathbf{x}_{ap}(v_1, v_2) = \begin{cases} \mathit{true} & \text{if } (v_1 = \mathit{true}) \vee (v_2 = \mathit{true}) \\ \mathit{false} & \text{otherwise} \end{cases}$$

where $ap \in \{a_1, r_1, a_2, r_2\}$. Finally, starting from s_0, if (1) the for loop of Algorithm 1 terminates after 1 communication round, and (2) the interleaving is M_1, M_2, then $\mathbf{x}([\![LS_2[0]]\!]) = \{r_1, a_1\}$, and evaluation of φ_{ra_2} by M_2 in LTL$_4$ results in $[\mathbf{x}([\![LS_2[0]]\!]) \models_4 \varphi_{ra_2}] = \top_p$.

3.2 Global Consistency

For any state s_j, when a set of monitors execute Algorithm 1, different interleavings, and hence different sets of verdicts, are possible. Global consistency is the property enabling to map the set of verdicts of the distributed monitors to *the* verdict of a centralized monitor that has the full view of states.

▶ **Definition 5.** A *monitor trace* in LTL$_4$ for α is a sequence $m = m_0 m_1 \cdots m_k$, where, for every $j \in [0, k]$, $m_j \subseteq \mathbb{B}_4$, and each element of each m_j is the verdict of some monitor $M_i \in \mathcal{M}$ by evaluating $[\mathbf{x}([\![LS_i[0]]\!])\mathbf{x}([\![LS_i[1]]\!]) \cdots \mathbf{x}([\![LS_i[j]]\!]) \models_4 \varphi]$. For example, Figure 3, shows a concrete finite trace α and its corresponding monitor trace.

▶ **Definition 6.** Let φ be an LTL formula, α be a finite trace in Σ^*, and m be any of its monitor traces. We say that m satisfies *global consistency* in LTL$_4$ iff there exists a function $\mu : 2^{\mathbb{B}_4} \to \{\top, \bot\}$ such that $\mu(m_{|\alpha|-1}) = [\alpha \models_F \varphi]$.

We now show that LTL$_4$ is unable to consistently monitor all LTL formulas. To see this, observe that in Figure 2, the shaded collective verdicts m_0 and m_0' are both equal to $\{\bot_p, \top_p\}$, but $[s_0 \models_4 \varphi] \neq [s_0' \models_4 \varphi]$. This clearly does not meet global consistency (see the proof of Lemma 7 for details).

▶ **Lemma 7.** *Not all* LTL *formulas can be consistently monitored by a 1-round distributed monitor with traces in* LTL$_4$*, even if monitors satisfy state coverage, and even if no monitors crash during the execution of the monitor.*

Lemma 7 holds for an arbitrary number of communication rounds as well. Indeed, additional rounds of communication will not result into reaching global consistency. This impossibility result is a direct consequence of the main lower bound in [10], which can be rephrased as follows.

▶ **Theorem 8.** *Not all* LTL *formulas can be consistently monitored by a distributed monitor with traces in* LTL$_4$*, even if monitors satisfy state coverage, even if no monitors crash during the execution of the monitor, and even if the monitors perform an arbitrarily large number of communication rounds.*

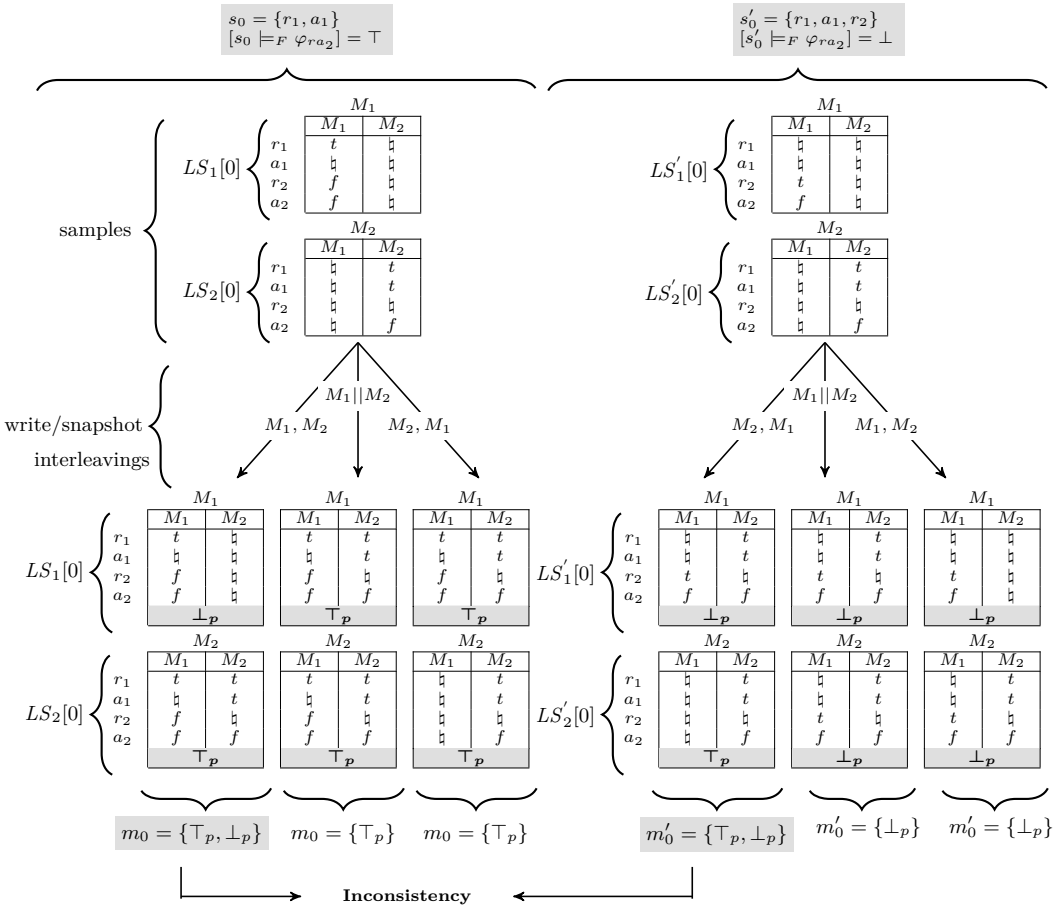


Figure 2 Example: Monitors M_1 and M_2 monitoring formula φ_{ra_2} from two different states s_0 and s'_0.

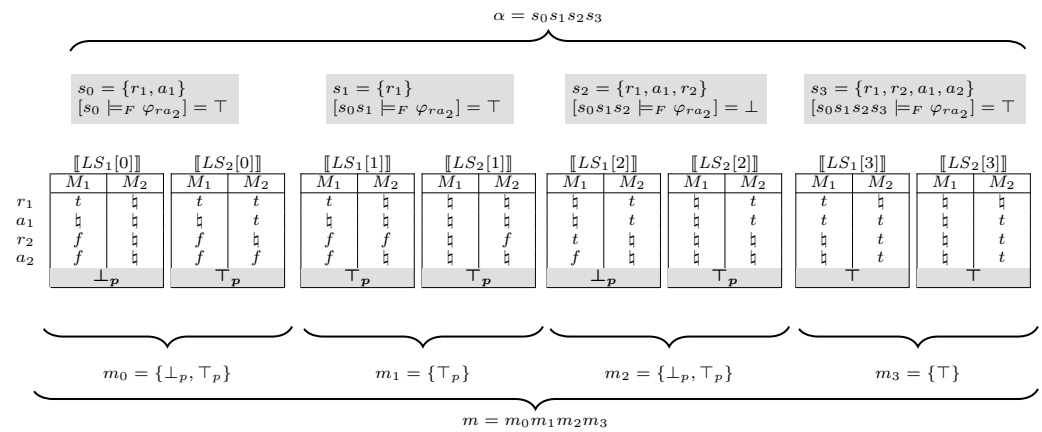


Figure 3 A monitor trace.

In the next section, we revisit the notion of *alternation number* introduced in [10] in order to identify formulas that can be monitored by LTL$_4$, and to design a multi-valued logic to monitor LTL formulas that cannot be monitored in LTL$_4$.

4 Alternation Number

We now define the notion of *alternation number* [10] in the context of LTL. In the next section, we shall show that the alternation number essentially determines an upper bound on the number of truth values needed to ensure consistency in distributed monitoring.

Let $\alpha \in \Sigma^*$ be a finite trace, α' be the longest proper prefix of α, and φ be an LTL formula. We set the *alternation number* of φ with respect to α as follows:

$$AN(\varphi, \alpha) = \begin{cases} 0 & \text{if } |\alpha| = 1 \\ AN(\varphi, \alpha') + 1 & \text{if } (|\alpha| \geq 2) \wedge ([\alpha' \models_F \varphi] \neq [\alpha \models_F \varphi]) \\ AN(\varphi, \alpha') & \text{otherwise} \end{cases}$$

The alternation number with respect to infinite traces is defined as follows. Let $\sigma \in \Sigma^\omega$ be an infinite trace. If for any prefix α of σ, there exists a finite extension α', such that $AN(\varphi, \alpha) < AN(\varphi, \alpha')$, then we set $AN(\varphi, \sigma) = \infty$. Otherwise, we set $AN(\varphi, \sigma) = AN(\varphi, \alpha)$ where α is such that there does not exist a finite extension α' of α such that $AN(\varphi, \alpha) < AN(\varphi, \alpha')$. Finally, the alternation number of φ with respect to a (possibly infinite) set A of traces is

$$AN(\varphi, A) = \max\left\{AN(\varphi, \alpha) \mid \alpha \in A\right\}$$

▶ **Definition 9.** The *alternation number* of an LTL formula φ is $AN(\varphi) = AN(\varphi, \Sigma^*)$.

Examples. We have $AN(\mathbf{G}\, p) = 1$ because, in any finite trace α, if the valuation of $\mathbf{G}\, p$ in FLTL changes from $\top$ to $\bot$, then, in no extension of α this value can change back to $\top$. We have $AN(\mathbf{G}(r \rightarrow \mathbf{F}a)) = \infty$, because any occurrence of $r \wedge \neg a$ evaluates the formula to $\bot$, and a subsequent occurrence of a evaluates the formula to $\top$ in FLTL. We have $AN(\varphi_{ra}) = AN(\mathbf{G}(\neg a \wedge \neg r) \vee [(\neg a\, \mathbf{U}\, r) \wedge \mathbf{F}a]) = 2$. Indeed, as long as $\neg r \wedge \neg a$ is true throughout a trace α, we have $[\alpha \models_{\mathrm{F}} \varphi_{ra}] = \top$. When $r \wedge \neg a$ becomes true, the valuation of φ_{ra} changes to $\bot$. If a becomes true subsequently, then φ_{ra} evaluates to $\top$. By the same type of arguments, we show $AN(\varphi_{ra_2}) = 4$.

Interestingly, the alternation number of an LTL formula φ can be determined from the structure of its LTL_4 monitor automaton M_4^φ.

▶ **Theorem 10.** *Let φ be an* LTL *formula. The alternation number of φ, $AN(\varphi)$, is equal to the length of the longest alternating walk in its* LTL_4 *monitor M_4^φ.*

Example. Let $\varphi_{ra} = \mathbf{G}(\neg a \wedge \neg r) \vee [(\neg a\, \mathbf{U}\, r) \wedge \mathbf{F}a]$. We have $AN(\varphi_{ra}) = 2$, and one can check on Figure 1 that indeed the length of the longest alternating walk in $M_4^{\varphi_{ra}}$ is 2.

5 Multi-Valued LTL for Consistent Distributed Monitoring

In this section, we introduce a family of multi-valued logics (called LTL_{2k+4}), for every $k \geq 0$, and relate it to the notion of alternation number. For every $k \geq 0$, the syntax of LTL_{2k+4} is identical to that of LTL. We present the semantics, monitor synthesis, and proof of global consistency of LTL_{2k+4} in Subsections 5.1, 5.2, and 5.3, respectively.

5.1 Semantics of LTL_{2k+4}

Truth values. The semantics of LTL_{2k+4} refines LTL_4. LTL_{2k+4} employs the following set of $2k+4$ truth values:

$$\mathbb{B}_{2k+4} = \{\bot, \top, \bot_0, \ldots, \bot_k, \top_0, \ldots, \top_k\}.$$

Intuitively, for $i \in [0,k]$, truth value $\bot_i$ means *possibly false* with *degree of certainty* i, and truth value $\top_i$ means *possibly true* with degree of certainty i, while $\top$ and $\bot$ have the same meaning as their LTL_3 counterparts. Thus, LTL_{2k+4} coincides with LTL_4 for $k=0$. Consider a non-empty finite trace $\alpha = s_0 s_1 \cdots s_n$ in Σ^*. We denote the valuation of a formula φ with respect to α in LTL_{2k+4} by $[\alpha \models_{2k+4} \varphi]$. Since, for any $i \in [0,k]$, $\bot_i$ implies '?' in LTL_3, we require that $[\alpha \models_{2k+4} \varphi] = \bot_i \;\rightarrow\; [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \bot$. The latter conjunct is to relate $\bot_i$ with the valuation of α in FLTL. Likewise, we require that, for any $i \in [0,k]$: $[\alpha \models_{2k+4} \varphi] = \top_i \;\rightarrow\; [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \top$. We determine the degree of certainty of $[\alpha \models_{2k+4} \varphi]$ inductively according to the judgement rules below, where $\alpha' = s_0 s_1 \cdots s_{n-1}$.

Observe that the degree of certainty does not change if the FLTL valuation does not change in α' and α, or change from $\bot$ to $\top$. On the contrary, the degree of certainty does change if the FLTL valuation changes in α' and α from $\top$ to $\bot$, respectively.

$$[\alpha \models_{2k+4} \varphi] = \begin{cases} \bot & \text{if} \quad [\alpha \models_3 \varphi] = \bot \\ \top & \text{if} \quad [\alpha \models_3 \varphi] = \top \\ \bot_0 & \text{if} \quad |\alpha| = 1 \wedge [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \bot \\ \top_0 & \text{if} \quad |\alpha| = 1 \wedge [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \top \\ \top_i \quad \text{with } i \in [0,k] & \text{if} \quad |\alpha| \geq 2 \wedge [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \top \wedge \\ & \qquad [\alpha' \models_{2k+4} \varphi] \in \{\top_i, \bot_i\} \\ \bot_i \quad \text{with } i \in [0,k) & \text{if} \quad |\alpha| \geq 2 \wedge [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \bot \wedge \\ & \qquad [\alpha' \models_{2k+4} \varphi] \in \{\bot_i, \top_{i-1}\} \\ \bot_k & \text{if} \quad |\alpha| \geq 2 \wedge [\alpha \models_3 \varphi] = ? \wedge [\alpha \models_F \varphi] = \bot \wedge \\ & \qquad [\alpha' \models_{2k+4} \varphi] \in \{\bot_k, \top_k, \top_{k-1}\} \end{cases}$$

5.2 Monitorability and Monitor Synthesis for LTL_{2k+4}

Pnueli and Zaks [18] characterize an LTL formula φ as *monitorable* for a finite trace α, if α can be extended to one that can be evaluated with respect to φ at run time. That is, an LTL formula φ is *monitorable* in LTL_3 if and only if: $\forall \alpha \in \Sigma^* : \exists \alpha' \in \Sigma^* : [\alpha\alpha' \models_3 \varphi] \neq ?$. We stick to the same definition for LTL_{2k+4}.

▶ **Definition 11.** Let φ be an LTL formula. The LTL_{2k+4} *monitor* of φ is the unique deterministic finite state machine $\mathcal{M}^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $\lambda : Q \to \mathbb{B}_{2k+4}$, such that, for every non-empty finite trace $\alpha \in \Sigma^*$, we have $[\alpha \models_{2k+4} \varphi] = \lambda(\delta(q_0, \alpha))$.

Algorithm 2 constructs LTL_{2k+4} monitors. Intuitively, our algorithm creates $k+1$ copies of LTL_4 [3] monitors by invoking Function ConstructMonitor, and cascades them in such a way that incrementing the degree of certainty is implemented as prescribed by our definition of LTL_{2k+4}. Observe that for a given value $i \in [0,k]$, Function ConstructMonitor renames truth value $\top_p$ (respectively, $\bot_p$) in LTL_4 to $\top_i$ (respectively, $\bot_i$) (see Lines 14-18). Cascading

Algorithm 2: Monitor construction for LTL_{2k+4}

Input: Alphabet Σ, LTL formula φ, $k \geq 0$
Output: LTL_{2k+4} monitor $M^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda)$

1 $(Q, q_0, \delta, \lambda) \leftarrow$ ConstructMonitor(Σ, φ, 0);
2 **for** $i \leftarrow 1$ **to** k **do**
3 $(\bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda}) \leftarrow$ ConstructMonitor(Σ, φ, i);
4 $Q \leftarrow Q \cup \bar{Q}$; $\delta \leftarrow \delta \cup \bar{\delta}$; $\lambda \leftarrow \lambda \cup \bar{\lambda}$;
5 **forall the** $q \in Q$, $\bar{q} \in \bar{Q}$ **do**
6 **if** $(\lambda(q) = \top_{i-1} \wedge \lambda(\bar{q}) = \bot_i)$ **then**
7 **forall the** $q' \in Q$, $a \in \Sigma$ **do**
8 **if** $\lambda(q') = \bot_{i-1} \wedge \delta(q, a) = q'$ **then**
9 $\delta = \delta - \{(q, a, q')\}$;
10 $\delta = \delta \cup \{(q, a, \bar{q})\}$;
11 **return** $M^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda)$;
12 **Function** `ConstructMonitor(`*alphabet* Σ, LTL *formula* φ, $i \geq 0$`)`
13 **Let** $\mathcal{M}^{\varphi}_4 = (\Sigma, Q, q_0, \delta, \lambda)$;
14 **forall the** $q \in Q$ **do**
15 **if** $(\lambda(q) = \top_p)$ **then**
16 $\lambda(q) \leftarrow \top_i$;
17 **if** $(\lambda(q) = \bot_p)$ **then**
18 $\lambda(q) \leftarrow \bot_i$;
19 **return** $(Q, q_0, \delta, \lambda)$;

the monitors in Algorithm 2 is as follows. Initially, we generate an LTL_4 monitor for $k = 0$ (Line 1). Then, in each step $i \in [1, k]$ of the for-loop, we generate a new LTL_4 monitor (cf. Line 3). We ensure incrementing the degree of certainty by removing monitor transitions (q, a, q'), where q is annotated by $\top_{i-1}$ and q' is annotated by $\bot_{i-1}$, and adding transitions $(q, a, \bar{q})$, where $\bar{q}$ is annotated by $\bot_i$ (Lines 5-10).

▶ **Theorem 12.** *Let φ be an* LTL *formula, and let $\mathcal{M}^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda)$ be its LTL_{2k+4} monitor such as constructed by Algorithm 2. Then, for any non-empty finite trace $\alpha \in \Sigma^*$, we have $\lambda(\delta(q_0, \alpha)) = [\alpha \models_{2k+4} \varphi]$.*

5.3 Monitoring Algorithm and Global Consistency in LTL_{2k+4}

Monitoring Algorithm Let $\alpha = s_0 s_1 \cdots s_k$ be a finite trace in Σ^*. As discussed in Section 3, for any state s_j, where $j \in [0, k]$, each monitor runs Algorithm 1 and emits a verdict. In order to employ LTL_{2k+4} and ensure consistency, each monitor has to compute the highest possible degree of certainty by considering all possible monitor communication interleavings that result in state s_j. Formally, the set of all interleavings that reach a state $s \in \Sigma$ is the set of sequences of partial states defined as follows:

$$\mathcal{I}_s = \big\{\mathcal{S}_0\mathcal{S}_1 \cdots \mathcal{S}_l \mid (\forall ap \in AP : \mathcal{S}_0(ap) = \natural) \wedge (\mathcal{S}_l = s) \wedge [\forall i \in [0, l) : \forall ap \in AP : (\mathcal{S}_i(ap) \neq \natural) \rightarrow (\forall m \in (i, l] : \mathcal{S}_i(ap) = \mathcal{S}_m(ap))]\big\}$$

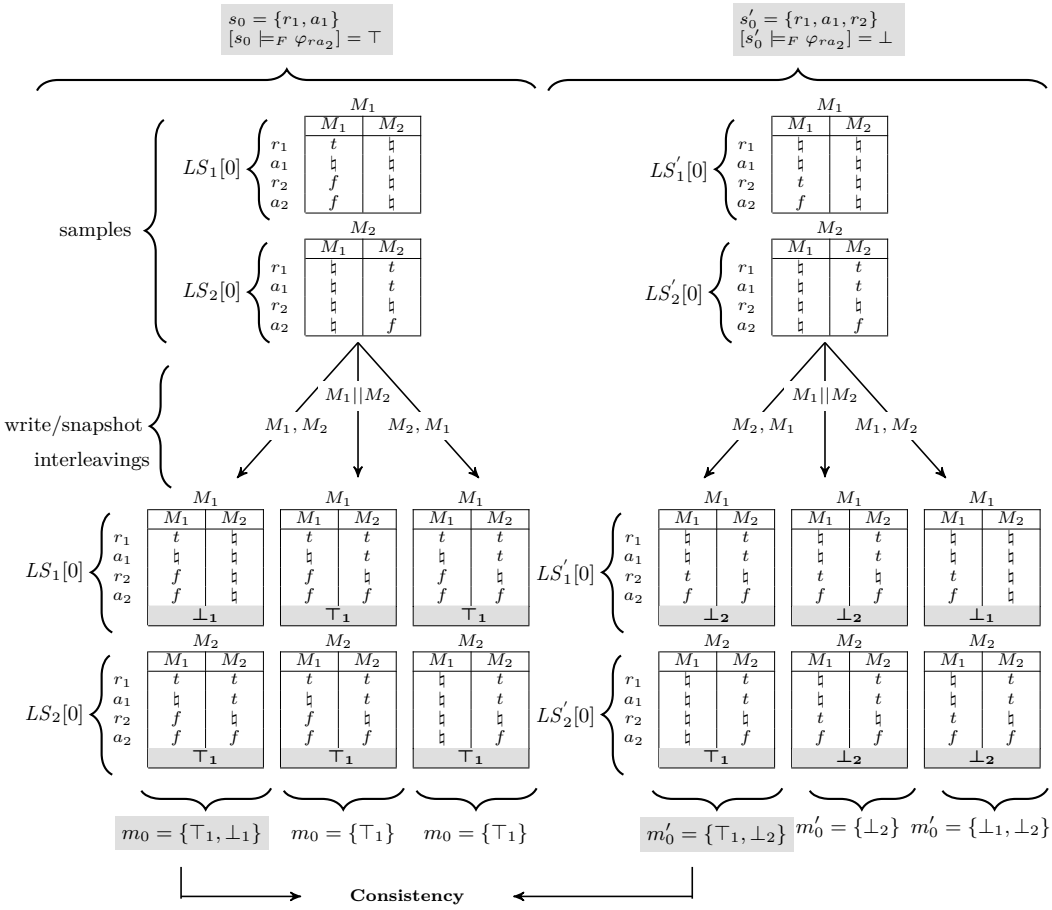


Figure 4 Global consistency of LTL_{2k+4} monitors M_1 and M_2 for formula φ_{ra_2}, where $k = 2$.

Now, for state s_j in α and formula φ, a monitor M_i computes $AN(\varphi, \mathcal{I}_{\mathbf{x}([\![LS_i[j]]\!])})$. This can be done by running each trace in $\mathcal{I}_{\mathbf{x}([\![LS_i[j]]\!])}$ on the LTL_{2k+4} monitor of φ. This is indeed the key idea to ensure global consistency.

▶ **Observation 13.** *For any state $s \in \Sigma$ and* LTL *formula φ, we have $AN(\varphi, \mathcal{I}_s) \leq AN(\varphi)$.*

Example. Figure 4 shows how monitors M_1 and M_2 evaluate formula φ_{ra_2} in LTL_{2k+4} with $k = 2$. Observe that the two sets of verdicts that were not distinguishable in Figure 2 (i.e., $m_0 = m_0' = \{\bot_p, \top_p\}$) are now distinguishable (i.e., $m_0 = \{\bot_1, \top_1\}$, while $m_0' = \{\top_1, \bot_2\}$), as we are now using 8 truth values instead of just 4. The ability of monitoring a formula in LTL_{2k+4} for a given $k \geq 0$ is strongly related to the alternation number of the formula.

Main Results. The following identifies an upper-bound on the number of truth values needed to monitor any LTL formula.

▶ **Theorem 14.** *An* LTL *formula φ can consistently be monitored by a wait-free distributed monitor in LTL_{2k+4}, if*

$$k \geq \lceil \frac{1}{2}(\min(AN(\varphi), n) - 1) \rceil$$

where n is the number of monitors.

An immediate consequence of Theorem 14 is for computing μ (Definition 6) for LTL_{2k+4}. For a set $m \in \mathbb{B}_{2k+4}$, one can compute $\mu(m)$ by identifying the supremum of m, for the total order $\bot_0 < \top_0 < \bot_1 < \top_1 < \ldots < \bot_k < \top_k$. It is straightforward to observe that such a μ results in global consistency for LTL_{2k+4}. Also, notice that Theorem 14 is best possible. It matches the following generalization of Theorem 8. The proof is similar to the lower bound of [10].

▶ **Theorem 15.** *For each $k \geq 0$, there is an* LTL *formula φ that cannot be consistently monitored by a wait-free distributed monitor in* LTL_{2k+4}*, if*

$$k < \lceil \frac{1}{2}(\min(AN(\varphi), n) - 1) \rceil$$

where n is the number of monitors.

6 Conclusion and Future Work

In this paper, we proposed a family of multi-valued logics LTL_{2k+4}, each one with $2k + 4$ truth values, for fault-tolerant distributed RV, refining existing finite LTL semantics. We presented an idealized setting where a set of unreliable monitors emit consistent verdicts in LTL_{2k+4} about the correctness of the system under inspection, if k is sufficiently large.

We note that wait-free computing is a powerful and simple abstraction to model and reason about distributed algorithms. All results in this paper can theoretically be transformed to more practical refinements such as message passing frameworks. Of course, further research is needed to develop such transformations. From a more practical perspective, it would be interesting to relax the timing model enabling monitors to observe, communicate, and emit verdicts between any two global states; to study frameworks for message passing systems, and to address more severe, even Byzantine failures.

References

1 Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

2 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley, 2004.

3 A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL Semantics for Runtime Verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

4 A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.

5 A. K. Bauer and Y. Falcone. Decentralised LTL monitoring. In *Proceedings of the 18th International Symposium on Formal Methods (FM)*, pages 85–100, 2012.

6 S. Berkovich, B. Bonakdarpour, and S. Fischmeister. GPU-based runtime verification. In *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1025–1036, 2013.

7 H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. A distributed abstraction algorithm for online predicate detection. In *Proceedings of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 101–110, 2013.

8 C. Colombo and Y. Falcone. Organising LTL monitors over distributed systems with a global clock. In *Proceedings of the 14th International Conference on Runtime Verification (RV)*, pages 140–155, 2014.

9 M. J. Fischer, N. .A. Lynch, and M. S. Peterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):373–382, 1985.

10 P. Fraigniaud, S. Rajsbaum, and C. Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *Proceedings of the 5th International Conference on Runtime Verification (RV)*, pages 92–107, 2014.

11 Pierre Fraigniaud, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. The opinion number of set-agreement. In *Proceedings of the 18th International Conference on Principles of Distributed Systems (OPODIS)*, pages 155–170, 2014.

12 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. *Distributed Computing*, 26(4):223–242, 2013. URL: `http://dx.doi.org/10.1007/s00446-013-0188-x`, `doi:10.1007/s00446-013-0188-x`.

13 M.H. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann-Elsevier, 2013.

14 Z. Manna and A. Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

15 N. Mittal and V. K. Garg. Techniques and applications of computation slicing. *Distributed Computing*, 17(3):251–277, 2005.

16 M. Mostafa and B. Bonakdarpour. Decentralized runtime verification of LTL specifications in distributed systems. In *Proceedings of the 29th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 494–503, 2015.

17 V. A. Ogale and V. K. Garg. Detecting temporal logic predicates on distributed computations. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, pages 420–434, 2007.

18 A. Pnueli and A. Zaks. PSL Model Checking and Run-Time Verification via Testers. In *14th Int. Symp. on Formal Methods (FM)*, pages 573–586, 2006.

19 K. Sen, A. Vardhan, G. Agha, and G.Rosu. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 418–427, 2004.

Lazy Reachability Analysis in Distributed Systems

Loïg Jezequel[1] **and Didier Lime**[2]

1 **Université de Nantes, IRCCyN UMR CNRS 6597, France**
Loig.Jezequel@irccyn.ec-nantes.fr
2 **École Centrale de Nantes, IRCCyN UMR CNRS 6597, France**
Didier.Lime@ec-nantes.fr

Abstract

We address the problem of reachability in distributed systems, modelled as networks of finite automata and propose and prove a new algorithm to solve it efficiently in many cases. This algorithm allows to decompose the reachability objective among the components, and proceeds by constructing partial products by lazily adding new components when required. It thus constructs more and more precise over-approximations of the complete product. This permits early termination in many cases, in particular when the objective is not reachable, which often is an unfavorable case in reachability analysis. We have implemented this algorithm in a first prototype and provide some very encouraging experimental results.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Reachability analysis, compositional verification, automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.17

1 Introduction

As distributed systems become more and more pervasive, the need for the verification of their correctness increases accordingly. The problem of verifying that some particular state of the system is reachable is a cornerstone in this endeavour. Yet even this simple problem is challenging in a distributed context, due to the exponential growth of the state-space of the system with the number of components, a problem often referred to as "state explosion".

To alleviate this issue several approaches have been proposed in the last two decades. In particular, partial order techniques, which allow to explore only part of the state-space while preserving completeness, have proved to be an efficient approach [12] and are implemented in some state-of-the-art tools, including LoLA [15]. Another technique called partial model-checking has been proposed in [1], which consists in incrementally quotienting temporal logic formulas by incorporating the behaviours of individual processes into that formula. This leads to a compositional verification scheme that has been recently extended and implemented in the PMC tool on top of the CADP toolbox [14]. This idea to incrementally take into account components of distributed systems is also present in works on compositional minimization [9, 6] and modular model checking [10, 7]. Recently, the IC3 algorithm [3] has been proposed to address the safety / reachability issue, with very promising results. This algorithm incrementally generates more and more precise over-approximations of the reachability relation, by computing stronger and stronger inductive assertions using SAT solving [3] or SMT solving [4]. Similar ideas of incremental refinements were also successfully used in AI planning [2, 11].

A common high-level scheme in the partial model-checking approach, the IC3 approach, and the hierarchical planning approach is to incrementally compute more and more precise approximate objects until sufficient precision permits to conclude. We propose a new approach

also based on this scheme. Contrarily to IC3, we deliberately use the classical tools of explicit state space exploration in finite automata-based models, with the motivation of ultimately combining our technique with some of the most successful improvements of these tools, like partial-order reduction techniques, and with well-known extensions to more expressive models like timed automata. In contrast to the partial model-checking approach, we take advantage of the simplicity of the reachability property we study, and adopt a lower-level approach, by focusing on the individual components of the system and their fine interactions.

Our contribution in this paper is therefore the following: first an algorithm, which projects a reachability property on the individual components, then lazily adds components to the projections and merges them together as interactions that are meaningful to the reachability property are discovered. We provide full proofs for completeness, soundness, and termination. Second, we report on a prototype implementation that permits a first evaluation of our algorithm. We have compared these performances with LoLA.

The paper is organized as follows: in Section 2 we give the basic definitions upon which our algorithm is built. In Section 3, we describe our algorithm and prove it. In Section 4 we report on experimental results, and we conclude in Section 5.

2 Preliminary definitions

2.1 LTSs and their products

We focus on labelled transition systems, synchronized on common actions, as models.

▶ **Definition 1.** A *labelled transition system* (LTS) is a tuple $\mathcal{L} = (S, \iota, T, \Sigma, \lambda)$ where S is a non-empty finite set of states, $\iota \in S$ is an initial state, Σ is an alphabet of transition labels, $T \subseteq S \times S$ is a finite set of transitions, and $\lambda : T \to \Sigma$ is a transition labeling function.

By a slight abuse of notations, we also denote by $\Sigma(\mathcal{L})$ the set of labels of $\mathcal{L}$ and by $\lambda(\mathcal{L})$ the set of labels effectively associated to at least one transition in $\mathcal{L}$.

▶ **Definition 2.** In such an LTS, a *path* is a sequence of transitions $\pi = t^1 \dots t^n$ such that: $\forall 1 \leq k \leq n, t^k = (s^k, s^{k+1}) \in T$ and $s^1 = \iota$. In this case we say that π *reaches* s^{n+1}. A state s is said to be *reachable* if there exists a path that reaches s.

▶ **Definition 3.** We say that two LTSs $\mathcal{L}_1 = (S_1, \iota_1, T_1, \Sigma_1, \lambda_1)$ and $\mathcal{L}_2 = (S_2, \iota_2, T_2, \Sigma_2, \lambda_2)$ are *isomorphic* if and only if there exists two bijections $f_S : S_1 \to S_2$ and $f_T : T_1 \to T_2$ so that: $f_S(\iota_1) = \iota_2$, and $\forall s_1, s_1' \in S_1, (s_1, s_1') \in T_1$ iff $(f_S(s_1), f_S(s_1')) \in T_2$, and $\lambda_1((s_1, s_1')) = \lambda_2((f_S(s_1), f_S(s_1')))$.

Our systems are built as parallel compositions of multiple LTSs.

▶ **Definition 4.** Let $\mathcal{L}_1, \dots, \mathcal{L}_n$ be LTSs such that $\forall 1 \leq i \leq n, \mathcal{L}_i = (S_i, \iota_i, T_i, \Sigma_i, \lambda_i)$. The *compound system* $\mathcal{L}_1 \parallel \dots \parallel \mathcal{L}_n$ is the LTS $(S, \iota, T, \Sigma, \lambda)$ such that $S = S_1 \times \dots \times S_n$, $\iota = (\iota_1, \dots, \iota_n)$, $\Sigma = \Sigma_1 \cup \dots \cup \Sigma_n$, and $t = ((s_1, \dots, s_n), (s_1', \dots, s_n')) \in T$ with $\lambda(t) = \sigma$ if and only if $\forall 1 \leq i \leq n$ if $\sigma \in \Sigma_i$ then $t_i = (s_i, s_i') \in T_i$ and $\lambda_i(t_i) = \sigma$ else $s_i = s_i'$.

Remark that $(\mathcal{L}_1 \parallel \mathcal{L}_2) \parallel \mathcal{L}_3$, $\mathcal{L}_1 \parallel (\mathcal{L}_2 \parallel \mathcal{L}_3)$, and $\mathcal{L}_1 \parallel \mathcal{L}_2 \parallel \mathcal{L}_3$ are isomorphic (they are identical up to renaming of states and transitions). It is thus possible to compute compound systems step by step, by adding LTSs to the composition one after the other.

▶ **Definition 5.** $\mathcal{L}_{id} = (S_{id}, \iota_{id}, T_{id}, \Sigma_{id}, \lambda_{id})$ with $S_{id} = \{id\}$, $\iota_{id} = id$, $T_{id} = \emptyset$, $\Sigma_{id} = \emptyset$, and λ_{id} is the unique function from $\emptyset$ to $\emptyset$.

Remark that $\mathcal{L}_{id}$ can be considered as the neutral element for the composition of LTSs: $\forall \mathcal{L}, \mathcal{L} \parallel \mathcal{L}_{id}$ and $\mathcal{L}$ are isomorphic. Also remark that for any LTS $\mathcal{L}$, there is an LTS containing only the initial state of $\mathcal{L}$ which is isomorphic to $\mathcal{L}_{id}$. We denote it by $\mathsf{id}(\mathcal{L})$.

2.2 Partial products and reachability of partial states

We define a notion of extension of LTSs, using a partial order relation.

▶ **Definition 6.** An LTS $\mathcal{L} = (S, \iota, T, \Sigma, \lambda)$ *extends* an LTS $\mathcal{L}'$, noted $\mathcal{L}' \sqsubseteq \mathcal{L}$, if and only if $\mathcal{L}'$ is isomorphic to some LTS $(S', \iota', T', \Sigma', \lambda')$ with $S' \subseteq S$, $T' \subseteq T$, $\Sigma' \subseteq \Sigma$, $\iota = \iota'$, and $\lambda' = \lambda|_{T'}$. If, moreover, $S' \neq S$, or $T' \neq T$, or $\Sigma' \neq \Sigma$, $\mathcal{L}$ is said to *strictly extend* $\mathcal{L}'$, noted $\mathcal{L}' \sqsubset \mathcal{L}$.

We define $\mathsf{ini}(\mathcal{L})$ as the LTS containing only the initial state of $\mathcal{L}$ but, contrarily to $\mathsf{id}(\mathcal{L})$, with $\Sigma(\mathsf{ini}(\mathcal{L})) = \Sigma(\mathcal{L})$. Note that we clearly have $\mathsf{ini}(\mathcal{L}) \sqsubseteq \mathcal{L}$.

Given a set of LTSs, we now define partial products as products in which some parts of the LTSs, or possibly some LTSs altogether, are not used:

▶ **Definition 7.** An LTS $\mathcal{L}'$ is a *partial product* of a compound system $\mathcal{L} = \mathcal{L}_1 \parallel \cdots \parallel \mathcal{L}_n$ if there exists m LTSs $\mathcal{L}'_{k_1}, \ldots, \mathcal{L}'_{k_m}$ (with $\{k_j \ : \ j \in [1..m]\} \subseteq [1..n]$) such that $\mathcal{L}'$ is isomorphic to $\mathcal{L}'_{k_1} \parallel \cdots \parallel \mathcal{L}'_{k_m}$ and $\forall j \in [1..m], \mathcal{L}'_{k_j} \sqsubseteq \mathcal{L}_{k_j}$.

Note that in the algorithm we propose in Section 3, we will actually always have, by construction, $\mathsf{ini}(\mathcal{L}_{k_j}) \sqsubseteq \mathcal{L}'_{k_j} \sqsubseteq \mathcal{L}_{k_j}$, which implies that all three LTSs have the same alphabet.

We focus on solving particular reachability problems where one is interested in reaching partial states.

▶ **Definition 8.** In a compound system $\mathcal{L}_1 \parallel \ldots \parallel \mathcal{L}_n$ we call any element from $S_1 \times \cdots \times S_n$ a *global state*. A *partial state* is an element from $(S_1 \cup \{\star\}) \times \cdots \times (S_n \cup \{\star\}) \setminus \{(\star, \ldots, \star)\}$.

We say that a partial state $(s'_1, \ldots, s'_n)$ *concretises* a partial state $(s_1, \ldots, s_n)$ if $\forall i, s_i \neq \star$ implies $s'_i = s_i$.

A partial state is therefore in some sense the specification of the set of global states that concretise it, i.e., that share the same values on dimensions not equal to $\star$ in the partial state. We use partial states to specify our reachability objectives.

▶ **Definition 9.** In a compound system $\mathcal{L}$, a partial state is said *reachable*, if there exists a reachable global state that concretises it. Given a set $\mathcal{R}$ of partial states, we call *reachability problem* ($RP^{\mathcal{R}}_{\mathcal{L}}$) the problem of deciding whether or not some element from $\mathcal{R}$ is reachable.

Given a reachability problem $RP^{\mathcal{R}}_{\mathcal{L}}$ we denote by $\mathbb{L}_g$ the set of indices of the LTSs involved in $\mathcal{R}$: for each $i \in \mathbb{L}_g$, there exists at least one partial state in which the element corresponding to $\mathcal{L}_i$ is not $\star$. In a reachability problem, if there exists a reachable global state in $\mathcal{L}$ that concretises an element from $\mathcal{R}$, we write $\mathcal{L} \to \mathcal{R}$. If this is not the case, we write $\mathcal{L} \nrightarrow \mathcal{R}$.

We conclude this section by establishing two basic results on partial products of LTSs that will be instrumental in proving that our approach is sound and complete.

Lemma 10 formalizes the fact that, due to the synchronization by shared labels mechanism, removing an LTS from a product produces an over-approximation of the reachability property projected on the remaining LTSs.

▶ **Lemma 10.** *Let $\mathcal{L} = \mathcal{L}_1 \parallel \cdots \parallel \mathcal{L}_n$ be a compound system in which some global state $s = (s_1, \ldots, s_n)$ is reachable. Let $\mathcal{L}'$ be the partial product of $\mathcal{L}$ obtained by removing $\mathcal{L}_i$ for some i. Similarly, let s' be the state of $\mathcal{L}'$ obtained from s by removing the i^{th} component s_i.*

Then s' is reachable in $\mathcal{L}'$.

Lemma 11 works in the opposite direction to Lemma 10: If we can find a subset of the LTSs, in which we can reach some given state, and that does not make use of any label appearing in an LTS not in this subset – condition (i) –, then we can add the missing LTSs to get the full product, while still ensuring the reachability of our given state. The result we prove is actually a bit stronger: one can preserve the reachability found in a *partial product* of our subset of LTSs, provided that if some label appears on a transition in the partial product, then it is present in the alphabets of all the components of the partial product that can be extended into an LTS that uses this label – condition (ii). Condition (i) ensures that we do not add any synchronization constraint to existing transitions when adding new LTS, while condition (ii) does the same but when extending the LTSs already in the subset.

▶ **Lemma 11.** *Let $\mathcal{L} = \mathcal{L}_1 \parallel \cdots \parallel \mathcal{L}_n$ be a compound system. Let $H \subseteq [1..n]$. Suppose to simplify the writing that $H = [1..h]$ and let $\mathsf{C} = \mathsf{C}_1 \parallel \cdots \parallel \mathsf{C}_h$ be a partial product of $(\parallel_{i \in H} \mathcal{L}_i)$ such that: (i) for all $i \notin H$, $\Sigma(\mathcal{L}_i) \cap \lambda(\mathsf{C}) = \emptyset$, and (ii) for all $i \in H$, $\Sigma(\mathcal{L}_i) \cap \lambda(\mathsf{C}) \subseteq \Sigma(\mathsf{C}_i)$.*

If some global state $s = (s_1, \ldots, s_n)$ is reachable in C with path π then the global state $s^* = (s_1^*, \ldots, s_n^*)$ defined by $s_i^* = s_i$ if $i \in H$ and s_i^* is an arbitrary state of $\mathcal{L}_i$ otherwise is reachable in $\mathcal{L}$ with the same path from the state $s^0 = (s_1^0, \ldots, s_n^0)$ such that s_i^0 is the initial state of $\mathcal{L}_i$ if $i \in H$ and $s_i^0 = s_i^*$ otherwise.

3 The Lazy Reachability Analysis Algorithm

In the following subsections, we present our algorithm. It makes use of the classical abstract list data-structure, with the usual operations: $\mathsf{hd}(), \mathsf{tl}(), \mathsf{len}()$ give respectively the *head*, *tail*, and *length* of a list. Operator : is the list constructor (prepend) and ++ is concatenation. $\mathsf{rev}()$ reverses a list. [ ] is the empty list and $L[i]$ is the i^{th} element of list L.

Algorithm 1 is the main function solving our reachability problems. It starts from a partition of the LTSs involved in the reachability objective. The idea here is to decompose this objective and verify it separately on each involved component with the hope that they do not interact. List Ls has (initially) one element per part of the initial partition. Each of those elements is a list of tuples $(\mathsf{A}, \mathsf{C}, I, J, K)$, described in details in the next subsection, that represent more and more concrete partial products (in the sense that they include more and more LTSs) of the system built around the LTSs in each partition, as we go towards the end of the list. In our algorithm, this list is walked along using the functional programming idiom of Huet's Zipper by decomposing it in Left, the current element, and Right, but it could as well be represented as a big array with a current index, etc.

We start with partial products consisting of only the initial states of each involved LTSs. The algorithm then performs two main tasks: concretisation and merging.

3.1 Concretisation

Concretisation consists in extending the partial products in two different directions: by computing more and more states and transitions for a given number of LTSs, and by adding LTSs. The (indices of the) LTSs currently used in the products are in the set J, and those being added are in set K. Set I serves as a memory of the initial partition of LTSs involved

Algorithm 1 Algorithm solving $RP^{\mathcal{R}}_{\mathcal{L}}$ ($\mathbb{L}_g$: indices of LTSs involved in $\mathcal{R}$)

```
1:  function Solve(L,R)
2:      choose a partition {I_1, ..., I_p} of L_g
3:      ∀k ∈ [1..p], let ID_k =‖ i∈I_k id(L_i) and INI_k =‖ i∈I_k ini(L_i).
4:      Ls ← [([ ], (ID_1, INI_1, I_1, ∅, I_1), [ ]), ..., ([ ], (ID_p, INI_p, I_p, ∅, I_p), [ ])]
5:      Complete ← False
6:      Consistent ← False
7:      while not Complete or not Consistent do
8:          Complete ← ∀k, Ls[k] is complete
9:          if not Complete then
10:             optional unless Consistent
11:                 mayHaveSol ← Concretise(Ls)
12:                 if not mayHaveSol then return False
13:                 end if
14:             end option
15:         end if
16:         Consistent ← Ls is consistent
17:         if not Consistent then
18:             optional unless Complete
19:                 Merge(Ls)
20:             end option
21:         end if
22:     end while
23:     return True
24: end function
```

in the objective. LTS C is the current partial product we have computed and A represents what we had computed at the previous level (before we added the LTSs in K).

The goal of the Concretise() function (Algorithm 2) is to find a partial product reaching the objective and using only LTSs that are either in J or in K. This is what we call *complete* and is formalized as follows:

▶ **Definition 12** (Completeness). The tuple $(\mathsf{Left}, (\mathsf{A}, \mathsf{C}, I, J, K), \mathsf{Right})$ is *complete* if $\exists \mathsf{C}^* \sqsubseteq \mathsf{C}$ such that $\mathsf{C}^* \to \mathcal{R}_{|J\cup K}$ and $\{i \notin J \cup K \ : \ \Sigma(\mathcal{L}_i) \cap \lambda(\mathsf{C}^*) \neq \emptyset\} = \emptyset$.

To ensure completeness, we have to add LTSs that share actions with our current partial product: this is the role of the case in line 10.

LTS A serves as a limit as to what we are allowed to compute at a given level. If we need to compute more, because we cannot find the (partial) objective, we have to backtrack to the previous level to increase this limit (line 19). If we cannot backtrack (line 16), then we have explored the whole product only missing some components, so we have an over-approximation, which implies that the property is false. If we successfully backtrack, and when we have extended again our partial product at that "lower" level, we can go forth again using the memory of what we had already computed (line 7).

Note the use of the Forget() function in line 19 that permits to "forget" part of that memory when we backtrack. This is useful if we are able to determine that we had made bad moves: for instance in the choice of the LTSs to add to the product. Many implementations of this function are possible provided they have the following property:

Algorithm 2 Auxiliary function CONCRETISE() for Algorithm 1

```
1: function CONCRETISE(Ls)
2:     choose k ∈ [0..len(Ls) − 1] such that Ls[k] is not complete
3:     (Left, (A, C, I, J, K), Right) ← Ls[k]
4:     if there exists C* s.t. C ⊏ C* ⊑ (‖ i∈K L_i) ‖ A and C* → R_|J∪K  then
5:         choose such a C*
6:         N ← {i ∉ J ∪ K : Σ(L_i) ∩ λ(C*) ≠ ∅}
7:         case Right ≠ [ ]
8:             (_, C', _, J', K') ← hd(Right)
9:             Ls[k] ← ((A, C*, I, J, K) : Left, (C*, C', I, J', K'), tl(Right))
10:        case Right = [ ] and N ≠ ∅
11:            choose ∅ ⊂ K' ⊆ N
12:            Ls[k] ← ((A, C*, I, J, K) : Left, (C*, ‖_{i∈J∪K∪K'} ini(L_i), I, J ∪ K, K'), [ ])
13:        case Right = [ ] and N = ∅
14:            Ls[k] ← (Left, (A, C*, I, J, K), [ ])
15:    else
16:        if Left = [ ] then return False
17:        else
18:            Right' ← FORGET((A, C, I, J, K) : Right)
19:            Ls[k] ← (tl(Left), hd(Left), Right')
20:        end if
21:    end if
22:    return True
23: end function
```

▶ **Property 13** (Good FORGET() implementation)**.** Let Lt be a list $[(\mathsf{A}_1, \mathsf{C}_1, I_1, J_1, K_1), \ldots, (\mathsf{A}_n, \mathsf{C}_n, I_n, J_n, K_n)]$ of tuples. Then FORGET(Lt) must be a list $[(\mathsf{A}'_1, \mathsf{C}'_1, I'_1, J'_1, K'_1), \ldots, (\mathsf{A}'_m, \mathsf{C}'_m, I'_m, J'_m, K'_m)]$ of tuples with $0 \leq m \leq n$ and $f : [1..m] \to [1..n]$ a non-decreasing function, such that:

- $\mathsf{A}'_1 = \mathsf{A}_1$, $J'_1 = J_1$,
- $\forall j \in [1..m]$,
  - $I'_j = I_1$ (remark that, by construction, $I_1 = I_2 = \cdots = I_n$),
  - $\emptyset \subset K'_j \subseteq (J_{f(j)} \cup K_{f(j)}) \setminus J'_j$
  - $\mathsf{C}'_j \sqsubseteq C_{f(j)}$ and $\| \; _{k \in J'_j \cup K'_j} \mathsf{ini}(\mathcal{L}_k) \sqsubseteq \mathsf{C}'_j \sqsubseteq (\|_{k \in K'_j} \mathcal{L}_k) \parallel \mathsf{A}'_j$
- $\forall j \in [2..m]$,
  - $\mathsf{A}'_j = \mathsf{C}'_{j-1}$
  - $J'_j = J'_{j-1} \cup K'_{j-1}$

In some sens, a good FORGET() implementation can be seen as a restriction of Lt to a subset of its elements. Each of these elements being itself restricted to a less concrete tuple (by taking subsets of C, J, and/or K).

The two most obvious implementations are either to never forget anything (then FORGET() is just the identity function) or to always forget everything (that is, take $m = 0$ in the above property – then FORGET() always returns the empty list). It is clear that these two choices satisfy Property 13.

Finally, in any element of Ls, we have a list of partial products that concern more and more LTSs. To sum up, a call to CONCRETISE() at least adds new LTSs to the current tuple $(\mathsf{A}, \mathsf{C}, I, J, K)$, or adds paths in C.

3.2 Merging

Merging occurs when two parts of a partition concretised independently use common LTSs. We have to ensure that these common LTSs behave consistently and we therefore merge the two partitions by computing the product of the two partial products. This use of common LTSs in different partitions we call inconsistency and it is formals as follows:

▶ **Definition 14** (Consistency). The list of triples $\mathsf{Ls} = [(\mathsf{Left}_1, (\_, \_, \_, J_1, K_1), \mathsf{Right}_1), \ldots, (\mathsf{Left}_n, (\_, \_, \_, J_n, K_n), \mathsf{Right}_n)]$ is *consistent* if $\forall i \neq j \in [1..n]$, $(J_i \cup K_i) \cap (J_j \cup K_j) = \emptyset$

Merging itself is done by the MERGE() function. As for the FORGET() function, many implementations are possible provided they satisfy Property 15:

▶ **Property 15** (Good MERGE() implementation). Let Ls be a non-consistent list of triples $[\mathcal{T}_1, \ldots, \mathcal{T}_n]$. For any triple $\mathcal{T}_k$, note $\mathcal{T}_k = (\mathsf{Left}_k, (\mathsf{A}_k, \mathsf{C}_k, I_k, J_k, K_k), \mathsf{Right}_k)$ and $\mathsf{Full}(\mathcal{T}_k) = \mathsf{rev}(\mathsf{Left}_k) +\!\!+ [(\mathsf{A}_k, \mathsf{C}_k, I_k, J_k, K_k)] +\!\!+ \mathsf{Right}_k$. Then MERGE($\mathsf{Ls}$) is any list $[\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_{j+1}, \ldots, \mathcal{T}_n, \mathcal{T}]$ of triples such that:

- $(J_i \cup K_i) \cap (J_j \cup K_j) \neq \emptyset$ (such i, j always exist because Ls is not consistent)
- There exist two non-decreasing functions $f_i : [1..m] \to [1..n_i]$ and $f_j : [1..m] \to [1..n_j]$ so that $\mathsf{Full}(\mathcal{T}) = [(\mathsf{A}''_1, \mathsf{C}''_1, I''_1, J''_1, K''_1), \ldots, (\mathsf{A}''_m, \mathsf{C}''_m, I''_m, J''_m, K''_m)]$, $\mathsf{Full}(\mathcal{T}_i) = [(\mathsf{A}_1, \mathsf{C}_1, I_1, J_1, K_1), \ldots, (\mathsf{A}_{n_i}, \mathsf{C}_{n_i}, I_{n_i}, J_{n_i}, K_{n_i})]$, and $\mathsf{Full}(\mathcal{T}_j) = [(\mathsf{A}'_1, \mathsf{C}'_1, I'_1, J'_1, K'_1), \ldots, (\mathsf{A}'_{n_j}, \mathsf{C}'_{n_j}, I'_{n_j}, J'_{n_j}, K'_{n_j})]$ with $m \leq n_i + n_j$ verify:
  - $\mathsf{A}''_1 = \mathsf{A}_1 \parallel \mathsf{A}'_1$, $J''_1 = J_1 \cup J'_1 = \emptyset$ (remark that, by construction, $J_1 = J'_1 = \emptyset$),
  - $I_1 \cup I'_1 \subseteq K''_1 \subseteq J_{f_i(1)} \cup K_{f_i(1)} \cup J'_{f_j(1)} \cup K'_{f_j(1)}$
  - $\forall k \in [1..m]$,
    - $I''_k = I_1 \cup I'_1$ (remark that, by construction, $I_1 = \cdots = I_{n_i}$ and $I'_1 = \cdots = I'_{n_j}$),
    - $\mathsf{C}''_k \sqsubseteq \mathsf{C}_{f_i(k)} \parallel \mathsf{C}_{f_j(k)}$ and $\parallel_{\ell \in J''_k \cup K''_k} \mathsf{ini}(\mathcal{L}_\ell) \sqsubseteq \mathsf{C}''_k \sqsubseteq (\parallel_{\ell \in K''_k} \mathcal{L}_\ell) \parallel \mathsf{A}''_k$
  - $\forall k \in [2..m]$,
    - $\emptyset \subset K''_k \subseteq (J_{f_i(k)} \cup K_{f_i(k)} \cup J'_{f_j(k)} \cup K'_{f_j(k)}) \setminus J''_k$
    - $\mathsf{A}''_k = \mathsf{C}''_{k-1}$
    - $J''_k = J''_{k-1} \cup K''_{k-1}$

Intuitively, a good MERGE() implementation should select two triples breaking the consistency of Ls and merge them. These two triples are in fact two histories of concretisations (i.e. the result of a sequence of calls to CONCRETISE()). Merging them basically consists in interleaving these two histories and then apply a FORGET()-like construct on this interleaving. This produces an history that could have been produced by a sequence of calls to CONCRETISE() starting from $([\,], (\parallel_{i \in I_1 \cup I'_1} \mathsf{id}(\mathcal{L}_i), \parallel_{i \in I_1 \cup I'_1} \mathsf{ini}(\mathcal{L}_i), I_1 \cup I'_1, \emptyset, I_1 \cup I'_1), [\,])$, i.e if $I_1 \cup I'_1$ had been an element of the initial partition chosen in Algorithm 1.

At the lowest level of implementation, the basic merging operation of two tuples $h_1 = (\mathsf{A}_1, \mathsf{C}_1, I_1, J_1, K_1)$ and $h_2 = (\mathsf{A}_2, \mathsf{C}_2, I_2, J_2, K_2)$ gives the tuple $(\mathsf{A}_1 \parallel \mathsf{A}_2, \mathsf{C}_1 \parallel \mathsf{C}_2, I_1 \cup I_2, J_1 \cup J_2, (K_1 \setminus J_2) \cup (K_2 \setminus J_1))$. Let us denote it by $h_1 * h_2$. Building on that, a minimal implementation satisfying Property 15 would be to select i and j such that $\mathsf{Ls}[i] = (\mathsf{Left}_i, h_i, \mathsf{Right}_i)$, with $h_i = (\mathsf{A}_i, \mathsf{C}_i, I_i, J_i, K_i)$ and $\mathsf{Ls}[j] = (\mathsf{Left}_j, h_j, \mathsf{Right}_j)$, with $h_j = (\mathsf{A}_j, \mathsf{C}_j, I_j, J_j, K_j)$, and $(J_i \cup K_i) \cap (J_j \cup K_j) \neq \emptyset$, remove them from Ls and replace them by the singleton list $([\,], \mathsf{hd}(\mathsf{Left}_i) * \mathsf{hd}(\mathsf{Left}_j), [\,])$. Another relevant choice, which is easy to compute and also trivially satisfies Property 15, would be to also keep the current elements by replacing both $\mathsf{Ls}[i]$ and $\mathsf{Ls}[j]$ by $([\mathsf{hd}(\mathsf{Left}_i) * \mathsf{hd}(\mathsf{Left}_j)], h_i * h_j, [\,])$.

When we have found a list of complete partial products that is consistent then we know that our objective is reachable.

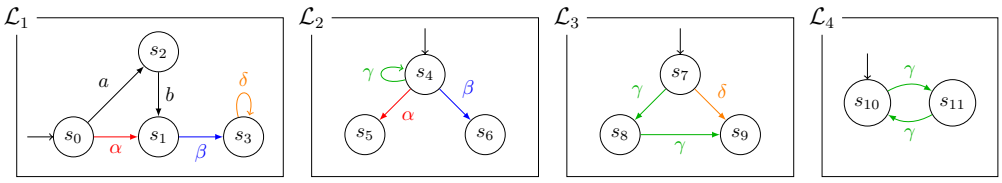


Figure 1 A compound system with four LTSs ($\mathcal{L}_1$ to $\mathcal{L}_4$).

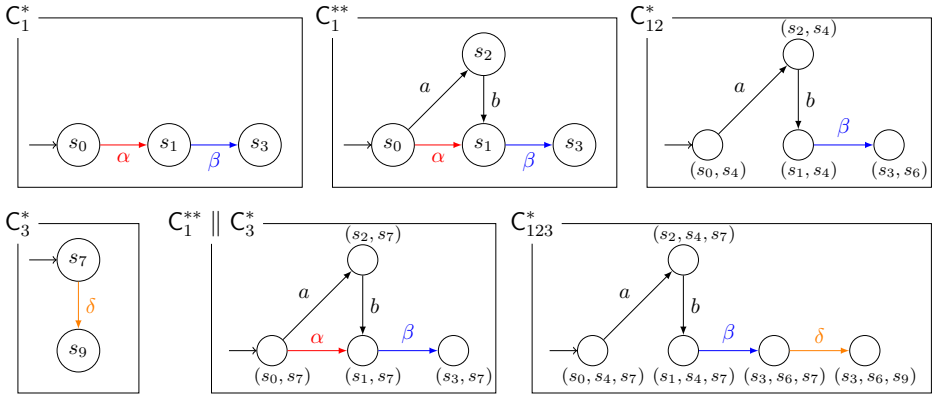


Figure 2 Some LTSs appearing during an execution of our algorithm on the example of Figure 1.

3.3 Example

Let us perform a sample execution of the algorithm on the system described in Figure 1. Suppose we want to reach the partial state $(s_3, \star, s_9, \star)$. Therefore we have the set of indices of the LTSs involved in the objective $\mathbb{L}_g = \{1, 3\}$ and we choose to partition it, for instance, as $\{\{1\}, \{3\}\}$.

Then, $\mathsf{Ls}[1]$ is the list $[([\,], (\mathsf{id}(\mathcal{L}_1), \mathsf{ini}(\mathcal{L}_1), \{1\}, \emptyset, \{1\}), [\,])]$ and, similarly, $\mathsf{Ls}[2]$ is the list $[([\,], (\mathsf{id}(\mathcal{L}_3), \mathsf{ini}(\mathcal{L}_3), \{3\}, \emptyset, \{3\}), [\,])]$. None of those elements is complete because they do not reach their part of the objective so we call Concretise().

In that function, we choose for instance $k = 1$ and compute an extension of $\mathsf{ini}(\mathcal{L}_1)$ that reaches the objective. Say we compute the extension C_1^* made of the path s_0, s_1, s_3 (Figure 2). Then, since labels α and β are shared with $\mathcal{L}_2$, and $2 \notin K_1 = \{1\}$, we have $\mathcal{N} = \{2\}$. Since $\mathsf{Right}_1 = [\,]$, we get to line 10, replace $\mathsf{ini}(\mathcal{L}_1)$ by C_1^*, add the tuple $(\mathsf{C}_1^*, \mathsf{ini}(\mathcal{L}_1) \parallel \mathsf{ini}(\mathcal{L}_2), \{1\}, \{1\}, \{2\})$ to the list represented by $\mathsf{Ls}[1]$, and set that tuple as the current element of that list. Finally, we return true.

Back in the main function, Ls is consistent for now, so we move to the next iteration. Again both $\mathsf{Ls}[1]$ and $\mathsf{Ls}[2]$ are not complete because their current elements do not reach their objective. Let us say we concretise again $\mathsf{Ls}[1]$. This time we cannot find an extension of $\mathsf{ini}(\mathcal{L}_1) \parallel \mathsf{ini}(\mathcal{L}_2)$ in the product of $\mathcal{L}_1$, $\mathcal{L}_2$ and C_1^*, which is empty. Then, since Left_1 is not empty, we go to line 18, call Forget() (suppose here it forgets nothing), and set again the head of the list represented by $\mathsf{Ls}[1]$ (which is also the head of Left_1) as the current element. Then we return true again.

Back in the main function, Ls is still consistent and both its element are not complete. We then call Concretise() and for the example choose again $k = 1$. This time we extend C_1^* by taking C_1^{**} as the whole of $\mathcal{L}_1$ except the δ transition (Figure 2). Since Right_1 is not empty this time, we go to line 7, update the tuples with C_1^{**}, and move the current element of the list right, then return true.

Back in the main function we still call Concretise() and choose again $k = 1$. This time, we can extend $\mathsf{ini}(\mathcal{L}_1) \parallel \mathsf{ini}(\mathcal{L}_2)$ with the LTS C^*_{12} made only of the path $(s_0, s_4), (s_2, s_4), (s_1, s_4), (s_3, s_6)$ (Figure 2). Set $\mathcal{N}$ is empty because no LTS other than $\mathcal{L}_1$ and $\mathcal{L}_2$ has labels β, a or b (used in this path). Right_1 is also empty so we just update the current element in $\mathsf{Ls}[1]$ with C^*_{12} (line 14) and return true.

Back in the main function, $\mathsf{Ls}[1]$ is now complete but not $\mathsf{Ls}[2]$. So, we call Concretise() and choose $k = 2$. We extend $\mathsf{ini}(\mathcal{L}_3)$ by C^*_3 made of the path s_7, s_9 (Figure 2). Then $\mathcal{N} = \{1\}$ because $\mathcal{L}_1$ shares label δ with C^*_3 and, as before, Right_2 being empty, we go to line 10, replace $\mathsf{ini}(\mathcal{L}_3)$ by C^*_3, add the tuple $(\mathsf{C}^*_3, \mathsf{ini}(\mathcal{L}_3) \parallel \mathsf{ini}(\mathcal{L}_1), \{3\}, \{3\}, \{1\})$ to the list represented by $\mathsf{Ls}[2]$, and set that tuple as the current element of that list. Then we return true.

Now, we have the index 1 in $J \cup K$ for both the current elements of $\mathsf{Ls}[1]$ and $\mathsf{Ls}[2]$, so we can choose to merge them. Let us do it: we use the second of the simple strategies outlined above: we keep and merge only the first and current elements of each list. After the call to Merge(), $\mathsf{Ls} = [([(\mathsf{id}(\mathcal{L}_1) \parallel \mathsf{id}(\mathcal{L}_3), \mathsf{C}^{**}_1 \parallel \mathsf{C}^*_3, \{1,3\}, \emptyset, \{1,3\})], (\mathsf{C}^{**}_1 \parallel \mathsf{C}^*_3, \mathsf{C}^*_{12} \parallel \mathsf{ini}(\mathcal{L}_3) \parallel \mathsf{ini}(\mathcal{L}_1), \{1,3\}, \{1,3\}, \{2\}), [\,])]$ and $\mathsf{C}^*_{12} \parallel \mathsf{ini}(\mathcal{L}_3) \parallel \mathsf{ini}(\mathcal{L}_1)$ consists only of state (s_0, s_4, s_7), because $\mathsf{ini}(\mathcal{L}_1)$ restricts C^*_{12} to its initial state and δ is not in C^*_{12}. That product has no path to a state $(s_3, \star, s_9)$, so the new $\mathsf{Ls}[1]$ is not complete and we need to call Concretise() one last time.

We will then be able to extend $\mathsf{C}^*_{12} \parallel \mathsf{ini}(\mathcal{L}_3) \parallel \mathsf{ini}(\mathcal{L}_1)$ to an LTS C^*_{123} containing state (s_3, s_6, s_9) but with no transition γ (Figure 2). Then $\mathcal{N}$ is empty, as well as Right_1, so we go through line 14 to update the current LTS, and return true. Finally, in the main function, we now have that Ls is consistent, since it contains only one element, and that element is complete because C^*_{123} contains only labels not shared with $\mathcal{L}_4$. So we terminate and return true.

Note that we never needed to consider products involving $\mathcal{L}_4$ – which is the reason why we call our analysis "lazy".

3.4 Soundness, completeness, termination

We now proceed to proving that our algorithm is sound and complete and that it terminates. We first state two utility lemmas.

▶ **Lemma 16.** *Let* $(\mathit{Left}, h, \mathit{Right})$ *be an element of* Ls *in* Solve$(\mathcal{L}, \mathcal{R})$ *and let* $\mathcal{L} = (\|_{i \in [1..n]} \mathcal{L}_i)$.

Let $(\mathsf{A}, \mathsf{C}, I, J, K)$ *be either* h*, or an element of* Left*, or an element of* Right*. Then* C *is a partial product of* $(\|_{i \in J \cup K} \mathcal{L}_i)$ *and, if we write* $\mathsf{C} = (\|_{i \in J \cup K} \mathsf{C}_i)$*, then* $\Sigma(\mathsf{C}_i) = \Sigma(\mathcal{L}_i)$.

▶ **Lemma 17.** *Let* $(\mathit{Left}, (\mathsf{A}, \mathsf{C}, I, J, K), \mathit{Right})$ *be an element of* Ls *in* Solve$(\mathcal{L}, \mathcal{R})$*. If* Left *is empty then* $I \subseteq K$*,* $J = \emptyset$ *and* $\mathsf{A} = \|_{i \in I} \mathsf{id}(\mathcal{L}_i)$.

And finally the main results:

▶ **Proposition 18.** If Solve$(\mathcal{L}, \mathcal{R})$ returns False then $\mathcal{R}$ is not reachable in $\mathcal{L}$.

Proof. The only way Solve$(\mathcal{L}, \mathcal{R})$ can return False is through line 12 in Algorithm 1, and in turn this means that the call to Concretise() in line 11 returned False. Now Concretise() will only return False through line 16 in Algorithm 2. To get there, there must exist some k such that $\mathsf{Ls}[k]$ can be decomposed as the triple $(\mathsf{Left}, (\mathsf{A}, \mathsf{C}, I, J, K), \mathsf{Right})$, with (1) Left being empty, and (2) there is no C^* s.t. $\mathsf{C} \sqsubset \mathsf{C}^* \sqsubseteq (\|_{i \in K} \mathcal{L}_i) \parallel \mathsf{A}$ and $\mathsf{C}^* \to \mathcal{R}_{|J \cup K}$.

By Lemma 16 we know that C is a partial product of $\mathcal{L}$. With (1) and Lemma 17, we have that $I \subseteq K$, $J = \emptyset$ and $\mathsf{A} = \|_{i \in I} \mathsf{id}(\mathcal{L}_i)$. So $(\|_{i \in K} \mathcal{L}_i) \parallel \mathsf{A} = (\|_{i \in K} \mathcal{L}_i)$. Then, with (2),

we can deduce that $\mathcal{R}_{|K}$ is not reachable in $(\|_{i\in K}\mathcal{L}_i)$, and finally with the contrapositive of Lemma 10, we get that $\mathcal{R}$ is not reachable in $\mathcal{L}$. ◀

▶ **Proposition 19.** If $\textsc{Solve}(\mathcal{L},\mathcal{R})$ returns True then $\mathcal{R}$ is reachable in $\mathcal{L}$.

Proof. The only way $\textsc{Solve}(\mathcal{L},\mathcal{R})$ can return True is through line 23 in Algorithm 1. This can only happen when Ls is such that (1) for all k, $\mathsf{Ls}[k]$ is complete and (2) Ls is consistent.

If we denote by $(\mathsf{A}_k,\mathsf{C}_k,I_k,J_k,K_k)$ the second component of $\mathsf{Ls}[k]$, and by H_k the union $J_k\cup K_k$, (1) translates to $\forall k$, there exists $\mathsf{C}_k^*\sqsubseteq\mathsf{C}_k$ such that C_k^* can reach $\mathcal{R}_{|H_k}$ and $\{i\notin H_k : \Sigma(\mathcal{L}_i)\cap\lambda(\mathsf{C}_k^*)\neq\emptyset\}=\emptyset$. Similarly, (2) translates to $\forall i,j, H_i\cap H_j=\emptyset$.

By Lemma 16, each C_k^* is a partial product of $(\|_{i\in H_k}\mathcal{L}_i)$ (and thus also of $\mathcal{L}$). Now consider some $i\in H_k$ and $\sigma\in\Sigma(\mathcal{L}_i)\cap\lambda(\mathsf{C}_k^*)$. By Lemma 16, we know that there exist some C_i such that $\mathsf{C}^*=(\|_{i\in H_k}\ \mathsf{C}_i)$ and $\Sigma(\mathsf{C}_i)=\Sigma(\mathcal{L}_i)$. Consequently, for all $i\in H_k$, $\Sigma(\mathcal{L}_i)\cap\lambda(\mathsf{C})\subseteq\Sigma(\mathsf{C}_i)$. From (2), we also have that $\forall i\notin H_k$, $\Sigma(\mathcal{L}_i)\cap\lambda(\mathsf{C})=\emptyset$ and we can thus use Lemma 11 and obtain that $\mathcal{R}_{|H_k}$ is reachable in $\mathcal{L}$, whatever the states of the components not in H_k (and leaving them unchanged). Therefore, by finally putting all components together, $\mathcal{R}$ is reachable in $\mathcal{L}$. ◀

Relation $\sqsubseteq$ is not sufficient to reflect progress in our algorithm. We therefore introduce a new relation $<_\ell$, built on top of $\sqsubseteq$, as a partial order over lists Ls (as the ones appearing in Algorithm 1). Relation $<_\ell$ does reflect progress in concretisation (by advancing in the history of concretisations (2), or by adding LTSs (3), or by adding paths in partial products (4,5)), and merging (by reducing the length of the list (1)).

▶ **Definition 20.** Given two lists of tuples Ls_1 and Ls_2 with the same type as Ls in Algorithm 1, we define $<_\ell$ such that $\mathsf{Ls}_1<_\ell\mathsf{Ls}_2$ if and only if $\mathsf{Ls}_1\neq\mathsf{Ls}_2$ and:

- $\mathsf{len}(\mathsf{Ls}_1)>\mathsf{len}(\mathsf{Ls}_2)$, or (1)
- $\mathsf{len}(\mathsf{Ls}_1)=\mathsf{len}(\mathsf{Ls}_2)$ and $\forall k, \mathsf{Ls}_1[k]\neq\mathsf{Ls}_2[k] \implies \mathsf{Ls}_1[k]<_t\mathsf{Ls}_2[k]$;

where $(\mathsf{Left}_1,h_1,\mathsf{Right}_1)<_t(\mathsf{Left}_2,h_2,\mathsf{Right}_2)$ if and only if, for $\mathsf{hLr}_i=\mathsf{rev}(\mathsf{Left}_i)+\!\!+[h_i]$,

- hLr_2 is a prefix of $\mathsf{rev}(\mathsf{Left}_1)$, or (2)
- $\exists\ell, \mathsf{hLr}_1[\ell]\neq\mathsf{hLr}_2[\ell]$ and, for the smallest such ℓ one has $\mathsf{hLr}_1[\ell]<_a\mathsf{hLr}_2[\ell]$;

where $(\mathsf{A}_1,\mathsf{C}_1,I_1,J_1,K_1)<_a(\mathsf{A}_2,\mathsf{C}_2,I_2,J_2,K_2)$ if and only if:

- $J_1\cup K_1\subset J_2\cup K_2$, or (3)
- $J_1\cup K_1=J_2\cup K_2$ and $\mathsf{A}_1\sqsubset\mathsf{A}_2$, or (4)
- $J_1\cup K_1=J_2\cup K_2$, $\mathsf{A}_1=\mathsf{A}_2$, and $\mathsf{C}_1\sqsubset\mathsf{C}_2$. (5)

If $\mathsf{Ls}_1<_\ell\mathsf{Ls}_2$ or $\mathsf{Ls}_1=\mathsf{Ls}_2$ we write $\mathsf{Ls}_1\leq_\ell\mathsf{Ls}_2$.

▶ **Proposition 21.** The calls to $\textsc{Solve}(\mathcal{L},\mathcal{R})$ always terminate (and return only True or False).

Sketch of the proof. The fact that, if a call to $\textsc{Solve}(\mathcal{L},\mathcal{R})$ terminates, it can only return True or False, comes from lines 23 (returning True) and 12 (returning False) of Algorithm 1 which are the only return statements of the $\textsc{Solve}(,)$ function.

In order to prove the termination one can show that (1) $\leq_\ell$ is an order relation over the Ls used in Algorithm 1, (2) the set of such lists appearing in any instance of Algorithm 1 is finite (and so, there are lists which are greater or incomparable to any other lists with respect to $\leq_\ell$), and (3) any step of the while loop of Algorithm 1 terminates and if the return of line 12 is not used, strictly increases Ls with respect to $\leq_\ell$. From (1) and (2) one then gets that, in any instance of Algorithm 1, there cannot exist an infinite strictly increasing chain of Ls with respect to $\leq_\ell$. Hence, from (3), Algorithm 1 always terminates. ◀

4 Experimental analysis

In order to get insight on the practical efficiency of our algorithm we developed a tool[1] (LaRA, for Lazy Reachability Analyzer) using it. We then compared the time efficiency of LaRA with that of other tools on several reachability analysis tasks in distributed systems. We originally selected three other tools for these experiments:

- LoLA[2]: A Petri net analyzer (it is straightforward to convert the compound systems we consider in this paper into (safe) Petri nets) which efficiently implements many techniques for model checking Petri nets. LoLA is arguably very effective for reachability analysis in Petri nets as it won the reachability track at the last model checking contest [13].
- PMC [14]: A tool for partial model checking that uses incremental techniques for dealing with the verification of distributed systems.
- The on the fly model checking capabilities of the CADP toolbox [8]

Early preliminary experiments revealed that, on all our benchmarks, LoLA outperformed PMC and CADP. For the larger experiments on which we report here we thus focused on comparing LoLA with our tool.

4.1 Implementation choices

LaRA consists of approximately 500 lines of Haskell code, using the standard Parsec parsing library, and the fgl graph library. Note that memory management in Haskell is automatic.

We have presented our algorithm in a manner as generic as possible including the possibility for many heuristic choices. For our first prototype presented here, we have chosen to completely compute each partial product before adding more components. This eliminates the need for backtracking, which greatly simplifies the code and, to some extent, favors the case when the desired state is not reachable. This choice is rather drastic and probably not optimal when exploring big partial products, in which a more on-the-fly approach would usually give better results. However, we believe it is reasonable, as our objective was to evaluate the influence of the laziness feature of our approach.

Before adding LTSs to the partial product, we trim it by keeping only the reachable and coreachable states. We add only one automaton each time, chosen arbitrarily in the set of LTSs synchronized on the path that synchronizes as few LTSs as possible.

4.2 Benchmarks

Our benchmarks where taken from a set of benchmarks proposed by Corbett in the 90's [5]. Among these, we selected the ones where scaling increases the number of components but does not change the size of individual components. The reason for this choice is that our early implementation is not made for dealing with large state spaces of individual components – as, again, our goal is to evaluate the impact of its laziness feature. Not embedding efficient search techniques it was hopeless to cope with finely tuned model checking tools. This left us with six models, described in Table 1. For each model we define a simple reachability property and state if it is verified by the model.

Both tools were used in a similar setting: on a machine with four Intel® Xeon® E5-2620 processors (six cores each) with 128GB of memory. Though this machine has some potential

[1] For reproducibility of experiments, LaRA is available at `http://lara.rts-software.org`
[2] `http://service-technology.org/lola/`

Table 1 Benchmarks description.

Model	Description	Size	Property	Verified?
Cyclic	Milner's cyclic scheduler, a set of tasks have to be scheduled in a cyclic order.	Number of tasks to be scheduled.	One task in two can be in their waiting state together.	Yes.
DAC	Divide and conquer computation, a task has to be completed by a set of processes. Each one can complete the task alone or fork.	Maximal number of processes involved in the solving of the task.	A given process can be involved in the solving and decide to complete it alone.	Yes.
Philo	Dining philosophers, in its eating cycle a philosopher takes and releases his left fork first.	Number of philosophers.	One philosopher in two can eat together.	Yes for even sizes. No for odd sizes.
PhiloDico	Variation of Philo, a dictionary turns around the table, preventing the philosopher holding it to take forks.	Number of philosophers.	One philosopher in two can eat together.	Yes for even sizes. No for odd sizes.
PhiloSync	Variation of Philo, philosophers take and release both their forks in a single step.	Number of philosophers.	One philosopher in two can eat together.	Yes for even sizes. No for odd sizes.
TokenRing	Classical mutual exclusion algorithm with a token circulating on a ring of processes.	Number of processes.	Two given processes can reach their critical section together.	No.

for parallel computing, all the experiments presented here are actually monothreaded. We put a time limit of 20 minutes for computations. For each experiment, each tool had as input a file in its own format: file generation and conversion are not taken into account in the processing times.

4.3 Positive results.

In most of the cases (namely Cyclic, Philo, PhiloDico, and PhiloSync) our tool outperformed LoLA with increasing size of models. On DAC, our results are comparable to those obtained with LoLA, and for very large instances we slightly outperform it. The experimental results for these cases are summarized in Table 2. For each model, timeout indicates the first instance of this model for which a tool reached the time limit of 20 minutes. Notice that, in the variants of the dining philosophers, there are two timeouts: one for instances of odd size and the other one for instances of even size. This is because the property we verify is false for odd sizes and true for even sizes, which makes a significant difference for LoLA.

4.4 Focus on TokenRing.

Table 3 presents the results obtained with various modeling of TokenRing. It compares runtimes of LoLA and our tool on Corbett's modeling. It appears that LaRA is far from efficient on this particular example. This is due to the fact that, without taking into account all the components, it is not possible for our tool to figure out that only one token exists in the system. So, for deciding that no two processes can be in their critical sections together, LaRA cannot be lazy and has to explore the full state space of the system.

Table 2 Comparison of runtimes of LoLA and LaRA on instances of increasing size of Cyclic, DAC, Philo, PhiloDico, and PhiloSync.

Size	Cyclic		DAC		Philo		PhiloDico		PhiloSync	
	LaRA	LoLA	LaRA	LoLA	LaRA	LoLA	LaRA	LoLA	LaRA	LoLA
15	0.01s	<0.01s	0.01s	<0.01s	0.04s	28.47s	0.10s	30.92s	0.02s	<0.01s
16	0.01s	<0.01s	0.01s	<0.01s	0.04s	<0.01s	0.05s	<0.01s	0.02s	<0.01s
17	0.01s	<0.01s	0.01s	<0.01s	0.05s	327.55s	0.10s	349.38s	0.02s	0.02s
18	0.01s	<0.01s	0.02s	<0.01s	0.04s	<0.01s	0.06s	<0.01s	0.03s	<0.01s
19	0.01s	<0.01s	0.01s	<0.01s	0.05s	**Timeout**	0.10s	**Timeout**	0.02s	0.05s
24	0.02s	<0.01s	0.01s	<0.01s	0.05s	<0.01s	0.08s	<0.01s	0.03s	<0.01s
25	0.02s	<0.01s	0.01s	<0.01s	0.06s		0.13s		0.03s	0.97s
35	0.03s	<0.01s	0.02s	<0.01s	0.08s		0.15s		0.04s	182.54s
45	0.03s	<0.01s	0.02s	<0.01s	0.11s		0.17s		0.06s	**Timeout**
1000	0.57s	2.55s	0.35s	0.56s	1.90s	2.44s	2.34s	2.50s	1.11s	2.38s
3000	2.68s	64.32s	1.08s	1.15s	6.87s	64.84s	8.56s	64.55s	4.82s	64.31s
6000	8.07s	514.89s	2.25s	1.62s	17.86s	520.86s	21.32s	523.54s	13.83s	519.21s
8000	13.37s	**Timeout**	2.97s	2.79s	27.63s	**Timeout**	32.21s	**Timeout**	22.15s	**Timeout**
10000	20.86s		3.72s	3.14s	39.73s		44.69s		33.10s	
30000	234.97s		11.24s	9.46s	334.79s		346.36s		319.15s	
50000	687.68s		19.10s	19.75s	1063.69s		1072.71s		946.86s	

Table 3 Runtimes on TokenRing.

Size	TokenRing	
	LaRA	LoLA
7	0.514s	<0.01s
8	1.716s	<0.01s
9	6.713s	<0.01s
10	25.810s	<0.01s
11	70.370s	<0.01s
12	322.440s	<0.01s
13	**Timeout**	<0.01s
1000		0.15s

5 Conclusion

We have presented a new approach for the verification of reachability in distributed systems. It builds on both decomposing the goal state into its projection on the different components of the system and lazily adding components in an iterative fashion to produce more and more precise over-approximations. This notably allows for early termination both when the state is reachable and when it is not. We have presented an algorithm based on this approach, together with proofs for completeness, soundness, and termination. We have also implemented this into an early prototype named LaRA. This rather naive implementation already gives very promising results, on which we report together with comparisons to LoLA, a state-of-art model-checker for Petri nets.

Further note that, when a reachability property is true, LaRA has computed, and can output, a consistent list of complete LTSs satisfying that property. An interesting plus-value is that adding whatever number of new components to that list would not change the outcome provided that none of those new components shares actions that are used in the list. Therefore in systems with a particular synchronization structure, like rings for instance, we can generalize the reachability result to any number of components in the ring : for instance to prove that a philosopher can eat we need to add the two forks around her and the other two philosophers that could also use those forks. Now, any additional fork or philosopher

beyond those do not share any action required to establish that the first philosopher can eat. We can then deduce that she can eat regardless of the number of philosophers around the table.

We have proposed and proved our algorithm in a generic and extendable way. In particular, it seems very likely that partial order or Decision Diagram-based symbolic techniques could be incorporated in this approach. The algorithm we propose also offers several opportunities for parallelisation. First, between two merge operations all concretisations in the different partitions can clearly be performed in parallel. Second, the different choices left open in the algorithm, such as the choice of a particular path to concretise, or a specific automaton to add to the product, can be resolved by some heuristics but may also better be handled by testing several of the different choices in parallel.

In addition to studying theses issues, further work includes extensions to more expressive formalisms, in particular (parametric) timed automata and time Petri nets, and to more complex properties.

Acknowledgments. We gratefully thank Frédéric Lang for the time he spent helping us to use his partial model checking tool. We also thank Karsten Wolf for offering help with LoLA. Finally, we thank the anonymous reviewers for their valuable comments.

References

1 H. R. Andersen. Partial model checking. In *LICS*, pages 398–407, 1995.
2 F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100, 1994.
3 A. R. Bradley. SAT-based model checking without unrolling. In *VMCAI*, pages 70–87, 2011.
4 A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV*, pages 277–293, 2011.
5 J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Trans. Software Eng.*, 22(3):161–180, 1996.
6 P. Crouzen and F. Lang. Smart reduction. In *FASE*, pages 111–126, 2011.
7 C. Flanagan and S. Qadeer. Thread-modular model checking. In *SPIN*, pages 213–224, 2003.
8 H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *STTT*, 15(2):89–107, 2013.
9 S. Graf and B. Steffen. Compositional minimization of finite state systems. In *CAV*, pages 186–196, 1990.
10 O. Grumberg and D. E. Long. Model checking and modular verification. *TOPLAS*, 16(3):843–871, 1994.
11 J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *JAIR*, 22(1):215–278, 2004.
12 G. J. Holzmann and D. Peled. An improvement in formal verification. In *FORTE*, pages 197–211, 1994.
13 F. Kordon, H. Garavel, L. M. Hillah, F. Hulin-Hubard, A. Linard, M. Beccuti, A. Hamez, E. Lopez-Bobeda, L. Jezequel, J. Meijer, E. Paviot-Adet, C. Rodriguez, C. Rohr, J. Srba, Y. Thierry-Mieg, and K. Wolf. Complete Results for the 2015 Edition of the Model Checking Contest. http://mcc.lip6.fr/2015/results.php, 2015.
14 F. Lang and R. Mateescu. Partial model checking using networks of labelled transition systems and boolean equation systems. *LMCS*, 9(4), 2013.
15 A. Lehmann, N. Lohmann, and K. Wolf. Stubborn sets for simple linear time properties. In *ICATPN*, pages 228–247, 2012.

Causally Consistent Dynamic Slicing

Roly Perera[*1], **Deepak Garg**[2], **and James Cheney**[†3]

1 **Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, UK**
rperera@inf.ed.ac.uk **and**
School of Computing Science, University of Glasgow, Glasgow, UK
roly.perera@glasgow.ac.uk

2 **Max Planck Institute for Software Systems, Saarbrücken, Germany**
dg@mpi-sws.org

3 **Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, UK**
jcheney@inf.ed.ac.uk

Abstract

We offer a lattice-theoretic account of dynamic slicing for π-calculus, building on prior work in the sequential setting. For any run of a concurrent program, we exhibit a Galois connection relating forward slices of the start configuration to backward slices of the end configuration. We prove that, up to lattice isomorphism, the same Galois connection arises for any causally equivalent execution, allowing an efficient concurrent implementation of slicing via a standard interleaving semantics. Our approach has been formalised in the dependently-typed language Agda.

1998 ACM Subject Classification D.1.3 Concurrent Programming; D.2.5 Testing and debugging

Keywords and phrases π-calculus; dynamic slicing; causal equivalence; Galois connection

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.18

1 Introduction

Dynamic slicing, due originally to Weiser [18], is a runtime analysis technique with applications in debugging, security and provenance tracking. The basic goal is to identify a sub-program, or *program slice*, that may affect an outcome of interest called the *slicing criterion*, such as the value of a variable. Dynamic slicing in concurrent settings is often represented as a graph reachability problem, thanks to influential work by Cheng [2]. However, most prior work on dynamic slicing for concurrency does not yield minimum slices, nor allows particularly flexible slicing criteria, such as arbitrary parts of configurations. Systems work on concurrent slicing [8, 13, 17] tends to be largely informal.

Perera et al [14] developed an approach where backward dynamic slicing is treated as a kind of (abstract) reverse execution or "rewind" and forward slicing as a kind of (abstract) re-execution or "replay". Forward and backward slices are related by a Galois connection, ensuring the existence of minimal slices. This idea is straightforward in the sequential setting of the earlier work. However, generalising it to concurrent programs is non-trivial. Suppose

* Perera was supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-13-1-3006. Perera was also supported by UK EPSRC project EP/K034413/1.

† Cheney was supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-13-1-3006.

	Scheduler thread 1	*Scheduler thread 2*	A_1	A_2
	$\mathbf{a_1}.c_1.(b_1.\overline{c_2}.\overline{r_1} + \overline{c_2}.b_1.\overline{r_1})$	$\overline{c_1}.a_2.c_2.(b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2})$	$\overline{\mathbf{a_1}}.\overline{b_1}.\overline{p_1}$	$\overline{a_2}.\overline{b_1}.\overline{p_2}$
⟶	$\mathbf{c_1}.(b_1.\overline{c_2}.\overline{r_1} + \overline{c_2}.b_1.\overline{r_1})$	$\overline{\mathbf{c_1}}.a_2.c_2.(b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2})$	$\overline{b_1}.\overline{p_1}$	$\overline{a_2}.\overline{b_1}.\overline{p_2}$
⟶	$b_1.\overline{c_2}.\overline{r_1} + \overline{c_2}.b_1.\overline{r_1}$	$\mathbf{a_2}.c_2.(b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2})$	$\overline{b_1}.\overline{p_1}$	$\overline{\mathbf{a_2}}.\overline{b_1}.\overline{p_2}$
⟶	$\mathbf{b_1}.\overline{c_2}.\overline{r_1} + \overline{c_2}.b_1.\overline{r_1}$	$c_2.(b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2})$	$\overline{b_1}.\overline{p_1}$	$\overline{\mathbf{b_1}}.\overline{p_2}$
⟶	$\overline{\mathbf{c_2}}.\overline{r_1}$	$\mathbf{c_2}.(b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2})$	$\overline{b_1}.\overline{p_1}$	$\overline{a_2}.\overline{b_1}.\overline{p_2}$
⟶	$a_1.c_1.(b_1.\overline{c_2}.\overline{r_1} + \overline{c_2}.b_1.\overline{r_1})$	$b_2.\overline{c_1}.\overline{r_2} + \overline{c_1}.b_2.\overline{r_2}$	$\overline{b_1}.\overline{p_1}$	$\overline{a_2}.\overline{b_1}.\overline{p_2}$

Figure 1 Stuck configuration, overlaid with backward slice with respect to final state of thread 1.

we run a concurrent computation, discover a bug, and then wish to compute a dynamic slice. It would clearly be impractical to require the slice be computed using the exact interleaving of the original run, particularly in a distributed setting. On the other hand, computing the slice using a brand-new concurrent execution may make different non-deterministic choices, producing a slice of a computation other than the one intended.

Intuitively, any execution which exhibits the same causal structure should be adequate for computing the slice, and any practical approach to concurrent slicing should take advantage of this. Danos and Krivine [4] make a similar observation about reversible concurrency, arguing that the most liberal notion of reversibility is one that just respects causality: an action can only be undone after all the actions that causally depend on it have been undone.

In this paper we formalise dynamic slicing for π-calculus, and show that any causally equivalent execution generates precisely the same slicing information. We do this by formalising slicing with respect to a particular execution $\tilde{t}$, and then proving that slicing with respect to any causally equivalent computation $\tilde{u}$ yields the same slice, after a unique "rewiring" which interprets the path witnessing $\tilde{t} \simeq \tilde{u}$ as a lattice isomorphism relating the two slices. The isomorphism is constructive, rewriting one slice into the other: this allows non-deterministic metadata (e.g. memory addresses or transaction ids) in the slicing execution to be aligned with the corresponding metadata in the original run. We build on an earlier "proof-relevant" formalisation of causal equivalence for π-calculus in Agda [15]. As long as causality is respected, an implementation of our system can safely use any technique (e.g. redex trails, proved transitions, or thread-local memories) to implement rewind and replay.

Example: scheduler with non-compliant task. While dynamic slicing cannot automatically isolate bugs, it can hide irrelevant detail and yield compact provenance-like explanations of troublesome parts of configurations. As an example we consider Milner's scheduler implementation [12, p. 65]. The scheduler controls a set of n tasks, executed by agents $A_1, \ldots, A_n$. Agent A_i sends the message a_i (*announce*) to the scheduler to start its task, and message b_i (*break*) to end its task. The scheduler ensures that the actions a_i occur cyclically starting with a_1, and that for each i the actions a_i and b_i alternate, starting with a_i. Although started sequentially, once started the tasks are free to execute in parallel.

Figure 1 shows five transitions of a two-thread scheduler, with the redex selected at each step highlighted in bold. The parts of the configuration which contribute to the final state of thread 1 are in black; the grey parts are discarded by our backward-slicing algorithm. Assume prefixing binds more tightly than either $\cdot \mid \cdot$ or $+$. To save space, we omit the ν-binders defining the various names, and write $x.\mathbf{0}$ simply as x. The names r_1, r_2, p_1 and p_2 are used to make recursive calls [12, p. 94]: a recursive procedure is implemented as a server which waits for an invocation request, spawns a new copy of the procedure body, and then

returns to the wait state. Here we omit the server definitions, and simply replace a successful invocation by the spawned body; thus in the final step of Figure 1, after the synchronisation on c_2 the invocation $\overline{r_1}$ is replaced by a fresh copy of the initial state of scheduler thread 1.

The final state of Figure 1 has no redexes, and so is stuck. The slice helps highlight the fact that by the time we come to start the second loop of scheduler 1, the task was terminated by message $\overline{b_1}$ from A_2, before any such message could be sent by A_1. We can understand the slice of the initial configuration (computed by "rewinding", or backward-slicing) as *sufficient* to explain the slice of the stuck configuration by noting that the former is able to *compute* the latter by "replay", or forward-slicing. In other words, writing a sliced part of the configuration as $\square$, and pretending the holes $\square$ are sub-computations which get stuck, we can derive

$$\mathbf{a_1}.c_1.(b_1.\square + \square) \mid \overline{c_1}.a_2.\square \mid \overline{\mathbf{a_1}}.\square \mid \overline{a_2}.\overline{b_1}.\overline{p_2} \quad \longrightarrow^* \quad \overline{a_2}.\square$$

without getting stuck. The slice on the left may of course choose to take the right-hand branch of the choice instead. But if we constrain the replay of the sliced program to follow the causal structure of the original unsliced run – to take the same branches of internal choices, and have the same synchronisation structure – then it will indeed evolve to the slice on the right. This illustrates the correctness property for backward slicing, which is that forward-slicing its result must recompute (at least) the slicing criterion.

For this example, the tasks are entirely atomic and so fixing the outcome of + has the effect of making the computation completely sequential. Less trivial systems usually have multiple ways they can evolve, even once the causal structure is fixed. A confluence lemma typically formalises the observational equivalence of two causally equivalent runs. However, a key observation made in [15] is that requiring causally equivalent runs to reach exactly the same state is too restrictive for π-calculus, in particular because of name extrusion. As we discuss in Section 3, two causally unrelated extrusion-rendezvous lead to states which differ in the relative position of two ν-binders, reflecting the two possible orderings of the rendezvous. Although technically unobservable to the program, interleaving-sensitive metadata, such as memory locations in a debugger or transaction ids in a financial application, may be important for domain-specific reasons. In these situations being able to robustly translate between the target states of the two executions may be useful.

Summary of contributions. Section 2 defines the core forward and backward dynamic slicing operations for π-calculus transitions and sequences of transitions (traces). We prove that they are related by a Galois connection, showing that backward and forward slicing, as defined, are minimal and maximal with respect to each other. Section 3 extends this framework to show that the Galois connections for causally equivalent traces compute the same slices up to lattice isomorphism. Section 4 discusses related work and Section 5 offers closing thoughts and prospects for follow-up work. Appendix A summarises the Agda module structure and required libraries; the source code can be found at `https://github.com/rolyp/concurrent-slicing`, release `0.1`.

2 Galois connections for slicing π-calculus programs

To summarise informally, our approach is to interpret, functorially, every transition diagram in the π-calculus into the category of lattices and Galois connections. For example the interpretation of the transition diagram on the left is the commutative diagram on the right:

Name	x, y	::=	$0 \mid 1 \mid \cdots$		Process	P, Q, R, S	::=	$\square$	erased
Payload	z	::=	$\square$	erased				$\mathbf{0}$	inactive
			x	retained				$\underline{x}.P$	input
Action	a	::=	$\square$	erased				$\overline{x}\langle z\rangle.P$	output
			$\underline{x}$	input				$P + Q$	choice
			$\overline{x}\langle z\rangle$	output				$P \mid Q$	parallel
			$\overline{x}$	bound output				νP	restriction
			τ	silent				$!P$	replication

Figure 2 Syntax of names, processes and actions.

where $\downarrow P$ means the lattice of slices of P, and $\overline{\mathsf{step}}_t : \downarrow P \longrightarrow \downarrow Q$ is a *Galois connection*, a kind of generalised order isomorphism. An order isomorphism between posets A and B is a pair of monotone functions $f : A \longrightarrow B$ and $g : B \longrightarrow A$ such that $f \circ g = \mathsf{id}_B$ and $g \circ f = \mathsf{id}_A$. Galois connections require only $f \circ g \geq \mathsf{id}_B$ and $g \circ f \leq \mathsf{id}_A$ where $\leq$ means the pointwise order. Galois connections are closed under composition.

The relationship to slicing is that these properties can be unpacked into statements of sufficiency and minimality: for example $f \circ g \geq \mathsf{id}_B$ means g (backward-slicing) is "sufficient" in that f (forward-slicing) is able to use the result of g to restore the slicing criterion, and $g \circ f \leq \mathsf{id}_A$ means g is "minimal" in that it computes the smallest slice with that property. One can dualise these statements to make similar observations about f.

We omit a treatment of structural congruence from our approach, but note that it slots easily into the framework, generating lattice isomorphisms in a manner similar to the "bound braid" relation $\rtimes$ discussed in Section 3, Definition 12.

2.1 Lattices of slices

The syntax of names, processes and actions is given in Figure 2. Slices are represented syntactically, via the $\square$ notation introduced informally in Section 1. Our formalisation employs de Bruijn indices [5], an approach with well-known strengths and weaknesses compared to other approaches to names such as higher-order abstract syntax or nominal calculi.

Names. Only names which occur in the "payload" (argument) position of a message may be erased. The erased name $\square$ gives rise to a (trivial) partial order $\leq$ over payloads, namely the partial order containing precisely $\square \leq z$ for any z. The set of *slices* of x is written $\downarrow x$ and defined to be $\{z \mid z \leq x\}$; because names are atomic $\downarrow x$ is simply the two-element set $\{\square, x\}$. The set $\downarrow x$ is a finite lattice with meet and join operations $\sqcap$ and $\sqcup$, and top and bottom elements x and $\square$ respectively. For any lattice, the meet and join are related to the underlying partial order by $z \leq z' \iff z \sqcup z' = z' \iff z \sqcap z' = z$. Lattices are closed under component-wise products, justifying the notation $\downarrow(z, z')$ for $\downarrow z \times \downarrow z'$.

$$\frac{}{\underline{x}.P \xrightarrow{\underline{x}} P} \qquad \frac{}{\overline{x}\langle z\rangle.P \xrightarrow{\overline{x}\langle z\rangle} P} \qquad \frac{P \xrightarrow{a} R}{P + Q \xrightarrow{a} R} \qquad \frac{P \xrightarrow{c} R}{P \mid Q \xrightarrow{c} R \mid Q}$$

$$(*)\ \frac{P \xrightarrow{b} R}{P \mid Q \xrightarrow{b} R \mid \mathsf{push}^* Q} \qquad (\S)\ \frac{P \xrightarrow{\underline{x}} R \qquad Q \xrightarrow{\overline{x}\langle z\rangle} S}{P \mid Q \xrightarrow{\tau} (\mathsf{pop}\ z)^* R \mid S} \qquad \frac{P \xrightarrow{\overline{(x+1)}\langle 0\rangle} R}{\nu P \xrightarrow{\overline{x}} R}$$

$$\frac{P \xrightarrow{\underline{x}} R \qquad Q \xrightarrow{\overline{x}} S}{P \mid Q \xrightarrow{\tau} \nu(R \mid S)} \qquad (\dagger)\ \frac{P \xrightarrow{\mathsf{push}^* c} R}{\nu P \xrightarrow{c} \nu R} \qquad (\ddagger)\ \frac{P \xrightarrow{\mathsf{push}^* b} R}{\nu P \xrightarrow{b} \nu(\mathsf{swap}^* R)}$$

$$\frac{P \mid !P \xrightarrow{a} R}{!P \xrightarrow{a} R}$$

Figure 3 Labelled transition relation $P \xrightarrow{a} R$ (symmetric variants omitted).

Processes. The $\leq$ relation and $\downarrow\cdot$ operation extend to processes, via payloads which may be $\square$, and a special undefined process also written $\square$. A slice of P is simply P with some sub-terms replaced by $\square$. The relation $\leq$ is the least compatible partial order which has $\square$ as least element; all process constructors both preserve and reflect $\leq$, so we assume an equivalent inductive definition of $\leq$ when convenient. A process has a closing context Γ enumerating its free variables; in the untyped de Bruijn setting Γ is just a natural number. Often it is convenient to conflate Γ with a set of that cardinality.

Actions. An action a labels a transition (Figure 3 below), and is either *bound* or *non-bound*. A bound action b is of the form $\underline{x}$ or $\overline{x}$ and opens a process with respect to x, taking it from Γ to $\Gamma + 1$. A non-bound action c is of the form $\overline{x}\langle z\rangle$ or τ and preserves the free variables of the process. The $\leq$ relation and $\downarrow\cdot$ operation extend to actions via $\square$ names, plus a special undefined action also written $\square$.

Renamings. In the lattice setting, a renaming $\rho : \Gamma \longrightarrow \Gamma'$ is any function from Γ to $\Gamma' \uplus \{\square\}$; we also allow σ to range over renamings. Renaming application $\rho^* P$ is extended with the equation $\rho^* \square = \square$. The $\leq$ relation and $\downarrow\cdot$ operation apply pointwise.

Labelled transition semantics. The late-style labelled transition semantics is given in Figure 3, and is distinguished only by its adaptation to the de Bruijn setting. The primary reference for a de Bruijn formulation of π-calculus is [9]; the consequences of such an approach are explored in some depth in [15]. One pleasing consequence of a de Bruijn approach is that the usual side-conditions associated with transition rules can be operationalised via renamings. We briefly explain this, along with other uses of renamings in the transition rules, and refer the interested reader to these earlier works for more details. Definition 1 defines the renamings used in Figure 3 and Definition 2 the application $\rho^* a$ of ρ to an action a.

▶ **Definition 1** (push, pop, and swap).

$$\begin{array}{lcl} \mathsf{push}_\Gamma & : & \boldsymbol{\Gamma} \longrightarrow \boldsymbol{\Gamma + 1} \\ \mathsf{push}\ x & = & x + 1 \end{array} \qquad \begin{array}{lcl} \mathsf{pop}_\Gamma\ \mathbf{z} & : & \boldsymbol{\Gamma + 1} \longrightarrow \boldsymbol{\Gamma} \\ \mathsf{pop}\ z\ 0 & = & z \\ \mathsf{pop}\ z\ (x+1) & = & x \end{array} \qquad \begin{array}{lcl} \mathsf{swap}_\Gamma & : & \boldsymbol{\Gamma + 2} \longrightarrow \boldsymbol{\Gamma + 2} \\ \mathsf{swap}\ 0 & = & 1 \\ \mathsf{swap}\ 1 & = & 0 \\ \mathsf{swap}\ (x+2) & = & x + 2 \end{array}$$

▶ **Definition 2** (Action renaming)**.** Define the following lifting of a renaming to actions.

$$\begin{array}{llcl} \cdot^* & & : & (\Gamma \longrightarrow \Gamma') \longrightarrow \mathsf{Action}\ \Gamma \longrightarrow \mathsf{Action}\ \Gamma' \\ \rho^* & \square & = & \square \\ \rho^* & \underline{x} & = & \underline{\rho x} \\ \rho^* & \overline{x} & = & \overline{\rho x} \\ \rho^* & \tau & = & \tau \\ \rho^* & \overline{x}\langle z\rangle & = & \overline{\rho x}\langle \rho z\rangle \end{array}$$

- push occurs in the transition rule which propagates a bound action through a parallel composition $P \mid Q$ (rule $(*)$ in Figure 3), and rewires Q so that the name 0 is reserved. The effect is to ensure that the binder being propagated by P is not free in Q.
- push also occurs in the rules which propagate an action through a ν-binder (rules (†) and (‡)), where it is applied to the action being propagated using the function defined in Definition 2. This ensures the action does not mention the binder it is propagating through. The use of $\cdot + 1$ in the name extrusion rule can be interpreted similarly.
- pop z is used in the event of a successful synchronisation (rule (§)), and undoes the effect of push, substituting the communicated name z for index 0.
- swap occurs in the rule which propagates a bound action through a ν-binder (rule (†)) and has no counterpart outside of the de Bruijn setting. As a propagating binder passes through another binder, their relative position in the syntax is exchanged, and so to preserve naming R is rewired with a "braid" that swaps 0 and 1.

Although its use in the operational semantics is unique to the de Bruijn setting, swap will also play an important role when we consider the relationship between slices of causally equivalent traces (Section 3 below), where it captures how the relative position of binders changes between different (but causally equivalent) interleavings.

2.2 Galois connections for slicing

We now compositionally assemble a Galois connection for each component of execution, starting with renamings, and then proceeding to individual transitions and entire traces, which relates forward and backward slices of the initial and terminal state.

Slicing renamings. The application ρx of a renaming to a name, and the lifting $\rho^* P$ of that operation to a process give rise to the Galois connections defined here.

▶ **Definition 3** (Galois connection for ρx)**.** Suppose $\rho : \Gamma \longrightarrow \Gamma'$ and $x \in \Gamma$. Define the following pair of monotone functions between $\downarrow(\rho, x)$ and $\downarrow(\rho x)$.

$$\begin{array}{llcl} \mathsf{app}_{\rho,x} & & : & \downarrow(\rho, x) \longrightarrow \downarrow(\rho x) \\ \mathsf{app}_{\rho,x} & (\sigma, \square) & = & \square \\ \mathsf{app}_{\rho,x} & (\sigma, x) & = & \sigma x \end{array} \qquad \begin{array}{llcl} \mathsf{unapp}_{\rho,x} & & : & \downarrow(\rho x) \longrightarrow \downarrow(\rho, x) \\ \mathsf{unapp}_{\rho,x} & z & = & (x \mapsto_\rho z, \rho_x^{-1} z) \end{array}$$

where

$$\begin{array}{llcll} x \mapsto_\rho \cdot & & : & \downarrow(\rho x) \longrightarrow \downarrow\rho & \\ (x \mapsto_\rho z) & x & = & z & \\ (x \mapsto_\rho z) & y & = & \square & (\text{if } y \neq x) \end{array} \qquad \begin{array}{llcll} \rho_x^{-1} & & : & \downarrow(\rho x) \longrightarrow \downarrow x & \\ \rho_x^{-1} & \square & = & \square & \\ \rho_x^{-1} & z & = & x & (\text{if } z \neq \square) \end{array}$$

It is convenient to decompose $\mathsf{unapp}_{\rho,x}$ into two components: $x \mapsto_\rho z$ denotes the least slice of ρ which maps x to z, and $\rho_x^{-1} z$ denotes the least slice of x such that $\rho x = z$.

▶ **Lemma 4.** *($app_{\rho,x}, unapp_{\rho,x}$) is a Galois connection.*

1. $app_{\rho,x} \circ unapp_{\rho,x} \geq id_{\rho x}$
2. $unapp_{\rho,x} \circ app_{\rho,x} \leq id_{\rho,x}$

▶ **Definition 5** (Galois connection for a renaming $\rho^* P$)**.**
Suppose $\rho : \Gamma \longrightarrow \Gamma'$ and $\Gamma \vdash P$. Define monotone functions between $\downarrow(\rho, P)$ and $\downarrow(\rho^* P)$ by structural recursion on $\downarrow P$, using the following equations. Here $\square_\rho$ denotes the least slice of ρ, namely the renaming which maps every $x \in \Gamma$ to $\square$.

$$
\begin{array}{lllll}
\mathsf{ren}_{\rho,P} & & : & \downarrow(\rho, P) \longrightarrow \downarrow(\rho^* P) \\
\mathsf{ren}_{\rho,P} & (\sigma, \square) & = & \square \\
\mathsf{ren}_{\rho,\mathbf{0}} & (\sigma, \mathbf{0}) & = & \mathbf{0} \\
\mathsf{ren}_{\rho,\underline{x}.P} & (\sigma, \underline{x}.R) & = & \underline{x}.\mathsf{ren}_{\rho+1,P}\ (\sigma, R) \\
\mathsf{ren}_{\rho,\overline{x}\langle z\rangle.P} & (\sigma, \overline{x}\langle z'\rangle.R) & = & \overline{x}\langle z''\rangle.\mathsf{ren}_{\rho,P}\ (\sigma, R)\ \textit{where}\ z'' = \mathsf{app}_{\rho,y}(\sigma, z') \\
\mathsf{ren}_{\rho,P+Q} & (\sigma, R + S) & = & \mathsf{ren}_{\rho,P}\ (\sigma, R) + \mathsf{ren}_{\rho,Q}\ (\sigma, S) \\
\mathsf{ren}_{\rho,P|Q} & (\sigma, R \mid S) & = & \mathsf{ren}_{\rho,P}\ (\sigma, R) \mid \mathsf{ren}_{\rho,Q}\ (\sigma, S) \\
\mathsf{ren}_{\rho,\nu P} & (\sigma, \nu R) & = & \nu(\mathsf{ren}_{\rho+1,P}\ (\sigma + 1, R) \\
\mathsf{ren}_{\rho,!P} & (\sigma, !R) & = & !(\mathsf{ren}_{\rho,P}\ (\sigma, R)) \\
 & & & \\
\mathsf{unren}_{\rho,P} & & : & \downarrow(\rho^* P) \longrightarrow \downarrow(\rho, P) \\
\mathsf{unren}_{\rho,P} & \square & = & (\square_\rho, \square) \\
\mathsf{unren}_{\rho,\mathbf{0}} & \mathbf{0} & = & (\square_\rho, \mathbf{0}) \\
\mathsf{unren}_{\rho,\underline{x}.P} & \underline{x}.R & = & (\rho', \underline{x}.P')\ \textit{where}\ \mathsf{unren}_{\rho+1,P}\ R = (\rho' + 1, P') \\
\mathsf{unren}_{\rho,\overline{x}\langle z\rangle.P} & \overline{x}\langle z'\rangle.R & = & (\rho' \sqcup (z \mapsto_\rho z'), \overline{x}\langle z''\rangle.P')\ \textit{where}\ \mathsf{unren}_{\rho,P}\ R = (\rho', P')\ \textit{and}\ z'' = \rho_z^{-1} z' \\
\mathsf{unren}_{\rho,P+Q} & (R + S) & = & (\rho_1 \sqcup \rho_2, P' + Q')\ \textit{where}\ \mathsf{unren}_{\rho,P}\ R = (\rho_1, P')\ \text{and}\ \mathsf{unren}_{\rho,Q}\ S = (\rho_2, Q') \\
\mathsf{unren}_{\rho,P|Q} & (R \mid S) & = & (\rho_1 \sqcup \rho_2, P' \mid Q')\ \textit{where}\ \mathsf{unren}_{\rho,P}\ R = (\rho_1, P')\ \text{and}\ \mathsf{unren}_{\rho,Q}\ S = (\rho_2, Q') \\
\mathsf{unren}_{\rho,\nu P} & \nu R & = & (\rho', \nu P')\ \textit{where}\ \mathsf{unren}_{\rho+1,P}\ R = (\rho' + 1, P') \\
\mathsf{unren}_{\rho,!P} & !R & = & (\rho', !P')\ \textit{where}\ \mathsf{unren}_{\rho,P}\ R = (\rho', P')
\end{array}
$$

▶ **Lemma 6.** *($\mathsf{ren}_{\rho,P}, \mathsf{unren}_{\rho,P}$) is a Galois connection.*

1. $\mathsf{ren}_{\rho,P} \circ \mathsf{unren}_{\rho,P} \geq \mathsf{id}_{\rho^* P}$
2. $\mathsf{unren}_{\rho,P} \circ \mathsf{ren}_{\rho,P} \leq \mathsf{id}_{\rho,P}$

Proof. In each case by induction on P, using Lemma 4 and the invertibility of $\cdot + 1$. ◀

Slicing transitions. Transitions also lift to the lattice setting, in the form of Galois connections defined by structural recursion over the proof that $t : P \xrightarrow{a} P'$. Figures 4 and 5 define the forward and backward slicing judgements. We assume a determinising convention where a rule applies only if no earlier rule applies.

The judgement $R_P \xrightarrow{a'_a} R'_{P'}$ asserts that there is a "replay" transition from $R \leq P$ to $(a', R') \leq (a, P)$, with R the input and (a', R') the output. The judgement $R'_P \xleftarrow{a'_a} R_{P'}$ asserts that there is a "rewind" transition from $(a', R) \leq (a, P')$ to $R' \leq P$, with (a', R) the input and R' the output. When writing R_P where $R \leq P$ we exploit the preservation and reflection of $\leq$ by all constructors, for example writing $\nu(R_P \mid S_Q)$ for $\nu(R \mid S)_{\nu(P|Q)}$.

For backward slicing, we permit the renaming application operator * to be used in a pattern-matching form, indicating a use of the lower adjoint unren: given a renaming application $\rho^* P$, the pattern $\sigma^* P'$ matches any slice R of $\rho^* P$ such that $\mathsf{unren}_{\rho,P}(R) = (\sigma, P')$.

▶ **Definition 7** (Galois connection for a transition)**.** Suppose $t : P \xrightarrow{a} P'$. Define the following pair of monotone functions between $\downarrow P$ to $\downarrow(a, P')$.

$$
\begin{array}{llllll}
\mathsf{step}_t & & : & \downarrow P \longrightarrow \downarrow(a, P') & \mathsf{unstep}_t & : & \downarrow(a, P') \longrightarrow \downarrow P \\
\mathsf{step}_t & R & = & (a', R')\ \textit{where}\ R_P \xrightarrow{a'_a} R'_{P'} & \mathsf{unstep}_t & (R, a') & = & R'\ \textit{where}\ R'_P \xleftarrow{a'_a} R_{P'}
\end{array}
$$

We omit the proofs that these equations indeed define total, deterministic, monotone relations.

▶ **Theorem 8** ($(\mathsf{step}_t, \mathsf{unstep}_t)$ is a Galois connection)**.**

1. $\mathsf{step}_t \circ \mathsf{unstep}_t \geq \mathsf{id}_{a,P'}$
2. $\mathsf{unstep}_t \circ \mathsf{step}_t \leq \mathsf{id}_P$

Proof. By induction on $t : P \xrightarrow{a'} P'$, using Lemma 6 for the cases involving renaming. ◀

$$\frac{}{\Box P \xrightarrow{\Box_a} \Box P'} \qquad \frac{}{\underline{x}.R_P \xrightarrow{\underline{x}} R_P} \qquad \frac{}{\overline{x}\langle z'_z\rangle.R_P \xrightarrow{\overline{x}\langle z'_z\rangle} R_P} \qquad \frac{R_P \xrightarrow{a'_a} R'_{P'}}{R_P + S_Q \xrightarrow{a'_a} R'_{P'}}$$

$$\frac{R_P \xrightarrow{c'_c} R'_{P'}}{R_P \mid S_Q \xrightarrow{c'_c} R'_{P'} \mid S_Q} \qquad \frac{R_P \xrightarrow{b'_b} R'_{P'}}{R_P \mid S_Q \xrightarrow{b'_b} R'_{P'} \mid \mathsf{push}^* S_Q}$$

$$\frac{R_P \xrightarrow{\underline{x}} R'_{P'} \qquad S_Q \xrightarrow{\overline{x}\langle z'_z\rangle} S'_{Q'}}{R_P \mid S_Q \xrightarrow{\tau} (\mathsf{pop}\ z'_z)^* R'_{P'} \mid S'_{Q'}} \qquad \frac{R_P \xrightarrow{\Box_{\underline{x}}} R'_{P'} \qquad S_Q \xrightarrow{\overline{x}\langle z'_z\rangle} S'_{Q'}}{R_P \mid S_Q \xrightarrow{\Box_\tau} (\mathsf{pop}\ z'_z)^* R'_{P'} \mid S'_{Q'}}$$

$$\frac{R_P \xrightarrow{a_{\underline{x}}} R'_{P'} \qquad S_Q \xrightarrow{\Box_{\overline{x}}\langle z\rangle} S'_{Q'}}{R_P \mid S_Q \xrightarrow{\Box_\tau} (\mathsf{pop}\ \Box_z)^* R'_{P'} \mid S'_{Q'}} \qquad \frac{R_P \xrightarrow{\overline{(x+1)}\langle 0\rangle} R'_{P'}}{\nu R_P \xrightarrow{\underline{x}} R'_{P'}} \qquad \frac{R_P \xrightarrow{a\overline{(x+1)}\langle 0\rangle} R'_{P'}}{\nu R_P \xrightarrow{\Box_{\overline{x}}} R'_{P'}}$$

$$\frac{R_P \xrightarrow{\underline{x}} R'_{P'} \qquad S_Q \xrightarrow{\overline{x}} S'_{Q'}}{R_P \mid S_Q \xrightarrow{\tau} \nu(R'_{P'} \mid S'_{Q'})} \qquad \frac{R_P \xrightarrow{a_{\underline{x}}} R'_{P'} \qquad S_Q \xrightarrow{a'_{\overline{x}}} S'_{Q'}}{R_P \mid S_Q \xrightarrow{\Box_\tau} \nu(R'_{P'} \mid S'_{Q'})} \qquad \frac{R_P \xrightarrow{\mathsf{push}^* c'_c} R'_{P'}}{\nu R_P \xrightarrow{c'_c} \nu R'_{P'}}$$

$$\frac{R_P \xrightarrow{\mathsf{push}^* b'_b} R'_{P'}}{\nu R_P \xrightarrow{b'_b} \nu(\mathsf{swap}^* R'_{P'})} \qquad \frac{R_P \mid !R_P \xrightarrow{a'_a} R'_{P'}}{!R_P \xrightarrow{a'_a} R'_{P'}}$$

Figure 4 Forward slicing judgement $R_P \xrightarrow{a'_a} R'_{P'}$.

Slicing traces. Finally we extend slicing to entire runs of a π-calculus program. A sequence of transitions $\tilde{t}$ is called a *trace*; the empty trace at P is written ε_P, and the composition of a transition $t : P \xrightarrow{a} R$ and trace $\tilde{t} : R \xrightarrow{\tilde{a}} S$ is written $t \cdot \tilde{t} : P \xrightarrow{a \cdot \tilde{a}} S$ where actions are composable whenever their source and target contexts match.

▶ **Definition 9** (Galois connection for a trace)**.** Suppose $\tilde{t} : P \xrightarrow{\tilde{a}} P'$. Define the following pair of monotone functions between $\downarrow P$ and $\downarrow P'$, using variants of step_t and unstep_t which discard the action slice (going forward) and which use $\Box$ as the action slice (going backward).

$$\begin{array}{lcll} \mathsf{fwd}_{\tilde{t}} & : & \downarrow P \longrightarrow \downarrow P' & \\ \mathsf{fwd}_{\varepsilon_P} & = & \mathsf{id}_{\downarrow P} & \\ \mathsf{fwd}_{t\cdot\tilde{t}}\ \Box & = & \Box & \\ \mathsf{fwd}_{t\cdot\tilde{t}}\ R & = & \mathsf{fwd}_{\tilde{t}}\ (\mathsf{step}'_t\ R) & (R \neq \Box) \end{array} \qquad \begin{array}{lcll} \mathsf{bwd}_{\tilde{t}} & : & \downarrow P' \longrightarrow \downarrow P & \\ \mathsf{bwd}_{\varepsilon_{P'}} & = & \mathsf{id}_{\downarrow P'} & \\ \mathsf{bwd}_{t\cdot\tilde{t}}\ \Box & = & \Box & \\ \mathsf{bwd}_{t\cdot\tilde{t}}\ R & = & \mathsf{unstep}'_t\ (\mathsf{bwd}_{\tilde{t}}\ R) & (R \neq \Box) \end{array}$$

$$\begin{array}{lcl} \mathsf{step}'_t & : & \downarrow \mathbf{P} \longrightarrow \downarrow \mathbf{P}' \\ \mathsf{step}'_t\ R & = & R'\ \textit{where}\ \mathsf{step}_t\ R = (a', R') \end{array} \qquad \begin{array}{lcl} \mathsf{unstep}'_t & : & \downarrow P' \longrightarrow \downarrow P \\ \mathsf{unstep}'_t\ R' & = & \mathsf{unstep}_t\ (\Box, R') \end{array}$$

At the empty trace ε_P the Galois connection is simply the identity on $\downarrow P$. Otherwise, we recurse into the structure of the trace $t \cdot \tilde{t}$, composing the Galois connection for the single transition t with the Galois connection for the tail of the computation $\tilde{t}$.

▶ **Theorem 10** ($(\mathsf{fwd}_{\tilde{t}}, \mathsf{bwd}_{\tilde{t}})$ is a Galois connection)**.**

1. $\mathit{fwd}_{\tilde{t}} \circ \mathit{bwd}_{\tilde{t}} \geq \mathit{id}_{P'}$
2. $\mathit{bwd}_{\tilde{t}} \circ \mathit{fwd}_{\tilde{t}} \leq \mathit{id}_P$

Note that the trace used to define forward and backward slicing for a computation is not an auxiliary data structure recording the computation, such as a redex trail or memory, but simply the proof term witnessing $P \xrightarrow{\tilde{a}} P'$.

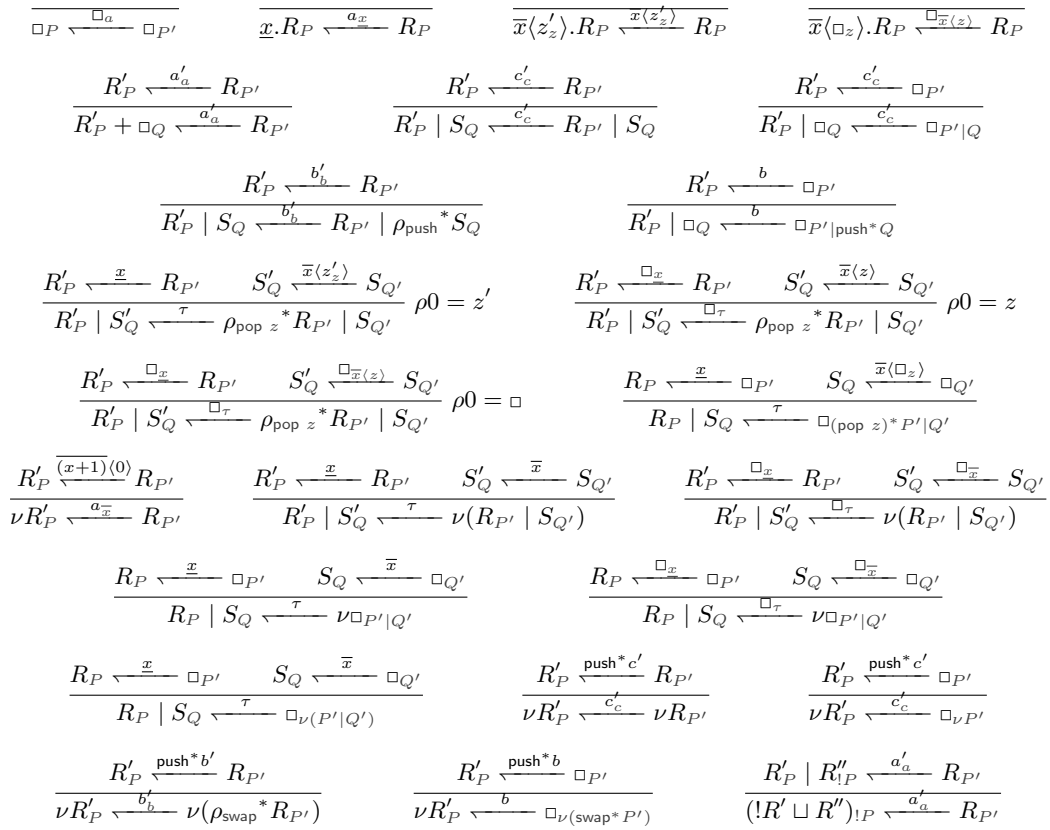


Figure 5 Backward slicing judgement $R'_P \xleftarrow{a'_a} R_{P'}$.

3 Slicing and causal equivalence

In this section, we show that when dynamic slicing a π-calculus program, slicing with respect to any causally equivalent execution yields essentially the same slice. "Essentially the same" here means modulo lattice isomorphism. In other words slicing discards precisely the same information regardless of which interleaving is chosen to do the slicing.

Proof-relevant causal equivalence. Causally equivalent computations are generated by transitions which share a start state, but which are independent. Following Lévy [11], we call such transitions *concurrent*, written $t \frown t'$. We illustrate this idea, and the non-trivial relationship that it induces between terminal states, by way of example. For the full definition of concurrency for π-calculus, we refer the interested reader to [15] or to the Agda definition[1]. For the sake of familiarity the example uses regular names instead of de Bruijn indices.

Example. Consider the process $P_0 \stackrel{\text{def}}{=} (\nu yz)\ (\overline{x}\langle y\rangle.P) \mid \overline{x}\langle z\rangle.Q$ for some unspecified processes P and Q. This process can take *two* transitions, which we will call t and t'. Transition

[1] https://github.com/rolyp/proof-relevant-pi/blob/master/Transition/Concur.agda

$t : P_0 \xrightarrow{\overline{x}(y)} P_1$ extrudes y on the channel x:

$$P_0 \xrightarrow{\overline{x}(y)} (\nu z)\ P \mid \overline{x}\langle z\rangle.Q \stackrel{\text{def}}{=} P_1$$

whereas transition $t' : P_0 \xrightarrow{\overline{x}(z)} P_1'$ extrudes z, also on the channel x:

$$P_0 \xrightarrow{\overline{x}(z)} (\nu y)\ (\overline{x}\langle y\rangle.P) \mid Q \stackrel{\text{def}}{=} P_1'$$

In both cases the output actions are bound, representing the extruding binder. Moreover, t and t' are *concurrent*, written $t \smile t'$, meaning they can be executed in either order. Having taken t, one can *mutatis mutandis* take t', and vice versa. Concurrency is an irreflexive and symmetric relation defined over transitions which are *coinitial* (have the same source state).

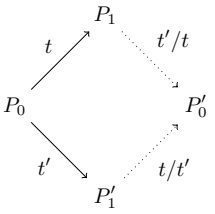
The qualification is needed because t' will need to be adjusted to operate on the target state of t, if t is the transition which happens first. If t' happens first then t will need to be adjusted to operate on the target state of t'. The adjusted version of t' is called the *residual* of t' after t, and is written t'/t. In this case t'/t can still extrude z:

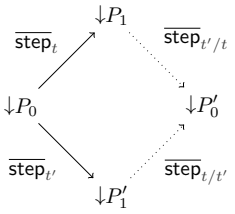
$$P_1 = (\nu z)\ P \mid \overline{x}\langle z\rangle.Q \xrightarrow{\overline{x}(z)} P \mid Q \stackrel{\text{def}}{=} P_0'$$

whereas the residual t/t' can still extrude y:

$$P_1' = (\nu y)\ (\overline{x}\langle y\rangle.P) \mid Q \xrightarrow{\overline{x}(y)} P \mid Q = P_0'$$

The independence of t and t' is confirmed by the fact that $t \cdot t'/t$ and $t' \cdot t/t'$ are *cofinal* (share a target state), as shown on the left below.





We say that the traces $\tilde{t} \stackrel{\text{def}}{=} t \cdot t'/t$ and $\tilde{u} \stackrel{\text{def}}{=} t' \cdot t/t'$ are *causally equivalent*, written $\tilde{t} \simeq \tilde{u}$. The commutativity of the right-hand square (Theorem 16 below) means the two interleavings are also equivalent for slicing purposes. Here $\overline{\mathsf{step}}_t$ denotes the Galois connection $(\mathsf{step}_t, \mathsf{unstep}_t)$.

However [15], which formalised causal equivalence for π-calculus, showed that causally equivalence traces do not always reach exactly the same state, but only the same state up to some permutation of the binders in the resulting processes. This will become clear if we consider another process $Q_0 \stackrel{\text{def}}{=} (x(y').R) \mid x(z').S$ able to synchronise with both of the extrusions raised by P_0 and consider the two different ways that $P_0 \mid Q_0$ can evolve.

First note that Q_0 can also take two independent transitions: $u : Q_0 \xrightarrow{x(y')} R \mid x(z').S \stackrel{\text{def}}{=} Q_1$ inputs on x and binds the received name to y'; and $u' : Q_0 \xrightarrow{x(z')} (x(y').R) \mid S \stackrel{\text{def}}{=} Q_1'$ also inputs on x and binds the received name to z'. (Assume z is not free in the left-hand side of Q_0 and that y is not free in the right-hand side.) The respective residuals $Q_1 = R \mid x(z').S \xrightarrow{x(z')} R \mid S \stackrel{\text{def}}{=} Q_0'$ and $Q_1' = (x(y').R) \mid S \xrightarrow{x(y')} R \mid S = Q_0'$ again converge on the same state Q_0', leading to a diamond for Q_0 similar to the one for P_0 above.

The subtlety arises when we put P_0 and Q_0 into parallel composition, since now we have two concurrent synchronisation possibilities. For clarity we give the derivations, which we call s and s':

$$\frac{t : P_0 \xrightarrow{\overline{x}(y)} P_1 \qquad u : Q_0 \xrightarrow{x(y')} Q_1}{s : P_0 \mid Q_0 \xrightarrow{\tau} (\nu y)\ P_1 \mid Q_1\{y/y'\}} \qquad \frac{t' : P_0 \xrightarrow{\overline{x}(z)} P_1' \qquad u' : Q_0 \xrightarrow{x(z')} Q_1'}{s' : P_0 \mid Q_0 \xrightarrow{\tau} (\nu z)\ P_1' \mid Q_1'\{z/z'\}}$$

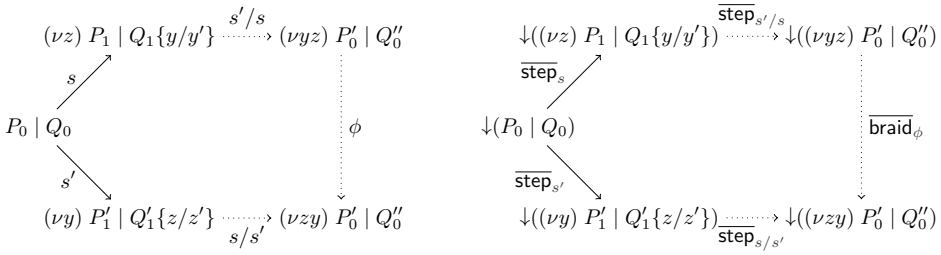
The labelled transition system is closed under renamings; thus the residual u'/u has an image in the renaming $\cdot\{y/y'\}$, and u/u' has an image in the renaming $\cdot\{z/z'\}$, allowing us to derive composite residual s'/s:

$$\frac{t'/t : P_1 \xrightarrow{\overline{x}(z)} P_0' \qquad \dfrac{u'/u : Q_1 \xrightarrow{x(z')} Q_0'}{(u'/u)\{y/y'\} : Q_1\{y/y'\} \xrightarrow{x(z')} Q_0'\{y/y'\}}}{s'/s : (\nu y)\ P_1 \mid Q_1\{y/y'\} \xrightarrow{\tau} (\nu yz)\ P_0' \mid Q_0'\{y/y'\}\{z/z'\}}$$

By similar reasoning we can derive s/s':

$$s/s' : (\nu z)\ P_1' \mid Q_1'\{y/y'\} \xrightarrow{\tau} (\nu zy)\ P_0' \mid Q_0'\{z/z'\}\{y/y'\}$$

By side-conditions on the transition rules the renamings $\cdot\{y/y'\}$ and $\cdot\{z/z'\}$ commute and so $Q_0'\{y/y'\}\{z/z'\} \stackrel{\text{def}}{=} Q_0'' = Q_0'\{z/z'\}\{y/y'\}$. However, the positions of binders y and z are transposed in the terminal states of s'/s and s/s'. Instead of the usual diamond shape, we have the pentagon on the left below, where ϕ is a *braid* representing the transposition of the binders. Lifted to slices, ϕ becomes the unique isomorphism $\overline{\mathsf{braid}}_\phi$ relating slices of the terminal states, as shown in the commutative diagram on the right:



In the de Bruijn setting, a braid like ϕ does not relate two processes of the form $(\nu yz)\ R$ and $(\nu yz)\ R$ but rather two processes of the form $\nu\nu R$ and $\nu\nu(\mathsf{swap}^* R)$: the transposition of the (nameless) binders is represented by the transposition of the roles of indices 0 and 1 in the body of the innermost binder.

▶ **Definition 11** (Bound braid $P \rtimes R$)**.** Inductively define the symmetric relation $P \rtimes R$ using the rules below.

$$\nu\nu\text{-}\mathsf{swap}_P\ \frac{}{\nu\nu P \rtimes \nu\nu P'}\ P = \mathsf{swap}^* P' \qquad \cdot + Q\ \frac{P \rtimes R}{P + Q \rtimes R + Q} \qquad P + \cdot\ \frac{Q \rtimes S}{P + Q \rtimes P + S}$$

$$\cdot \mid Q\ \frac{P \rtimes R}{P \mid Q \rtimes R \mid Q} \qquad P \mid \cdot\ \frac{Q \rtimes S}{P \mid Q \rtimes P \mid S} \qquad \nu\cdot\ \frac{P \rtimes R}{\nu P \rtimes \nu R} \qquad !\cdot\ \frac{P \rtimes R}{!P \rtimes !R}$$

Following [15], we adopt a compact term-like notation for $\rtimes$ proofs, using the rule names which occur to the left of each rule in Definition 11. For the extrusion example above, ϕ (in de Bruijn indices notation) would be a leaf case of the form $\nu\nu\text{-}\mathsf{swap}_{\cdot|\cdot}$.

▶ **Definition 12** (Lattice isomorphism for bound braid)**.** Suppose $\phi : Q \rtimes Q'$. Define the following pair of monotone functions between $\downarrow Q$ and $\downarrow Q'$ by structural recursion on ϕ.

braid_ϕ		:	$\downarrow Q \longrightarrow \downarrow Q'$	$\mathsf{unbraid}_\phi$		:	$\downarrow Q' \longrightarrow \downarrow Q$
$\mathsf{braid}_{\nu\nu\text{-}\mathsf{swap}_P}$	$(\nu\nu R)$	=	$\nu\nu(\mathsf{ren}_{\mathsf{swap},P}(R))$	$\mathsf{unbraid}_{\nu\nu\text{-}\mathsf{swap}_P}$	$(\nu\nu R)$	=	$\nu\nu(\mathsf{ren}_{\mathsf{swap},P}(R))$
$\mathsf{braid}_{\phi+S}$	$(R+S)$	=	$\mathsf{braid}_\phi\ R + S$	$\mathsf{unbraid}_{\phi+S}$	$(R+S)$	=	$\mathsf{unbraid}_\phi\ R + S$
$\mathsf{braid}_{R+\psi}$	$(R+S)$	=	$R + \mathsf{braid}_\psi\ S$	$\mathsf{unbraid}_{R+\psi}$	$(R+S)$	=	$R + \mathsf{unbraid}_\psi\ S$
$\mathsf{braid}_{\phi\mid S}$	$(R \mid S)$	=	$\mathsf{braid}_\phi\ R \mid S$	$\mathsf{unbraid}_{\phi\mid S}$	$(R \mid S)$	=	$\mathsf{unbraid}_\phi\ R \mid S$
$\mathsf{braid}_{R\mid\psi}$	$(R \mid S)$	=	$R \mid \mathsf{braid}_\psi\ S$	$\mathsf{unbraid}_{R\mid\psi}$	$(R \mid S)$	=	$R \mid \mathsf{unbraid}_\psi\ S$
$\mathsf{braid}_{\nu\phi}$	(νR)	=	$\nu(\mathsf{braid}_\phi\ R)$	$\mathsf{unbraid}_{\nu\phi}$	(νR)	=	$\nu(\mathsf{unbraid}_\phi\ R)$
$\mathsf{braid}_{!\phi}$	$(!R)$	=	$!(\mathsf{braid}_\phi\ R)$	$\mathsf{unbraid}_{!\phi}$	$(!R)$	=	$!(\mathsf{unbraid}_\phi\ R)$

▶ **Lemma 13.**

1. $\mathit{braid}_\phi \circ \mathit{unbraid}_\phi = \mathit{id}_{\downarrow Q'}$
2. $\mathit{unbraid}_\phi \circ \mathit{braid}_\phi = \mathit{id}_{\downarrow Q}$

Proof. Induction on ϕ. In the base case use the idempotence of swap lifted to lattices. ◀

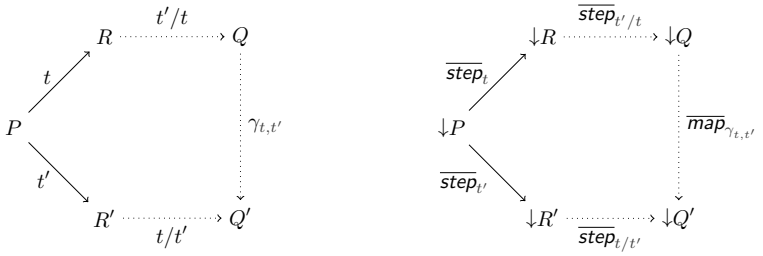
▶ **Definition 14** (Lattice isomorphism for cofinality map)**.** Suppose $t \smile t'$ with $\mathsf{tgt}(t'/t) = Q$ and $\mathsf{tgt}(t/t') = Q'$. By Theorem 1 of [15], there exists a unique $\gamma_{t,t'}$ witnessing $Q = Q'$, $Q \asymp Q'$ or $Q \rtimes Q'$. Define the following pair of monotone functions between $\downarrow Q$ and $\downarrow Q'$.

$$
\begin{array}{lcl@{\qquad}lcl}
\mathsf{map}_{\gamma_{t,t'}} & : & \downarrow Q \longrightarrow \downarrow Q' & \mathsf{unmap}_{\gamma_{t,t'}} & : & \downarrow Q' \longrightarrow \downarrow Q \\
\mathsf{map}_{Q=Q'} & = & \mathsf{id}_{\downarrow Q} & \mathsf{unmap}_{Q=Q'} & = & \mathsf{id}_{\downarrow Q} \\
\mathsf{map}_{Q \asymp Q'} & = & \mathsf{ren}_{\mathsf{swap},Q} & \mathsf{unmap}_{Q \asymp Q'} & = & \mathsf{unren}_{\mathsf{swap},Q} \\
\mathsf{map}_{\phi:Q \rtimes Q'} & = & \mathsf{braid}_\phi & \mathsf{unmap}_{\phi:Q \rtimes Q'} & = & \mathsf{unbraid}_\phi
\end{array}
$$

▶ **Lemma 15.**

1. $\mathit{map}_{\gamma_{t,t'}} \circ \mathit{unmap}_{\gamma_{t,t'}} = \mathit{id}_{\downarrow Q'}$
2. $\mathit{unmap}_{\gamma_{t,t'}} \circ \mathit{map}_{\gamma_{t,t'}} = \mathit{id}_{\downarrow Q}$

▶ **Theorem 16.** *Suppose $t \smile t'$ as on the left. Then the pentagon on the right commutes.*



Lattice isomorphism for arbitrary causal equivalence. Concurrent transitions $t \smile t'$ induce an "atom" of causal equivalence, $t \cdot t'/t \simeq t' \cdot t/t'$. The full relation is generated by closing under the trace constructors (for horizontal composition) and transitivity (for vertical composition). In [15] this yields a composite form of cofinality map γ_α where $\alpha : \tilde{t} \simeq \tilde{u}$ is an arbitrary causal equivalence. We omit further discussion for reasons of space, but note that γ_α is built by composing and translating (by contexts) atomic cofinality maps, and so gives rise, by composition of isomorphisms, to a lattice isomorphism between $\downarrow\mathsf{tgt}(\tilde{t})$ and $\downarrow\mathsf{tgt}(\tilde{u})$.

4 Related work

Reversible process calculi. Reversible process calculi have recently been used for speculative execution, debugging, transactions, and distributed protocols that require backtracking. A key challenge is to permit backwards execution to leverage concurrency whilst ensuring causal consistency. In contrast to our work, reversible calculi focus on mechanisms for reversibility, such as the thread-local memories used by Danos and Krivine's reversible CCS [4], Lanese et al's $\rho\pi$ [10], and Cristescu et al's reversible π-calculus [3]. We intentionally remain agnostic about implementation strategy, whilst providing a formal guarantee that causally consistent rewind and replay are a suitable foundation for any implementation.

Concurrent program slicing. An early example of concurrent dynamic slicing is Duesterwald et al, who consider a language with synchronous message-passing [7]. They give a notion of correctness with respect to a slicing criterion, but find that computing least slices is undecidable, in contrast to our slices which are extremal by construction. Following Cheng [2], most subsequent work has recast dynamic slicing as a dependency-graph reachability problem; our approach is to slice with respect to a particular interleaving, but show how to derive the slice corresponding to any execution with the same dependency structure.

Goswami and Mall consider shared-memory concurrency [8], and Mohapatra et al tackle slicing for concurrent Java [13], but both present only algorithms, with no formal guarantees. Tallam et al develop an approach based on dependency graphs, but again offer only algorithms and empirical results [17]. Moreover most prior work restricts the slicing criteria to the (entire) values of particular variables, rather than arbitrary parts of configurations.

Provenance and slicing. Our interest in slicing arises in part due to connections with provenance, and recent applications of provenance to security [1]. Others have also considered provenance models in concurrency calculi, including Souliah et al [16] and Dezani-Ciancaglini et al [6]. Further study is needed to relate our approach to provenance and security.

5 Conclusion

The main contribution of this paper is to extend our previous approach to slicing based on Galois connections to π-calculus, and show that the resulting notion of slice is invariant under causal equivalence. For this latter step, we build on a prior formalisation of causal equivalence for π-calculus [15]. Although de Bruijn indices significantly complicate the resulting definitions, the formalism is readily implemented in Agda. This paper provides a foundation for future development of rigorous provenance tracing or dynamic slicing techniques for practical concurrent programs, which we plan to investigate in future work.

Acknowledgements. The U.S. Government and University of Edinburgh are authorized to reproduce and distribute reprints for their purposes notwithstanding any copyright notation thereon. Umut Acar helped with problem formulation and an earlier approach. Vít Šefl provided valuable Agda technical support. Our anonymous reviewers provided useful comments.

References

1 U. A. Acar, A. Ahmed, J. Cheney, and R. Perera. A core calculus for provenance. *Journal of Computer Security*, 21:919–969, 2013. Full version of a POST 2012 paper.

2 J. Cheng. Slicing concurrent programs: A graph-theoretical approach. In *Automated and Algorithmic Debugging*, number 749 in LNCS, pages 223–240. Springer-Verlag, 1993.

3 I. Cristescu, J. Krivine, and D. Varacca. A compositional semantics for the reversible pi-calculus. In *LICS*, pages 388–397, June 2013.

4 V. Danos and J. Krivine. Reversible communicating systems. In *Concurrency Theory, 15th International Conference, CONCUR '04*, LNCS, pages 292–307. Springer, 2004.

5 N.G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

6 M. Dezani-Ciancaglini, R. Horne, and V. Sassone. Tracing where and who provenance in linked data: A calculus. *Theoretical Computer Science*, 464:113–129, 2012.

7 E. Duesterwald, R. Gupta, and M. L. Soffa. Distributed slicing and partial re-execution for distributed programs. In *Proceedings of the 5th International Workshop on Languages and Compilers for Parallel Computing*, pages 497–511. Springer, 1993.

8 D. Goswami and R. Mall. Dynamic slicing of concurrent programs. In *High Performance Computing – HiPC 2000*, volume 1970 of *LNCS*, pages 15–26. Springer, 2000.

9 D. Hirschkoff. Handling substitutions explicitly in the pi-calculus. In *2nd International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, 1999.

10 I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order π. In *Concurrency Theory, 21st International Conference, CONCUR '10*, pages 478–493. Springer, 2010.

11 J.-J. Lévy. Optimal reductions in the lambda-calculus. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, New York, NY, USA, 1980.

12 R. Milner. *Communicating and mobile systems: the π calculus*. Cambridge University Press, Cambridge, UK, 1999.

13 D.P. Mohapatra, Rajib Mall, and Rajeev Kumar. An efficient technique for dynamic slicing of concurrent Java programs. In *Applied Computing*, volume 3285 of *LNCS*, pages 255–262. Springer, 2004.

14 R. Perera, U. A. Acar, J. Cheney, and P. B. Levy. Functional programs that explain their work. In *17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 365–376. ACM, 2012.

15 R. Perera and J. Cheney. Proof-relevant pi-calculus, 2016. Submitted to *Mathematical Structures in Computer Science*. `http://arxiv.org/abs/1604.04575`.

16 I. Souilah, A. Francalanza, and V. Sassone. A formal model of provenance in distributed systems. In *TAPP 2009*, Berkeley, CA, USA, 2009. USENIX Association.

17 S. Tallam, C. Tian, and R. Gupta. Dynamic slicing of multithreaded programs for race detection. In *24th IEEE International Conference on Software Maintenance (ICSM 2008)*, pages 97–106. IEEE, 2008.

18 M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE '81, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

A Agda module structure

Figure 6 summarises the module structure of the repository `concurrent-slicing`, which contains the Agda formalisation. The module structure of the auxiliary repositories is described in [15]. All repositories can be found at the URL `https://github.com/rolyp`.

Auxiliary repositories	
`agda-stdlib-ext 0.0.3`	Extensions to Agda library
`proof-relevant-pi 0.3`	Concurrent transitions, residuals and causal equivalence
Core modules	
`Action.Lattice`	Action slices $a' \in \downarrow a$
`Action.Concur.Lattice`	Action residual, lifted to slices
`Action.Ren.Lattice`	Action renaming, lifing to slices
`Braiding.Proc.Lattice`	Bound braids, lifted to slices via braid_ϕ and $\mathsf{unbraid}_\phi$
`ConcurrentSlicing`	Include everything; compile to build project
`ConcurrentSlicingCommon`	Common imports from standard library
`Example`	Milner's scheduler example
`Example.Helper`	Utility functions for examples
`Lattice`	Lattice typeclass
`Lattice.Product`	Component-wise product of lattices
`Name.Lattice`	Name slices $y \in \downarrow x$
`Proc.Lattice`	Process slices $P' \in \downarrow P$
`Proc.Ren.Lattice`	Process renaming, lifted to slices via $\mathsf{ren}_{\rho,P}$ and $\mathsf{unren}_{\rho,P}$
`Ren.Lattice`	Renaming slices $\sigma \in \downarrow \rho$ and application to slices ($\mathsf{app}_{\rho,x}$ and $\mathsf{unapp}_{\rho,x}$)
`Ren.Lattice.Properties`	Additional properties relating to renaming slices
`Transition.Lattice`	Slicing functions step_t and unstep_{gt}
`Transition.Ren.Lattice`	Renaming of transitions, lifted to lattices
`Transition.Concur.Cofinal.Lattice`	Braidings $\gamma_{t,t'}$ lifted to slices
`Transition.Seq.Lattice`	Slicing functions $\mathsf{fwd}_{\tilde{t}}$ and $\mathsf{bwd}_{\tilde{t}}$
Common sub-modules	
`.GaloisConnection`	Galois connection between lattices defined in parent module

Figure 6 `concurrent-slicing` module overview, release `0.1`.

Topological Self-Stabilization with Name-Passing Process Calculi

Christina Rickmann[1], Christoph Wagner[2], Uwe Nestmann[3], and Stefan Schmid[4]

1 **Technische Universität Berlin, Germany**
`c.rickmann@tu-berlin.de`
2 **Technische Universität Berlin, Germany**
`christoph.wagner@tu-berlin.de`
3 **Technische Universität Berlin, Germany**
`uwe.nestmann@tu-berlin.de`
4 **Aalborg University, Denmark**
`schmiste@cs.aau.dk`

Abstract

Topological self-stabilization is the ability of a distributed system to have its nodes themselves establish a meaningful overlay network. Independent from the initial network topology, it converges to the desired topology via forwarding, inserting, and deleting links to neighboring nodes.

We adapt a linearization algorithm, originally designed for a shared memory model, to asynchronous message-passing. We use an extended localized π-calculus to model the algorithm and to formally prove its essential self-stabilization properties: closure and weak convergence for every arbitrary initial configuration, and strong convergence for restricted cases.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Distributed Algorithms, Fault Tolerance, Topological Self-Stabilization, Linearization, Process Calculi

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.19

1 Introduction and Technical Preliminaries

Distributed algorithms are designed to be executed on networked hardware consisting of several connected processes like computers, processors or threads [7]. With the importance of distributed algorithms and the increasing complexity, the need for robust, error-prone solutions rises. The field of self-stabilization [5] offers such fault tolerance. We adapt the algorithm of [4] to a more realistic setting, i.e., we are using a local memory model with asynchronous message-passing opposed to the shared memory model of [4]. Based on an adapted version of the localized π-calculus [8], we formalize the algorithm and use methods similar to [3, 12] to prove it correct.

The approach of self-stabilizing systems was first introduced by [2]. According to [3], the idea of a self-stabilizing system is simply as follows: when started in an arbitrary state it always converges to a desired state. This leads to the ability to tolerate any transient fault, including process crash with recovery, transmission errors like loss or corruption, and corrupted random-access memory. A transient fault is any event that may changes the state of the system, but not its behavior i.e., the program code. The state after the end of the last fault can be considered as a new initial state and the system must recover if no new

faults occur for a sufficiently long period of time. Another characteristic of self-stabilizing algorithms is that they must not terminate [3] and processes must continuously communicate with neighboring nodes. As a consequence, the participating processes can not know whether the system is stabilized (i.e., is in a correct configuration). Dolev defines a self-stabilizing system in [3] as follows:

▶ **Definition 1** (Self-Stabilizing System)**.** A self-stabilizing system can be started in any arbitrary configuration and will eventually exhibit a desired "legal" behavior. We define the desired legal behavior by a set of legal executions denoted LE. A set of legal executions is defined for a particular system and a particular task. Every system execution of a self-stabilizing system should have a suffix that appears in LE. A configuration c is *safe* with respect to a task LE and an algorithm if every fair execution of the algorithm that starts from c belongs to LE (*closure*). An algorithm is *self-stabilizing* for a task LE if every fair execution of the algorithm reaches a safe configuration with respect to LE (*[strong] convergence*).

According to [3], so-called *potential functions* are a classic approach to prove convergence. The idea is to define a function over the configuration set and to prove that this function monotonically decreases (or increases) with every executed step. Additionally, it has to be shown that after the function reaches a certain threshold, the system is in a safe configuration. Since the closure property states that every step from a safe or correct configuration leads again to a correct configuration, closure is usually proven through invariants. An easier to achieve and easier to prove property is weak stabilization. According to [6], a system is weakly stabilizing if for every initial configuration there is an execution that reaches a safe or correct configuration. This property is called *weak convergence*.

Topological self-stabilization describes a particular class of self-stabilizing systems. The goal is that the nodes themselves establish a meaningful overlay network, independent from the initial network topology, via forwarding, inserting, and deleting links to neighboring nodes. One of such desired network topologies is a chain. Given a fixed set of nodes V with unique identifiers (ids) and a total order ($\leq$), the goal is to build an ordered list of the nodes according to their ids. Hence, the (undirected) *linear/chain graph* $G_L = (V, E_L)$ is defined as $\{u, v\} \in E_L$ iff $u = succ(v) \vee v = succ(u)$ (where $succ(v)$ defines the $\leq$-next id after v). Since the successor of any node is (if existent) uniquely defined, the linear graph is also unique for a given node set V. According to [4], a linearization algorithm is defined as follows:

▶ **Definition 2** (Linearization)**.** A *linearization algorithm* is a distributed self-stabilizing algorithm where an *initial configuration* forms any (undirected) connected graph $G_0 = (V, E_0)$, the only *legal configuration* is the linear topology $G_L = (V, E_L)$ on the nodes V, and actions only update the neighborhoods of the nodes.

Gall et al. introduce in [4] two variants of a self-stabilizing algorithm for graph linearization, named LIN_{all} and LIN_{max}. Both are based on the idea that whenever a node has two neighbors, both of which have a smaller (or both a greater) id, it establishes a link between them and deletes its link to the smaller (respectively greater) one. These steps are called left and right linearization and are depicted in Figure 1. The variants of the algorithm only differ in which linearization steps are enabled.

The algorithm only works in a setting where all nodes have write access to the whole memory as the shared variables are written not only by the nodes themselves but also by their neighbors. Such a shared memory model does not seem a good match for distributed systems like peer-to-peer systems, which usually rely on message-passing, and where communication links may be asymmetric. We redesigned and modelled the algorithm for an asynchronous

(a) Left Linearization. **(b)** Right Linearization.

Figure 1 Linearization steps.

message-passing system. Our algorithm corresponds to the LIN_{all} variant, but it would be equally possible—with a small adjustment—to implement LIN_{max} .

In order to prevent a faulty design of an algorithm and to confirm the correctness of proofs the usage of formal methods is imperative. The π-calculus is a well-known and widely-used process calculus to model concurrent and distributed systems. According to [9] it is designed to naturally express processes with a changing link infrastructure, as the communication between processes carries information that leads to a change in the linkage of processes. We model our algorithm in an extension of the localized π-calculus [8], a distributable variant [1] of the π-calculus [11]. We extend the calculus with data (similar to [12]) in order to explicitly keep track of the neighbors of each node and thus of the systems topology. Each node can receive messages via a channel with the same name as its own id. To enable a neighboring node to communicate with another neighbor it is sufficient to send it the corresponding id. We prove the self-stabilization properties closure, weak convergence, and restricted cases of strong convergence utilizing state-based reasoning rather than an action-based style (cf. [12]).

Related Work. Self-stabilization in the context of distributed computing was first introduced by [2]. The fundamentals of topological self-stabilization and linearization for this work originated in [4], which is also the foundation for the algorithm and the main idea of the proofs. The basics in designing a self-stabilizing algorithm and main proof techniques, as well as a general introduction and overview can be found in [3]. We used an extended localized π-calculus to model our algorithm in an unambiguous way and as a formal basis for proofs. Self-stabilization does not offer masking fault tolerance, i.e., it does not ensure liveness and safety of the whole system, but it still ensures liveness, hence nonmasking fault tolerance. Even though masking fault tolerance is strictly stronger than nonmasking, it might not always be achievable or too costly [5], making nonmasking fault tolerance—thus self-stabilizing algorithms—a good alternative. The basic localized π-calculus is due to [8] and extended in a way similar to [12]. Furthermore, we introduce standard forms for reachable states—again based on ideas from [12], and inspired by [9]—which enables us to explicitly and structurally keep track of the global state and therefore the topology of the system.

Contributions and Overview. We adapt a self-stabilizing algorithm for graph linearization: whereas the original algorithm works only in a very restrictive shared memory model, our algorithm is applicable in a completely asynchronous message-passing system.

We formally prove the closure property, i.e., if the system reaches a correct configuration, it stays in a correct configuration if no fault occurs. Furthermore, we prove strong convergence for restricted cases. Assume an initial configuration that is connected while taking the messages in transit into account. Strong convergence holds whenever every process knows, first, at most its desired neighbors, and second, at least its desired neighbors. For the general case, i.e., an arbitrary connected initial configuration, we prove weak convergence. With strong convergence for the corner cases and weak convergence in general, we have all essentials for proving strong convergence in general. Approaches are discussed in [10].

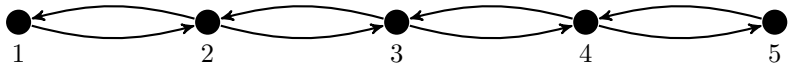


Figure 2 Desired network topology, whereby the nodes are ordered according to their ids.

First, we introduce our model (Section 2) and the redesigned algorithm for asynchronous message-passing (Section 3). Then, we present selected proven properties (Section 4); the complete proofs can be found in [10]. Finally, we summarize our approach and hint at future work (Section 5).

2 Model for Asynchronous Message-Passing

We assume n processes in the system and every process has a unique id. To be as general as possible, we only assume the existence of a total order on these ids.

▶ Assumption (Ids). Every process has a unique constant id.

▶ **Definition 3** (Process identifiers $\mathcal{P}$)**.** Let $\mathcal{P}$ be the (non-empty) finite *set of unique identifiers* of the processes in the system. Let $\leq$ be a total order on $\mathcal{P}$. Let $|\mathcal{P}| = n \in \mathbb{N}$ then there exists an index function (bijection) $i : \mathcal{P} \to \{1, \ldots, n\}$ and $\forall p \in \mathcal{P}.i(p) = |\{q \in \mathcal{P} | q \leq p\}|$ i.e., $i(p)$ describes the position of p with respect to $\leq$.

We define the *predecessor* and the *successor* $pred, succ : \mathcal{P} \to (\mathcal{P} \cup \bot)$ of a process as respectively the next smaller and next greater process according to the total order (and $\bot$ if there is none). We call such a pair of processes *consecutive*. The overlay network that the processes (represented as nodes) shall establish is an ordered doubly-linked list according to the total order on the ids (example depicted in Figure 2). Every process has an unidirectional link (represented as edge) to its predecessor and its successor (with exception of the smallest resp. greatest process) and there are no other links. We call this topology the (directed) linear graph. The undirected linear graph is the undirected variant. Here, for every pair of consecutive processes at least one of them has a link to the other one.

▶ **Definition 4** (Desired Topology Graph)**.** The *(directed) linear graph* $G_{LIN} = (\mathcal{P}, E)$ is defined as $E = \{(p, q) | p, q \in \mathcal{P} \wedge (p = succ(q) \vee q = succ(p))\}$ and the *undirected* U_{LIN} is defined accordingly.

The *distance* $dist : \mathcal{P} \times \mathcal{P} \to \mathbb{N}$ is the number of nodes between the two processes according to the total order. A process has a distance of zero to itself and the distance between a pair of consecutive processes is one. The length of a (directed or undirected) edge is defined as the distance between the connected nodes.

Extended Localized Pi-Calculus. To model the algorithm, we introduce an extension of the name-passing localized π-calculus, the extended localized π-calculus $\text{eL}\pi = \langle \mathcal{P}_{\text{eL}}, \longmapsto \rangle$. The extension is based on ideas similar to [12], which allows us to define a kind of standard form for a configuration of our algorithm. The local state of all processes and the messages in transit, and therefore the global state of the system, is directly accessible via the parameters of the corresponding process definition. This in turn allows state-based proofs, which is more traditional for distributed algorithms [7], instead of the action-based style of process calculi.

▶ Notation (Multisets). Let $a, b, c \in S$ be arbitrary elements of an arbitrary set S. We denote with $M = \{|a, a, b, c|\}$ a multiset and use $\mathbb{N}^S$ as the type of such a set. Furthermore, the

union $\cup$ of two multisets is the multiset where all appearances of elements in both are added and the difference $\setminus$ is the multiset where all appearances of elements in the first multiset are decreased by those in the second (but at least zero). Since sets are only special cases with multiplicity one for all elements, we also use combinations of sets and multisets.

We assume the existence of a countably infinite set $\mathbf{A}$ containing all channel names, function names, and variables. $K(X)$ denotes a parameterized process constant, which is defined with respect to a finite set of process equations D of the form $\{K_j(X) = P_j\}_{j\in J}$. Since we use parameterized process constants, we exclude replication and use instead recursion via process definitions to model repetitive behavior.

▶ **Definition 5** (Syntax of the extended Localized π-Calculus: $\mathcal{P}_{\mathrm{eL}}$).

$$
\begin{array}{lll}
\textsc{Data Values} \ \mathbf{V} & v & ::= \bot \mid 0 \mid 1 \mid c \mid (v,v) \mid \{v,\ldots,v\} \mid \\
 & & \quad \{\!|v,\ldots,v|\!\}, \text{ with } c \in \mathbf{A} \\
\textsc{Variable Pattern} & X & ::= x \mid (X,X), \text{ with } x \in \mathbf{A} \\
\textsc{Expressions} & e & ::= v \mid X \mid (e,e) \mid f(e), \text{ with } f \in \mathbf{A} \\
\textsc{Processes} \ \mathbf{P} & P & ::= 0 \mid P \mid P \mid c(X).P \mid \overline{c}\langle v\rangle \mid (\nu c)\, P \mid \\
 & & \quad \textbf{if } e \textbf{ then } P \textbf{ else } P \mid \textbf{let } X = e \textbf{ in } P \mid K(e) \\
\textsc{Process Equations} & D & = \{K_j(X) = P_j\}_{j\in J} \text{ a finite set of process definitions}
\end{array}
$$

where in $c(X).P$ every variable x that appears in X may not occur free in P in input position.

Names received as an input and restricted names are *bound names*. The remaining names are *free names*. Accordingly, we assume three sets, the sets of names $\mathsf{n}(P)$ and its subsets of free names $\mathsf{fn}(P)$ and bound names $\mathsf{bn}(P)$, with each term P. To avoid name capture or clashes, i.e., to avoid confusion between free and bound names or different bound names, bound names can be mapped to fresh names by *α-conversion*. We write $P \equiv_\alpha Q$ if P and Q differ only by α-conversion. The substitution of value v for a variable pattern X in expression e or process P is written $\{^v/_X\}e$ and $\{^v/_X\}P$ respectively. Note that only data values can be substituted for names and that all variables of the pattern X must be free in P (while possibly applying α-conversion to avoid capture or name clashes). Let $[\![e]\!]$ denote the evaluation of expression e which allows results in a data value, defined in the standard way.

▶ **Definition 6** (Structural Congruence for the Localized π-Calculus). The structural congruence for the extended localized π-calculus is based on the structural congruence for the π-calculus:

$$
\begin{array}{cc}
P \equiv Q \text{ if } P \equiv_\alpha Q \qquad P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu n)\, 0 \equiv 0 \\
P \mid (\nu n)\, Q \equiv (\nu n)\, (P \mid Q), \text{if } n \notin \mathsf{fn}(P) \qquad (\nu n)\, (\nu m)\, P \equiv (\nu m)\, (\nu n)\, P \\
\textbf{if } e \textbf{ then } P \textbf{ else } Q \equiv P, \text{ if } [\![e]\!] = 1 \qquad \textbf{if } e \textbf{ then } P \textbf{ else } Q \equiv Q \text{ if } [\![e]\!] = 0 \\
\textbf{let } X = e \textbf{ in } P \equiv \{^{[\![e]\!]}/_X\}P \qquad K(e) \equiv \{^{[\![e]\!]}/_X\}P \text{ if } (K(X) = P) \in D
\end{array}
$$

We are only interested in the interaction between the processes and not with any further environment. Therefore, we only present a reduction semantics for our extended localized π-calculus, based on the reduction semantics of the π-calculus.

▶ **Definition 7** (Reduction Semantics of the extended Localized π-Calculus: $\longmapsto$). Defined as:

$$
\texttt{comm:}\ \frac{}{c(X).P \mid \overline{c}\langle v\rangle \longmapsto \{^v/_X\}P} \qquad \texttt{res:}\ \frac{P \longmapsto P'}{(\nu c)\, P \longmapsto (\nu c)\, P'}
$$

$$
\texttt{struct:}\ \frac{P \equiv Q \qquad Q \longmapsto Q' \qquad Q' \equiv P'}{P \longmapsto P'} \qquad \texttt{par:}\ \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q}
$$

▶ **Definition 8** (Steps)**.** We call a single application of this reduction semantics, $P \longmapsto P'$, a step and write $\Longmapsto$ for the reflexive and transitive closure of $\longmapsto$. We use *execution* to refer to a reduction starting from a particular term.

Every structural extension like function calls, if-then-else-statements, and let-in-statements are evaluated in the structural congruence. The evaluation of these constructs is not considered a step on its own. Hence, internal computations are executed as parts of other steps.

The fault tolerance of self-stabilization is based on the property that the initial state can be arbitrarily. It is sufficient to show that every arbitrary state can serve as an initial state to ensure this form of fault tolerance since the state after every fault can be seen as a new initial state. Thus, we assume for the proofs as usual that there are no faults. An infinite message delay can be seen as message loss. Therefore, every message that is sent is received after finite time. Since the message-passing model is asynchronous, there are no further assumptions regarding the delivery time of messages.

▶ Assumption (No Message Loss). Every message is received after a finite but arbitrary number of steps.

Furthermore, we need an assumption of fairness, as otherwise nodes could starve. A process starves if it never executes a step. Furthermore, a subprocess of a node could starve, e.g. if the process is only consuming messages, but never tries to find a linearization step itself. Without a fairness assumption, it is not possible to show any progress in the system.

▶ Assumption (Fairness). Every continuously enabled subprocess will eventually (after an arbitrary but finite number of steps) execute a step.

3 Linearization Algorithm for Asynchronous Message-Passing

The utilization of a calculus enables us to model the algorithm unambiguously and allows us to formally prove properties of the algorithm. Although in the calculus itself all channels are bidirectional, we only use them in an unidirectional manner. Together with the output-capability restriction of the localized π-calculus this helps us to ensure that every process can be implemented in an asynchronous setting on a different location [1].

Each node can receive messages from other nodes via a channel with the same name as its id. One could think of the ids as serving as the IP address of the corresponding process. To enable a neighbor to communicate with another process, one sends it the corresponding id. Thus, $\mathbf{A}$ contains the ids of all processes in the system i.e., $\mathcal{P} \subseteq \mathbf{A}$.

To model local variables, we use restricted channels for every process. Each variable is represented through a message in transit that only can be sent and received by the corresponding process. The value of such a local variable is modeled by the value of the matching message in transit. Thus, receiving the message corresponds to reading the variable and sending corresponds to writing. In our algorithm, every process p has one local variable nb_p, describing the neighborhood of the process i.e., it contains the ids of all processes that p knows and can send messages to.

Each process can be in one of two states. In the state $Alg\,(p, nb)$ the process p is able to receive a message from another process in the system. In the state $Alg'\,(p, nb, x)$ the process p can add a previously received process id x to its current neighborhood nb. In doing so, a process blocks the reception of messages until the the previously received id is added to the neighborhood. In both states, the process can additionally try to find a linearization step in its current neighborhood nb based on internal computations. If it finds no possible linearization step, it sends *keep-alive*-messages to its current neighbors.

(a) Directed Left Linearization. **(b)** Directed Right Linearization.

Figure 3 Directed linearization steps.

In the algorithm, $\mathit{LeftN} : \mathcal{P} \times 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ calculates the *left neighborhood* of a process i.e., all neighbors with a smaller id and corresponding $\mathit{RightN} : \mathcal{P} \times 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ the *right neighborhood* of a process i.e., all neighbors with a greater id. $\mathit{findLin} : \mathcal{P} \times 2^{\mathcal{P}\times\mathcal{P}} \to 2^{\mathcal{P}\times\mathcal{P}}$ calculates all possible linearization steps in the neighborhood of a process, based on the input set: $\mathit{findLin}(p,y) = \{(q,r) | q,r \in y \wedge q < r \wedge (q,r \in \mathit{LeftN}(p,y) \vee \mathit{RightN}(p,y))\}$.
The function $\mathit{select} : 2^{\mathcal{P}\times\mathcal{P}} \to (\mathcal{P} \times \mathcal{P})$ returns one arbitrary of these linearization steps.

▶ **Definition 9** (Subprocesses). For every process $p \in \mathcal{P}$ we denote $\mathit{Alg}_{match}(p)$, $\mathit{Alg}_{rec}(p)$, and $\mathit{Alg}_{add}(p,\cdot)$ as *subprocesses* of p. Note that all subprocess are input guarded.

$$\mathit{Alg}(p, initNb) = (\nu nb_p)\left(\ \overline{nb_p}\langle initNb\rangle \mid \mathit{Alg}_{rec}(p) \mid \mathit{Alg}_{match}(p)\ \right)$$
$$\mathit{Alg}'(p, initNb, x) = (\nu nb_p)\left(\ \overline{nb_p}\langle initNb\rangle \mid \mathit{Alg}_{add}(p,x) \mid \mathit{Alg}_{match}(p)\ \right)$$
$$\mathit{Alg}_{rec}(p) = p(x).\mathit{Alg}_{add}(p,x) \qquad \mathit{Alg}_{add}(p,x) = nb_p(y).\left(\overline{nb_p}\langle y \cup \{x\}\rangle \mid \mathit{Alg}_{rec}(p)\right)$$

$$\begin{aligned}\mathit{Alg}_{match}(p) = nb_p(y).\big(\ &\textbf{let } x = \mathit{select}(\mathit{findLin}(p,y)) \textbf{ in}\\ &\textbf{if } x = \bot \textbf{ then } \prod_{j\in y} \overline{j}\langle p\rangle \mid \overline{nb_p}\langle y\rangle\\ &\textbf{else if } x = (j,k) \textbf{ then}\\ &\quad\textbf{if } j < k \wedge k < p \textbf{ then } \overline{j}\langle k\rangle \mid \overline{nb_p}\langle y \setminus \{j\}\rangle\\ &\quad\textbf{else if } j < k \wedge p < j \textbf{ then } \overline{k}\langle j\rangle \mid \overline{nb_p}\langle y \setminus \{k\}\rangle\\ &\quad\quad\textbf{else } \overline{nb_p}\langle y\rangle\\ &\textbf{else } \overline{nb_p}\langle y\rangle\\ \mid \mathit{Alg}_{match}(p)\ \big)\end{aligned}$$

The subprocess $\mathit{Alg}_{rec}(p)$ models the ability of a process to receive a message. When it receives a message with content x, it continues as subprocess $\mathit{Alg}_{add}(p,x)$, thus the process changes its state. $\mathit{Alg}_{add}(p,x)$ reads the current value of the neighborhood of p and adds the previous received process id x. Afterwards, the process is again able to receive any message.

The subprocess $\mathit{Alg}_{match}(p)$ defines the behavior, based on the internal computations of $\mathit{findLin}(p, nb)$, in case process p tries to find a linearization step in its neighborhood nb. If there is no possible linearization step, *select* returns $\bot$ and the process sends *keep-alive*-messages to its current neighbors. In case *select* returns a tuple, it defines a left or right linearization step respectively and p sends the further away process the id of the other process and deletes the receiver from its neighborhood (as depicted in Figure 3). The other two else cases are only implemented to obtain a complete case distinction. It is ensured by the definitions that these branches are never explored. The sending of the message $\overline{nb_p}\langle y\rangle$ ensures that, if there would be a possibility to explore these branches, nothing changes. The same value that was read in the previous step (i.e., received by the $nb_p(y)$ message) is written (i.e., sent) again and therefore the neighborhood remains unchanged. In all cases, the process is directly able to try to find another linearization step.

The system is composed of n such processes. The global states that serve as starting points for the executions of our algorithm are called initial configurations. Later we show that every global state can serve as such a starting point as required for self-stabilization.

▶ **Definition 10** (Initial Configuration)**.** Let

- $\mathcal{P}$ be the *set of unique identifiers* and $P, P' \subseteq \mathcal{P}$ with $P \cup P' = \mathcal{P}$ and $P \cap P' = \emptyset$,
- $init : \mathcal{P} \to 2^{\mathcal{P}}$ a function that defines for every process $p \in \mathcal{P}$ the neighborhood i.e., which process ids are known by p,
- $Msgs \in \mathbb{N}^{\mathcal{P}\times\mathcal{P}}$ a multiset that describes the messages in transit and
- $add : \mathcal{P} \rightharpoonup \mathcal{P}$ a partial function with $\forall p \in P'.\exists q \in \mathcal{P}.(p,q) \in add$ and $\forall p \in P.\forall q \in \mathcal{P}.(p,q) \notin add$ that describes the adding in progress i.e., where $add(p) = q$ describes that p wants to add q to its neighborhood.

Then, an *initial configuration* of the algorithm is defined as the process term:

$$Alg_{all}(P, P', init, Msgs, add) = \prod_{j\in P} Alg(j, init(j)) \mid \prod_{j\in P'} Alg'(j, init(j), add(j)) \mid \prod_{(j,k)\in Msgs} \overline{j}\langle k\rangle$$

In an initial configuration there is for every process $p \in \mathcal{P}$ exactly one $\overline{nb_p}\langle\cdot\rangle$-message. This message can not be lost or duplicated through a previous fault as it models a variable. A transient fault can lead to an arbitrary value of a variable but not to its disappearance or duplication. This does not restrict the fault tolerance of the algorithm. Self-stabilization tolerates transient faults but no permanent ones. The disappearance or duplication of a variable could only be caused by corruption of the program code itself which would be a permanent fault. The value of this message can be an arbitrary set of $\mathcal{P}$ (without p itself), reflecting the arbitrary initial neighborhood of p. These messages of all processes describe the initial network topology. A configuration describes the global state of the system, consisting of the states of all processes and messages in transit.

▶ **Definition 11** (Configuration)**.** Let $\mathcal{I}$ be the set of all initial configurations. We call every process term C that can be reached from any arbitrary initial configuration, i.e., $\exists I \in \mathcal{I}.I \Longmapsto C$, *configuration*. We denote the set of all such configurations with $\mathcal{T}$.

▶ **Definition 12** (Reachability)**.** We call a configuration C' *reachable* from a configuration C iff $C \Longmapsto C'$. Further, we say a configuration with a predicate P is *reached* from configuration C iff in every execution there is a configuration C' with $C \Longmapsto C'$ and P holds for C'.

For every configuration C there are parameters with the same properties as in Definition 10 so that C is structurally equivalent to $Alg_{all}(\cdot,\cdot,\cdot,\cdot,\cdot)$. We call $Alg_{all}(\cdot,\cdot,\cdot,\cdot,\cdot)$ the standard form of a configuration and use it as representative of all structurally equivalent configurations. Therefore, every configuration is structurally equivalent to an initial configuration.

▶ **Lemma 13** (Standard form)**.** *Starting from an arbitrary initial configuration every reachable configuration is structurally equivalent to a term* $Alg_{all}(P, P', nb, Msgs, add)$ *whereby* $P, P' \subseteq \mathcal{P}$ *with* $P \cup P' = \mathcal{P}$ *and* $P \cap P' = \emptyset$*,* $nb : \mathcal{P} \to 2^{\mathcal{P}}$ *is a function that defines for every process* $p \in \mathcal{P}$ *the neighborhood,* $Msgs \in \mathbb{N}^{\mathcal{P}\times\mathcal{P}}$ *is the multiset of messages in transit,* $add : \mathcal{P} \rightharpoonup \mathcal{P}$ *is a partial function with* $\forall p \in P'.\exists q \in \mathcal{P}.(p,q) \in add$ *and* $\forall p \in P.\forall q \in \mathcal{P}.(p,q) \notin add$*.*

▶ Notation (Configuration Components). Let A be an arbitrary configuration then there are parameters P, P', nb, $Msgs$, add with $A \equiv Alg_{all}(P, P', nb, Msgs, add)$ and we denote in the following : $P_A = P$, $P'_A = P'$, $nb_A = nb$, $Msgs_A = Msgs$ and $add_A = add$.

(a) Topology without Messages. **(b)** Topology with Messages.

Figure 4 Topology with and without messages whereby solid lines represent the edges.

▶ **Corollary 14** (Steps). *For every configuration A there are always exactly the following steps (up to structural congruence) possible:*

- $\forall p \in \mathcal{P}.select\,(findLin\,(p, nb_A(p))) = \bot \implies$
  $A \longmapsto Alg_{all}\,(P_A, P'_A, nb_A, Msgs_A \cup \{(j,p) | j \in nb_A(p)\}, add_A)$
- $\forall p \in \mathcal{P}.select\,(findLin\,(p, nb_A(p))) = (j,k) \wedge (j < k \wedge k < p) \implies A \longmapsto A'$ *with*
  $A' \equiv Alg_{all}\,(P_A, P'_A, nb, Msgs_A \cup \{(j,k)\}, add_A)$ *and* $nb(x) = \begin{cases} nb_A(x), & \text{if } x \neq p \\ nb_A(p) \setminus \{j\}, & \text{if } x = p \end{cases}$
- $\forall p \in \mathcal{P}.select\,(findLin\,(p, nb_A(p))) = (j,k) \wedge (j < k \wedge p < j) \implies A \longmapsto A'$ *with*
  $A' \equiv Alg_{all}\,(P_A, P'_A, nb, Msgs_A \cup \{(k,j)\}, add_A)$ *and* $nb(x) = \begin{cases} nb_A(x), & \text{if } x \neq p \\ nb_A(p) \setminus \{k\}, & \text{if } x = p \end{cases}$
- $\forall p \in \mathcal{P}.$*if select* $(findLin\,(p, nb_A(p)))$ *does not match any previous case* $\implies A \longmapsto A$
- $\forall p \in P_A.\exists q \in \mathcal{P}.(p,q) \in Msgs_A \implies$
  $A \longmapsto Alg_{all}\,(P_A \setminus \{p\}, P'_A \cup \{p\}, nb_A, Msgs_A \setminus \{(p,q)\}, add_A \cup \{(p,q)\})$
- $\forall p \in P'_A.\exists q \in \mathcal{P}.(p,q) \in add_A \wedge\ A \longmapsto A'$ *with* $nb(x) = \begin{cases} nb_A(x), & \text{if } x \neq p \\ nb_A(p) \cup \{q\}, & \text{if } x = p \end{cases}$ *and*
  $A' \equiv Alg_{all}\,(P_A \cup \{p\}, P'_A \setminus \{p\}, nb, Msgs_A, add_A \setminus \{(p,q)\})$

Topology of Configuration. The system is in a legal state i.e., correct configuration, if the network topology of the system is the desired one, i.e., the linear graph. Thus, we need to define the network topology of a configuration. The network topology graph of a configuration describes to whom each process can send messages directly and therefore corresponds to the neighborhood sets of all processes. To define the several cases in which strong convergence holds and lower the preconditions as much as possible, we introduce variants of the network topology of a configuration. They differ in whether we regard the direction of edges, and if we take messages in transit into account or not. The network topology graphs with messages describe the neighborhoods if all current messages in transit were received and processed (depicted in Figure 4). This variant of the topology is, for example, also needed to define a correct configuration. It is not enough that the current topology corresponds to the linear graph, it also has to be ensured that no undesired connection will be established through a message in transit.

▶ Notation (Topology (without messages)). In figures of the topology graph without messages a solid line represents a process in the neighborhood, a dotted line a message in transit, and a dashed line an adding in progress, i.e., an id that is already received but not yet added to the neighborhood. In the topology graph with messages these are all defined as edges and represented through a solid line.

▶ **Definition 15** (Network Topology Graph (without Messages)). Let A be an arbitrary configuration. The *(directed) topology graph (without messages)* $T(A) = (\mathcal{P}, E)$ is defined as: $E = \{(p,q) | p, q \in \mathcal{P} \wedge q \in nb_A(p)\}$

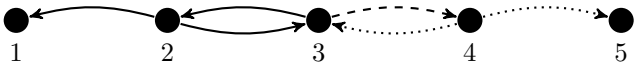


Figure 5 Topology of an undirected correct configuration.

▶ **Definition 16** (Network Topology Graph with Messages)**.** Let A be an arbitrary configuration. The *(directed) topology graph with messages* $T^M(A) = (\mathcal{P}, E)$ is defined as: $E = \{(p,q) | p,q \in \mathcal{P} \wedge (q \in nb_A(p) \vee (p,q) \in Msgs_A \vee add_A(p) = q)\}$.

The undirected variants, i.e., the *undirected topology graph* $U(A)$ and the *undirected topology graph with messages* $U^M(A)$, are defined correspondingly.

▶ Notation (Topology Graph Components). Let A be an arbitrary configuration, we introduce $E_{T(A)}, E_{T^M(A)}, E_{U(A)}$ and $E_{U^M(A)}$ to denote the edges of the different topology graphs $T(A) = (\mathcal{P}, E)$, $T^M(A) = (\mathcal{P}, E')$, $U(A) = (\mathcal{P}, E'')$ and $U^M(A) = (\mathcal{P}, E''')$, i.e., we denote in the following: $E_{T(A)} = E$, $E_{T^M(A)} = E'$, $E_{U(A)} = E''$ and $E_{U^M(A)} = E'''$.

If the topology graph with messages of the initial configuration is weakly connected, the topology graph with messages of all reachable configurations is weakly connected. The only steps that remove edges are linearization steps. If a process executes a linearization step, the removed edge can be simulated by the introduced edge and the edge to the other neighbor of the executing process. Thus, linearization can not result in partitioning the network.

▶ **Lemma 17** (Connectivity)**.** *Let A_0 be an initial configuration and A an arbitrary reachable configuration. Then, it holds that if $U^M(A_0)$ is connected, also $U^M(A)$ is connected.*

Correct Configuration. A configuration is correct if every process knows only its consecutive processes. To ensure that no other connections will be established, it must also hold that every message in transit contains the id of a consecutive process of the receiver. Thus, the network topology with and without messages must be the desired linear graph. With exception of the number of these messages and the state of the processes, the correct configuration is unique.

▶ **Definition 18** (Correct Configuration)**.** Let A be an arbitrary configuration. A is a *correct configuration* iff $T^M(A) = G_{LIN} \ \wedge \ T(A) = G_{LIN}$

▶ **Lemma 19** (Uniqueness, Directed Case)**.** *The correct configuration is unique up to structural congruence, the number of messages in the system and the state of the processes.*

A weaker property is described by an undirected correct configuration. Here, we only demand that the undirected topology with message must be the undirected linear graph (as depicted in Figure 5). Hence, the neighborhood of each process is a subset of its consecutive processes and the messages in transit must satisfy the same requirement as before. To ensure connectivity, between each pair of consecutive processes there must be at least one connection while taking the messages in transit into account. Similarly to a correct configuration, an undirected configuration is uniquely defined with exception of the number of messages, the state of a process, and the type of the connection between a consecutive pair of processes.

▶ **Definition 20** (Undirected Correct Configuration)**.** Let A be an arbitrary configuration. A is an *undirected correct configuration* iff $U^M(A) = U_{LIN}$

▶ **Lemma 21** (Uniqueness, Undirected Case)**.** *The undirected correct configuration is unique up to structural congruence, the number of messages in the system, the state of the processes, and the type of connections.*

4 Results

We want to prove the algorithm correct. In order to show that the algorithm is self-stabilizing, we have to prove *convergence* and *closure* (Definition 1). To be a linearization algorithm according to Definition 2, the system is in a legal state if and only if the topology is the linear graph. The complete proofs together with all omitted results can be found in [10].

Closure. Several closure properties are based on two facts: every process may only remove processes from its own neighborhood and a process never removes the id of a desired neighbor i.e., its predecessor and successor. Further, if every process knows only (a subset of) desired neighbors, there are no more possible linearization steps in the system. All processes only send and receive *keep-alive*-messages to and from desired neighbors respectively.

The only steps that remove edges i.e., ids from the neighborhood, are linearization steps. Whenever a process does not execute any linearization steps, its neighborhood can be expanded by reception of messages but not shrink.

If a process executes a linearization step it never removes a correct neighbor as it always removes the process that is further away. Thus, if a process knows a correct neighbor, it remains in the neighborhood for every reachable configuration. It follows directly that in the directed and undirected topology without messages edges between consecutive neighbors are preserved.

▶ **Corollary 22** (Preservation of Correct Edges). *Let A be an arbitrary configuration. For every reachable configuration R i.e., $A \Longmapsto R$, it holds: $\forall p \in \mathcal{P}.\forall p' \in \{succ(p), pred(p)\}.(p,p') \in E_{T(A)} \implies (p,p') \in E_{T(R)}$ and $\forall p \in \mathcal{P}.\{p, succ(p)\} \in E_{U(A)} \implies \{p, succ(p)\} \in E_{U(R)}$*

This preservation holds further for correct edges in the topologies with messages. Every id carried by a messages cannot get lost and is received and processed eventually. Therefore, also edges between desired neighbors that represent adding or messages are preserved.

▶ **Lemma 23** (Preservation of Correct Edges with Messages). *Let A be an arbitrary configuration. For every reachable configuration R holds: $\forall p \in \mathcal{P}.\forall p' \in \{succ(p), pred(p)\}.(p,p') \in E_{T^M(A)} \implies (p,p') \in E_{T^M(R)}$ and $\forall p \in \mathcal{P}.\{p, succ(p)\} \in E_{U^M(A)} \implies \{p, succ(p)\} \in E_{U^M(R)}$.*

The topology of a configuration contains the desired topology i.e., the linear graph is a subgraph, if every correct edge is already established but possibly undesired edges are still existent in addition. Since every correct edge is always preserved, this property is invariant.

▶ **Corollary 24** (Closure for $T^M \subseteq G_{LIN} \wedge T \subseteq G_{LIN}$). *If A is a configuration with $G_{LIN} \subseteq T^M(A) \wedge G_{LIN} \subseteq T(A)$ it holds for every reachable configuration R i.e., $A \Longmapsto R$, that $G_{LIN} \subseteq T^M(R) \wedge G_{LIN} \subseteq T(R)$.*

In the topology of an undirected correct configuration every edge is correct but there may be correct edges missing. Through preservation of correct edges this property is invariant. Each configuration that is reachable from an undirected correct configuration is itself an undirected correct configuration. Further, none already established correct edge has been removed.

▶ **Lemma 25** (Closure for Undirected Correct Configuration). *Let A be an arbitrary undirected correct configuration. It holds for every reachable configuration C i.e., $A \Longmapsto C$, that C is also an undirected correct configuration.*

The topology is the linear graph if and only if the configuration is a correct configuration as in Definition 18. *Closure* states once a correct configuration is reached, provided no fault occurs, the system stays in a correct configuration.

▶ **Theorem 26** (Closure for Correct Configurations)**.** *Let A be a correct configuration then it holds for every reachable configuration C i.e., $A \Longrightarrow C$, that C is also a correct configuration.*

Convergence We prove strong convergence for restricted cases and weak convergence in general. Strong convergence is proven if either there are possibly correct edges missing but no non-correct edges existent in the topology with messages i.e., the topology is a subgraph of the linear graph, or there are possibly still non-correct edges but at least all correct edges are contained in the topology with messages i.e., the linear graph is a subgraph of the topology.

If there are only correct edges missing, there is at least one connection between every pair of consecutive processes. No more possible linearization steps exist and every process sends *keep-alive*-messages to the current known subset of desired neighbors. All these messages are received and processed eventually and all missing correct edges are eventually established.

▶ **Lemma 27** (Convergence for $U^M = U_{LIN}$)**.** *Let A be an arbitrary configuration. If the undirected network topology with messages is the desired undirected topology i.e., $U^M(A) = U_{LIN}$, then a correct configuration C is reached after a finite number of steps i.e., $A \Longrightarrow C \ \wedge \ T^M(C) = G_{LIN} \ \wedge \ T(C) = G_{LIN}$.*

If there are possibly still undesired edges in the topology but at least all correct edges are established, the only processes that can send *keep-alive*-messages are processes that only know their desired neighbors. Every process knows at least its desired neighbors and whenever a process has additional neighbors, there is a linearization step. With every linearization step the topology gets closer to the desired topology which is shown via a potential function.

▶ **Lemma 28** (Convergence for $G_{LIN} \subseteq T^M \wedge G_{LIN} \subseteq T$)**.** *Let A be an arbitrary configuration. If $G_{LIN} \subseteq T^M(A) \wedge G_{LIN} \subseteq T(A)$, then a correct configuration C is reached after a finite number of steps i.e., $A \Longrightarrow C \ \wedge \ T^M(C) = G_{LIN} \ \wedge \ T(C) = G_{LIN}$.*

Convergence also holds if the desired topology is a subgraph of the topology with messages. Since every message that is in transit will eventually be received and processed, we always reach a configuration with the linear graph as a subgraph of the topology without messages.

▶ **Lemma 29** (Convergence for $G_{LIN} \subseteq T^M$)**.** *Let A be an arbitrary configuration. If $G_{LIN} \subseteq T^M(A)$, then a correct configuration C is reached after a finite number of steps i.e., $A \Longrightarrow C \ \wedge \ T^M(C) = G_{LIN} \ \wedge \ T(C) = G_{LIN}$.*

The proofs use the fact that *keep-alive*-messages are only exchanged between desired neighbors. Proving strong convergence in general is much more difficult as the sending of *keep-alive*-messages to undesired neighbors can cause the reestablishing of undesired connections. Therefore, we show weak convergence i.e., for every initial configuration there are executions that converge to a correct configuration. For this, we define a perfect oracle. It cannot be implemented and should only be seen as a restriction on the set of executions.

▶ **Definition 30** (Perfect Oracle O)**.** A *perfect oracle* is a global omniscient instance that whenever the system is not in an (undirected) correct configuration only let the processes send *keep-alive*-messages to resolve deadlocks and otherwise suppresses all *keep-alive*-messages.

Figure 6 Linearization steps in the undirected topology with messages but not in the directed.

▶ Remark. We show in Theorem 31 that starting from an arbitrary initial configuration a correct configuration is reached after a finite number of steps. A perfect oracle O does not suppress the sending of *keep-alive*-messages in a correct configuration. Once the system is in a correct configuration, it stays in a correct configuration according to the closure property as shown in Theorem 26. Hence, a perfect oracle is not contradictory to the fairness assumption.

We show that every execution that is admissible under the restriction of a perfect oracle converges to a correct configuration. Since for every configuration this set of executions is non-empty, we prove weak convergence in general. We introduced three potential functions and showed that whenever at least one of them is minimal, the system reaches a correct configuration in a finite number of steps. For every configuration, exactly one of the three cases true: there is no more linearization step in the undirected topology with messages, there is a linearization step in the directed topology with messages, or there is a linearization step in the undirected but not in the directed topology with messages. We show that the system neither can stay infinite long in the second case nor can infinitely often alternate between the second and the third case without reaching a configuration in which at least one of the three potential functions is minimal and thus convergence ensured.

▶ **Theorem 31** (Convergence with Perfect Oracle)**.** *Let I be an arbitrary connected, i.e., $U^M(I)$ is connected, initial configuration and A an arbitrary reachable configuration i.e., $I \Longmapsto A$. Assume there is a perfect oracle O. Then a correct configuration C is reached after a finite number of steps i.e., $A \Longmapsto C \ \wedge \ T^M(C) = G_{LIN} \ \wedge \ T(C) = G_{LIN}$.*

5 Conclusion

We adapted the algorithm for shared memory of [4] such that it works in an asynchronous message-passing system. The algorithm of [4] requires a system where all processes must have access to the whole memory, which is very restrictive. In the redesigned algorithm, processes communicate via message-passing and we do not make any assumptions about the time a process needs to execute a step or for message delivery. This makes it applicable in a completely asynchronous message-passing system which is a significantly weaker requirement and corresponds more to real life system conditions.

An algorithm is self-stabilizing, if it satisfies the properties of closure and convergence. We formally proved the closure property, i.e., if the system reaches a correct configuration, it stays in a correct configuration if no fault occurs. There are two forms of convergence. Starting with an arbitrary initial configuration, strong convergence requires that in every execution a correct configuration is reached, whereas weak convergence only claims the existence of such an execution. We proved strong convergence for restricted cases. First, whenever the topology with messages of a connected initial configuration only lacks desired edges but no undesired ones exist, i.e., every process knows at most its desired neighbors, strong convergence holds. Second, strong convergence also holds, whenever in the topology with messages of an initial configuration there are just too many edges, but no desired ones are missing. For the general case, i.e., an arbitrary connected initial configuration, we proved weak convergence. For this proof, we introduced a global omniscient entity, called a perfect oracle. We showed that every execution that is admissible under the assumption of a perfect

oracle ensures strong convergence. Since for every initial configuration this is a non-empty set of executions, weak convergence holds in general.

We extended the localized π-calculus, which provides us through its clearly defined syntax and semantics with the possibility to model the algorithm in a precise and unambiguous manner. It is also the basis for formally proving properties about the algorithm. The usage of standard forms [12] of configurations helps us to simplify the proof by identifying every possible reachable process term with a structurally equivalent representative and significantly reduces the number of cases to be analyzed. Further, this enables us to explicitly and conveniently keep track of the global state of the system. This allows us to execute our proofs in a state-based fashion, which is more traditional for distributed algorithms [7], rather than in an action-based style, which would be more typical when using process calculi [11].

Future Work. As we strongly conjecture strong convergence to hold in general, the primary goal is a convergence proof for the general case that works without any oracle at all. The problem is that *keep-alive*-messages can reestablish edges that were already removed through linearization steps. Nevertheless, if a process executes a linearization step, it prevents the further away process eventually from ever sending *keep-alive*-messages to it again. Neither *keep-alive*-messages nor linearization steps establishing edges that are longer as the current longest edge in the topology with messages. The edge that is established through a linearization step is even strictly shorter than the at least temporarily removed one.

The main difficulty is: a potential function that decreases strictly with every linearization step cannot be monotonically decreasing with every step. Thus, potential functions alone are not sufficient to prove strong convergence. A promising approach lies in finding good properties for livelock freedom. With such properties, a general proof can likely be achieved in various ways, as discussed in [10]. For example, livelock freedom properties could be used to show that linearization steps involving the current longest edges are eventually enabled and executed. If it can be shown that the current longest edges are eventually removed permanently, strong convergence would hold.

In preparation for a general proof, it could be interesting to lower the restrictions on the set of executions by an oracle and consider a weaker oracle to acquire further insight in properties that could be helpful for a proof without any oracle.

References

1 P.-D. Brodmann. Distributability of Asynchronous Process Calculi. Master's thesis, Technische Universität Berlin, Germany, October 2014.

2 E. W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Communications of the ACM*, 17(11):643–644, November 1974.

3 S. Dolev. *Self-Stabilization.* The MIT Press, 2000.

4 D. Gall et al. A Note on the Parallel Runtime of Self-Stabilizing Graph Linearization. *Theory of Computing Systems*, 55(1):110–135, 2014.

5 F. C. Gärtner. Fundamentals of Fault-tolerant Distributed Computing in Asynchronous Environments. *ACM Comput. Surv.*, 31(1):1–26, March 1999.

6 M. G. Gouda. The Triumph and Tribulation of System Stabilization. In *Proc. of the 9th International WDAG*, WDAG '95, pages 1–18, 1995.

7 N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.

8 M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In *Proc. of ICALP*, volume 1443 of *LNCS*, pages 856–867. Springer, 1998.

9 R. Milner. *communicating and mobile systems: the pi-calculus.* Cambridge UP, 1999.

10 C. Rickmann. Topological Self-Stabilization with Name-Passing Process Calculi. Master's thesis, Technische Universität Berlin, Germany, October 2015. arxiv.org/abs/1604.04197.

11 D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge UP, 2001.

12 C. Wagner and U. Nestmann. States in Process Calculi. In *Proc. of EXPRESS/SOS*, volume 160 of *EPTCS*, pages 48–62, 2014.

Linear Distances between Markov Chains*

Przemysław Daca[1], Thomas A. Henzinger[2], Jan Křetínský[3], and Tatjana Petrov[4]

1 **IST Austria, Klosterneuburg, Austria**
2 **IST Austria, Klosterneuburg, Austria**
3 **Institut für Informatik, Technische Universität München, Germany**
4 **IST Austria, Klosterneuburg, Austria**

Abstract

We introduce a general class of distances (metrics) between Markov chains, which are based on linear behaviour. This class encompasses distances given topologically (such as the total variation distance or trace distance) as well as by temporal logics or automata. We investigate which of the distances can be approximated by observing the systems, i.e. by black-box testing or simulation, and we provide both negative and positive results.

1998 ACM Subject Classification G.3 Probability and Statistics, F.4.3 Formal Languages

Keywords and phrases probabilistic systems, verification, statistical model checking, temporal logic, behavioural equivalence

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.20

1 Introduction

Behaviour of processes is traditionally compared using various notions of equivalence, such as trace equivalence, bisimulation, etc. However, the concept of equivalence is often too coarse for quantitative systems, such as Markov chains. For instance, probabilities of failures of particular hardware components are typically only empirically estimated and the slightest imprecision in the estimate may result in breaking the equivalence between processes. Moreover, if the (possibly black-box) processes are indeed different we would like to measure *how much they differ*. This has led to lifting the Boolean idea of behavioural equivalence to a finer, quantitative notion of behavioural *distance* between processes. The distance between processes s and t is typically formalized as $\sup_{p\in\mathcal{C}} |p(s) - p(t)|$ where $\mathcal{C}$ is a class of properties of interest and $p(s)$ is a quantitative value of the property p in process s [13]. This notion has been introduced in [13] for Markov chains and further developed in various settings, such as Markov decision processes [16], quantitative transition systems [12], or concurrent games [11].

Several kinds of distances have been investigated for Markov chains. On the one hand, branching distances, e.g. [1, 13, 26, 25, 4, 3, 2, 18], lift the equivalence given by the probabilistic bisimulation of Larsen and Skou [22]. On the other hand, there are *linear* distances, in particular the total variation distance [8, 6] and trace distances [20, 5]. Linear distances are particularly appropriate when (i) we are interested in linear-time properties, and (ii) we want to estimate the distance based only on simulation runs from the initial distribution of

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), the Austrian Science Fund (FWF) under grants project S11402-N23 (RiSE and SHiNE) and Z211-N23 (Wittgenstein Award), by the Czech Science Foundation Grant No. P202/12/G061, and by the SNSF Advanced Postdoc. Mobility Fellowship – grant number P300P2_161067.

the system, i.e. in a black-box setting. (Recall that for branching distances, the underlying probabilistic bisimulation corresponds to testing equivalence where not only runs from the initial distribution can be observed, but it is also possible to dump the current state of the system, and later restart the simulation from this state [22].)

In this paper, we introduce a simple framework for linear distances between Markov chains, using the formula above, where $p(s)$ is the probability of satisfying p when starting a simulation run in state s (when p is seen as a language of ω-words it is the probability to generate a trace belonging to p). We consider several classes $\mathcal{C}$ of languages of interest, characterized from several points of view, e.g. topologically, by linear-time logics, or by automata, thus rendering our framework versatile.

We investigate when a given distance can be estimated in a black-box setting, i.e. only from simulations. One of the main difficulties is that the class $\mathcal{C}$ typically includes properties with arbitrarily long horizon or even infinite-horizon properties, whereas every simulation run is necessarily finite. Note that we do not employ any simplifications such as imposed fixed horizon or discounting, typically used for obtaining efficient algorithms, e.g., [13, 26, 3], and the undiscounted setting is fundamentally more complex [25]. Since even simpler tasks are impossible for unbounded horizon in the black-box setting without any further knowledge, we assume we only know a lower bound on the minimum transition probability p_{min}. Note that knowledge of p_{min} has been justified in [10].

Our contribution is the following:

- We introduce a systematic linear-distance framework and illustrate it with several examples, including distances previously investigated in the literature.
- The main technical contributions are (i) a negative result stating that the total variation distance cannot be estimated by simulating the systems, and (ii) a positive result that the trace distance can be estimated.
- These results are further exploited to provide both negative and positive results for each of the settings where the language class is given topologically, by LTL (linear temporal logic) fragments, and by automata. We also show that the negative result on the total variation distance can be turned into a positive result if the transition probabilities have finite precision.

1.1 Related work

There are two main linear distances considered for Markov chains: the total variation distance and trace distance. Several algorithms have been proposed for both of them in the case when the Markov chains are known (white-box setting). We are not aware of any work where the distances are estimated only from simulating the systems (black-box setting).

Firstly, for the *total variation distance* in the white-box setting, [8] shows that deciding whether it equals one can be done in polynomial time, but computing it is NP-hard and not known to be decidable, however, it can be approximated; [6] considers this distance more generally for semi-Markov processes, provides a different approximation algorithm, and shows it coincides with distances based on (i) metric temporal logic, and (ii) timed automata languages.

Secondly, the *trace distance* is based on the notion of trace equivalence, which can be decided in polynomial time [15] (however, trace refinement of Markov decision processes is already undecidable [17]). Several variants of trace distance are considered in [20] where it is taken as a limit of finite-trace distances, possibly using discounting or averaging. In [5] the finite-trace distance is shown to coincide with distances based on (i) LTL, and (ii)

LTL without the U operator, i.e., only using the X operator and Boolean connectives. This distances is also shown to be NP-hard and not known to be decidable, similarly to the total variation distance. Finally, an approximation algorithm is shown (again in the white-box setting), where the over-approximates are branching-time distances, showing an interesting connection between the branching and linear distances.

In [21] the distinguishability problem is considered, i.e. given two Markov chains whether there is a monitor that reads a single sample and with high probability decides which chain produced the sequence. This is indeed possible when the total variation distance between the chains equals one, and [21] shows how to construct such monitors. In contrast, our negative results shows that it is not possible to decide with high probability whether the total variation distance equals one when the two Markov are black-box.

Linear distances have been proposed also for quantitative transition systems, e.g. [11]. Moreover, there are other useful distances based on different fundaments; for instance, the Skorokhod distance [7, 23, 14] measures the discrete differences between systems while allowing for timing distortion; Kullback-Leibler divergence [20] is useful from the information-theoretic point of view. Finally, distances have been also studied with respect to applications in linear-time model checking [24, 5].

1.2 Outline

After recalling the basic notions in Section 2, we introduce our framework and illustrate it with examples in Section 3. We define our problem formally in Section 4. In Sections 5 and 6 we provide the proofs of our technically principal negative and positive result, respectively. Section 7 extends the results in the settings of topology, logics and automata, and discusses general conditions for estimability. We conclude in Section 8.

Proofs omitted due to space constraints can be found in [9].

2 Preliminaries

We consider a finite set Ap of atomic propositions and denote $\Sigma = 2^{Ap}$.

▶ **Definition 1** (Markov chain). A *(labelled) Markov chain (MC)* is a tuple $\mathcal{M} = (S, \mathbf{P}, \mu, L)$, where

- S is a finite set of *states*,
- $\mathbf{P} : S \times S \to [0, 1]$ is a *transition* probability matrix, such that for every $s \in S$ it holds $\sum_{s' \in S} \mathbf{P}(s, s') = 1$,
- μ is an *initial* probability distribution over S,
- $L : S \to \Sigma$ is a *labelling* function.

A *run* of $\mathcal{M}$ is an infinite sequence $\rho = s_1 s_2 \cdots$ of states, such that $\mu(s_1) > 0$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 1$; we let $\rho[i]$ denote the state s_i. A *path* in $\mathcal{M}$ is a finite prefix of a run of $\mathcal{M}$. An *ω-word* is an infinite sequence $a_1 a_2 \cdots \in \Sigma^\omega$ of symbols from Σ; a *word* is a finite prefix $w \in \Sigma^*$ of an ω-word. We extend the labelling notation so that for a path $\pi \in S^k$, the projected sequence $L(\pi)$ is the word $w \in \Sigma^k$, where $w[i] = L(\pi[i])$, and the inverse map is $L^{-1}(w) = \{\pi \in S^k \mid L(\pi) = w\}$. Given a path $\pi = s_1 \cdots s_n$, we denote the *k-prefix* of π by $\pi \downarrow k = s_1 \cdots s_k$, and similarly for prefixes of words.

Each path π in $\mathcal{M}$ determines the set of runs $\mathsf{Cone}(\pi)$ consisting of all runs that start with π. To $\mathcal{M}$ we assign the probability space $(\mathsf{Runs}, \mathcal{F}, \mathbb{P}_\mathcal{M})$, where Runs is the set of all runs in $\mathcal{M}$, $\mathcal{F}$ is the σ-algebra generated by all $\mathsf{Cone}(\pi)$, and $\mathbb{P}_\mathcal{M}$ is the unique probability measure such that $\mathbb{P}_\mathcal{M}(\mathsf{Cone}(s_1 \cdots s_n)) = \mu(s_1) \cdot \prod_{i=1}^{n-1} \mathbf{P}(s_i, s_{i+1})$, where the empty product

equals 1. We will omit the subscript in $\mathbb{P}_{\mathcal{M}}$ if the Markov chain is clear from the context. Further, we write $\mathbb{P}^s_{\mathcal{M}}$ for the probability measure, where $\mu(s) = 1$ and $\mu(s') = 0$ for $s' \neq s$. Finally, we overload the notation and for a path π write $\mathbb{P}(\pi)$ meaning $\mathbb{P}(\mathsf{Cone}(\pi))$, and for a (ω)-word w, we write $\mathbb{P}(w)$ meaning $\mathbb{P}(L^{-1}(w))$.

3 Framework for Linear Distances

In this section we introduce our framework for linear distances. For $i \in \{1, 2\}$, let $\mathcal{M}_i = (S, \mathbf{P}_i, \mu_i, L)$ denote a Markov chain[1] and $(\mathsf{Runs}, \mathcal{F}, \mathbb{P}_i)$ the induced probability space. Since single runs of Markov chains typically have measure 0, we introduce linear distances using measurable sets of runs:

▶ **Definition 2** ($\mathcal{L}$-distance)**.** For a class $\mathcal{L} \subseteq \mathcal{F}$ of measurable ω-languages[2], the $\mathcal{L}$-distance $\mathsf{D}_{\mathcal{L}}$ is defined by

$$\mathsf{D}_{\mathcal{L}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{X \in \mathcal{L}} |\mathbb{P}_1(X) - \mathbb{P}_2(X)| \ .$$

Note that every $\mathsf{D}_{\mathcal{L}}$ is a pseudo-metric[3]. However, two different MCs can have distance 0, for instance, when they induce the same probability space.

The definition of $\mathcal{L}$-distances can be instantiated either (i) by a direct topological description of $\mathcal{L}$, or indirectly (ii) by a class $\mathcal{A}$ of automata inducing the class of recognized languages $\mathcal{L} = \{\mathsf{L}(A) \mid A \in \mathcal{A}\}$, or (iii) by a set of formulae $\mathfrak{L}$ of a linear-time logic inducing the languages of models $\mathcal{L} = \{\mathsf{L}(\varphi) \mid \varphi \in \mathfrak{L}\}$ where $\mathsf{L}(\varphi)$ denotes the language of ω-words satisfying the formula φ.

We now discuss several particularly interesting instantiations:

▶ **Example 3** (Total variation)**.** One extreme choice is to consider all measurable languages, resulting in the *total variation distance* $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{X \in \mathcal{F}(\Sigma)} |\mathbb{P}_1(X) - \mathbb{P}_2(X)|$.

▶ **Example 4** (Trace distances)**.** The other extreme choices are to consider (i) only the generators of $\mathcal{F}(\Sigma)$, i.e. the cones $\{w\Sigma^\omega \mid w \in \Sigma^*\}$, resulting in the *finite-trace distance* $\mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{w \in \Sigma^+} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|$; or (ii) only the elementary events, i.e. Σ^ω, resulting in the *infinite-trace distance* $\mathsf{D}_{\mathsf{IT}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{w \in \Sigma^\omega} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|$.

▶ **Example 5** (Topological distances)**.** There are many possible choices for $\mathcal{L}$ between the two extremes above, such as *clopen sets* Δ_1, which are finite unions of cones (being both closed and open), *open sets* Σ_1, which are infinite unions of cones, *closed sets* Π_1, or classes higher in the *Borel hierarchy* such as the class of ω*-regular* languages (within Δ_3), or languages given by thresholds for a *long-run average reward* (within Σ_3).

▶ **Example 6** (Automata distances)**.** The class of ω-regular languages can also be given in terms of automata, for instance by the class of all deterministic *Rabin automata* (DRA). Similarly, the closed sets Π_1 correspond to the class of deterministic Büchi automata with all states final. Further, we can restrict the class of all DRA to those of *size at most* k for a fixed $k \in \mathbb{N}$, denoting the resulting distance by $\mathsf{D}_{DRA \leq k}$.

---

[1] To avoid clutter, the chains are defined over the same state space with the same labelling, which can be w.l.o.g. achieved by their disjoint union.

[2] Formally, the measurable space of ω-languages is given by the set Σ^ω equipped with a σ-algebra $\mathcal{F}(\Sigma)$ generated by the set of cones $\{w\Sigma^\omega \mid w \in \Sigma^*\}$. This ensures, for every measurable ω-language X, that $L^{-1}(X)$ is measurable in every MC.

[3] It is symmetric, it satisfies the triangle inequality, and the distance between identical MCs is 0.

▶ **Example 7** (Logical distances). The class of ω-regular languages can also be given in terms of logic, by the monadic second-order logic (with order). Further useful choices include the *first-order logic with order*, corresponding to the star-free languages and to the *linear temporal logic* (LTL), or its fragments such as LTL with only **X** or only **F** and **G** operators etc.

▶ Remark. The introduced distances can also be considered in the discrete setting, resulting in various notions of equivalence. For instance, the *finite-trace equivalence* E_{FT} can be derived from the finite-trace distance by the following discretization:

$$\mathsf{E}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = \begin{cases} 0 & \text{if } \mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = 0 \\ 1 & \text{otherwise, i.e., } \mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) > 0. \end{cases}$$

4 Problem Statement

Linear distances can be very useful when we want to compare a black-box system with another system, e.g. a white-box specification or a black-box previous version of the system. Indeed, in such a setting we can typically obtain simulation runs of the system and we must establish a relation between the systems based on these runs only. This is in contrast with branching distances where either both systems are assumed white-box or there are strong requirements on the testing abilities, such as dumping the current state of the system, arbitrary many restarts from there, and nesting this branching arbitrarily. Therefore, we focus on the setting where we can obtain only finite prefixes of runs and we use statistics to (i) deduce information on the whole infinite runs, and (ii) estimate the distance of the systems.

For a given distance function $\mathsf{D}_{\mathcal{L}}$, the goal is to construct an almost-surely terminating algorithm that given

- any desired finite number of sampled simulation run from Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ of any desired finite length,
- lower bound $p_{\mathsf{min}} > 0$ on the minimum (non-zero) transition probability,
- confidence $\alpha \in (0, 1)$,
- interval width $\delta \in (0, 1)$,

computes an interval I such that $|I| \leq \delta$ and $\Pr[\mathsf{D}_{\mathcal{L}}(\mathcal{M}_1, \mathcal{M}_2) \in I] \geq 1 - \alpha$.

A distance function is called *estimable*, if there exists an algorithm in the above sense, and *inestimable* otherwise.

5 Inestimability: Total variation distance

We show that for the total variation distance D_{TV} there exists no "statistical" algorithm (in the above sense) which is correct for all inputs $(\mathcal{M}_1, \mathcal{M}_2, \alpha, \delta)$. Our argument consists of two steps:

1. We construct two chains such that $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}_1, \mathcal{M}_2) = 1$, namely the two MCs shown in Figure 1 (similar to [20]): one with $\tau = 0$ and the other with small $\tau > 0$.
2. We show that any potentially correct algorithm will give with high probability an incorrect output for some choice of τ, α, δ.

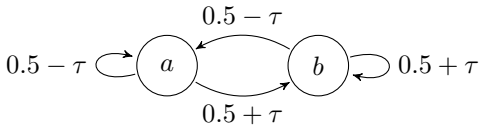


Figure 1 A Markov chain with labelling displayed in states.

Maximizing event. We start by showing that even an arbitrarily small difference in transition probabilities between two Markov chains may result in total variation distance of 1. Consider two Markov chains as in Figure 1, where $\mathcal{M}_1$ has $\tau = 0$, and $\mathcal{M}_2$ has $\tau > 0$. We assume that the initial distribution for each chain is its stationary distribution. In this setting, every simulation step is like an independent trial with probability $0.5 - \tau$ (resp. $0.5 + \tau$) of seeing a (resp. b).

Let X_n (resp. Y_n) denote the number of b symbols in a random path of length n sampled from $\mathcal{M}_1$ (resp. $\mathcal{M}_2$). By the central limit theorem the distributions of X_n and Y_n are converging to the normal distribution when $n \to \infty$:

$$X_n \approx \mathcal{N}(0.5n, 0.5^2 n) \qquad Y_n \approx \mathcal{N}((0.5+\tau)n, n(0.25-\tau^2)).$$

For $n \in \mathbb{N}$ let the event E_n mean "there is at most $c_n = (0.5 + \tau/2)n$ symbols b in the path prefix of length n." The probabilities of event E_n in the two Markov chains are:

$$\mathbb{P}_{\mathcal{M}_1}(E_n) = \mathbb{P}_{\mathcal{M}_1}(X_n \leq c_n) = \Phi(\tau\sqrt{n}) \qquad \mathbb{P}_{\mathcal{M}_1}(E_n) = \mathbb{P}_{\mathcal{M}_1}(Y_n \leq c_n) = \Phi(\frac{-0.5\tau\sqrt{n}}{\sqrt{0.25-\tau^2}}),$$

where Φ is the CDF of the standard normal distribution. For $n \to \infty$ the probability of E_n in $\mathcal{M}_1$ and $\mathcal{M}_2$ converges to 1 and 0, respectively, so the total variation distance converges to 1.

Negative result for total variation distance. Now we show that there is no statistical procedure for estimating total variation distance that would almost-surely terminate.

▶ **Theorem 8.** *For any $\delta < 1$ and $\alpha < \frac{1}{2}$, there is no algorithm for computing a $1-\alpha$ confidence interval of size δ for the total variation distance that almost-surely terminates.*

Proof. Let us write $\mathcal{M}(\tau)$ for a Markov chain in Figure 1 with the parameter τ and the initial distribution being stationary.

For $\alpha < \frac{1}{2}$ we define the following decision problem B_α:

- The input to B_α is a single path from $\mathcal{M}(\tau)$ of arbitrary length, where τ is unknown,
- The task of B_α is to output answer **Yes** with probability $\geq 1-\alpha$ if $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau))) = 1$, output answer **No** with probability $\geq 1-\alpha$ if $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau)) = 0$. Note that $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau))$ can equal only 0 or 1.

The remaining part of proof is done in two parts. In the first part, we show that there is no algorithm that solves B_α and almost-surely terminates. In the second part we reduce the problem B_α to computing a confidence interval for the total variation distance.

Part I. Suppose the opposite of the claim: that for some $\alpha < \frac{1}{2}$ there is an algorithm which solves B_α and almost-surely terminates. We represent the algorithm for solving B_α as a deterministic Turing machine TM, which works as follows:

1. The input tape of TM contains a (single) randomly sampled run of $\mathcal{M}(\tau)$,
2. TM reads a part of the run from the tape and eventually returns **Yes**/**No** answer.

The input to the TM is random, therefore we can assign a probability distribution to the computations of TM. To this end, we represent the answer of TM by the random variable $X : \mathsf{Runs} \mapsto \{\mathbf{Yes}, \mathbf{No}\}$, and we use the random variable $Y : \mathsf{Runs} \mapsto \mathbb{N} \cup \{\infty\}$ to represent the number of path symbols TM reads before terminating, where ∞ means that TM does not terminate.

Suppose we run TM on the Markov chain $\mathcal{M}(0)$. We write $\mathbb{P}_1$ for the probability measure of TM on this input. The total variation distance between the two Markov chains $\mathcal{M}(0)$ is 0, so with probability $\geq 1 - \alpha$ TM returns answer **No**, i.e. $\mathbb{P}_1(X = \mathbf{No}) \geq 1 - \alpha$.

By assumption TM almost-surely terminates on every input, so $\mathbb{P}_1(Y \in \mathbb{N}) = 1$. Let q be the following quantile:

$$q = \min\{c \in \mathbb{N} \ : \ \mathbb{P}_1(Y \leq c) \geq 0.5 + \alpha\}.$$

▶ Claim. $q \in \mathbb{N}$.

It follows that:

$$\mathbb{P}_1(X = \mathbf{No} \wedge Y \leq q) = 1 - \mathbb{P}_1(X = \mathbf{Yes} \vee Y > q) \geq 1 - \mathbb{P}_1(X = \mathbf{Yes}) - \mathbb{P}_1(Y > q) \geq 0.5. \quad (1)$$

Turing machine TM is deterministic, so if it terminates after reading prefix π of some run ρ, then it terminates after reading prefix π of any run. As a consequence, the event $Y \leq q$ can be represented as a union of ℓ cones where $\ell \leq |\Sigma|^q = 2^q$ since $\Sigma = \{a, b\}$ in $\mathcal{M}$:

$$\{\rho \ : \ Y(\rho) \leq q\} = \bigcup_{i=1}^{\ell} \mathsf{Cone}(\pi_i),$$

where all $\pi_i \in \Sigma^q$ are distinct. The event $X = \mathbf{No} \wedge Y \leq q$ is a refinement of the event $Y \leq q$, so it may also be represented as

$$\{\rho \ : \ X = \mathbf{No} \wedge Y(\rho) \leq q\} = \bigcup_{i=1}^{m} \mathsf{Cone}(\pi_i), \quad (2)$$

where $m \leq \ell \leq 2^q$. Since every path in $\mathcal{M}(0)$ of length q has probability 0.5^q, we get by (2)

$$\mathbb{P}_1(X = \mathbf{No} \wedge Y(\rho) \leq q) = \mathbb{P}_1(\bigcup_{i=1}^{m} \mathsf{Cone}(\pi_i)) = \sum_{i=1}^{m} \mathbb{P}_1(\pi_i) = m0.5^q.$$

Then by (1) it follows that $m \geq 2^{q-1}$.

Now, we run TM on the Markov chain $\mathcal{M}(\epsilon)$ where $\epsilon = 0.5 - \alpha^{\frac{1}{q}} 2^{\frac{1-q}{q}}$ if $q > 0$ and $\epsilon = 0.25$ in the degenerated case of $q = 0$.

▶ Claim. $\epsilon > 0$.

Let us write $\mathbb{P}_2$ for the probability measure of TM on the input $\mathcal{M}(\epsilon)$. The distance between $\mathcal{M}(0)$ and $\mathcal{M}(\epsilon)$ is 1, since $\epsilon > 0$. As a consequence, TM should return answer **Yes** on this input with probability $\geq 1 - \alpha$, or equivalently answer **No** with probability $< \alpha$. We show, however, that the probability of **No** is $\geq \alpha$:

$$\mathbb{P}_2(X = \mathbf{No} \wedge Y \leq q) = \sum_{i=1}^{m} \mathbb{P}_2(\pi_i) \qquad \text{by (2)}$$

$$= \sum_{i=1}^{m} (0.5+\epsilon)^{u_i}(0.5-\epsilon)^{q-u_i} \qquad u_i \text{ is number of } b\text{'s in } \pi_i$$
$$\geq \sum_{i=1}^{m} (0.5-\epsilon)^{q} = m(0.5-\epsilon)^{q}$$
$$\geq 2^{q-1}(0.5-\epsilon)^{q} = \alpha. \qquad \text{by } m \geq 2^{q-1}..$$

We obtain a contradiction, thus the assumed machine TM does not exist.

Part II. Suppose for a contradiction that for some $\alpha < \frac{1}{2}, \delta < 1$ there exists an algorithm $\mathsf{Alg}_{\alpha,\delta}$ that solves the problem defined in the theorem and almost-surely terminates. Then then this algorithm can solve the problem B_α in the following way:

1. Use $\mathsf{Alg}_{\alpha,\delta}$ to compute a confidence interval I for the total variation distance between $\mathcal{M}(0)$ and $\mathcal{M}(\tau)$. Algorithm $\mathsf{Alg}_{\alpha,\delta}$ can sample any number of paths from $\mathcal{M}(0)$. Observe that in $\mathcal{M}(\tau)$ probability of seeing states a and b remains constant over time. Thus, sampling multiple paths from $\mathcal{M}(\tau)$ by $\mathsf{Alg}_{\alpha,\delta}$ can be replaced by sampling a single path from $\mathcal{M}(\tau)$.
2. Output **Yes** if $1 \in I$, **No** if $0 \in I$.

We have shown that for any $\alpha < \frac{1}{2}$ the problem B_α cannot by solved by an algorithm that almost-surely terminates. As a consequence, the algorithm $\mathsf{Alg}_{\alpha,\delta}$ cannot exist. ◀

From Part II, it follows that there is no statistical algorithm even for fixed α and δ.

6 Estimability: Finite-trace distance

In Section 6.1 we show how to estimate the distance given by traces of a fixed length. In Section 6.2 we show how to reduce the problem of computing the finite-trace distance D_{FT} (where traces of arbitrary lengths are considered) to computing a constant number of fixed-length distances.

6.1 Estimates for fixed length

Given two Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ we wish to estimate the finite-trace distance for fixed length $k \in \mathbb{N}$

$$\mathsf{D}^k_{\mathsf{FT}} = \sup_{w \in \Sigma^k} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|.$$

There is $m = |\Sigma|^k$ words in Σ^k (we enumerate them as $w_1, \cdots, w_m$), so the traces of length k follow a multinomial distribution, i.e. for $i = 1, 2$ $\sum_{j=1}^{m} \mathbb{P}_i(w_j) = 1$.

We present a statistical procedure that estimates $\mathsf{D}^k_{\mathsf{FT}}$ with arbitrary precision. For $j \leq |\Sigma|^k$ we call a *contrast* Δ_j the difference in probabilities of trace w_j between $\mathcal{M}_1$ and $\mathcal{M}_2$: $\Delta_j = |\mathbb{P}_1(w_j) - \mathbb{P}_2(w_j)|$. The distance $\mathsf{D}^k_{\mathsf{FT}}$ is the maximum over all such contrasts $\mathsf{D}^k_{\mathsf{FT}} = \max_{j \leq m} \Delta_j$. We use the statistical procedure of [19] to simultaneously estimate all contrasts. We sample random paths from both Markov chains, and let n_i^j denote the number of observations of trace w_j in a Markov chain $\mathcal{M}_i$. We write $n_i = \sum_{j \leq m} n_i^j$ for the sum of all observations in $\mathcal{M}_i$. The estimator of $\mathbb{P}_i(w_j)$ is $\tilde{p}_i^j = \frac{n_i^j}{n_i}$, and the estimator of Δ_j is $\tilde{\Delta}_j = |\tilde{p}_1^j - \tilde{p}_2^j|$.

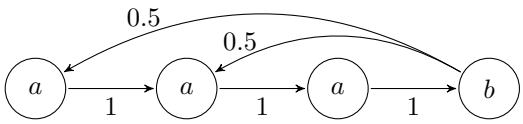


Figure 2 Markov chain with 4 states. The leftmost state is 6-deterministic, but not deterministic.

▶ **Theorem 9** ([19]). *As $n_1, n_2 \to \infty$ the probability approaches $1-\alpha$ that simultaneously for all contrasts*

$$|\Delta_j - \tilde{\Delta}_j| \leq S_j M \qquad \text{where} \qquad S_j = \sqrt{\frac{\tilde{p}_1^j - (\tilde{p}_1^j)^2}{n_1} + \frac{\tilde{p}_2^j - (\tilde{p}_2^j)^2}{n_2}},$$

and M is the square root of the $\frac{1-\alpha}{100}$ percentile of the χ^2 distribution with $|\Sigma|^k$ degrees of freedom.

The procedure for estimating $\mathsf{D}_{\mathsf{FT}}^k$ works as follows. For $\epsilon, \alpha > 0$ we sample paths from $\mathcal{M}_1$ and $\mathcal{M}_2$ until, by Theorem 9, with probability $1-\alpha$ for all contrasts $|\Delta_j - \tilde{\Delta}_j| \leq \epsilon$. Then with probability $1-\alpha$ it holds that $|\mathsf{D}_{\mathsf{FT}}^k - \max_{j \leq m} \tilde{\Delta}_j| \leq \epsilon$.

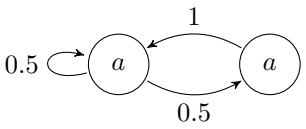
6.2 Estimates for unbounded length

Intuitively, the longer the path, the less probable it is, and the less distance it can cause. However, this is only true if along the path probabilistic choices are made repeatedly.

▶ **Definition 10.** In a Markov chain $\mathcal{M}$, a state $s \in S$ is *k-deterministic*, if there exists a word w of length k, such that $\mathbb{P}^s(w) = 1$. Otherwise, s is *k-branching*. A state $s \in S$ is *deterministic*, if it is k-deterministic for all $k \in \mathbb{N}$.

▶ **Lemma 11.** *If $s \in S$ is k-branching, it is also $(k+1)$-branching. Dually, if it is k-deterministic, it is also $(k-1)$-deterministic.*

▶ **Example 12.** Every state is trivially 1-deterministic. In Figure 3, the leftmost state is 3-deterministic and 4-branching. The states of the MC on the right are deterministic.



▶ **Lemma 13.** *Consider a state s in a Markov chain $\mathcal{M}$ with n states. If state s is n^2-deterministic, then it is deterministic.*

Before proceeding to the proof, notice that even though it may seem that every branching state must be $n+1$ branching, this is not the case in general. Observe the counterexample in Figure 2. The leftmost state is 6-deterministic (only the word *aaabaa* can be generated), while $n = 4$.

Proof. Consider state s that is n^2-deterministic and assume for contradiction that s is not deterministic. Let $N > n^2$ be the smallest number such that s is N-branching, and thus not $(N-1)$-branching. Then there exist two paths $\pi = s_1, s_2, \ldots, s_N$ and $\pi' = s_1, s'_2, \ldots, s'_N$ such that $s_1 = s$ and for $i = 1, 2, \ldots, N-1$, we have $L(s_i) = L(s'_i)$ and $L(s_N) \neq L(s'_N)$. Looking at a sequence of pairs $(s_1, s_1), (s_2, s'_2), \ldots, (s_{N-1}, s'_{N-1})$, since there are at most n^2 possible pairs of states over S, by the pigeon-hole principle at least two pairs will be

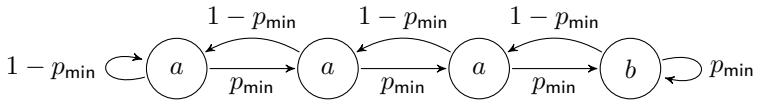


Figure 3 Markov chain, s.t. $\mathbb{P}(a) = \mathbb{P}(aa) = \mathbb{P}(aaa) = 1$, $\mathbb{P}(aaab) = p_{\mathsf{min}}^3$, $\mathbb{P}(aaaa) = 1 - p_{\mathsf{min}}^3$.

repeating in the observed sequence, say $(s_i, s_i') = (s_j, s_j')$, where $i < j$. But then the paths $\pi'' = s_1, s_2, \ldots, s_i, s_{j+1}, \ldots, s_N$ and $\pi''' = s_1, s_2, \ldots, s_i, s_{j+1}, \ldots, s_N$ have $M < N$ states and they witness that s_1 is M-branching, which by Lemma 11 is in contradiction with s being $(N-1)$-deterministic. ◀

▶ **Lemma 14.** *If a state $s \in S$ is k-branching, then any word of length k starting from s has probability at most $(1 - p_{\mathsf{min}}^{k-1})$, i.e., $\forall w \in \Sigma^k \;:\; \mathbb{P}^s(w) \leq 1 - p_{\mathsf{min}}^{k-1}$.*

To illustrate this, observe the Markov chain in Figure 3 with leftmost initial state.

Proof. Let $w \in \Sigma^k$. Since s is k-branching, there exists a word $w' \in \Sigma^k$ such that $w' \neq w$ and $\mathbb{P}^s(w') > 0$. Hence there exists at least one path with $k-1$ transitions, producing the trace w', and thus $\mathbb{P}^s(w') \geq p_{\mathsf{min}}^{k-1}$. Finally, $\mathbb{P}^s(w) \leq 1 - \mathbb{P}^s(w') \leq 1 - p_{\mathsf{min}}^{k-1}$. ◀

We show that, for estimating the finite trace distance with the required precision ϵ, it suffices to infer probabilities of the words up to some finite length k, which depends on ϵ. The idea is that paths that become deterministic before step k do not change their probability afterwards, while all other paths together have the probability bounded by ϵ.

▶ **Lemma 15.** *Let s be a n^2-deterministic state in a Markov chain $\mathcal{M}$ with n states. Then there are words u, z, such that $|z| + |u| \leq n$, $|u| \geq 1$, and $\mathbb{P}^s(zu^\omega) = 1$.*

This motivates the following definition, where $\mathsf{pref}(w)$ denotes the set of all prefixes of the (ω-)word w.

▶ **Definition 16.** A word $w \in \Sigma^+$ is called *k-ultimately periodic in a Markov chain $\mathcal{M}$* if $\mathbb{P}(w) > 0$ and there exists a word u such that $w \in \mathsf{pref}(\Sigma^k u^\omega)$ and $1 \leq |u| \leq n$, where n is the number of states in $\mathcal{M}$. ◀

Intuitively, for sufficiently long word w and large ϵ, if $\mathbb{P}(w) > \epsilon$ and w is k-ultimately periodic, then it enters within k steps a BSCC, which is bisimilar to a cycle (all transition probabilities are 1). One can also prove that this is the only way for a ω-word to achieve a probability greater than ϵ.

For a word w we write $B^k(w)$ for the set of paths that are labelled by w, have a positive probability and where all states up to step k are n^2-branching:

$$B^k(w) = \{\pi = s_1 \cdots s_{|w|} \in L^{-1}(w) \mid \mathbb{P}(\pi) > 0 \wedge \forall i \leq \min(k, |w|).\ s_i \text{ is } n^2\text{-branching}\} \ .$$

In a similar way, we write $D^k(w)$ for the set of paths that enter a (n^2-)deterministic state before step k

$$D^k(w) = \{\pi = s_1 \cdots s_{|w|} \in L^{-1}(w) \mid \mathbb{P}(\pi) > 0 \wedge \exists i \leq \min(k, |v|).\ s_i \text{ is } n^2\text{-deterministic}\} \ .$$

For any k, we can partition paths labeled by w into B^k-paths and D^k-paths:

$$\mathbb{P}(w) = \sum_{\pi \in L^{-1}(w)} \mathbb{P}(\pi) = \sum_{\pi \in B^k(w)} \mathbb{P}(\pi) + \sum_{\pi \in D^k(w)} \mathbb{P}(\pi) \ . \tag{3}$$

Now we show that the probability of B^k-paths diminishes exponentially with length k:

▶ **Lemma 17.** *Consider a Markov chain $\mathcal{M}$ with n states. For every $k \in \mathbb{N}$ and word w, if $|w| > k$ then*

$$\sum_{\pi \in B^k(w)} \mathbb{P}(\pi) \leq (1 - p_{\mathsf{min}}^{n^2})^{\lfloor \frac{k}{n^2} \rfloor} .$$

▶ **Lemma 18.** *Let w be a word in a Markov chain $\mathcal{M}$ with n states. For every $\epsilon > 0$, if $\mathbb{P}(w) > \epsilon$ and $|w| > k$ then w is k-ultimately periodic in $\mathcal{M}$, where $k = n^2 \lceil \frac{\log \epsilon}{\log(1-p_{\mathsf{min}}^{n^2})} \rceil + n$.*

Proof. Assume that $|w| > k$. We split paths labelled by w into $B^{k-n}(w)$ and $D^{k-n}(w)$ as in (3):

$$\mathbb{P}(w) = \sum_{s_1 \cdots s_{|w|} \in L^{-1}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) = \sum_{\substack{s_1 \cdots s_{|w|} \in \\ B^{k-n}(w)}} \mathbb{P}(s_1 \cdots s_{|w|}) + \sum_{\substack{s_1 \cdots s_{|w|} \in \\ D^{k-n}(w)}} \mathbb{P}(s_1 \cdots s_{|w|}) . \quad (4)$$

By Lemma 17 we get

$$\sum_{s_1 \cdots s_{|w|} \in B^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) \leq \epsilon . \quad (5)$$

Now, from the assumption $\mathbb{P}(w) > \epsilon$, (4) and (5), it follows that

$$\sum_{s_1 \cdots s_{|w|} \in D^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) > 0 .$$

This implies that there is a path $\pi = s_1 \cdots s_{|w|} \in D^{k-n}(w)$. By definition of $D^{k-n}(w)$, π has a n^2-deterministic state before step $k-n$, and w.l.o.g. let s_{k-n} be that state. By Lemma 15, every positive word from state s_{k-n} is a prefix of zu^ω for some words z, u such that $|z|+|u| \leq n$. Therefore $w \in \mathsf{pref}(yzu^\omega)$, where $y = L(s_1 \cdots s_{k-n})$, i.e. w is $|k|$-ultimately periodic. ◀

▶ **Lemma 19.** *Consider a Markov chain $\mathcal{M}$ with n states. Let w be a k-ultimately periodic word in $\mathcal{M}$, and x be a prefix of w such that $|x| > k + n$. Then*

$$\mathbb{P}(x) - \mathbb{P}(w) \leq (1 - p_{\mathsf{min}}^{n^2})^{\lfloor \frac{k-n}{n^2} \rfloor} .$$

▶ **Theorem 20.** *Consider Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ that have at most n states. For $\epsilon > 0$ it holds that*

$$|\mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) - \max_{i \leq k} \mathsf{D}^i_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2)| \leq \epsilon, \quad \textit{where} \quad k = n^2 \lceil \frac{\log \epsilon}{\log(1 - p_{\mathsf{min}}^{n^2})} \rceil + 2n.$$

Proof. We show that for any word $w \in \Sigma^+$:

$$\Big| |\mathbb{P}_1(w) - \mathbb{P}_2(w)| - |\mathbb{P}_1(w \downarrow k) - \mathbb{P}_2(w \downarrow k)| \Big| \leq \epsilon . \quad (6)$$

For $|w| \leq k$ (6) holds trivially. Suppose that $|w| \geq k$ and consider two cases.

1. If $\mathbb{P}_i(w \downarrow k) > \epsilon$, then by Lemma 18 $w \downarrow k$ is $(k-n)$-ultimately periodic. Then by Lemma 19 $\mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon$.

2. If $\mathbb{P}_i(w \downarrow k) \leq \epsilon$, then clearly $\mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon$.
Both cases can be summarised by

$$\mathbb{P}_i(w) \leq \mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon \ . \tag{7}$$

W.l.o.g assume that $\mathbb{P}_1(w) \geq \mathbb{P}_2(w)$. Then by (7)

$$\mathbb{P}_1(w \downarrow k) - \mathbb{P}_2(w \downarrow k) \geq \mathbb{P}_1(w) - \mathbb{P}_2(w) - \epsilon,$$

which implies (6). ◀

7 Consequences and Discussion

We now discuss the consequences of the (in)estimability results for several specific subclasses of ω-regular languages, captured topologically, logically, or by automata. We also remark on the estimability in case when the transition probabilities have finite precision.

7.1 Topology

Negative result for clopen sets. Note that the proof of inestimability was based on the ability to express the events E_n for any $n \in \mathbb{N}$:

$$E_n = \text{“there is at most } c_n = (0.5 + \tau/2)n \text{ symbols } b \text{ in the prefix path of length } n\text{.”}$$

Observe that each E_n can be expressed as finite union of cones, each expressing exact positions of a's and b's in the first n steps. For instance, for $\tau = 0.2$, the event E_2, “there is at most 1 symbol b in the first 2 steps,” can be described by the union $\mathsf{Cone}(aa) \cup \mathsf{Cone}(ab) \cup \mathsf{Cone}(ba)$.

Since finite unions of cones form exactly the clopen sets, the lowest class Δ_1 in the Borel hierarchy, it follows that distances based on any class in the hierarchy are inestimable.

Positive result for the infinite-trace distance. Using the result on finite-trace distance, we can prove that the infinite-trace distance D_{IT} of Example 4 is also estimable. Indeed, the distance is non-zero only due to k-ultimately periodic ω-words with positive probability. By Lemma 19 we can provide confidence intervals for these probabilities through the k-prefixes using the fixed-length distance $\mathsf{D}^k_{\mathsf{FT}}$.

7.2 Logic

Negative result for LTL. The LTL distance as in Example 7 is again inestimable since we can express the event E_n in LTL by a finite composition of operators $\mathbf{X}, \wedge, \vee$ (notably this fragment induces the same distance as LTL [5]). Indeed, for instance, for $\tau = 0.2$, the event E_{10}, “there is at most 6 symbols b in the path prefix of length 10,” is equivalent to “at least 4 symbols a in the path prefix of length n,” and it can be described by a disjunction of $\binom{10}{4}$ formulae, each determining the possible position of symbols a, resulting in a formula $(a \wedge \mathbf{X}a \wedge \mathbf{X}^2a \wedge \mathbf{X}^3a) \vee (a \wedge \mathbf{X}a \wedge \mathbf{X}^2a \wedge \mathbf{X}^4a) \vee \ldots \vee (\mathbf{X}^7a \wedge \mathbf{X}^8a \wedge \mathbf{X}^9a \wedge \mathbf{X}^{10}a)$.

Positive result for LTL(FG,GF). The distance generated by the fragment of LTL described by combining operators **FG** and **GF** and Boolean operators is estimable. Notice that the probability of the property $\varphi \equiv \mathbf{FG}\varphi'$ equals the probability of reaching a BSCC such that φ' holds in all of its states, while the probability of property $\varphi \equiv \mathbf{GF}\varphi'$ equals the probability that every BSCC contains a state which satisfies φ'. Hence, properties expressed in this fragment of LTL can be checked by inferring all BSCCs of a chain and a simple analysis of them. The statistical estimation of all BSCCs for labelled Markov chains where only the minimal transition probability is known is possible and is shown in [10].

7.3 Automata

Negative result for automata distances. For the class of all deterministic Rabin automata (DRA), the distance (as in Example 6) is inestimable. This is implied by the inestimability for clopen sets or for LTL. Further, we can also directly encode the event E_n that "at least k symbols a are observed in the path of length n" by an automaton: the DRA counts how many symbols a are seen in the prefix up to length n; this can be done with $k \cdot n$ states where the automaton is in a state $s_{k',n'}$ if and only if in the $n' \leq n$ prefix of the input word, there are $k' \leq k$ symbols a.

Positive result for fixed-size automata. When restricting to the class of DRA of size at most $k \in \mathbb{N}$, the distance $\mathsf{D}_{DRA \leq k}$ can be estimated. A naive algorithm amounts to enumerating all automata up to given size k, then applying statistical model checking to infer the probability of satisfying the automata in each of the Markov chains, and checking for which automaton the probability difference in the two chains is maximized. Statistically inferring the probability of whether a (black-box) Markov chain satisfies a property given by a DRA is a subroutine of the procedure for statistical model checking Markov chains for LTL, described in [10].

7.4 Finite Precision

When the transition probabilities have finite precision, e.g. are given by at most two decimal digits, several negative results turn positive. Finite precision allows us to learn the MCs exactly with high probability, by rounding the learnt transition probabilities to the closest multiple of the precision. Subsequently, we can approximate the distance by the algorithms applicable in the white-box setting. In case of the total variation distance, one can apply the approximation algorithm of [8]; for trace distances, the approximation algorithm of [5] is also available. In particular, for the special case of the trace equivalence E_{FT} we can leverage the fact that Markov chains are equivalent when all their traces up to length $|\mathcal{M}_1| + |\mathcal{M}_2| - 1$ have equal probability. With the assumption of finite precision one can get by sampling the exact distribution of such traces with high confidence. Note that the same algorithm can not be applied without assuming finite precision, since arbitrarily small difference in chains cannot be detected.

8 Conclusions and Future Work

We have introduced a linear-distance framework for Markov chains and considered estimating the distances in the black-box setting from simulation runs. We investigated several distances, delimiting the (in)estimability boarder for distances given topologically, logically, and by automata. As the next step, it is desirable to look for practical algorithms that would

converge fast on practical benchmarks. Another direction is to characterize the largest language for which the distance can be estimated, and, dually, the smallest language that cannot be estimated.

References

1 Alessandro Abate. Approximation metrics based on probabilistic bisimulations for general state-space Markov processes: A survey. *Electr. Notes Theor. Comput. Sci.*, 297:3–25, 2013.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. The BisimDist library: Efficient computation of bisimilarity distances for Markovian models. In *QEST*, pages 278–281, 2013.

3 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Computing behavioral distances, compositionally. In *MFCS*, pages 74–85, 2013.

4 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In *TACAS*, pages 1–15, 2013.

5 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Converging from branching to linear metrics on Markov chains. In *ICTAC*, pages 349–367, 2015.

6 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On the total variation distance of semi-Markov chains. In *FoSSaCS*, pages 185–199, 2015.

7 Paul Caspi and Albert Benveniste. Toward an approximation theory for computerised control. In *EMSOFT*, pages 294–304, 2002.

8 Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In *CSL-LICS*, pages 33:1–33:10, 2014.

9 Przemyslaw Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Linear distances between Markov chains. Technical Report abs/1605.00186, arXiv.org, 2014.

10 Przemyslaw Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. *TACAS*, pages 112–129, 2016.

11 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching metrics for quantitative transition systems. In *ICALP*, pages 97–109, 2004.

12 Luca de Alfaro, Rupak Majumdar, Vishwanath Raman, and Mariëlle Stoelinga. Game relations and metrics. In *LICS*, pages 99–108, 2007.

13 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In *CONCUR*, pages 258–273, 1999.

14 Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the Skorokhod metric. In *CAV*, pages 234–250, 2015.

15 Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008.

16 Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *AAAI*, pages 950–951, 2004.

17 Nathanaël Fijalkow, Stefan Kiefer, and Mahsa Shirmohammadi. Trace refinement in labelled Markov decision processes. In *FOSSACS*, pages 303–318, 2016.

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

19 Leo A Goodman. Simultaneous confidence intervals for contrasts among multinomial populations. *The Annals of Mathematical Statistics*, pages 716–725, 1964.

20 Manfred Jaeger, Hua Mao, Kim G. Larsen, and Radu Mardare. Continuity properties of distances for Markov processes. In *QEST*, pages 297–312, 2014.

21 Stefan Kiefer and A. Prasad Sistla. Distinguishing hidden markov chains. In *31th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, 2016.

22 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. In *POPL*, pages 344–352, 1989.

23 Rupak Majumdar and Vinayak S. Prabhu. Computing the Skorokhod distance between polygonal traces. In *HSCC*, pages 199–208, 2015.

24 Ilya Tkachev and Alessandro Abate. On approximation metrics for linear temporal model-checking of stochastic systems. In *HSCC*, pages 193–202, 2014.

25 Franck van Breugel, Babita Sharma, and James Worrell. Approximating a behavioural pseudometric without discount for probabilistic systems. In *FOSSACS*, pages 123–137, 2007.

26 Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006.

Complete Axiomatization for the Bisimilarity Distance on Markov Chains*

Giorgio Bacci[1], Giovanni Bacci[2], Kim G. Larsen[3], and Radu Mardare[4]

1 **Dept. of Computer Science, Aalborg University, Denmark.**
`grbacci@cs.aau.dk`
2 **Dept. of Computer Science, Aalborg University, Denmark.**
`giovbacci@cs.aau.dk`
3 **Dept. of Computer Science, Aalborg University, Denmark.**
`klg@cs.aau.dk`
4 **Dept. of Computer Science, Aalborg University, Denmark.**
`mardare@cs.aau.dk`

Abstract

In this paper we propose a complete axiomatization of the bisimilarity distance of Desharnais et al. for the class of finite labelled Markov chains. Our axiomatization is given in the style of a quantitative extension of equational logic recently proposed by Mardare, Panangaden, and Plotkin (LICS'16) that uses equality relations $t \equiv_\varepsilon s$ indexed by rationals, expressing that "t is approximately equal to s up to an error ε". Notably, our quantitative deductive system extends in a natural way the equational system for probabilistic bisimilarity given by Stark and Smolka by introducing an axiom for dealing with the Kantorovich distance between probability distributions.

1998 ACM Subject Classification F.3.2 Algebraic Approaches to Semantics.

Keywords and phrases Markov chains, Behavioral distances, Axiomatization.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.21

1 Introduction

A very attractive approach toward the study of the behavior of systems consists in expressing behavioral properties in an equational algebraic fashion. The attractiveness of the equational reasoning comes from the fact that one can deal with different notions of behaviors (such as non-deterministic, probabilistic, etc.) in a compositional way, by introducing new algebraic operators and their corresponding axioms as a sequence of successive refinements.

There is a well-established literature considering complete axiomatizations of several semantic theories [15, 3, 19, 4, 1, 16, 6, 18]. Amongst the aforementioned references, the studies [19, 1, 16, 18] consider operators for the definitions of recursive behaviors and offer implicational equational proof systems for probabilistic bisimulation equivalence. It is well-known that for reasoning about the behavior of probabilistic system, a notion of distance measuring the dissimilarities of two systems is preferable to that of equivalence, since the latter is not robust w.r.t. small variations of numerical values (see e.g. [7] for more details).

* Work supported by the EU 7th Framework Programme (FP7/2007-13) under Grants Agreement nr.318490 (SENSATION), nr.601148 (CASSTING), the Sino-Danish Basic Research Center IDEA4CPS funded by Danish National Research Foundation and National Science Foundation China, the ASAP Project (4181-00360) funded by the Danish Council for Independent Research, the ERC Advanced Grant LASSO, and the Innovation Fund Denmark center DiCyPS.

The first proposals for a complete axiomatization of behavioral distances for quantitative systems are due to Larsen et al. [12] and D'Argenio et al. [5], respectively axiomatizing the weighted and probabilistic bisimulation metrics. The approaches pursued in these works, however, are rather specific and based on *ad hoc* assumptions.

Recently, Mardare, Panangaden, and Plotkin [14] —with the purpose of developing a general research program for a quantitative algebraic theory of effects [17]— proposed the concepts of *quantitative equational theory* and *quantitative algebra* as models for these theories. The key idea behind their approach is to use equations of the form $t \equiv_\varepsilon s$ annotated with a rational number ε to be interpreted as "t is approximately equal to s up to an error ε". Their main result is that completeness for a quantitative theory always holds on the freely generated algebra of terms equipped with a metric that is freely induced by the axioms. Due to this result, they were able to prove completeness for many interesting axiomatizations, such as the Hausdorff, the total variation, the p-Wasserstein, and the Kantorovich metrics.

In this paper, we contribute to the quest of complete axiomatizations of behavioral metrics, by proposing a quantitative equational theory in the sense of [14] that is proved to be complete w.r.t. the bisimilarity distance of Desharnais et al. [7] for finitely presentable labelled Markov chains. The signature of operators that we consider is the one of [19], consisting of a prefix operator, a binary probabilistic choice operator, and a recursion operator. The set of axioms we use is that of barycentric algebras relative to the probabilistic choice operator and Milner's axioms for recursion [15]. To deal with the Kantorovich distance —that is the basic ingredient for the definition of the bisimilarity distance— we use the axiom (IB) from [14]. The resulting axiomatization is simpler than the one presented in [5] for probabilistic transition systems and it extends [5] by allowing recursive behaviors.

For the proof of completeness we could not apply the general proof technique of [14], since the recursion operator is not sound w.r.t. the axiom of non-expansiveness, that is required to fit within the quantitative algebraic framework of [14]. To prove completeness we needed to appeal to specific properties of the functional operator used to define the distance, namely, that it preserves infima of countable decreasing chains, a.k.a. ω-cocontinuity. Interestingly, the proof technique we use seems to be generic on the functional operator that defines the distance, provided that it is ω-cocontinuous.

Moreover, we show that the class of expressible behaviors, namely those that can be described as syntactic terms of this signature, corresponds up to bisimilarity to the class of finite and finitely supported labelled Markov chains. This establishes a strong correspondence between syntactic terms and a clearly defined semantic class of probabilistic systems.

2 Preliminaries and Notation

For $R \subseteq X \times X$ an equivalence relation, we denote by $X/_R$ its quotient set. For two sets X and Y, we denote by $X \uplus Y$ their disjoint union.

A *discrete sub-probability* on X is a function $\mu\colon X \to [0,1]$, such that $\mu(X) \leq 1$, where, for $E \subseteq X$, $\mu(E) = \sum_{x \in E} \mu(x)$; it is a *probability distribution* if $\mu(X) = 1$. The support of μ is the set $supp(\mu) = \{x \in X \mid \mu(x) > 0\}$. We denote by $\Delta(X)$ and $\mathcal{D}(X)$ the set of probability and finitely-supported sub-probability distributions on X, respectively.

A 1-bounded *pseudometric* on X is a function $d\colon X \times X \to [0,1]$ such that, for any $x, y, z \in X$, $d(x,x) = 0$, $d(x,y) = d(y,x)$ and $d(x,y) + d(y,z) \geq d(x,z)$; d is a *metric* if, in addition, $d(x,y) = 0$ implies $x = y$. The pair (X,d) is called *(pseudo)metric space*. For $n \in \mathbb{N}$, the *n-th product (pseudo)metric space* of (X,d) is defined as (X^n, d') where $d'((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \max_{i=1}^n d(x_i,y_i)$. The kernel of a (pseudo)metric d is the set $\ker(d) = \{(x,y) \mid d(x,y) = 0\}$.

3 Quantitative Algebras and their Equational Theories

We recall the notions of quantitative equational theory and quantitative algebras from [14].

Let Σ be an algebraic signature of function symbols $f\colon n \in \Sigma$ of arity $n \in \mathbb{N}$. Fix a countable set of *metavariables* X, ranged over by $x, y, z, \ldots \in X$. We denote by $\mathbb{T}(\Sigma, X)$ the set of Σ-terms freely generated over X; terms will be ranged over by $t, s, u, \ldots$ A *substitution of type* Σ is a function $\sigma\colon X \to \mathbb{T}(\Sigma, X)$ that is homomorphically extended to terms as $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$; by $\mathcal{S}(\Sigma)$ we denote the set of substitutions of type Σ.

A *quantitative equation of type* Σ is an expression of the form $t \equiv_\varepsilon s$, where $t, s \in \mathbb{T}(\Sigma, X)$ and $\varepsilon \in \mathbb{Q}_+$. Let $\mathcal{E}(\Sigma)$ denote the set of quantitative equations of type Σ and let range over its subsets by $\Gamma, \Theta, \Pi, \ldots \subseteq \mathcal{E}(\Sigma)$.

Let $\vdash \subseteq 2^{\mathcal{E}(\Sigma)} \times \mathcal{E}(\Sigma)$ be a binary relation from the powerset of $\mathcal{E}(\Sigma)$ to $\mathcal{E}(\Sigma)$. We write $\Gamma \vdash t \equiv_\varepsilon s$ if $(\Gamma, t \equiv_\varepsilon s) \in \vdash$, and $\Gamma \nvdash t \equiv_\varepsilon s$ otherwise; by $\vdash t \equiv_\varepsilon s$ we denote $\emptyset \vdash t \equiv_\varepsilon s$, and by $\Gamma \vdash \Theta$ we mean that $\Gamma \vdash t \equiv_\varepsilon s$, for all $t \equiv_\varepsilon s \in \Theta$. The relation $\vdash$ is called *quantitative deduction system of type* Σ if it satisfies the following axioms and rules

$$
\begin{aligned}
&\text{(Refl)} && \vdash t \equiv_0 t\,,\\
&\text{(Symm)} && \{t \equiv_\varepsilon s\} \vdash s \equiv_\varepsilon t\,,\\
&\text{(Triang)} && \{t \equiv_\varepsilon u, u \equiv_{\varepsilon'} s\} \vdash t \equiv_{\varepsilon+\varepsilon'} s\,,\\
&\text{(Max)} && \{t \equiv_\varepsilon s\} \vdash t \equiv_{\varepsilon+\varepsilon'} s\,, \text{ for all } \varepsilon' > 0\,,\\
&\text{(Arch)} && \{t \equiv_{\varepsilon'} s \mid \varepsilon' > \varepsilon\} \vdash t \equiv_\varepsilon s\,,\\
&\text{(NExp)} && \{t_1 =_\varepsilon s_1, \ldots, t_n =_\varepsilon s_n\} \vdash f(t_1, \ldots, t_n) \equiv_\varepsilon f(s_1, \ldots, s_n)\,, \text{ for all } f\colon n \in \Sigma\,,\\
&\text{(Subst)} && \text{If } \Gamma \vdash t \equiv_\varepsilon s, \text{ then } \sigma(\Gamma) \vdash \sigma(t) \equiv_\varepsilon \sigma(t), \text{ for all } \sigma \in \mathcal{S}(\Sigma)\,,\\
&\text{(Cut)} && \text{If } \Gamma \vdash \Theta \text{ and } \Theta \vdash t \equiv_\varepsilon s, \text{ then } \Gamma \vdash t \equiv_\varepsilon s\,,\\
&\text{(Assum)} && \text{If } t \equiv_\varepsilon s \in \Gamma, \text{ then } \Gamma \vdash t \equiv_\varepsilon s\,.
\end{aligned}
$$

where $\sigma(\Gamma) = \{\sigma(t) \equiv_\varepsilon \sigma(s) \mid t \equiv_\varepsilon s \in \Gamma\}$.

An expression of the form $\{t_1 \equiv_{\varepsilon_1} s_1, \ldots, t_n \equiv_{\varepsilon_n} s_n\} \vdash t \equiv_\varepsilon s$ —i.e., with finite set of hypotheses— is called *basic quantitative inference*. A *quantitative equational theory* is a set $\mathcal{U}$ of basic quantitative inferences closed under $\vdash$-deducibility. A set $\mathcal{A}$ of basic inferences is said to axiomatize a quantitative equational theory $\mathcal{U}$, if $\mathcal{U}$ is the smallest quantitative equational theory containing $\mathcal{A}$. A theory $\mathcal{U}$ is called *inconsistent* if $\vdash x \equiv_0 y \in \mathcal{U}$, for distinct metavariables $x, y \in X$, it is called *consistent* otherwise[1]. The models of quantitative equational theories are given by the following structures.

▶ **Definition 1** (Quantitative Algebra). A *quantitative* Σ*-algebra* is a tuple $\mathcal{A} = (A, \Sigma^{\mathcal{A}}, d^{\mathcal{A}})$, consisting of a pseudometric space $(A, d^{\mathcal{A}})$, with $d^{\mathcal{A}}\colon A \times A \to [0, \infty]$, and a set of non-expansive *interpretations* $\Sigma^{\mathcal{A}} = \{f^{\mathcal{A}}\colon A^n \to A \mid f\colon n \in \Sigma\}$ for the function symbols in Σ.

Quantitative Σ-algebras extend standard Σ-algebras with a notion of distance. Morphisms of quantitative algebras are non-expansive homomorphisms.

A quantitative algebra $\mathcal{A} = (A, \Sigma^{\mathcal{A}}, d^{\mathcal{A}})$ *satisfies* the quantitative inference $\Gamma \vdash t \equiv_\varepsilon s$, written $\Gamma \models_{\mathcal{A}} t \equiv_\varepsilon s$, if for any assignment of the meta-variables $\iota\colon X \to A$,

$$
\big(\text{for all } t' \equiv_{\varepsilon'} s' \in \Gamma,\ d^{\mathcal{A}}(\iota(t'), \iota(s')) \le \varepsilon'\big) \qquad \text{implies} \qquad d^{\mathcal{A}}(\iota(t), \iota(s)) \le \varepsilon\,,
$$

[1] Note that for an inconsistent theory $\mathcal{U}$, by Subst, we have $\vdash t \equiv_0 s \in \mathcal{U}$, for all $t, s \in \mathbb{T}(\Sigma, X)$.

where, for a term $t \in \mathbb{T}(\Sigma, X)$, $\iota(t)$ denotes the homomorphic interpretation of t in $\mathcal{A}$. A quantitative algebra $\mathcal{A}$ is said to be a *model* for a quantitative theory $\mathcal{U}$, if $\Gamma \models_{\mathcal{A}} t \equiv_\varepsilon s$, for all $\Gamma \vdash t \equiv_\varepsilon s \in \mathcal{U}$.

In [14] it is shown that any quantitative theory $\mathcal{U}$ has a universal model $\mathcal{T}_{\mathcal{U}}$ (the freely generated $\vdash$-model) satisfying exactly those quantitative equations belonging to $\mathcal{U}$. Moreover, in [14] it is proven a strong completeness theorem for quantitative equational theories $\mathcal{U}$, stating that a basic inference is satisfied by all the algebras satisfying $\mathcal{U}$ iff it belongs to $\mathcal{U}$.

Furthermore, in [14] several interesting examples of quantitative equational theories have been proposed. The one we will focus on later in this paper is the so called interpolative barycentric equational theory (see §10 in [14]).

4 The Quantitative Algebra of Probabilistic Behaviors

In this section we present the quantitative algebra of open Markov chains. Open Markov chains extend the familiar notion of discrete-time labelled Markov chain with "open" states taken from a fixed countable set $\mathcal{X}$ of names ranged over by $X, Y, Z, \ldots \in \mathcal{X}$. Names indicate states at which the behavior of the Markov chain can be extended by substitution of another Markov chain, in a way which will be made precise later.

4.1 Open Markov Chains and Bisimilarity Distance

In what follows we fix a set $\mathcal{L}$ of labels, ranged over by $a, b, c, \ldots \in \mathcal{L}$. Recall that $\mathcal{D}(M)$ denotes the set of *finitely supported* discrete sub-probability distributions over a set M.

▶ **Definition 2** (Open Markov Chain)**.** An *open Markov chain* $\mathcal{M} = (M, \tau)$ consists of a set M of *states* and a *transition probability function* $\tau \colon M \to \mathcal{D}((\mathcal{L} \times M) \uplus \mathcal{X})$.

Intuitively, if $\mathcal{M}$ is in a state $m \in M$ it moves with action $a \in \mathcal{L}$ to a state $n \in M$, with probability $\tau(m)(a, n)$ and to a name $X \in \mathcal{X}$ with probability $\tau(m)(X)$. A name $X \in \mathcal{X}$ is said to be *unguarded* in a state $m \in M$, if $\tau(m)(X) > 0$. Clearly, $\mathcal{L}$-labelled sub-probabilistic Markov chains are encoded as open Markov chains by letting $\tau(m)(\mathcal{X}) = 0$, for all $m \in M$.

A *pointed open Markov chain*, denoted by $(\mathcal{M}, m)$, is a Markov chain $\mathcal{M} = (M, \tau)$ with a distinguished *initial* state $m \in M$. We use $\mathcal{M} = (M, \tau)$ and $\mathcal{N} = (N, \theta)$ to range over open Markov chains and $(\mathcal{M}, m)$, $(\mathcal{N}, n)$ to range over the set **OMC** of pointed open Markov chains. In the following we will often refer to the constituents of $\mathcal{M}$ and $\mathcal{N}$ implicitly.

Next we recall the probabilistic bisimulation of Larsen and Skou [13].

▶ **Definition 3** (Bisimulation)**.** An equivalence relation $R \subseteq M \times M$ is a *bisimulation* on $\mathcal{M}$ if whenever $m \, R \, m'$, then, for all $a \in \mathcal{L}$, $X \in \mathcal{X}$ and $C \in M/_R$,

- $\tau(m)(X) = \tau(m')(X)$,
- $\tau(m)(\{a\} \times C) = \tau(m')(\{a\} \times C)$.

Two states $m, m' \in M$ are *bisimilar* w.r.t. $\mathcal{M}$, written $m \sim_{\mathcal{M}} m'$, if there exists a bisimulation relation on $\mathcal{M}$ relating them.

We say that two pointed open Markov chains $(\mathcal{M}, m), (\mathcal{N}, n) \in \mathbf{OMC}$ are bisimilar, written $(\mathcal{M}, n) \sim (\mathcal{N}, n)$, if m and n are bisimilar w.r.t. the disjoint union of $\mathcal{M}$ and $\mathcal{N}$, defined as expected. The bisimilarity relation $\sim \subseteq \mathbf{OMC} \times \mathbf{OMC}$ is an equivalence (see e.g. [2]).

The notion of bisimulation can be lifted to pseudometrics by means of a straightforward extension of the bisimilarity distance of Desharnais et al. [7] over open Markov chains —we refer the interested reader to [7, 22] for more details about its properties— that is based on the Kantorovich distance $\mathcal{K}(d)(\mu, \nu) = \min\left\{\int d \, \mathrm{d}\omega \mid \omega \in \Omega(\mu, \nu)\right\}$ between probability

measures $\mu, \nu \in \Delta(A)$ w.r.t. the underlying distance d on A. In the definition, $\Omega(\mu, \nu)$ is the set of *couplings* for (μ, ν), i.e., a probability distributions $\omega \in \Delta(A \times A)$ such that, for all $E \subseteq A$, $\omega(E \times A) = \mu(E)$ and $\omega(A \times E) = \nu(E)$.

▶ Remark. The definition of $\mathcal{K}(d)$ above is tailored on probability distributions, whereas in the present setting we are dealing with sub-probability distributions $\mu \in \mathcal{D}(A)$. To use $\mathcal{K}(d)$ on $\mathcal{D}(A)$ it is standard to add a bottom element $\perp$ in A, that is assumed to be at maximum distance from all elements $a \in A$, written $A_\perp$; and define $\mu^* \in \Delta(A_\perp)$ as the unique probability distribution such that, for all $E \subseteq A$, $\mu^*(E) = \mu(E)$ and, $\mu^*(\perp) = 1 - \mu(A)$. ◀

The set of 1-bounded pseudometrics over a set M ordered point-wise by $d \sqsubseteq d'$ iff for all $m, n \in M$, $d(m, n) \leq d'(m, n)$ is a complete partial order, with bottom given by the 0-constant pseudometric $\mathbf{0}$ and join being the point-wise supremum. We define the bisimilarity pseudometric $\mathbf{d}_{\mathcal{M}} \colon M \times M \to [0, 1]$ over $\mathcal{M} = (M, \tau)$ as the least fixed-point of the following functional operator on 1-bounded pseudometrics

$$\Psi_{\mathcal{M}}(d)(m, m') = \mathcal{K}(\Lambda(d))(\tau^*(m), \tau^*(m')) \qquad \text{(Kantorovich Operator)}$$

where $\Lambda(d)$ is the greatest 1-bounded pseudometric on $\big((\mathcal{L} \times M) \uplus \mathcal{X}\big)_\perp$ such that, for all $a \in \mathcal{L}$ and $t, s \in \mathbb{T}$, $\Lambda(d)((a, t), (a, s)) = d(t, s)$. Hereafter, whenever $\mathcal{M}$ is clear from the context we will simply write $\mathbf{d}$ and Ψ in place of $\mathbf{d}_{\mathcal{M}}$ and $\Psi_{\mathcal{M}}$, respectively.

The well definition of $\mathbf{d}$ is guaranteed by the first half of the next lemma and Knaster-Tarski fixed-point theorem. We also prove that Ψ is ω-continuous, i.e., it preserves suprema of countable increasing chains. Note that by this and Kleene fixed-point theorem, the bisimilarity distance can be alternatively characterized as $\mathbf{d} = \bigsqcup_{n \in \omega} \Psi^n(\mathbf{0})$.

▶ **Lemma 4.** *The operator Ψ is monotonic and ω-continuous.*

Proof. Monotonicity of Ψ follows from the monotonicity of $\mathcal{K}$ and Λ. ω-continuity follows from [21, Theorem 1] by showing that Ψ is non expansive, i.e., for all $d, d' \colon M \times M \to [0, 1]$, $\|\Psi(d') - \Psi(d)\| \leq \|d' - d\|$, where $\|f\| = \sup_x |f(x)|$ is the supremum norm. It suffices to prove that for all $d \sqsubseteq d'$ and $m, m' \in M$, $\Psi(d')(m, m') - \Psi(d)(m, m') \leq \|d' - d\|$:

$$\begin{aligned}
&\Psi(d')(m, m') - \Psi(d)(m, m') \\
&= \mathcal{K}(\Lambda(d'))(\tau^*(m), \tau^*(m')) - \mathcal{K}(\Lambda(d))(\tau^*(m), \tau^*(m')) && \text{(by def. } \Psi\text{)}
\end{aligned}$$

by choosing $\omega \in \Omega(\tau^*(m), \tau^*(m'))$ such that $\mathcal{K}(\Lambda(d))(\tau^*(m), \tau^*(m')) = \int \Lambda(d)\, \mathrm{d}\omega$,

$$\begin{aligned}
&= \mathcal{K}(\Lambda(d'))(\tau^*(m), \tau^*(m')) - \textstyle\int \Lambda(d)\, \mathrm{d}\omega \\
&\leq \textstyle\int \Lambda(d')\, \mathrm{d}\omega - \int \Lambda(d)\, \mathrm{d}\omega && \text{(by def. of } \mathcal{K}(\Lambda(d'))\text{)} \\
&= \textstyle\int (\Lambda(d') - \Lambda(d))\, \mathrm{d}\omega && \text{(by linearity)}
\end{aligned}$$

and since, for all $(\alpha, \beta) \notin E = \{((a, n), (a, n')) \mid a \in \mathcal{L}, n, n' \in M\}$, $\Lambda(d')(\alpha, \beta) = \Lambda(d)(\alpha, \beta)$,

$$\begin{aligned}
&= \textstyle\int_E (\Lambda(d') - \Lambda(d))\, \mathrm{d}\omega \\
&\leq \textstyle\int_E \|d' - d\|\, \mathrm{d}\omega && \text{(by def. } \Lambda\text{)} \\
&\leq \|d' - d\|\,. && \text{(by linearity and } \textstyle\int_E 1\, \mathrm{d}\omega \leq 1\text{)}
\end{aligned}$$

◀

The next Lemma states that $\mathbf{d}$ lifts the bisimilarity relation to a pseudometric.

▶ **Lemma 5.** $\mathbf{d}(m, m') = 0$ *iff* $m \sim m'$.

Proof. ($\Leftarrow$) We prove that $R = \{(m, m') \mid \mathbf{d}(m, m') = 0\}$ (i.e., $\ker(\mathbf{d})$) is a bisimulation. Clearly, R is an equivalence, and also $\ker(\Lambda(d))$ is so. Assume $(m, m') \in R$. By definition of Ψ, we have that $\mathcal{K}(\Lambda(\mathbf{d}))(\tau^*(m), \tau^*(m')) = 0$. By [8, Lemma 3.1], for all $\ker(\Lambda(d))$-equivalence classes $D \subseteq ((\mathcal{L} \times M) \uplus \mathcal{X})_\perp$, $\tau^*(m)(D) = \tau^*(m')(D)$. By definition of Λ, this implies that, for all $a \in \mathcal{L}$, $X \in \mathcal{X}$ and $C \in M/_R$, $\tau(m)(X) = \tau(m')(X)$ and, moreover, $\tau(m)(\{a\} \times C) = \tau(m')(\{a\} \times C)$.

($\Rightarrow$) Let $R \subseteq M \times M$ be a bisimulation on $\mathcal{M}$, and define $d_R \colon M \times M \to [0, 1]$ by $d_R(m, m') = 0$ if $(m, m') \in R$ and $d_R(m, m') = 1$ otherwise. We show that $\Psi(d_R) \sqsubseteq d_R$. If $(m, m') \notin R$, then $d_R(m, m') = 1 \geq \Psi(d_R)(m, m')$. If $(m, m') \in R$, then for all $a \in \mathcal{L}$, $X \in \mathcal{X}$ and $C \in M/_R$, $\tau(m)(X) = \tau(m')(X)$, $\tau(m)(\{a\} \times C) = \tau(m')(\{a\} \times C)$. This implies that for all $\ker(\Lambda(d_R))$-equivalence class $D \subseteq ((\mathcal{L} \times M) \uplus \mathcal{X})_\perp$, $\tau^*(m)(D) = \tau^*(m')(D)$. By [8, Lemma 3.1], we have $\mathcal{K}(\Lambda(d_R))(\tau^*(m), \tau^*(m')) = 0$. This implies that $\Psi(d_R) \sqsubseteq d_R$. Since $\sim$ is a bisimulation, $\Psi(d_\sim) \sqsubseteq d_\sim$, so that, by Tarski's fixed point theorem, $\mathbf{d} \sqsubseteq d_\sim$. By definition of $d_\sim$ and $\mathbf{d} \sqsubseteq d_\sim$, $m \sim m'$ implies $\mathbf{d}(m, m') = 0$. ◀

The definition above can be extended to the collection **OMC** of open Markov chains as $\mathbf{d}_{\mathbf{OMC}} \colon \mathbf{OMC} \times \mathbf{OMC} \to [0, 1]$ by using the bisimilarity distance on the disjoint union of their open Markov chains structures and by taking the distance between their initial states.

4.2 The Algebra of Open Markov Chains

Next we turn to simple algebra of pointed Markov chains. The signature of this algebra is defined as follows,

$$
\begin{aligned}
\Sigma = {} & \{X \colon 0 \mid X \in \mathcal{X}\} \cup && \text{(NAMES)} \\
& \{a.(\cdot) \colon 1 \mid a \in \mathcal{L}\} \cup && \text{(PREFIX)} \\
& \{+_e \colon 2 \mid e \in [0, 1]\} \cup && \text{(PROBABILISTIC CHOICE)} \\
& \{\mathsf{rec}\, X \colon 1 \mid X \in \mathcal{X}\}\,, && \text{(RECURSION)}
\end{aligned}
$$

consisting of a constant X for each name in $\mathcal{X}$; prefix $a.\cdot$ and a recursion $\mathsf{rec}\, X$ unary operators, for each $a \in \mathcal{L}$ and $X \in \mathcal{X}$; and a probabilistic choice $+_e$ binary operator for each $e \in [0, 1]$. For $t \in \mathbb{T}(\Sigma, M)$, $fn(t)$ denotes the set of free names in t, where the notions of *free* and *bound name* are defined in the standard way, with $\mathsf{rec}\, X$ acting as a binding construct. A term is *closed* if it does not contain any free variable. Throughout the paper we consider two terms as syntactically identical if they are identical up to renaming of their bound names. For $t, s_1, \ldots, s_n \in \mathbb{T}(\Sigma, M)$ and an n-vector $\overline{X} = (X_1, \ldots, X_n)$ of distinct names, $t[\overline{s}/\overline{X}]$ denotes the simultaneous *capture avoiding substitution* of X_i in t with s_i, for $i = 1, \ldots, n$. A name X is *guarded*[2] in a term t if every free occurrence of X in t occurs within a context the following forms: $a.[\cdot]$, $s +_1 [\cdot]$, or $[\cdot] +_0 s$.

Since from now on we will only refer to terms constructed over the signature Σ, we will simply write $\mathbb{T}(M)$ and $\mathbb{T}$, in place of $\mathbb{T}(\Sigma, M)$ and $\mathbb{T}(\Sigma, \emptyset)$, respectively.

Before giving the interpretation for these operations in **OMC**, we define an operator on open Markov chains, taking $\mathcal{M} = (M, \tau)$ to the open Markov chain $\mathbb{U}(\mathcal{M}) = (\mathbb{T}(M), \mu_{\mathcal{M}})$, where $\mu_{\mathcal{M}} \colon \mathbb{T}(M) \to \mathcal{D}((\mathcal{L} \times \mathbb{T}(M)) \uplus \mathcal{X})$ is defined as the least solution (over the complete

[2] This notion, coincides with the one in [19], though our definition may seem more involved due to the fact that we allow the probabilistic choice operators $+_e$ with e ranging in the closed interval $[0, 1]$.

partial order of the set of transition probability functions over $\mathbb{T}(M)$, ordered point-wise as $\tau \sqsubseteq \tau'$ iff $\tau(t)(E) \leq \tau'(t)(E)$, for all $t \in \mathbb{T}(M)$, and $E \subseteq (\mathcal{L} \times \mathbb{T}(M)) \uplus \mathcal{X})$ of the equation

$$\mu_{\mathcal{M}} = \mathcal{P}_{\mathcal{M}}(\mu_{\mathcal{M}}) \,,$$

where $\mathcal{P}_{\mathcal{M}}$ is defined by induction on $\mathbb{T}(M)$, for arbitrary transition probability functions θ over $\mathbb{T}(M)$, as follows:

$$\begin{aligned} \mathcal{P}_{\mathcal{M}}(\theta)(m) &= \tau(m) & \mathcal{P}_{\mathcal{M}}(\theta)(a.t) &= \delta_{(a,t)} \\ \mathcal{P}_{\mathcal{M}}(\theta)(X) &= \delta_X & \mathcal{P}_{\mathcal{M}}(\theta)(t +_e s) &= e\theta(t) + (1-e)\theta(s) \\ & & \mathcal{P}_{\mathcal{M}}(\theta)(\mathsf{rec}\, X.t) &= \theta(t[\mathsf{rec}\, X.t/X]) \,, \end{aligned}$$

where δ_X and $\delta_{(a,t)}$ denote the Dirac distributions pointed at $X \in \mathcal{X}$ and $(a,t) \in \mathcal{L} \times \mathbb{T}(M)$, respectively. The definition of $\mu_{\mathcal{M}}$ corresponds essentially to the transition probability of the operational semantics of probabilistic processes given by Stark and Smolka in [19]. The only difference with their definition is that $\mu_{\mathcal{M}}$ is defined over $\mathbb{T}(M)$ rather than $\mathbb{T}(\emptyset)$; and that our formulation simplifies theirs by skipping the definition of a labelled transition system. We refer the interested reader to [19] for more information on the definition of $\mu_{\mathcal{M}}$. Here we limit ourself to recalling that $\mu_{\mathcal{M}}(\mathsf{rec}\, X.X)((a,t)) = 0$, for all $a \in \mathcal{L}$ and $t \in \mathbb{T}(M)$ and $\mu_{\mathcal{M}}(\mathsf{rec}\, X.X)(Y) = 0$, for all $Y \in \mathcal{X}$, that is, $\mathsf{rec}\, X.X$ is a terminating state in $\mathbb{U}(\mathcal{M})$.

▶ **Definition 6** (Universal open Markov chain)**.** Let $\mathcal{M}_\emptyset = (\emptyset, \tau_\emptyset)$ be the open Markov chain with $\tau_\emptyset$ the empty transition function. The *universal open Markov chain* is $\mathbb{U}(\mathcal{M}_\emptyset)$.

The reason why it is called universal will be clarified later. As for now just note that $\mathbb{U}(\mathcal{M}_\emptyset)$ has $\mathbb{T}$ as the set of states, and that its transition probability function corresponds to the one defined in [19]. To ease the notation we will denote $\mathbb{U}(\mathcal{M}_\emptyset)$ as $\mathbb{U} = (\mathbb{T}, \mu_{\mathbb{T}})$.

Next we give an algebraic interpretation over $\mathbf{OMC}$ to the operations in Σ. For arbitrary $(\mathcal{M}, m), (\mathcal{N}, n) \in \mathbf{OMC}$ and $f \colon n \in \Sigma$ define $f^{\mathsf{omc}} \colon \mathbf{OMC}^n \to \mathbf{OMC}$ as follows:

$$\begin{aligned} X^{\mathsf{omc}} &= (\mathbb{U}, X) \,, & (\mathcal{M}, m) +_e^{\mathsf{omc}} (\mathcal{N}, n) &= (\mathbb{U}(\mathcal{M} \oplus \mathcal{N}), m +_e n) \,, \\ (a.(\mathcal{M}, m))^{\mathsf{omc}} &= (\mathbb{U}(\mathcal{M}), a.m) \,, & (\mathsf{rec}\, X.(M, m))^{\mathsf{omc}} &= (\mathbb{U}(\mathcal{M}^*_{X,m}), \mathsf{rec}\, X.m) \,, \end{aligned}$$

where $\mathcal{M} \oplus \mathcal{N}$ denotes the disjoint union of $\mathcal{M}$ and $\mathcal{N}$, and for $\mathcal{M} = (M, \tau)$, $\mathcal{M}^*_{X,m}$ is the open Markov chain $(M, \tau^*_{X,m})$ with transition function defined, for all $m' \in M$ and $E \subseteq (\mathcal{L} \times M) \uplus \mathcal{X}$, as $\tau^*_{X,m}(m')(E) = \tau(m')(X)\tau(m)(E \backslash \{X\}) + (1 - \tau(m')(X))\tau(m')(E \backslash \{X\})$. Intuitively, $\tau^*_{X,m}$ modifies τ by removing the name $X \in \mathcal{X}$ from the support of all $\tau(m')$ and replacing the removed probability mass with the probabilistic behavior of m.

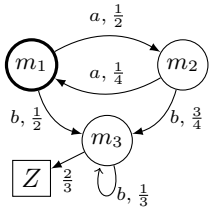
▶ **Definition 7.** The *quantitative algebra of open Markov chains* is $(\mathbf{OMC}, \Sigma^{\mathsf{omc}}, \mathbf{d}_{\mathbf{OMC}})$.

The (initial) semantics for terms $t \in \mathbb{T}$ to pointed open Markov chains is given via the function $[\![\cdot]\!] \colon \mathbb{T} \to \mathbf{OMC}$, defined by induction on terms as follows

$$\begin{aligned} [\![X]\!] &= X^{\mathsf{omc}} & [\![t +_e s]\!] &= [\![t]\!] +_e^{\mathsf{omc}} [\![s]\!] \,, \\ [\![a.t]\!] &= (a.[\![t]\!])^{\mathsf{omc}} \,, & [\![\mathsf{rec}\, X.t]\!] &= (\mathsf{rec}\, X.[\![t]\!])^{\mathsf{omc}} \,. \end{aligned} \qquad \text{(SEMANTICS)}$$

For an example of how a term is interpreted to a pointed open Markov chain see Figure 1.

Note that the freely-generated algebra of Σ-terms, namely $(\mathbb{T}, \Sigma)$ can be turned into a quantitative algebra as $\mathbb{U} = (\mathbb{T}, \Sigma, \mathbf{d}_{\mathbb{U}})$, where $\mathbf{d}_{\mathbb{U}}$ is the bisimilarity distance defined over the universal open Markov chain $\mathbb{U}$. The next result states the universality of $\mathbb{U}$.



$$t_1 = \mathsf{rec}\, X.(a.t_2 +_{\frac{1}{2}} b.t_3)$$
$$t_2 = a.X +_{\frac{1}{4}} b.t_3$$
$$t_3 = \mathsf{rec}\, Y.(b.Y +_{\frac{1}{3}} Z)$$

Figure 1 The term t_1 is interpreted to the pointed open Markov chain $[\![t_1]\!] \sim (\mathcal{M}, m_1)$ depicted on the left (restricted only to the states reachable from t_1).

▶ **Theorem 8** (Universality). *Let $t, s \in \mathbb{T}$. Then $[\![t]\!] \sim (\mathbb{U}, t)$ and $\mathbf{d}_{\mathbf{OMC}}([\![t]\!], [\![s]\!]) = \mathbf{d}_{\mathbb{U}}(t, s)$.*

Proof (sketch). The proof of $[\![t]\!] \sim (\mathbb{U}, t)$ is by induction on t. The base case is trivial. The cases for the prefix and probabilistic choice operations are completely routine from the definition of the interpretations and the operator $\mathbb{U}\colon \mathbf{OMC} \to \mathbf{OMC}$ (in each case a bisimulation can be constructed from those given by the inductive hypothesis). The only nontrivial case is when $t = \mathsf{rec}\, X.t'$. The proof carries over in two steps. First one shows that $(\mathbb{U}, \mathsf{rec}\, X.t') \sim (\mathsf{rec}\, X.(\mathbb{U}, t'))^{\mathsf{omc}}$; then, by using the inductive hypothesis $[\![t']\!] \sim (\mathbb{U}, t')$, that $(\mathsf{rec}\, X.(\mathbb{U}, t'))^{\mathsf{omc}} \sim (\mathsf{rec}\, X.[\![t']\!])^{\mathsf{omc}}$. Since $[\![\mathsf{rec}\, X.t']\!] = (\mathsf{rec}\, X.[\![t']\!])^{\mathsf{omc}}$, by transitivity of the bisimilarity relation $[\![\mathsf{rec}\, X.t']\!] \sim (\mathbb{U}, \mathsf{rec}\, X.t')$.

The proof of $\mathbf{d}_{\mathbf{OMC}}([\![t]\!], [\![s]\!]) = \mathbf{d}_{\mathbb{U}}(t, s)$ follows by Lemma 5 and the above result. Indeed

$$\begin{aligned}
\mathbf{d}_{\mathbf{OMC}}([\![t]\!], [\![s]\!]) &= \mathbf{d}([\![t]\!], [\![s]\!]) && \text{(def. } \mathbf{d}\text{)} \\
&\leq \mathbf{d}([\![t]\!], (\mathbb{U}, t)) + \mathbf{d}((\mathbb{U}, t), (\mathbb{U}, s)) + \mathbf{d}((\mathbb{U}, s), [\![s]\!]) && \text{(triangular ineq.)} \\
&= \mathbf{d}((\mathbb{U}, t), (\mathbb{U}, s)) && ([\![t]\!] \sim (\mathbb{U}, t),\ [\![s]\!] \sim (\mathbb{U}, s) \ \&\ \text{Lemma 5}) \\
&= \mathbf{d}_{\mathbb{U}}(t, s)\,. && \text{(def. } \mathbf{d}\text{)}
\end{aligned}$$

By a similar argument we also have $\mathbf{d}_{\mathbf{OMC}}([\![t]\!], [\![s]\!]) \geq \mathbf{d}_{\mathbb{U}}(t, s)$, hence the thesis. ◀

The above result states that it is totally equivalent to reason about the behavior of $[\![t]\!]$ by just considering the state corresponding to the term t in the universal model $\mathbb{U}$. Hence, due to Theorem 8, in the rest of the paper whenever we refer to the distance between two terms we will use $\mathbf{d}_{\mathbb{U}}$, often simply denoted as $\mathbf{d}$. Similarly, $\Gamma \models_{\mathbf{OMC}} t \equiv_\varepsilon s$ is equivalent to $\Gamma \models_{\mathbb{U}} t \equiv_\varepsilon s$, and it will be denoted just by $\Gamma \models t \equiv_\varepsilon s$.

▶ Remark. We already noted that the universal open Markov chain $\mathbb{U}$ corresponds to the operational semantics of probabilistic expressions given by Stark and Smolka [19]. In the light of Theorem 8, the soundness and completeness results for axiomatic equational system w.r.t. probabilistic bisimilarity over probabilistic expressions given in [19], can be moved without further efforts to the class of open Markov chains of the form $[\![t]\!]$.

5 Axiomatization of the Bisimilarity Distance

This section presents a quantitative deductive system, namely the one satisfying the axioms in Figure 2, and prove it to be sound and complete w.r.t. the bisimilarity distance $\mathbf{d}$.

The axioms (B1), (B2), (SC), (SA) are that of barycentric algebras [20], used to axiomatize probability distributions. The axioms (Unfold), (Unguard), (Fix), (Cong) are used to axiomatize recursive behaviors and correspond to those proposed by Milner [15]. All together, these axioms have been used by Stark and Smolka [19] to provide a complete axiomatization of probabilistic bisimilarity. To this set of axioms we add the axiom (IB)

$$
\begin{aligned}
&\text{(B1)} && \vdash t +_1 s \equiv_0 t\,,\\
&\text{(B2)} && \vdash t +_e t \equiv_0 t\,,\\
&\text{(SC)} && \vdash t +_e s \equiv_0 s +_{1-e} t\,,\\
&\text{(SA)} && \vdash (t +_e s) +_{e'} u \equiv_0 t +_{ee'} (s +_{\frac{e'-ee'}{1-ee'}} u)\,,\ \text{for } e, e' \in [0,1)\,,\\
&\text{(Unfold)} && \vdash \mathsf{rec}\, X.t \equiv_0 t[\mathsf{rec}\, X.t/X]\,,\\
&\text{(Unguard)} && \vdash \mathsf{rec}\, X.t +_e X \equiv_0 \mathsf{rec}\, X.t\,,\\
&\text{(Fix)} && \{s \equiv_0 t[s/X]\} \vdash s \equiv_0 \mathsf{rec}\, X.t\,,\ \text{for } X \text{ guarded in } t,\\
&\text{(Cong)} && \{t \equiv_0 s\} \vdash \mathsf{rec}\, X.t \equiv_0 \mathsf{rec}\, X.s\,,\\
&\text{(Top)} && \vdash t \equiv_1 s\,,\\
&\text{(IB)} && \{t \equiv_\varepsilon s, t' \equiv_{\varepsilon'} s'\} \vdash t +_e t' \equiv_{\varepsilon''} s +_e s'\,,\ \text{for } \varepsilon'' \geq e\varepsilon + (1-e)\varepsilon'.
\end{aligned}
$$

Figure 2 Quantitative axioms for the bisimilarity pseudometric.

of [14], that, in combination with the barycentric axioms, axiomatizes the Kantorovich distance between finitely-supported probability distributions (see §10 in [14] for more details). Finally the axiom (Top) is used to bound the distance between terms.

A significant difference w.r.t. the original framework of quantitative deductive systems of Mardare, Panangaden, and Plotkin, recalled in Section 3, is that we do not impose non-expansiveness for the operator $\mathsf{rec}\, X$ (i.e., the axiom (NExp) associated to $\mathsf{rec}\, X$ is dropped). This is replaced by the weaker axiom (Cong). The intuitive reason why (NExp) is not sound for $\mathsf{rec}\, X$ is that the recursion magnifies the differences of the behaviors of its arguments. We refer the interested reader to [9] for an exhaustive explanation of this phenomenon.

By [20, Theorem 2], any barycentric algebra has a one-to-one embedding into a convex subset of a suitable vector space. By this result, we can conveniently its elements as n-ary convex combinations of terms $t_1, \ldots, t_n \in \mathbb{T}$, as $\sum_{i=1}^n e_i \cdot t_i \in \mathbb{T}$, provided that $e_i \in [0,1]$ and $\sum_{i=1}^n e_i = 1$. We refer the reader to [19, 10, 11] for an analytic discussion of this notation.

5.1 Soundness

In this section we show the soundness of our quantitative deductive system w.r.t. the bisimilarity distance. As noticed in Remark 4.2, the soundness of the axioms already present in the deductive system of Stark and Smolka follow without further changes from [19], due to Theorem 8 and Lemma 5.

▶ **Theorem 9** (Soundness). *If* $\vdash t \equiv_\varepsilon s$ *then* $\models t \equiv_\varepsilon s$.

Proof. As usual, we must show that each axiom and rule of inference is valid. The axioms (Refl), (Symm), (Triang), (Max), and (Arch) are sound since $\mathbf{d}$ is a pseudometric (Lemma 5). The soundness of the classical logical deduction rules (Subst), (Cut), and (Assum) is immediate. By Lemma 5, the kernel of $\mathbf{d}$ is $\sim$. Hence the axioms of barycentric algebras (B1), (B2), (SC), and (SA) all along with the axioms (Unfold), (Unguard), (Cong), and (Fix) follow directly by the soundness theorem proven in [19]. The axiom (Top) is immediate consequence of the fact that $\mathbf{d}$ is 1-bounded. Note that (IB) subsumes the axiom (NExp+$_e$) —the two coincide when $\varepsilon = \varepsilon'$. It only remains to show the soundness of (NExp-pref) for the prefix

operator and (IB). To prove (NExp-pref) it suffices to show that $\mathbf{d}(t,s) \geq \mathbf{d}(a.t, a.s)$:

$$\begin{aligned}
\mathbf{d}(a.t, a.s) &= \mathcal{K}(\Lambda(\mathbf{d}))(\mu^*_{\mathbb{T}}(a.t), \mu^*_{\mathbb{T}}(a.s)) && (\mathbf{d} \text{ fixed-point \& def. } \Psi)\\
&= \mathcal{K}(\Lambda(\mathbf{d}))(\delta_{(a,t)}, \delta_{(a,s)}) && (\text{def. } \mu_{\mathbb{T}} \text{ \& } \mathcal{P}_{\mathbb{U}})\\
&= \Lambda(\mathbf{d})((a,t),(a,s)) && (\text{def. } \mathcal{K})\\
&= \mathbf{d}(t,s)\,. && (\text{def. } \Lambda)
\end{aligned}$$

Finally, the soundness of (IB) follows by $e\mathbf{d}(t,s) + (1-e)\mathbf{d}(t',s') \geq \mathbf{d}(t +_e t', s +_e s')$

$$\begin{aligned}
&e\mathbf{d}(t,s) + (1-e)\mathbf{d}(t',s')\\
&\quad= e\Psi(\mathbf{d})(t,s) + (1-e)\Psi(\mathbf{d})(t',s') && (\mathbf{d} \text{ fixed point})\\
&\quad= e\mathcal{K}(\Lambda(\mathbf{d}))(\mu^*_{\mathbb{T}}(t), \mu^*_{\mathbb{T}}(s)) + (1-e)\mathcal{K}(\Lambda(\mathbf{d}))(\mu^*_{\mathbb{T}}(t'), \mu^*_{\mathbb{T}}(s')) && (\text{def. } \Psi)
\end{aligned}$$

then, for $\omega \in \Omega(\mu^*_{\mathbb{T}}(t), \mu^*_{\mathbb{T}}(s))$ and $\omega' \in \Omega(\mu^*_{\mathbb{T}}(t'), \mu^*_{\mathbb{T}}(s'))$ optimal couplings for $\mathcal{K}(\Lambda(\mathbf{d}))$, and by noticing that $e\omega + (1-e)\omega' \in \Omega(e\mu^*_{\mathbb{T}}(t) + (1-e)\mu^*_{\mathbb{T}}(t'), e\mu^*_{\mathbb{T}}(s) + (1-e)\mu^*_{\mathbb{T}}(s'))$ we have

$$\begin{aligned}
&= e\textstyle\int \Lambda(\mathbf{d})\ \mathrm{d}\omega + (1-e)\int \Lambda(\mathbf{d})\ \mathrm{d}\omega'\\
&= \textstyle\int \Lambda(\mathbf{d})\ \mathrm{d}(e\omega + (1-e)\omega') && (\text{linearity})\\
&\geq \mathcal{K}(\Lambda(\mathbf{d}))(e\mu^*_{\mathbb{T}}(t) + (1-e)\mu_{\mathbb{T}}(t'), e\mu^*_{\mathbb{T}}(s) + (1-e)\mu_{\mathbb{T}}(s')) && (\text{def. } \mathcal{K} \text{ and above})\\
&= \mathcal{K}(\Lambda(\mathbf{d}))(\mathcal{P}_{\mathbb{U}}(\mu_{\mathbb{T}})^*(t +_e t'), \mathcal{P}_{\mathbb{U}}(\mu_{\mathbb{T}})^*(s +_e s')) && (\text{def. } \mathcal{P}_{\mathbb{U}})\\
&= \mathcal{K}(\Lambda(\mathbf{d}))(\mu^*_{\mathbb{T}}(t +_e t'), \mu^*_{\mathbb{T}}(s +_e s')) && (\mu_{\mathbb{T}} \text{ fixed-point})\\
&= \mathbf{d}(t +_e t', s +_e s') && (\text{def. } \Psi \text{ \& } \delta \text{ fixed-point})
\end{aligned}$$

The above concludes the proof. ◀

5.2 Completeness

This section is devoted to prove the completeness of our axiomatization w.r.t. the bisimilarity distance. The proof relies on the completeness theorem of the axiomatization of probabilistic bisimilarity in [19], and the one for interpolative barycentric algebras in [14].

The next theorem, due to Milner [15], and restated in the probabilistic setting by Stark and Smolka [19, Theorem 1] is essential for proving the completeness of our axiomatization.

▶ **Theorem 10** (Unique Solution of Equations). *Let $\overline{X} = (X_1, \ldots, X_k)$ and $\overline{Y} = (Y_1, \ldots, Y_h)$ be distinct names, and $\overline{t} = (t_1, \ldots, t_k)$ terms with free names in $(\overline{X}, \overline{Y})$ in which each X_i is guarded. Then there exist terms $\overline{s} = (s_1, \ldots, s_k)$ with free names in $\overline{Y}$ such that*

$$\vdash s_i \equiv_0 t[\overline{s}/\overline{X}]\,, \qquad \text{for all } i \leq k.$$

Moreover, if for some terms $\overline{u} = (u_1, \ldots, u_k)$ with free variables in $\overline{Y}$, $\vdash u_i \equiv_0 t[\overline{u}/\overline{X}]$, for all $i \leq k$, then $\vdash s_i \equiv_0 u_i$, for all $i \leq k$.

The next theorem is the equational characterization theorem of Stark and Smolka. In our formulation the statement is simpler than [19, Theorem 2] since in our axiomatization we have the unit laws for $+_1$ and $+_0$, derivable from the axioms (B1) and (SC).

▶ **Theorem 11** (Equational Characterization). *For any term t, with free names in $\overline{Y}$, there exist terms $t_1, \ldots, t_k$ with free names in $\overline{Y}$, such that $\vdash t \equiv_0 t_1$ and*

$$\vdash t_i \equiv_0 \textstyle\sum_{j=1}^{h(i)} p_{ij} \cdot s_{ij} + \sum_{j=1}^{l(i)} q_{ij} \cdot Y_{g(i,j)}\,, \qquad \text{for all } i \leq k,$$

where the terms s_{ij} and names $Y_{g(i,j)}$ are enumerated without repetitions, and s_{ij} is either $\mathsf{rec}\, X.X$ or has the form $a_{ij}.t_{f(i,j)}$.

Recall that (NExp) is not sound for the recursion operator. Nevertheless, the completeness of the axiomatization can be carried out regardless, thanks to the fact that the bisimilarity distance can alternatively be obtained as $\delta = \bigsqcap_{k\in\omega} \tilde{\Psi}^k(\mathbf{1})$, i.e., as the ω-limit of the decreasing sequence $\mathbf{1} \sqsupseteq \tilde{\Psi}(\mathbf{1}) \sqsupseteq \tilde{\Psi}^2(\mathbf{1}) \sqsupseteq \dots$ of the operator

$$\tilde{\Psi}(d)(m,m') = \begin{cases} 0 & \text{if } m \sim m', \\ \Psi(d)(m,m') & \text{otherwise.} \end{cases}$$

▶ **Lemma 12.** *The operator $\tilde{\Psi}$ is monotone and ω-cocontinuous. Moreover, $\delta = \bigsqcap_{k\in\omega} \tilde{\Psi}^k(\mathbf{1})$.*

Proof. Monotonicity and ω-cocontinuity follow similarly to Lemma 4 and [21, Theorem 1]. By ω-cocontinuity $\bigsqcap_{k\in\omega} \tilde{\Psi}^k(\mathbf{1})$ is a fixed point. By Lemma 5 and $\delta = \Psi(\delta)$, also δ is a fixed point of $\tilde{\Psi}$. We show that they coincide by proving that $\tilde{\Psi}$ has a unique fixed point.

Assume that $\tilde{\Psi}$ has two fixed points d and d' such that $d \sqsubset d'$. Define $R \subseteq M \times M$ as $m\ R\ m'$ iff $d'(m,m') - d(m,m') = \|d' - d\|$. By the assumption made on d and d' we have that $\|d' - d\| > 0$ and $R \cap \sim = \emptyset$. Consider arbitrary $m, m' \in M$ such that $m\ R\ m'$, then

$$\begin{aligned} \|d' - d\| &= d'(m,m') - d(m,m') \\ &= \tilde{\Psi}(d')(m,m') - \tilde{\Psi}(d)(m,m') && \text{(by } d = \tilde{\Psi}(d) \text{ and } d' = \tilde{\Psi}(d')\text{)} \\ &= \Psi(d')(m,m') - \Psi(d)(m,m') && \text{(by } m \not\sim m' \text{ and def. } \tilde{\Psi}\text{)} \\ &\leq \textstyle\int_E (\Lambda(d') - \Lambda(d))\, \mathrm{d}\omega\,, && \text{(as proved in Lemma 4)} \end{aligned}$$

where we recall that $E = \{((a,n),(a,n')) \mid a \in \mathcal{L}, n, n' \in M\}$.

Observe that $(\Lambda(d') - \Lambda(d))((a,n),(a,n')) = d'(n) - d(n') \leq \|d' - d\|$, for all $n, n' \in M$ and $a \in \mathcal{L}$. Since $\|d' - d\| > 0$ the inequality $\|d' - d\| \leq \int_E (\Lambda(d') - \Lambda(d))\, \mathrm{d}\omega \leq \|d' - d\|$ holds only if the support of ω is included in $E_R = \{((a,n),(a,n')) \mid a \in \mathcal{L} \text{ and } n\ R\ n'\}$. Since the argument holds for arbitrary $m, m' \in M$ such that $m\ R\ m'$, we have that R is a bisimulation, which is in contradiction with the initial assumptions. ◀

Now we are ready to prove the main result of this section.

▶ **Theorem 13** (Completeness). *If $\models t \equiv_\varepsilon s$, then $\vdash t \equiv_\varepsilon s$.*

Proof. Let $t, s \in \mathbb{T}$ and $\varepsilon \in \mathbb{Q}_+$. We have to show that if $\mathbf{d}(t,s) \leq \varepsilon$ then $\vdash t \equiv_\varepsilon s$. The case $\varepsilon \geq 1$ trivially follows by (Top) and (Max). Let $\varepsilon < 1$. By Theorem 11, there exist terms $t_1, \dots, t_k$ and $s_1, \dots, s_r$ with free names in $\overline{X}$ and $\overline{Y}$, respectively, such that $\vdash t \equiv_0 t_1$, $\vdash s \equiv_0 s_1$, and

$$\vdash t_i \equiv_0 \textstyle\sum_{j=1}^{h(i)} p_{ij} \cdot t'_{ij} + \sum_{j=1}^{l(i)} q_{ij} \cdot X_{g(i,j)}\,, \qquad \text{for all } i \leq k, \tag{1}$$

$$\vdash s_u \equiv_0 \textstyle\sum_{v=1}^{n(u)} e_{uv} \cdot s'_{uv} + \sum_{v=1}^{m(u)} d_{uv} \cdot Y_{w(u,v)}\,, \qquad \text{for all } u \leq r, \tag{2}$$

where the terms t'_{ij} (resp. s'_{uv}) and names $X_{g(i,j)}$ (resp. $Y_{w(u,v)}$) are enumerated without repetitions, and t'_{ij} (resp. s'_{uv}) have either the form $a_{ij}.t'_{f(i,j)}$ (resp. $b_{uv}.s'_{z(u,v)}$) or $\mathsf{rec}\, Z.Z$. By induction on $\alpha \in \mathbb{N}$, we prove that

$$\vdash t_i \equiv_\varepsilon s_u\,, \qquad \text{for all } i \leq k,\ u \leq r, \text{ and } \varepsilon \geq \tilde{\Psi}^\alpha(\mathbf{1})(t_i, s_u)\,. \tag{3}$$

(Base case: $\alpha = 0$) $\tilde{\Psi}^0(\mathbf{1})(t_i,s_u) = \mathbf{1}(t_i,s_u)$. Since $\mathbf{1}(t_i,s_u) = 0$ whenever $t_i = s_u$ and $\mathbf{1}(t_i,s_u) = 1$ if $t_i \neq s_u$, then (3) follows by the axioms (Refl), (Top) and (Max).
(Inductive step: $\alpha \geq 0$). Assume that (3) holds for α. We want to show $\vdash t_i \equiv_\varepsilon s_u$, for all $\varepsilon \geq \tilde{\Psi}^{\alpha+1}(\mathbf{1})(t_i,s_u)$. Since our deduction system includes the one of Stark and Smolka,

whenever $t_i \sim_{\mathbb{U}} s_u$, by completeness w.r.t. $\sim_{\mathbb{U}}$, namely [19, Theorem 3], we obtain $\vdash t_i \equiv_0 s_u$. By (Max) $\vdash t_i \equiv_\varepsilon s_u$, for all $\varepsilon \geq \tilde{\Psi}^{\alpha+1}(\mathbf{1})(t_i, s_u) = 0$. Let consider the case $t_i \not\sim_{\mathbb{U}} s_u$. By inductive hypothesis, we have that (3) holds. For each name $X_{g(i,j)}$ and $Y_{w(u,v)}$ occurring in (1) and (2), respectively, by (Top) we have $\vdash X_{g(i,j)} \equiv_1 Y_{w(u,v)}$ whenever $X_{g(i,j)} \neq Y_{w(u,v)}$, and by (Refl) we have $\vdash X_{g(i,j)} \equiv_0 Y_{w(u,v)}$ whenever $X_{g(i,j)} = Y_{w(u,v)}$. For each term $a_{ij}.t'_{f(i,j)}$ and $b_{uv}.s'_{z(u,v)}$ occurring in (1) and (2), respectively, by inductive hypothesis and (NExp) we can deduce $\vdash a_{ij}.t'_{f(i,j)} \equiv_\varepsilon b_{uv}.s'_{z(u,v)}$, for all $\varepsilon \geq \tilde{\Psi}^{\alpha}(\mathbf{1})(t'_{f(i,j)}, s'_{z(u,v)})$, whenever $a_{ij} = b_{uv}$. If $a_{ij} \neq b_{uv}$, by (Top) we get $\vdash a_{ij}.t'_{f(i,j)} \equiv_1 b_{uv}.s'_{z(u,v)}$. As for $\mathsf{rec}\, Z.Z$, by (Top) we have $\vdash \mathsf{rec}\, Z.Z \equiv_1 \beta$ for all terms $\beta \neq \mathsf{rec}\, Z.Z$ occurring in the right hand side of (1) and (2); and by (Refl) we have $\vdash \mathsf{rec}\, Z.Z \equiv_0 \mathsf{rec}\, Z.Z$.

Note that in this manner —possibly using (Max)— we have deduced $\vdash \beta \equiv_{\varepsilon'} \gamma$, for all $\varepsilon' \geq \Lambda(\tilde{\Psi}^{\alpha}(\mathbf{1}))(\beta, \gamma)$, where β and γ are arbitrary terms occurring in the right hand side of (1) and (2), respectively. Since our quantitative deductive system includes all the axioms of interpolative barycentric algebras in the sense of [14], by completeness w.r.t. the Kantorovich distance (see §10 in [14]), for all $t_i \not\sim_{\mathbb{U}} s_u$,

$$\vdash t_i \equiv_\varepsilon s_u\,, \qquad \text{for all } \varepsilon \geq \mathcal{K}(\Lambda(\tilde{\Psi}^{\alpha}(\mathbf{1})))(\mu^*_{\mathbb{T}}(t_i), \mu^*_{\mathbb{T}}(s_u)) = \tilde{\Psi}^{\alpha+1}(\mathbf{1})(t_i, s_u)\,. \tag{4}$$

By Lemma 12 and (3), applying (Arch) we have $\vdash t_i \equiv_\varepsilon s_u$, for all $\varepsilon \geq \mathbf{d}(t_i, s_u)$. By $\vdash t \equiv_0 t_1$, $\vdash s \equiv_0 s_1$, and (Triang), we deduce $\vdash t \equiv_\varepsilon s$, for all $\varepsilon \geq \mathbf{d}(t, s)$. ◀

6 The Class of Expressible Open Markov Chains

In this last section we show that the class of expressible open Markov chains corresponds up to bisimilarity to the class of finite (and finitely supported) open Markov chains. Specifically, this means that any finite open Markov chain (hence, also "closed" Markov chains) can be represented, up to bisimilarity, as Σ-terms; so that by Theorems 9 and 13 we can reason about their quantitative operational semantics in a purely algebraic way via the axiomatic system presented in Section 5[3].

A pointed Markov chain $(\mathcal{M}, m)$ is said *expressible* if there exists a term $t \in \mathbb{T}$ such that $[\![t]\!] \sim (\mathcal{M}, m)$. The next result is a corollary of Theorems 8, 10, and 9.

▶ **Corollary 14.** *If $(\mathcal{M}, m)$ is finite then it is expressible.*

Proof. We have to show that there exists $t \in \mathbb{T}$ such that $[\![t]\!] \sim (\mathcal{M}, m)$. Since the set of states $M = \{m_1, \ldots, m_k\}$ is finite and, for each $m_i \in M$, $\tau(m_i)$ is finitely supported, then the sets of unguarded names $\{Y^i_1, \ldots, Y^i_{h(i)}\} = supp(\tau(m_i)) \cap \mathcal{X}$ and labelled transitions $\{\alpha^i_1, \ldots, \alpha^i_{l(i)}\} = supp(m_i) \cap (\mathcal{L} \times M)$ of m_i are finite. Let us associate with each α^i_j a name X^i_j, for all $i \leq k$ and $j \leq l(i)$. For each $i \leq k$, we define the terms

$$t_i = \textstyle\sum_{j=1}^{l(i)} \tau(m_i)(\alpha^i_j) \cdot a^i_j.X^i_j + \sum_{j=1}^{h(i)} \tau(m_i)(Y^i_j) \cdot Y^i_j\,,$$

where $\alpha^i_j = (a^i_j, m^i_j)$, for all $i \leq k$ and $i \leq l(i)$. By Theorem 10, for $i \leq k$, there exists terms $\overline{s^i} = (s^i_1, \ldots, s^i_{l(i)})$ such that $\vdash s_i \equiv_0 t_i[\overline{s^i}/\overline{X^i}]$, so that by soundness (Theorem 9), $[\![s_i]\!] \sim [\![t_i[\overline{s^i}/\overline{X^i}]]\!]$. Hence, by Theorem 8, we have $(\mathbb{U}, s_i) \sim (\mathbb{U}, t_i[\overline{s^i}/\overline{X^i}])$.

Let $\overline{m^i} = (m^i_1, \ldots, m^i_{l(i)})$ and $\overline{X^i} = (X^i_1, \ldots, X^i_{l(i)})$, for $i \leq k$. It is a routine check to prove that the smallest equivalence relation R_i containing $\{(m_i, t_i[\overline{m^i}/\overline{X^i}]) \mid i \leq k\}$ is

[3] The results in this section can be alternatively obtained as in [18] by observing that open Markov chains are coalgebras of a quantitative functor.

a bisimulation for $(\mathcal{M}, m_i)$ and $(\mathbb{U}(\mathcal{M}), t_i[\overline{m^i}/\overline{X^i}])$, hence $(\mathcal{M}, m_i) \sim (\mathbb{U}(\mathcal{M}), t_i[\overline{m^i}/\overline{X^i}])$. Similarly, one can prove $(\mathbb{U}(\mathcal{M}), t_i[\overline{m^i}/\overline{X^i}]) \sim (\mathbb{U}, t_i[\overline{s^i}/\overline{X^i}])$ by taking the smallest equivalence relation containing $\{(t_i[\overline{m^i}/\overline{X^i}], t_i[\overline{s^i}/\overline{X^i}]) \mid i \leq k\}$ and $\{(m^i_j, s^i_j) \mid i \leq k, j \leq l(i)\}$. By transitivity of $\sim$, $(\mathcal{M}, m_i) \sim [\![s_i]\!]$, for all $i \leq k$, hence $(\mathcal{M}, m)$ is expressible. ◀

The converse (up to bisimilarity) of the above result can also be proved, and it follows as a corollary of Theorems 8, 9, and 11.

▶ **Corollary 15.** *If $(\mathcal{M}, n)$ is expressible then it is finite up-to-bisimilarity.*

Proof. Let $t \in \mathbb{T}$. We have to show that there exists $(\mathcal{M}, m) \in \mathbf{OMC}$ with a finite set of states such that $[\![t]\!] \sim (\mathcal{M}, m)$. From Theorem 11, there exist $t_1, \ldots, t_k$ with free names in $\overline{Y}$, such that $\vdash t \equiv_0 t_1$ and

$$\vdash t_i \equiv_0 \textstyle\sum_{j=1}^{h(i)} p_{ij} \cdot s_{ij} + \sum_{j=1}^{l(i)} q_{ij} \cdot Y_{g(i,j)}\,, \qquad \text{for all } i \leq k,$$

where the terms s_{ij} and names $Y_{g(i,j)}$ are enumerated without repetitions, and s_{ij} is either $\mathsf{rec}\, X.X$ or has the form $a_{ij}.t_{f(i,j)}$. Let $Z_1, \ldots, Z_k$ be fresh names distinct from $\overline{Y}$, and define t'_i as the term obtained by replacing in the right end side of the equation above each occurrence of t_i with Z_i. Then, clearly $\vdash t_i \equiv_0 t'_i[\overline{t}/\overline{Z}]$. By soundness (Theorem 9), we have that $[\![t_i]\!] \sim [\![t'_i[\overline{t}/\overline{Z}]]\!]$, so that, by Theorem 8, $(\mathbb{U}, t_i) \sim (\mathbb{U}, t'_i[\overline{t}/\overline{Z}])$.

Define $\mathcal{M} = (M, \tau)$ by setting $M = \{t_1, \ldots, t_k\}$, $m = t_1$, and, for all $i \leq k$, taking as $\tau(t_i)$ the smallest sub-probability distribution on $(\mathcal{L} \times M) \uplus \mathcal{X}$ such that $\tau(t_i)((a_{ij}, t_{f(i,j)})) = p_{ij}$ and $\tau(t_i)(Y_{g(i,e)}) = q_{ie}$, for all $i \leq k$, $j \leq h(i)$, and $e \leq l(i)$. Notice that since the equation above is without repetitions, τ is well defined. Moreover, $1 - \tau(m_i)((\mathcal{L} \times M) \uplus \mathcal{X}) = p_{iw}$ whenever there exists $w \leq h(i)$ such that $s_{iw} = \mathsf{rec}\, X.X$. It is not difficult to prove that $(\mathcal{M}, t_i) \sim (\mathbb{U}, t'_i[\overline{t}/\overline{Z}])$ (take the smallest equivalence relation containing the pairs $(t_i, t'_i[\overline{t}/\overline{Z}])$, for $i \leq k$), so that by transitivity of $\sim$, $(\mathcal{M}, t_i) \sim [\![t_i]\!]$, for all $i \leq k$. By $\vdash t \equiv_0 t_1$ and Theorem 9, we also have $[\![t]\!] \sim [\![t_1]\!]$, so that $[\![t]\!] \sim (\mathcal{M}, m)$. ◀

7 Conclusions and Future Work

In this paper we proposed a complete axiomatization for the bisimilarity distance of Desharnais et al. The axiomatic system comes as a natural generalization of the one proposed by Stark and Smolka [19] for probabilistic bisimilarity, where we only added the axiom (IB) from [14] for dealing with the Kantorovich distance. Although the use of the recursion operator does not fit the general framework of Mardare et al. [14], we believe that the proof technique employed in the present paper may be general enough to accommodate the axiomatization of other behavioral distances for probabilistic systems, such as the total variation distance. Moreover, in the light of the results in Section 6, it would be interesting to see to what extent one could approach infinitary behaviors by means of finitary ones, and how such an axiomatization would look like. These questions are left open for future work.

Acknowledgments. We thank the anonymous reviewers for their comments and suggestions. The first two authors are indebted to Lotte Legarth for the support provided and the delicious meals.

References

1 Luca Aceto, Zoltán Ésik, and Anna Ingólfsdóttir. Equational axioms for probabilistic bisimilarity. In *AMAST 2002*, LNCS, pages 239–253, 2002.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Bisimulation on Markov Processes over Arbitrary Measurable Spaces. In *Horizons of the Mind. A Tribute to Prakash Panangaden*, volume 8464 of *LNCS*, pages 76–95. Springer, 2014.

3 Jos C. M. Baeten, Jan A. Bergstra, and Scott A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Inf. Comput.*, 121(2):234–255, 1995.

4 Emanuele Bandini and Roberto Segala. Axiomatizations for probabilistic bisimulation. In *ICALP*, LNCS, pages 370–381, 2001.

5 Pedro R. D'Argenio, Daniel Gebler, and Matias D. Lee. Axiomatizing bisimulation equivalences and metrics from probabilistic SOS rules. In *FoSSaCS*, LNCS, pages 289–303, 2014.

6 Yuxin Deng and Catuscia Palamidessi. Axiomatizations for probabilistic finite-state behaviors. *Theor. Comput. Sci.*, 373(1-2):92–114, 2007.

7 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

8 Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov Decision Processes. In *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence*, UAI, pages 162–169. AUAI Press, 2004.

9 Daniel Gebler, Kim Guldstrand Larsen, and Simone Tini. Compositional metric reasoning with probabilistic process calculi. In *FoSSaCS*, LNCS, pages 230–245, 2015.

10 C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195. IEEE Computer Society, 1989.

11 Klaus Keimel and Gordon Plotkin. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science*, page ??, 2015. (to appear).

12 Kim G. Larsen, Uli Fahrenberg, and Claus R. Thrane. Metrics for weighted transition systems: Axiomatization and complexity. *Theor. Comput. Sci.*, 412(28):3358–3369, 2011.

13 Kim Guldstrand Larsen and Arne Skou. Bisimulation Through Probabilistic Testing. In *POPL*, pages 344–352, 1989.

14 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. Quantitative algebraic reasoning. In *LICS*, page ?? IEEE Computer Society, 2016. (to appear).

15 Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.

16 Michael Mislove, Joël Ouaknine, and James Worrell. Axioms for probability and nondeterminism. *Electronic Notes in Theoretical Computer Science*, 96:7–28, 2004.

17 Gordon D. Plotkin and John Power. Semantics for algebraic operations. *Electr. Notes Theor. Comput. Sci.*, 45:332–345, 2001.

18 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Quantitative kleene coalgebras. *Inf. Comput.*, 209(5):822–849, 2011.

19 Eugene W. Stark and Scott A. Smolka. A complete axiom system for finite-state probabilistic processes. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, MIT Press, pages 571–596, 2000.

20 Marshall H. Stone. Postulates for the barycentric calculus. *Annali di Matematica Pura ed Applicata*, 29(1):25–30, 1949.

21 Franck van Breugel. On behavioural pseudometrics and closure ordinals. *Inf. Process. Lett.*, 112(19):715–718, 2012.

22 Franck van Breugel and James Worrell. Towards Quantitative Verification of Probabilistic Transition Systems. In *ICALP*, volume 2076 of *LNCS*, pages 421–432, 2001.

Computing Probabilistic Bisimilarity Distances via Policy Iteration*

Qiyi Tang[1] and Franck van Breugel[2]

1 **DisCoVeri Group, Department of Electrical Engineering and Computer Science York University, Toronto, Canada**
2 **DisCoVeri Group, Department of Electrical Engineering and Computer Science York University, Toronto, Canada**

Abstract

A transformation mapping a labelled Markov chain to a simple stochastic game is presented. In the resulting simple stochastic game, each vertex corresponds to a pair of states of the labelled Markov chain. The value of a vertex of the simple stochastic game is shown to be equal to the probabilistic bisimilarity distance, a notion due to Desharnais, Gupta, Jagadeesan and Panangaden, of the corresponding pair of states of the labelled Markov chain. Bacci, Bacci, Larsen and Mardare introduced an algorithm to compute the probabilistic bisimilarity distances for a labelled Markov chain. A modification of a basic version of their algorithm for a labelled Markov chain is shown to be the policy iteration algorithm applied to the corresponding simple stochastic game. Furthermore, it is shown that this algorithm takes exponential time in the worst case.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.1.1 Models of Computation, G.3 Probability and Statistics

Keywords and phrases labelled Markov chain, simple stochastic game, probabilistic bisimilarity, pseudometric, value function, policy iteration

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.22

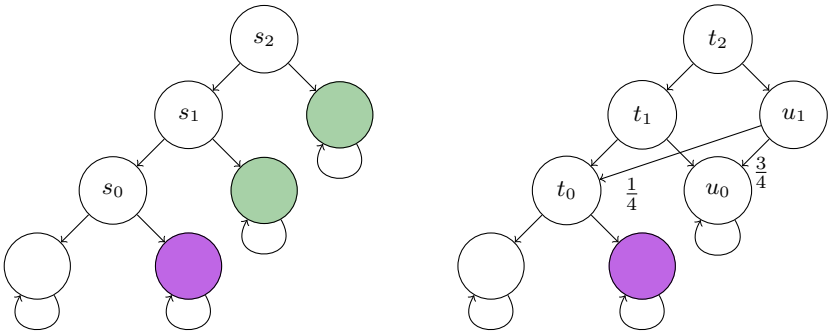
1 Introduction

A *behavioural equivalence* answers the fundamental question "when are two states of a model considered behaviourally the same?" The most prominent behavioural equivalence is *bisimilarity*, due to Milner [21] and Park [24]. We refer the reader to, for example, [25, page 1–4], for an extensive discussion of the importance of behavioural equivalences such as bisimilarity.

A *behavioural pseudometric* is a quantitative generalization of a behavioural equivalence. Such a pseudometric assigns to each pair of states a number in the unit interval $[0, 1]$. The smaller this number, the more alike the states behave. Those states that have distance zero are considered behaviourally equivalent. As first observed by Giacalone, Jou and Smolka [14], behavioural equivalences are *not robust* for models that include quantitative information such as time and probabilities. For these models, behavioural pseudometrics are an essential complement to behavioural equivalences. For a more detailed discussion of the merits of behavioural pseudometrics, we refer the reader to, for example, [23, Chapter 8].

* This research has been supported by a grant of the Natural Sciences and Engineering Research Council of Canada.

Systems with probabilistic behaviour are often modelled as *labelled Markov chains*. An example of such a Markov chain is depicted below. In a labelled Markov chain, each state has a label. In the example, the label is represented by the colour of the state. These labels are used to capture that particular properties of interest hold in some states and do not hold in other states. In diagrams, like the one below, only if the probabilities of the outgoing transitions of a state are not all the same, as is the case for state u_1, we denote the actual probabilities.



The most prominent behavioural equivalence for labelled Markov chains is *probabilistic bisimilarity*, due to Larsen and Skou [19]. Numerous quantititative generalizations of this behavioural equivalence have been proposed, the *probabilistic bisimilarity pseudometric* due to Desharnais et al. [12] being the most notable one. In this paper, we focus on this probabilistic bisimilarity pseudometric.

In order to exploit behavioural pseudometrics such as the probabilistic bisimilarity pseudometric, it is essential to be able to approximate or compute these behavioural distances. The first algorithm to approximate these distances was presented by Van Breugel, Sharma and Worrell in [5]. In their algorithm, the distance between states s and t, denoted $\delta(s,t)$, is computed as follows. Since $\delta(s,t) < q$, for some rational q, can be expressed in the existential fragment of the first order theory over the reals as shown by Van Breugel et al., and this theory is decidable as shown by Tarski [28], one can use binary search to approximate $\delta(s,t)$. The satisfiability problem for the existential fragment of the first order theory over the reals can be solved in polynomial space [7]. The algorithm of Van Breugel et al. can only handle labelled Markov chains with a handful of states.

Subsequently, Chen, Van Breugel and Worrell [8] presented a polynomial time algorithm to compute the distances. They showed that the distances are rational and that those distances can be computed by means of Khachiyan's ellipsoid method [18]. In particular, they showed that the distance function can be expressed as the solution of a linear program. In this case, the separation algorithm, which is an integral part of the ellipsoid method, boils down to solving a minimum cost flow problem. The network simplex algorithm solves the latter problem in polynomial time [22]. In practice, it is several orders of magnitude slower than the algorithm of Bacci et al. which we will discuss next.

Bacci, Bacci, Larsen and Mardare [2] put forward yet another algorithm to compute the bisimilarity distances. In their paper, they showed that their algorithm, in contrast to the two algorithms mentioned above, can handle labelled Markov chains of 50 states. Their algorithm can be viewed as a basic algorithm, enhanced with an optimization. The key idea behind this optimization is to compute the distances "on-the-fly." Roughly speaking, to compute $\delta(s,t)$ we only need to compute $\delta(u,v)$ where s and t can reach u and v in n transitions for some $n > 0$. In this paper, we will not consider this optimization, but focus on the basic algorithm only.

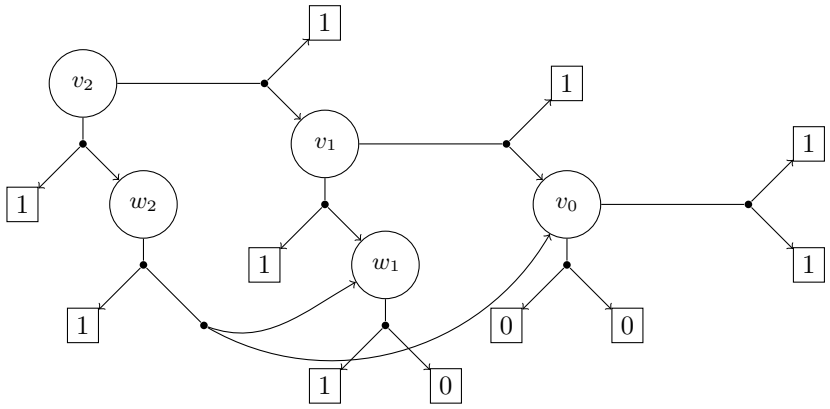
Stochastic games were introduced by Shapley [26]. A simplified version of these games, called *simple stochastic games*, were studied by Condon [9]. A simple stochastic game is played with a single token by two players, called min and max, on a finite directed graph. The graph has five types of vertices: min, max and average vertices, 0-sinks and 1-sinks. The min, max and average vertices have two outgoing edges, whereas the 0-sinks and 1-sinks have no outgoing edges. Whenever the token is in a min (max) vertex, the token is moved to one of the two successors of the vertex, chosen by the min (max) player. If the token is in an average vertex, the successor is chosen randomly. The min (max) player's objective is to minimize (maximize) the probability of reaching a 1-sink.

In its strategy, the min player chooses for each min vertex one of its two successors. Similarly, a strategy for the max player assigns to each max vertex one of its successors. Strategies are also known as *policies*. Given a strategy for the min player and a strategy for the max player, we can define a function that maps the vertices to the interval $[0, 1]$. This function maps each vertex v to the probability that the max player wins the game, provided that the game starts in vertex v and the min and max player play according to their strategies. When the strategy of both players is optimal, this function is called the *value function*.

A variety of algorithms has been developed to compute the value function of a simple stochastic game. Several of these algorithms use *policy iteration*. As long as there exists a choice in the strategy of the min or max player that is not locally optimal, switch that choice for one that is locally optimal. Hoffman and Karp [15] introduced a policy iteration algorithm for stochastic games in which all non-optimal choices are switched in each iteration. Condon [10] presented a similar algorithm, known as *simple policy iteration*, that switches only one non-optimal choice per iteration.

Most of the main results of this paper rely on a transformation mapping each labelled Markov chain to a simple stochastic game. The resulting simple stochastic game does not have any max vertices. Each min vertex v of the simple stochastic game corresponds to a pair (s, t) of states of the labelled Markov chain. In Section 4, we will show that the value of v is the distance of s and t. In [6], Van Breugel and Worrell present a similar transformation. They map a probabilistic automaton, which is a more general model as it includes not only probabilistic choices but also nondeterminstic choices, to a simple stochastic game. They also show that values and distances correspond, and use this correspondence to prove a complexity result for computing behavioural distances for probabilistic automata, but they do not present any algorithm.

Below we present (part of) the simple stochastic game corresponding to the labelled Markov chain presented earlier in this introduction. The min vertices v_i correspond to the state pairs (s_i, t_i) and the min vertices w_i correspond to the state pairs (s_{i-1}, u_{i-1}). The average vertices are denoted by bullets. The 0- and 1-sinks are labelled with zeroes and ones.



In Section 5, we will prove that a small modification of the basic algorithm of Bacci et al. corresponds to Condon's simple policy iteration. That is, the modified basic algorithm for computing the distances of all state pairs of a labelled Markov chain can be viewed as simple policy iteration applied to the corresponding simple stochastic game. As a consequence, the proof by Condon [9, Lemma 4] that simple policy iteration computes the value function of a simple stochastic game also shows that the modified basic algorithm computes the distances.

Let us highlight a technical detail here. Condon's proof that simple policy iteration computes the value function of a simple stochastic game relies on the assumption that the game halts with probability one. That is, no matter which strategy the min and max player use, the game reaches a 0- or 1-sink with probability one. To be able to use Condon's proof in our setting, we need to show that the simple stochastic game resulting from the labelled Markov chain halts with probability one. As we will see, this is accomplished by mapping state pairs (s, t) of the labelled Markov chain, for which s and t are probabilistic bisimilar, to a 0-sink in the simple stochastic game. Hence, before running the basic algorithm of Bacci et al., we first need to decide which states are probabilistic bisimilar, which can be done in polynomial time [3]. This is the small, yet essential, modification of their algorithm, to which we alluded earlier.

In Section 6, we will show that in the worst case, our algorithm takes exponential time. Many similar lower bounds have been proved for closely related algorithms by showing that the algorithms can be viewed as binary counters. We refer the reader to, for example, the thesis of Friedmann [13] for several such proofs. For simple stochastic games, Melekopoglou and Condon [20] showed that simple policy iteration takes exponential time in the worst case. We cannot directly use their result since no labelled Markov chain maps to the simple stochastic games they use in their proof. As we mentioned before, the labelled Markov chain depicted earlier gives rise to the simple stochastic game (of which a part is) depicted above. That simple stochastic game implements a 3-bit counter, as we will discuss next.

For the above simple stochastic game, a strategy of the min player consists of either going to the right (represented by 0) or down (represented by 1) in the vertices v_2, v_1 and v_0. In the table below, we present for each strategy the values of the vertices v_2, v_1 and v_0. Going from one column to the next, the strategy for either v_2, v_1 or v_0 is switched. As a result, none of the values increase and one of the values decreases. The table contains all eight 3-bit combinations and, hence, the simple stochastic game can be viewed as a 3-bit counter. As we will show in Section 6, for each $n \in \mathbb{N}$ we can construct a labelled Markov chain of size $O(n)$ that gives rise to a simple stochastic game that implements an n-bit counter. Hence, from a theoretical point of view, the algorithm of Bacci et al. is inferior to the algorithm of Chen et al.

strategy	v_2	0	1	1	0	0	1	1	0
	v_1	0	0	1	1	1	1	0	0
	v_0	0	0	0	0	1	1	1	1
value	v_2	1	$\frac{15}{16}$	$\frac{15}{16}$	$\frac{7}{8}$	$\frac{7}{8}$	$\frac{13}{16}$	$\frac{13}{16}$	$\frac{3}{4}$
	v_1	1	1	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$
	v_0	1	1	1	1	0	0	0	0

As Bacci et al. already showed in [2], in practice their algorithm outperforms the other algorithms to compute bisimilarity distances. Recall that we only consider their basic algorithm. We have shown that our modification of this algorithm also performs very well in practice. We ran the algorithm on a variety of labelled Markov chains obtained from examples of probabilistic model checkers such as PRISM[1] and MRMC[2]. The performance of our modified algorithm, which has to decide probabilistic bisimilarity first, is comparable to the performance of their basic algorithm. We expect the same results by adapting our basic algorithm to the on-the-fly setting.

2 The Probabilistic Bisimilarity Pseudometric

In this section, we review the model of interest, labelled Markov chains, and the probabilistic bisimilarity pseudometric due to Desharnais et al. [12]. We denote the set of probability distributions on a set S by $\mathit{Dist}(S)$.

▶ **Definition 1.** A labelled Markov chain is a tuple $\langle S, L, \tau, \ell \rangle$ consisting of

- a set S of states,
- a set L of labels,
- a transition function $\tau : S \to \mathit{Dist}(S)$,
- a labelling function $\ell : S \to L$.

We restrict our attention to labelled Markov chains with finitely many states and the transition probabilities of which are rationals. For the remainder of this section, we fix such a labelled Markov chain $\langle S, L, \tau, \ell \rangle$.

▶ **Definition 2.** Let $\mu, \nu \in \mathit{Dist}(S)$. The set $\Omega(\mu, \nu)$ of couplings of μ and ν is defined by

$$\Omega(\mu,\nu) = \left\{ \omega \in \mathit{Dist}(S \times S) \;\middle|\; \sum_{t \in S} \omega(s,t) = \mu(s) \wedge \sum_{s \in S} \omega(s,t) = \nu(t) \right\}.$$

Note that $\omega \in \Omega(\mu, \nu)$ is a joint probability distribution with marginals μ and ν.

▶ **Definition 3.** The function $\mathcal{T} : \mathit{Dist}(S) \times \mathit{Dist}(S) \times [0,1]^{S \times S} \to [0,1]$ is defined by

$$\mathcal{T}(\mu,\nu,d) = \min_{\omega \in \Omega(\mu,\nu)} \sum_{u,v \in S} \omega(u,v)\, d(u,v).$$

[1] `www.prismmodelchecker.org`
[2] `www.mrmc-tool.org`

The function $\mathcal{T}$ can be viewed as the minimal cost of a transportation problem. Consider two disjoint copies of S, one representing sources and the other representing targets. For each s, $t \in S$, $\mu(s)$ represents the supply at source s and $\nu(t)$ represents the demand at target t. The cost of transporting one unit from source s to target t is captured by $d(s,t)$. The amount transported from source s to target t is captured by $\omega(s,t)$. The distance function $\mathcal{T}(\mu,\nu,d)$ is known as the Kantorovich metric [17].

▶ **Definition 4.** The function $\Delta : [0,1]^{S\times S} \to [0,1]^{S\times S}$ is defined by

$$\Delta(d)(s,t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \mathcal{T}(\tau(s),\tau(t),d) & \text{otherwise} \end{cases}$$

To define the behavioural pseudometric, we use the Knaster-Tarski fixed point theorem (see, for example, [11, Chapter 2]). To apply that theorem, we need to define an order on $[0,1]^{S\times S}$. For d, $e \in [0,1]^{S\times S}$ we write $d \sqsubseteq e$ if $d(s,t) \leq e(s,t)$ for all s, $t \in S$. The set $[0,1]^{S\times S}$ endowed with the order $\sqsubseteq$ forms a complete lattice. Since Δ is a monotone function, we can conclude from the Knaster-Tarski fixed point theorem that Δ has a least fixed point. We denote this fixed point by δ. This is the probabilistic bisimilarity pseudometric of Desharnais et al.

3 An Alternative Characterization of δ

Next, we provide an alternative characterization of the probabilistic bisimilarity pseudometric δ which can be found in the extended version of [8]. This characterization can also be found in the work of Bacci et al. [2, Theorem 9]. It provides the basis for their algorithm. The characterization relies on couplings, a notion from the theory of Markov chains.

▶ **Definition 5.** The set $\mathcal{C}$ of couplings of the labelled Markov chain $\langle S, L, \tau, \ell\rangle$ is defined by

$$\mathcal{C} = \{\, T \in \mathit{Dist}(S\times S)^E \mid \forall (s,t) \in E : T(s,t) \in \Omega(\tau(s),\tau(t)) \,\}$$

where $E = \{\, (s,t) \in S \times S \mid \ell(s) = \ell(t) \,\}$.

For the remainder of this section, we fix a labelled Markov chain $\langle S, L, \tau, \ell\rangle$. We also fix a coupling $T \in \mathcal{C}$ of the labelled Markov chain. Note that $\langle S \times S, T\rangle$ can be viewed as a Markov chain, where the states $(s,t) \notin E$ are absorbing.

▶ **Definition 6.** The function $\Gamma^T : [0,1]^{S\times S} \to [0,1]^{S\times S}$ is defined by

$$\Gamma^T(d)(s,t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \displaystyle\sum_{u,v\in S} T(s,t)(u,v)\, d(u,v) & \text{otherwise} \end{cases}$$

Since $[0,1]^{S\times S}$ is a complete lattice and Γ^T is a monotone function, we can conclude from the Knaster-Tarski fixed point theorem that Γ^T has a least fixed point. We denote this fixed point by γ^T. Note that $\gamma^T(s,t)$ is the probability of reaching a state $(u,v) \notin E$ from the state (s,t) in the Markov chain $\langle S \times S, T\rangle$.

For all μ, $\nu \in \mathit{Dist}(S)$, we denote the set of vertices of the convex polytope $\Omega(\mu,\nu)$ by $V(\Omega(\mu,\nu))$. We define

$$V(\mathcal{C}) = \{\, T \in \mathit{Dist}(S\times S)^E \mid \forall (s,t) \in E : T(s,t) \in V(\Omega(\tau(s),\tau(t))) \,\}.$$

The probabilistic bisimilarity pseudometric δ can be characterized as follows.

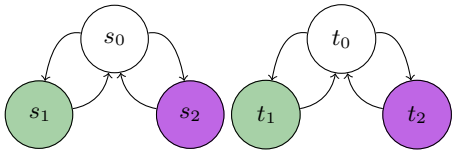
▶ **Theorem 7.** $\delta = \min_{T \in V(\mathcal{C})} \gamma^T$.

The basic algorithm of Bacci et al. relies on the fact that if a coupling T is locally optimal, that is, $\Delta(\gamma^T) = \gamma^T$, then it is globally optimal as well, that is, $\gamma^T = \delta$ (see [2, Lemma 18]). However, as we will show next, this is not the case in general.

We denote the Dirac distribution concentrated at the pair of states (s,t) by $\delta_{(s,t)}$, that is, $\delta_{(s,t)}(u,v) = 1$ if $s = u$, $t = v$ and $\delta_{(s,t)}(u,v) = 0$ otherwise.

▶ **Theorem 8.** *There exists a labelled Markov chain and $T \in V(\mathcal{C})$ such that $\Delta(\gamma^T) = \gamma^T$ and $\gamma^T \neq \delta$.*

Proof. Consider the following labelled Markov chain.



Note that

$$V(\Omega(\tau(s_0),\tau(t_0))) = \{\tfrac{1}{2}\delta_{(s_1,t_1)} + \tfrac{1}{2}\delta_{(s_2,t_2)}, \tfrac{1}{2}\delta_{(s_1,t_2)} + \tfrac{1}{2}\delta_{(s_2,t_1)}\}$$
$$V(\Omega(\tau(s_1),\tau(t_1))) = \{\delta_{(s_0,t_0)}\} \text{ and } V(\Omega(\tau(s_2),\tau(t_2))) = \{\delta_{(s_0,t_0)}\}$$

Now take T, $U \in V(\mathcal{C})$ such that

$$T(s_0,t_0) = \tfrac{1}{2}\delta_{(s_1,t_2)} + \tfrac{1}{2}\delta_{(s_2,t_1)} \text{ and } U(s_0,t_0) = \tfrac{1}{2}\delta_{(s_1,t_1)} + \tfrac{1}{2}\delta_{(s_2,t_2)}$$

Then we have

$$\gamma^T(s_0,t_0) = \gamma^T(s_1,t_1) = \gamma^T(s_2,t_2) = \gamma^T(s_1,t_2) = \gamma^T(s_2,t_1) = 1$$
$$\gamma^U(s_0,t_0) = 0$$

Furthermore,

$$\begin{aligned}\Delta(\gamma^T)(s_0,t_0) &= \min\{\tfrac{1}{2}\gamma^T(s_1,t_2) + \tfrac{1}{2}\gamma^T(s_2,t_1), \tfrac{1}{2}\gamma^T(s_1,t_1) + \tfrac{1}{2}\gamma^T(s_2,t_2)\} = 1\\ \Delta(\gamma^T)(s_1,t_1) &= \gamma^T(s_0,t_0) = 1\\ \Delta(\gamma^T)(s_2,t_2) &= \gamma^T(s_0,t_0) = 1\end{aligned}$$

◀

4 Simple Stochastic Games

In this section, we present the transformation that maps each labelled Markov chain to a simple stochastic game such that distances correspond to values. Let us first formally define simple stochastic games.

▶ **Definition 9.** A simple stochastic game is a tuple (V, E, π) consisting of

- a finite directed graph (V, E) such that
  - V is partitioned into the sets
    - $V_{\min}$ of min vertices,
    - $V_{\max}$ of max vertices,
    - V_{rnd} of random vertices,
    - V_0 of 0-sinks, and
    - V_1 of 1-sinks,

- the vertices in V_0 and V_1 have outdegree zero and all other vertices have outdegree at least one,
- a function $\pi : V_{\text{rnd}} \to \mathit{Dist}(V)$ such that for each vertex $v \in V_{\text{rnd}}$, $\pi(v)(w) > 0$ iff $(v, w) \in E$.

In this paper, we consider a slightly more general definition than the one given by Condon in [9]. In particular, the outdegree of min, max and random vertices is at least one (instead of exactly two), there may be multiple 0-sinks and 1-sinks (rather than exactly one), and the outgoing edges of a random vertex are labelled with rationals (rather than $\frac{1}{2}$). However, a simple stochastic game as defined above can be transformed in polynomial time into a simple stochastic game as defined in [9], as shown by Zwick and Paterson [29, page 355].

In the construction below, we need the notion of probabilistic bisimilarity. This notion can be captured as follows.

▶ **Definition 10.** An equivalence relation $R \subseteq S \times S$ is a probabilistic bisimulation if for all $(s, t) \in R$, $\ell(s) = \ell(t)$ and there exists $\omega \in \Omega(\tau(s), \tau(t))$ such that $\text{support}(\omega) \subseteq R$, where $\text{support}(\omega) = \{\, (u, v) \in S \times S \mid \omega(u, v) > 0 \,\}$. Probabilistic bisimilarity, denoted $\sim$, is defined as the largest probabilistic bisimulation.

In [16, Theorem 4.6] it is shown that the above characterization of probabilistic bisimilarity is equivalent to the standard definition given in [19]. Now, we are ready to introduce the transformation that maps each labelled Markov chain to a simple stochastic game.

▶ **Definition 11.** Let $\langle S, L, \tau, \ell \rangle$ be a labelled Markov chain. The simple stochastic game $\langle V, E, \pi \rangle$ is defined by

- $V_{\min} = \{\, (s, t) \in S \times S \mid s \not\sim t \wedge \ell(s) = \ell(t) \,\}$,
- $V_{\max} = \emptyset$,
- $V_{\text{rnd}} = \bigcup \{\, V(\Omega(\tau(s), \tau(t))) \mid s, t \in S \wedge s \not\sim t \wedge \ell(s) = \ell(t) \,\}$,
- $V_0 = \{\, (s, t) \in S \times S \mid s \sim t \,\}$,
- $V_1 = \{\, (s, t) \in S \times S \mid \ell(s) \neq \ell(t) \,\}$,
- $E = \{\, \langle (s, t), \omega \rangle \mid (s, t) \in V_{\min} \wedge \omega \in V(\Omega(\tau(s), \tau(t))) \,\} \cup \{\, \langle \omega, (u, v) \rangle \mid \omega(u, v) > 0 \,\}$, and
- $\pi(\omega)(u, v) = \omega(u, v)$.

Note that the resulting simple stochastic game does not have any max vertices.[3] As a result, the max player never gets to move the token. Therefore, there is no need for a strategy of the max player and, hence, we only need to consider the strategy of the min player. Recall that a strategy of the min player maps each min vertex to one of its successors. That is, such a strategy maps (s, t), with $s \not\sim t$ and $\ell(s) = \ell(t)$, to $\omega \in V(\Omega(\tau(s), \tau(t)))$. Hence, we can view $T \in V(\mathcal{C})$ as such a strategy by ignoring its values for (s, t) with $s \sim t$.

For the remainder of this section, we fix a labelled Markov chain $\langle S, L, \tau, \ell \rangle$ and a coupling $T \in V(\mathcal{C})$. The following definition of Θ^T is very similar to the definition of Γ^T given in Definition 6.

[3] The simple stochastic game of Definition 11 is a special type of simple stochastic game since it does not have any max vertices. It can also be viewed as a Markov decision process. Note, though, that it is also a special type of the Markov decision process since the reward function maps all transitions to zero apart from those that reach a 1-sink which it maps to one.

▶ **Definition 12.** The function $\Theta^T : [0,1]^{S\times S} \to [0,1]^{S\times S}$ is defined by

$$\Theta^T(d)(s,t) = \begin{cases} 0 & \text{if } s \sim t \\ 1 & \text{if } \ell(s) \neq \ell(t) \\ \sum_{u,v\in S} T(s,t)(u,v)\, d(u,v) & \text{otherwise} \end{cases}$$

Since $[0,1]^{S\times S}$ is a complete lattice and Θ^T is a monotone function, we can conclude from the Knaster-Tarski fixed point theorem that Θ^T has a least fixed point. We denote this fixed point by θ^T.

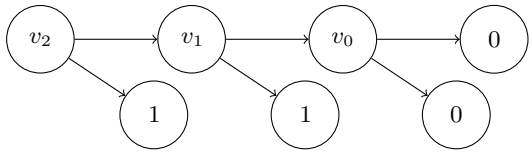
Note that $\theta^T(s,t)$ is the value of the min vertex (s,t) with respect to the strategy T. As we will show next, the optimal strategy gives rise to the probabilistic bisimilarity pseudometric δ, that is, the probabilistic bisimilarity distances of the states s and t of the labelled Markov chain are the values of the vertices (s,t) of the corresponding simple stochastic game.

▶ **Theorem 13.** $\delta = \min_{T\in V(\mathcal{C})} \theta^T$.

5 Simple Policy Iteration

As we already mentioned in the introduction, Condon's simple policy iteration algorithm computes the values of a simple stochastic game, provided that the simple stochastic game halts with probability one. As we have shown in the previous section, the probabilistic bisimilarity distances of a labelled Markov chain are the values of the corresponding simple stochastic game defined in Definition 11. Hence, if that simple stochastic game halts with probability one, then we can use simple policy iteration to compute the probabilistic bisimilarity distances.

A simple stochastic game and a pair of strategies for the min and max player give naturally rise to a Markov chain, where the vertices of the simple stochastic game are the states of the Markov chain (see, for example, [9, Section 2.1] for details). We will call this the coupled Markov chain. For example, the coupled Markov chain induced by the simple stochastic game presented in the introduction, where the strategy for the min player (the values of the vertices v_2, v_1 and v_0) is 001, is given below.



A simple stochastic game halts with probability one if for each pair of strategies, in the coupled Markov chain each state reaches a 0-sink or 1-sink with probability one.

▶ **Theorem 14.** *The simple stochastic game defined in Definition 11 halts with probability one.*

Proof. Towards a contradiction, assume that the simple stochastic game does not halt with probability one. Then there exists a strategy for the min player, that is, an element $T \in V(\mathcal{C})$, and a vertex (s,t) which does not reach a 0- or 1-sink with probability one in the coupled Markov chain. Each state in a Markov chain reaches with probability one a closed communication class, also known as a bottom strongly connected component (see, for example, [4, Theorem 10.27]). Note that a 0-sink and a 1-sink each forms a closed

communication class. Since (s,t) does not reach a 0- or 1-sink with probability one, (s,t) reaches a closed communication class C not consisting of a 0- or 1-sink. We can also show that for each $(s,t) \in C$, we have $s \sim t$. However, in that case, (s,t) is a 0-sink, which contradicts the fact that C does not consist of a 0- or 1-sink.

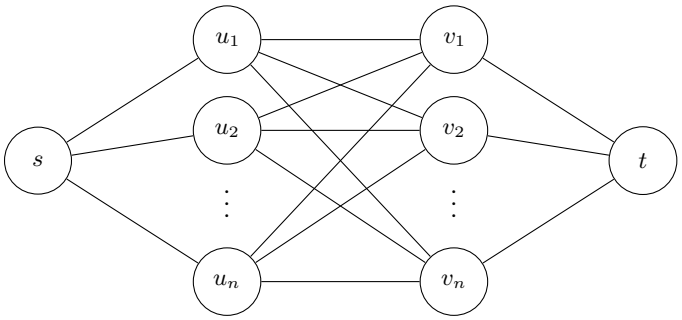
◀

Simple policy iteration starts with an arbitrary strategy, that is, an arbitrary $T \in V(\mathcal{C})$ (see line 1). As long as there is a min vertex which is not locally optimal with respect to the current strategy, the strategy at that vertex is improved to the locally optimal choice. Note that a min vertex (s,t) is not locally optimal if there exists a different choice for that vertex, that is, $\omega \in V(\Omega(\tau(s),\tau(t))$, so that the value of the vertex decreases. This is captured in line 2. In line 3, we compute a locally optimal choice and update the strategy.

1 $T \leftarrow$ `an element of` $V(\mathcal{C})$
2 `while` $\exists (s,t) \in V_{\min} : \theta^T(s,t) > \Delta(\theta^T)(s,t)$
3 $\quad T(s,t) \leftarrow \underset{\omega \in V(\Omega(\tau(s),\tau(t)))}{\arg\min} \sum_{u,v \in S} \omega(u,v)\, \theta^T(u,v)$

This is our modification of the basic algorithm of Bacci et al.

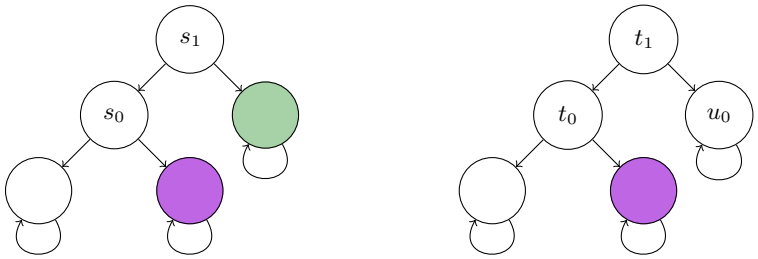
In line 1, an initial strategy $T \in V(\mathcal{C})$ can be computed by the North-West corner method in polynomial time (see, for example, [27, page 180]). In line 2, rather than choosing an arbitrary min vertex that is not locally optimal, in the simple policy iteration a select procedure can be defined as follows: we number all the vertices in $V_{\min}$ and select the one with the highest number [20, Section 2]. Note that θ^T can be computed in polynomial time [4, Section 10.1.1]. In line 3, the computation can be viewed as a minimum-cost flow problem, where s is the source vertex and t is the sink vertex. Below we present the flow network, the sets $\{u_1, \cdots, u_n\}$ and $\{v_1, \cdots, v_n\}$ are copies of S. For the edge (s,u_i) and (v_j,t), the capacity is $\tau(s)(u_i)$ and $\tau(t)(v_j)$ respectively. There is no cost transporting along these edges. Each edge $(u_i,v_j) \in S \times S$ has a capacity of $\min(\tau(s)(u_i), \tau(t)(v_j))$ and $\theta^T(u_i,v_j)$ is the cost of edge (u_i,v_j). The minimum cost of transporting one unit from s to t is captured by $\Delta(\theta^T)(s,t)$, which can be solved using the network simplex algorithm and is strongly polynomial time [1, Section 11.8]. Note that each piece is polynomial so that the complexity of the algorithm is determined by the number of iterations, as we will see in the next section.



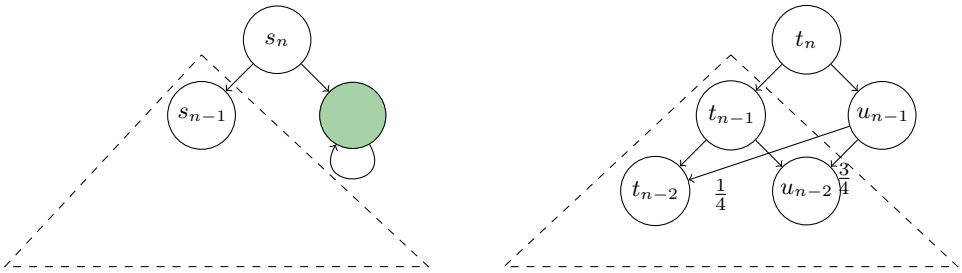
6 An Exponential Lower Bound

Below, we will prove an exponential lower bound for the algorithm we presented in the previous section. In particular, for each $n \in \mathbb{N}$ we will construct a labelled Markov chain of size $O(n)$. Furthermore, we will show that our simple policy iteration algorithm takes $\Omega(2^n)$ iterations for the resulting simple stochastic game.

▶ **Definition 15.** For $n \in \mathbb{N}$, the labelled Markov chain $\mathcal{M}_n$ is defined as follows by induction on n. The labelled Markov chain $\mathcal{M}_0$ is defined as



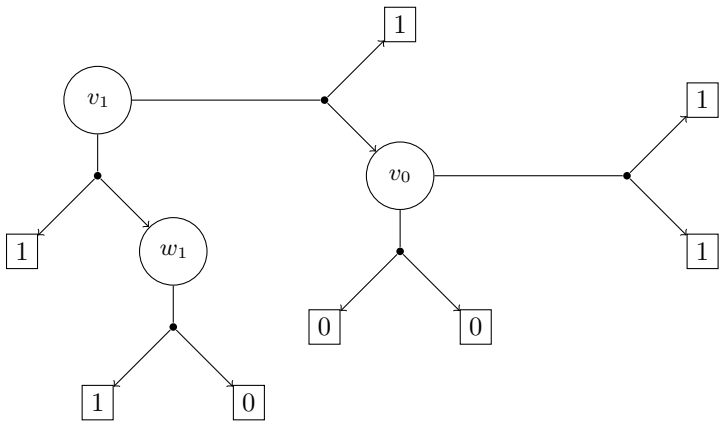
If $n > 0$ then the labelled Markov chain $\mathcal{M}_n$ is defined as



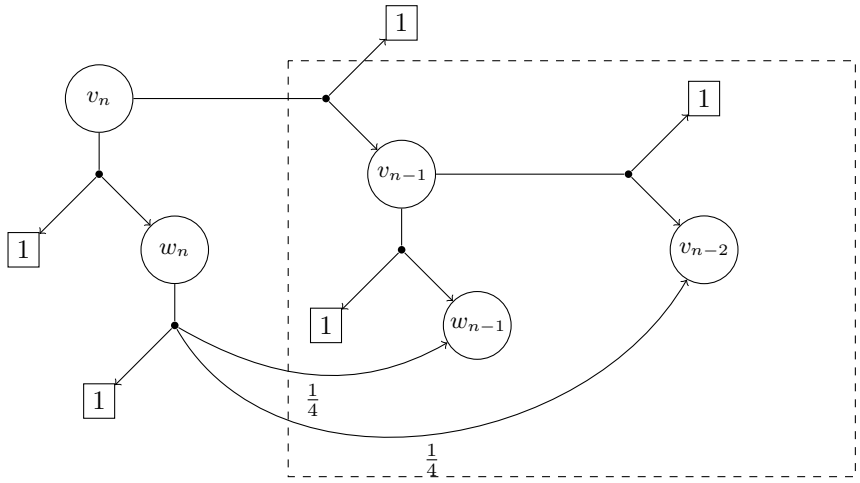
where the two dashed triangles together represent the labelled Markov chain $\mathcal{M}_{n-1}$.

In the introductory section, we presented $\mathcal{M}_1$. Note that $\mathcal{M}_n$ has $4n + 10$ states and $7n + 14$ transitions and, hence, is of size $O(n)$. Next, we give the simple stochastic games corresponding to the above defined labelled Markov chains according to transformation presented in Definition 11.

▶ **Definition 16.** For $n \in \mathbb{N}$, the simple stochastic game $\mathcal{G}_n$ is defined as follows by induction on n. The simple stochastic game $\mathcal{G}_0$ is defined as

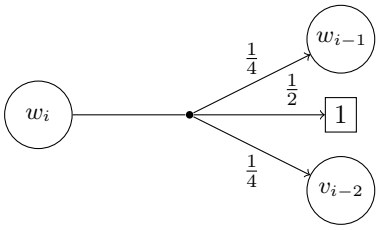


If $n > 0$ then the simple stochastic game $\mathcal{G}_n$ is defined as



where the dashed rectangle represents the simple stochastic game $\mathcal{G}_{n-1}$.

In the introductory section, we presented $\mathcal{G}_1$. In the above definition, we use v_i to denote the min node (s_i, t_i) and w_i to denote the min node (s_{i-1}, u_{i-1}). The transformation given in Definition 11 applied to labelled Markov chain $\mathcal{M}_n$ gives rise to a simple stochastic game of which $\mathcal{G}_n$ is only a part. In particular, for $2 \leq i \leq n$ a random vertex and the following edges have not been included in $\mathcal{G}_n$, as they are never selected in any of the strategies we construct in our proofs.



Next, we consider the strategies for the simple stochastic game $\mathcal{G}_n$. Recall that the Dirac distribution δ_v is defined by $\delta_v(w) = 1$ if $w = v$ and $\delta_v(w) = 0$ otherwise. In order to avoid clutter, for $1 \leq i \leq n$, instead of $T(v_i) = \frac{1}{2}\delta_1 + \frac{1}{2}\delta_{v_{i-1}}$ we write $T(i) = 0$ and instead of $T(v_i) = \frac{1}{2}\delta_1 + \frac{1}{2}\delta_{w_i}$ we write $T(i) = 1$. Also, instead of $T(v_0) = \frac{1}{2}\delta_1 + \frac{1}{2}\delta_1$ we write $T(0) = 0$ and instead of $T(v_0) = \frac{1}{2}\delta_0 + \frac{1}{2}\delta_0$ we write $T(0) = 1$.

A vertex is switchable if it is not locally optimal.

▶ **Definition 17.** The vertex v_i is switchable with respect to T if $\theta^T(v_i) > \theta^{\overline{T}_i}(v_i)$, where

$$\overline{T}_i(j) = \begin{cases} 1 - T(j) & \text{if } j = i \\ T(j) & \text{otherwise} \end{cases}$$

Rather than starting from an arbirary strategy, we pick a specific initial strategy (line 1–2). Furthermore, rather than choosing an arbitrary min vertex that is not locally optimal, we pick the v_i which is not locally optimal with the largest index (line 4).

```
1  for  i = 0, ..., n
2    T(i) ← 0
3  while  ∃0 ≤ i ≤ n : v_i is switchable with respect to T
4    m ← max{ i | v_i is switchable with respect to T }
5    T(m) ← 1 − T(m)
```

The above simple policy iteration algorithm applied to the simple stochastic game $\mathcal{G}_n$ gives rise to exponentially many iterations.

▶ **Theorem 18.** *For each $n \in \mathbb{N}$, there exists a labelled Markov chain of size $O(n)$ such that simple policy iteration takes $\Omega(2^n)$ iterations.*

7 Conclusion

Based on a correspondence between labelled Markov chains and simple stochastic games, we have shown that a modification of the basic algorithm of Bacci et al. for computing probabilistic bisimilarity distances of a labelled Markov chain is simple policy iteration applied to the corresponding simple stochastic game. The correspondence between labelled Markov chains and simple stochastic games also allows us to use Condon's correctness proof of simple policy iteration to show that our modification of the basic algorithm of Bacci et al. is correct. As we have shown in Theorem 8, that modification is essential. Such a modification also needs to be applied to the on-the-fly algorithm of Bacci et al.

Although Bacci et al. had already shown that their algorithm performs very well in practice, we show that in the worst case, their basic algorithm takes exponential time. Our example can also be used to show that their on-the-fly algorithm takes exponential time in the worst case as well. Our experimental results confirm that our modification gives rise to very little overhead. In several cases our modified algorithm even performs better than theirs. A detailed study of the performance of our modification of the on-the-fly algorithm is left for future research.

As we already mentioned in the introduction, the difference between Hoffman and Karp's policy iteration and Condon's simple policy iteration is that the former switches all locally non-optimal vertices in every iteration, whereas the latter only switches one of them. The question whether policy iteration, applied to a labelled Markov chain to compute its behavioural distances, may also give rise to exponentially many iterations is still open and left for future research. Note that an affirmative answer to this question would also prove an exponential lower bound for policy iteration for simple stochastic games, a problem that has been open for several decades.

Acknowledgments. The authors would like to thank Norm Ferns for extensive discussions. The authors are also grateful to the referees for their constructive feedback.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Upper Saddle River, NJ, USA, 1993.

2 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, March 2013. Springer-Verlag.

3 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, July/August 1996. Springer-Verlag.

4 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, MA, USA, 2008.

5 Franck van Breugel, Babita Sharma, and James Worrell. Approximating a behavioural pseudometric without discount. *Logical Methods in Computer Science*, 4(2), April 2008.

6 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.

7 John Canny. Some algebraic and geometric computations in PSPACE. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 460–467, Chicago, IL, USA, May 1988. ACM.

8 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, March/April 2012. Springer-Verlag.

9 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, February 1992.

10 Anne Condon. On algorithms for simple stochastic games. In Jin-Yi Cai, editor, *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

11 Brian Davey and Hilary Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom, 2002.

12 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, June 2004.

13 Oliver Friedmann. *Exponential Lower Bounds for Solving Infinitary Payoff Games and Linear Programs*. PhD thesis, Ludwig-Maximilians-University, Munich, Germany, 2011.

14 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, April 1990. North-Holland.

15 Alan Hoffman and Richard Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, January 1966.

16 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, July 1991. IEEE.

17 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, October 1958.

18 Leonid Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

19 Kim Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

20 Mary Melekopoglou and Anne Condon. On the complexity of the policy iteration algorithm. Computer Science Technical Report 941, University of Wisconsin, Madison, WI, USA, June 1990.

21 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1980.

22 James Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, August 1997.

23 Prakash Panangaden. *Labelled Markov Processes*. Imperial College Press, London, United Kingdom, 2009.

24 David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, Germany, March 1981. Springer-Verlag.

25 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, Cambridge, United Kingdom, 2012.

26 Lloyd Shapley. Stochastic games. *Proceedings of the Academy of Sciences*, 39(10):1095–1100, October 1953.

27 James K. Strayer. *Linear programming and its applications.* Springer-Verlag, New York, NY, USA, 1989.

28 Alfred Tarski. *A decision method for elementary algebra and geometry.* University of California Press, Berkeley, CA, USA, 1951.

29 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1/2):343–359, May 1996.

Robustly Parameterised Higher-Order Probabilistic Models*

Fredrik Dahlqvist[1], Vincent Danos[2], and Ilias Garnier[3]

1 **University College London**
`f.dahlqvist@ucl.ac.uk`
2 **Ecole Normale Supérieure**
`vincent.danos@ens.fr`
3 **University of Edinburgh**
`igarnier@inf.ed.ac.uk`

Abstract

We present a method for constructing robustly parameterised families of higher-order probabilistic models. Parameter spaces and models are represented by certain classes of functors in the category of Polish spaces. Maps from parameter spaces to models (parameterisations) are continuous and natural transformations between such functors. Naturality ensures that parameterised models are invariant by change of granularity – ie that parameterisations are intrinsic. Continuity ensures that models are robust with respect to their parameterisation. Our method allows one to build models from a set of basic functors among which the Giry probabilistic functor, spaces of cadlag trajectories (in continuous and discrete time), multisets and compact powersets. These functors can be combined by guarded composition, product and coproduct. Parameter spaces range over the polynomial closure of Giry-like functors. Thus we obtain a class of robust parameterised models which includes the Dirichlet process, point processes and other classical objects of probability theory such as the de Finetti theorem. By extending techniques developed in prior work, we show how to reduce the questions of existence, uniqueness, naturality, and continuity of a parameterised model to combinatorial questions only involving finite spaces.

1998 ACM Subject Classification Semantics of programming languages

Keywords and phrases Topological semantics, probabilistic models

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.23

1 Introduction

In a widely read paper [1], Beylin and Dybjer reinterpret coherence for monoidal categories as a result of normalisation on a linear non-commutative lambda-calculus. They prove that the structural arrows of a monoidal category are characterised by their domain and codomain. In this paper we follow a parallel path for probabilistic models. There are several differences with Dybjer's correspondence, however.

First, we work in **Pol**, the *concrete* category of Polish spaces Polish spaces form a classic and convenient environment to construct a large variety of stochastic models. We exploit this potential to build up a sufficiently expressive stock of structure arrows. Specifically, we build up structure arrows based on a two-sorted polynomial type theory of 'parameter' functors and 'model' ones. Model functors include as primitives the Vietoris functor of compact non-determinism, the Giry functor of probabilities, and most interestingly the

* This work was funded by the ERC grant RULE.

Skorokhod functor of cadlag functions - i.e. with countably many jumps and values in any Polish space - which captures traces of processes both in discrete and continuous time. Model functors can be combined by products, coproducts, and guarded composition; parameter functors are less malleable (perhaps because we do not understand them well yet). Hence, our type theory allows for iterations of 'compact' non-determinism (as captured by the Vietoris functor) and probabilism (the Giry monad) - which have been a fundamental pursuit in quantitative models of concurrency theory since Segala's early work [18, 10, 19, 14]. In addition, topology (and the possible recourse to metrics) allows one to talk about continuity and to quantify notions of approximate bisimulation. Our arrows come pre-equipped with metric interpretations which several and in particular van Breugel et al. have convincingly argued are fundamental in probabilistic modeling [6, 20].

Thus, our stock of arrows is remarkably expressive. Of course, this would amount to little, without a characterization of arrows equality. Here again, our result is slightly different from Dybjer's. We do not show that natural transformations are uniquely determined by their 'types'. Instead, using the 'machine' built in earlier work [5, 4], we show that structure arrows are completely characterised by their behaviours on finite spaces. In effect, our result makes the structure arrow equality problem (the equivalent of the normalisation of linear lambda-terms if we follow our analogy) purely combinatorial. For some special choices of parameter and model functors (of the Giry type), we can even show rigidity [4], that is to say, types do characterise arrows. Regarding existence, we provide a converse to the above result, and prove that not only are structure arrows wholly determined by the finite case, but that finite data is *enough* to define such arrows.

The standpoint presented here departs from the more common "forward semantics" approach where one knows already what one wants to semanticize. Here we embrace a less trodden path (but usually rewarding, as in the Ehrhard-Regnier invention of differential linear logic via the study of spaces of sequences, or Girard's own carving of linear logic via coherence spaces) of "reverse semantics" where the mathematical tractability of the semantic universe is the primary tool for constructing the universe of computational discourse.

The outline of the paper is as follows: we start with a 'slicing' of the category of Polish spaces in convenient layers, and recall the basic points of our existence and unicity 'Machine', spelling out the conditions on our parameter and model functors. With this behind us, we attack the description of the type theory and develop a string of propositions which justify that choice by building its semantics. We conclude with our "normalisation" theorem and an application of our framework to the celebrated de Finetti theorem, a key result in probability theory and statistics.

2 Preliminaries

We work in the category **Pol** of completely metrisable and separable topological spaces and continuous maps. There is the obvious Borel functor $\mathcal{B} : \mathbf{Pol} \to \mathbf{Meas}$ mapping a Polish space to the measurable space with the Borel σ-algebra and mapping continuous maps to measurable ones, together with the underlying set functor $U : \mathbf{Pol} \to \mathbf{Set}$ with the obvious forgetful action. **Pol** has all countable limits and all countable coproducts [3, IX].

2.1 Pol endofunctors

We introduce some **Pol** endofunctors which are going to be used as examples in the rest of the paper and form the primitive bricks for our structure arrows. First, there is the Giry functor G [9, 17] which maps a space X to the space of Borel probability measures (with the

topology induced by the Kantorovich metric) over X. Related to G are the finite nonzero positive measure functor $\mathsf{M}^+ \cong \mathsf{G} \times \mathbb{R}_{>0}$. A parallel construction is that of the Vietoris functor V which maps X to the "hyperspace" of its compact subsets topologised with the Hausdorff distance [13, 4.F]. The finite multiset functor B and the related finite list functor W are also **Pol** endofunctors [4]. Finally, we have a pair of functional functors. The Skorokhod functor D, which maps X to the space of cadlag (right-continuous with left limits) functions from $[0, \infty)$ to X equipped with the J_1 topology [7], which is fundamental to the study of continuous-time stochastic processes. And for any compact Polish set X, the functor $\mathsf{C}(X, -)$ which maps any Polish space Y to that of continuous maps from X to Y.

Our family of functors covers both probabilistic behaviour through G, compact non-determinism through V and spaces of trajectories through D and C. Moreover, working in the category of Polish spaces makes the analogies between these functors all the more striking: G is metrised by Kantorovich, V by Hausdorff and the same holds of D using a metric allowing "time transport". Recent work [16] might shed additional light on these similarities.

2.2 The structure of Pol

We slice **Pol** into the following full subcategories: finite Polish spaces $\mathbf{Pol}_f$; compact zero-dimensional spaces $\mathbf{Pol}_{cz}$; zero-dimensional spaces $\mathbf{Pol}_z$; and compact spaces $\mathbf{Pol}_c$:

$$\begin{array}{ccccccc} \mathbf{Pol}_f & \hookrightarrow & \mathbf{Pol}_{cz} & \hookrightarrow & \mathbf{Pol}_z & \hookrightarrow & \mathbf{Pol} \\ & \searrow & & & & \nearrow & \\ & & & \mathbf{Pol}_c & & & \end{array} \tag{1}$$

Finite spaces are equipped with the discrete topology. Compact zero-dimensional Polish spaces (such as the Cantor set $2^{\mathbb{N}}$) can be characterised as projective limits of finite Polish spaces. Zero-dimensional spaces (such as the Baire space $\mathbb{N}^{\mathbb{N}}$) are those spaces which admit a *base* of their topology constituted of clopen sets. These subcategories have interesting structures: any zero-dimensional space equipped with a choice of a countable base of clopen sets can be mapped to its compactification, which is compact zero-dimensional [4]. In the other direction, any Polish space equipped with a choice of a countable base can be mapped to a zero-dimensional Polish refinement of its topology. We give some more details on these operations in the next section, together with characterisations of Polish and compact zero-dimensional spaces as respectively colimits and limits of particular diagrams.

2.2.1 Characterisations of zero-dimensional spaces

A countable codirected diagram in $\mathbb{A}$ (an $\mathbb{A}$ *ccd* for short) is a functor $D : I^{op} \to \mathbb{A}$ where I is a countable directed partial order and $\mathbb{A}$ is a subcategory of **Pol**. The characterisation of objects of $\mathbf{Pol}_{cz}$ can be formulated as follows:

▶ **Proposition 1** ([11]). *A space X is compact zero-dimensional if and only if there exists a $\mathbf{Pol}_f$ ccd D such that $X \cong \lim D$.*

Any Polish space can be written as the colimit of a diagram in $\mathbf{Pol}_z$:

▶ **Proposition 2** ([5], Proposition 3.2). *Let X be a Polish space and $\mathcal{F}$ a countable base of the topology of X. Let $Z_X(\mathcal{F}) \triangleq (U(X), \langle Bool(\mathcal{F})\rangle)$ be the space having the same underlying set as X and the topology generated by the Boolean algebra generated by $\mathcal{F}$. The following holds: (i) $Z_X(\mathcal{F})$ is zero-dimensional Polish, (ii) $\mathcal{B}(Z_X(\mathcal{F})) = \mathcal{B}(X)$. We call $Z_X(\mathcal{F})$ the* zero-dimensionalisation *of X with respect to $\mathcal{F}$.*

The family of zero-dimensionalisations of a space X indexed by all countable bases of X forms a codirected diagram. This diagram is indexed by $Bases(X)$ the set of all countable bases of X partially ordered by inclusion; $Bases(X)$ is directed by closing the union of two bases under finite intersection [4, Def. 2.10]. If $\mathcal{F}$, $\mathcal{G}$ are such bases then, if $\mathcal{F} \subseteq \mathcal{G}$, the identity function is continuous from $Z_X(\mathcal{G})$ to $Z_X(\mathcal{F})$. This defines a codirected diagram from the directed partial order $Bases(X)$ to $\mathbf{Pol}_z$, that we still denote by Z_X. The following statement states that any Polish space is the colimit of its diagram of zero-dimensionals:

▶ **Theorem 3** ([5], Th. 3.5; [4], Proposition 2.11)**.** *For every Polish space X, $X \cong \operatorname{colim} Z_X$.*

2.3 Converging in $\mathsf{G}(X)$

We recall some standard facts about convergence in $\mathsf{G}(X)$ for X Polish. The *boundary* of a set $A \subseteq X$ is the set-theoretic difference between its closure and its interior, and is denoted by $\partial_X A$. By the Portmanteau theorem ([2], Th. 2.1), a sequence $(p_n)_{n\in\mathbb{N}}$ of probability measures converges to $p \in \mathsf{G}(X)$ iff $p_n(A) \to p(A)$ for each Borel set A which is a *p-continuity set*, i.e. which verifies $p(\partial_X A) = 0$. Note that for all p, p-continuity sets form a Boolean algebra that we denote $\mathcal{C}_X(p)$ ([17], Lemma 6.4). We have the following facts:

▶ **Lemma 4.** *Countable Polish spaces are zero-dimensional.*

Proof. Let X be Polish, $x \in X$ and U be open in X. It is not difficult to use the metric to define a function $f : X \to [0,1]$ with $f(x) = 0$ and $f(y) = 1$ when $y \in U^c$. If X is countable, then $f[X]$ is countable and thus there exists $p \in [0,1]$ such that $p \notin f[X]$ and it is easy to check that $f^{-1}([0,p)) = f^{-1}([0,p])$ is clopen and included in U, and the conclusion follows. ◀

▶ **Lemma 5.** *Let X, Y be Polish, $p \in \mathsf{G}(X)$ and let $f : X \to Y$ be continuous. If B is a $\mathsf{G}(f)(p)$-continuity set then $f^{-1}(B)$ is a p-continuity set.*

Proof. Direct consequence of the inequality $\partial_X(f^{-1}(B)) \subseteq f^{-1}(\partial_Y B)$. ◀

▶ **Lemma 6.** *Let X be Polish and uncountable and let $\{p_i\}_{i\in I} \subseteq \mathsf{G}(X)$ be a countable family of probability measures. There exists a countable base $\mathcal{F}$ of X such that $\mathcal{F} \subseteq Boole(\mathcal{F}) \subseteq \cap_i \mathcal{C}_X(p_i)$.*

Proof. Let $B(x,\epsilon)$ be the open ball of radius $\epsilon > 0$ centered on x. Observe that for $0 < \epsilon < \epsilon'$, $\partial_X B(x,\epsilon) \cap \partial_X B(x,\epsilon') = \emptyset$. Therefore, for a given p, there can at most be countably many radiuses ϵ_k such that the $B(x,\epsilon_k)$ are *not* p-continuity sets, as otherwise the total mass of $\cup_k \partial_X B(x,\epsilon_k)$ would diverge. Using that a countable union of countable sets is countable, there are at most countably many radiuses ϵ_k such that $B(x,\epsilon_k) \notin \cap_{i\in I} \mathcal{C}_X(p_i)$. For any dense subset $E \subseteq \mathbb{R}_{>0}$, the open balls $\mathcal{N}(x) = \{B(x,\epsilon)\}_{\epsilon\in E}$ characterise convergence to x. For our purposes, it is enough to take E such that E does not intersect the forbidden radii $\{\epsilon_k\}$, which can always be done. The sought base $\mathcal{F}$ is obtained by considering a countable dense subset D of X and taking $\mathcal{F}$ to be the closure under finite intersections of $\{\mathcal{N}(x) \mid x \in D\}$. Since continuity sets form a boolean algebra, we get $Boole(\mathcal{F}) \subseteq \cap_i \mathcal{C}_X(p_i)$. ◀

3 The Machine

The parameterised models we are interested in use 'the Machine' [4], a powerful theorem allowing one to extend a class of natural transformations from finite Polish spaces to arbitrary ones. This extension theorem hinges on particular conditions on the domain and

codomain functors of the natural transformation, corresponding respectively to constraints on parameters and models. Accordingly, we will call domain functors '$\mathcal{P}$-functors' and codomain functors '$\mathcal{M}$-functors'. Below, we list these conditions. In Section 4, we will study closure properties of these conditions, and derive a syntax for parameters and models.

3.1 Parameter condition

The Machine applies to natural transformations whose $\mathcal{P}$-functor commutes with colimits of diagrams of zero-dimensionals (Section 2.2.1). We call this property *Z-cocontinuity*:

▶ **Definition 7** (Z-cocontinuity)**.** An endofunctor $F : \mathbf{Pol} \to \mathbf{Pol}$ is Z-cocontinuous if for all space X, there exists an isomorphism $F(\operatorname{colim} Z_X) \cong \operatorname{colim} FZ_X$.

In order for the Machine to apply, $\mathcal{P}$-functors must also preserve epis. The *parameter condition* is the conjunction of these two conditions.

▶ **Definition 8** (Parameter conditions)**.** An endofunctor $F : \mathbf{Pol} \to \mathbf{Pol}$ satisfies the *parameter condition* (or equivalently, is a $\mathcal{P}$-functor) if (i) F is Z-cocontinuous and (ii) F preserves epis.

▶ **Example 9.** The following are $\mathcal{P}$-functors: (i) the identity functor (it is Z-cocontinuous by Theorem 3); (ii) the Polish Giry functor G (Z-cocontinuity is proved in Ref. [5, Th. 3.7]), and the related finite positive nonzero measure functor M^+; (iii) the multiset functor B (see [4]). (iv) for any discrete (and thus at most countable) space X, $\mathsf{C}(X, -)$ is trivially Z-cocontinuous.

3.2 Model condition

The Machine also requires $\mathcal{M}$-functors to verify a list of conditions, corresponding to constraints on models.

The model condition

Before defining the model condition, we introduce some terminology related to commutation of functors with some limits.

▶ **Definition 10.** Let $\mathbb{A}$ be a subcategory of $\mathbf{Pol}$. An endofunctor $G : \mathbf{Pol} \to \mathbf{Pol}$ is $\mathbb{A}$-continuous if for all *ccd* (Section 2.2) $D : I^{op} \to \mathbb{A}$, $G(\lim D) \cong \lim GD$.

$\mathcal{M}$-functors are endofunctors that satisfies the following.

▶ **Definition 11** (Model conditions)**.** An endofunctor $G : \mathbf{Pol} \to \mathbf{Pol}$ satisfies the *model condition* (or equivalently, is an $\mathcal{M}$-functor) if: (i) G preserves monos, (ii) G preserves embeddings, (iii) G preserves intersections, (iv) G is $\mathbf{Pol}_f$-continuous.

▶ **Example 12.** The following are $\mathcal{M}$-functors: (i) the Giry functor G, and finite measure functors M^+; (ii) the multiset and list functors B, W; (iii) the Vietoris functor V; (iv) the Skorokhod functor D and the continuous map functor $\mathsf{C}(X, -)$ from a compact Polish space X (see [4]).

Observe that in Definition 11, all conditions are preserved by composition of endofunctors except the last one. We will come back to this in Section 4.

3.3 The Machine

The Machine states that natural transformations (parameterised models) between $\mathcal{M}$-functors and $\mathcal{P}$-functors are entirely characterised by their components on finite spaces.

▶ **Theorem 13** ([4]). *Let F_1 be a $\mathcal{P}$-functor and F_2 be a $\mathcal{M}$-functor; one has $Nat(F_1, F_2) \cong Nat(F_1|_{\mathbf{Pol}_f}, F_2|_{\mathbf{Pol}_f})$.*

▶ **Example 14.** Let us give some examples of finitely characterised natural transformations.

- The unite of the Giry monad $\eta : \mathbb{1} \Rightarrow \mathsf{G}$ is entirely characterised by its finite components. We conjecture that the parameter condition is closed under composition by G, which would imply that multiplication $\mu : \mathsf{G}^2 \Rightarrow \mathsf{G}$ of the Giry monad is also characterised on the whole of **Pol** by its finite components.
- The normalisation $\nu : \mathsf{M}^+ \Rightarrow \mathsf{G}$ defined by $\nu_X : \mathsf{M}^+X \to \mathsf{G}X, \mu \mapsto \frac{\mu}{\mu(X)}$ is also finitely characterised. Moreover, the Machine allows to prove that it is the unique natural transformation between M^+ and G [4, Th. 5.2].

Classical objects of statistics can be framed as natural transformations:

- the i.i.d. distribution on sequences of samples $iid : \mathsf{G} \Rightarrow \mathsf{G}(-^{\mathbb{N}})$ (Section 5);
- the Dirichlet process $\mathscr{D} : \mathsf{M}^+ \Rightarrow \mathsf{G}^2$ a cornerstone of Bayesian nonparametrics [8];
- the Poisson process $\mathscr{P} : \mathsf{M}^+ \Rightarrow \mathsf{GB}$ which is the prototypical point process. Using the Machine, it is enough to define the Poisson process on finite sets, this is done via

$$\mathscr{P}_n : \mathsf{M}^+(n) \simeq (\mathbb{R}_+)^n \to \mathsf{GB}(n) \simeq \mathsf{G}(\mathbb{N}^n), (\lambda_1, \dots, \lambda_n) \mapsto \mathrm{Po}(\lambda_1) \times \dots \times \mathrm{Po}(\lambda_n)$$

where $\mathrm{Po}(\lambda)$ is the Poisson measure on $\mathbb{N}$ with parameter λ.

4 A grammar for parameterised models

We turn now to the main question of the paper, which is to find operations on functors under which the parameter and model conditions are closed. For parameters, this result takes the form of a simple grammar over functors. For models, we develop a simple type system over polynomial terms generated from a family of functors, well-typedness implying the model condition. This syntax for parameter and models lifts to natural transformations, giving rise to a language of natural combinators for parameterised models.

As **Pol** has all countable limits and coproducts, the category of **Pol** endofunctors is closed under at most countable coproducts and products (recall that if $F, G : \mathbf{Pol} \to \mathbf{Pol}$ are two endofunctors, their coproduct $F + G$ acts on objects by $(F + G)(X) = F(X) + G(X)$ and on morphisms by $(F + G)(f) = F(f) + G(f)$, and similarly for products). Endofunctors are also trivially closed under composition.

4.1 Closure properties of the parameter condition

Let us start with parameter conditions. At the time of writing, we do not know whether these are closed under products and/or functor composition. However, we show that they are closed under coproducts. We also derive specific results for particular functors that altogether yield a sufficiently expressive class of parameterisations.

Finite coproducts preserve the parameter condition

The following facts are easily verified:

▶ **Proposition 15.** *If G, H preserve epis, then so does $G + H$.*

▶ **Proposition 16.** *The parameter condition is preserved by finite coproducts.*

Products of Giry-like functors satisfy the parameter condition

Countable products of Giry-like functors (i.e. G, M^+) satisfy the parameter condition. The case of finite products follow trivially from the same result.

▶ **Proposition 17.** *Let* $\{F_k\}_{k\in\mathbb{N}}$ *be given with* $F_k \in \{\mathsf{G}, \mathsf{M}^+\}$. *Then* $\prod_k F_k$ *satisfies the parameter condition.*

Proof. It is enough to treat the case of G as M^+ is naturally isomorphic to $\mathsf{G} \times \mathbb{R}_{>0}$. Preservation of epis by G lifts to products of G. Let us prove Z-cocontinuity. We reuse the proof method of ([5], Th. 3.7). It is enough to exhibit, for all X and all countable family of converging sequences $\{p_n^{(k)} \to p^{(k)}\}_{k\in\mathbb{N}}$ in $\mathsf{G}(X)^{\mathbb{N}}$, a base of $\mathcal{F} \in \mathit{Bases}(X)$ s.t. the sequence converge in $\mathsf{G}(Z_X(\mathcal{F}))^{\mathbb{N}}$. It follows from Lemma 4 that for countable Polish spaces there is nothing to show, and we can thus assume w.l.o.g. that X is uncountable. Applying Lemma 6, we get a base $\mathcal{F}$ s.t. $\mathit{Boole}(\mathcal{F}) \subseteq \cap_k \mathcal{C}_X(p^{(k)})$. Noting that $\mathit{Boole}(\mathcal{F})$ is a base of $Z_X(\mathcal{F})$, an application of Th. 2.2 in (Billingsley [2]) concludes. ◀

Giry-like functors over products satisfy the parameter condition

Our grammar for parameterisations admits a way of specifying quantitative relations on points of the underlying space, as shown below in Proposition 18 (which generalises to Giry-like functors M^+). In what follows, we treat countable products. The case of finite products follows easily from the same proof.

▶ **Proposition 18.** $\mathsf{G}(-^{\mathbb{N}})$ *satisfies the parameter condition.*

Proof. Preservation of epis still follows from the properties of G. To prove Z-cocontinuity, we follow the proof scheme of Proposition 17. We denote $\pi_k : X^{\mathbb{N}} \to X, \pi_{1\ldots k} : X^{\mathbb{N}} \to X^k$ the canonical projections. Let X be Polish and let $(p_n)_{n\in\mathbb{N}} \to p$ be a converging sequence in $\mathsf{G}(X^{\mathbb{N}})$. By Lemma 6, there exists a base $\mathcal{F}$ of X such that for all $k > 0$, $\mathit{Boole}(\mathcal{F}) \subseteq \cap_k \mathcal{C}_X(\mathsf{G}(\pi_k)(p))$. The sets of the form $\pi_k^{-1}(O)$ with O ranging in $\mathit{Boole}(\mathcal{F})$ induce a base $\mathcal{H}$ of $Z_X(\mathcal{F})^{\mathbb{N}}$. Using Lemma 5 plus the fact that continuity sets are closed under finite intersections, we deduce that $\mathcal{H} \subseteq \mathcal{C}_{X^{\mathbb{N}}}(p)$. Therefore, for all $V \in \mathcal{H}$, $p_n(V) \to p(V)$. Using Th. 2.2 in (Billingsley [2]), we get that $p_n \to p$ in $\mathsf{G}(Z_X(\mathcal{F})^{\mathbb{N}})$. We conclude that $\mathsf{G}(-^{\mathbb{N}})$ is Z-cocontinuous. ◀

4.2 Closure properties of the model condition

We turn now to closure properties of model conditions. As we are going to show, all model conditions (Definition 11) are closed under all polynomial operations with the exception of the last one, namely $\mathbf{Pol}_f$-continuity.

Finite coproducts preserve the model condition

We prove preservation of the model condition under finite coproducts. We proceed with the other parts of the model condition, namely preservation of embeddings, intersections and $\mathbf{Pol}_f$-continuity. The following proposition is routine.

▶ **Proposition 19.** *(i) If* G, H *preserve monos, then so does* $G + H$. *(ii) If* G, H *preserve embeddings, then so does* $G + H$. *(iii) If* G, H *preserve embeddings and intersections, then so does* $G + H$.

The following one is a bit more technical and deserves a proof.

▶ **Proposition 20.** *If G, H are $\mathbb{A}$-continuous, then so is $G + H$.*

Proof. Let $(X_i)_{i\in I}$ be a ccd with limit X, we're assuming that $GX = \lim_i GX_i$ and $HX = \lim_i HX_i$. The diagrams $(GX_i)_{i\in I}$ and $(HX_i)_{i\in I}$ define a diagram $(GX_i + HX_i)_{i\in I}$ where for each $i < j$ there exists a unique arrow $Gp_{i,j} + Gp_{i,j} : GX_i + HX_i \to GX_j + HX_j$, with $p_{i,j} : X_i \to X_j$ the connecting morphism of the diagram $(X_i)_{i\in I}$. The coproduct $GX + HX$ is a cone over this diagram via the maps $Gp_i + Hp_i : GX + HX \to GX_i + HX_i$ constructed from the canonical projections $p_i : X \to X_i$ by the universal property of coproducts. There must therefore exist a unique continuous map $\phi : GX + HX \to \lim_i(GX_i + HX_i)$ which maps a thread in GX to the obvious thread in $\lim_i(GX_i + HX_i)$ and similarly for threads in HX. It is clear that ϕ is injective, moreover it is easy to see that threads in $\lim_i(GX_i + HX_i)$ must be the form $(x_i)_{i\in I}$ with every $x_i \in GX_i$ or with every $x_i \in HX_i$, and in particular ϕ is surjective. Thus ϕ is continuous and bijective, and it only remains to show that it is open. Let U be open in $GX + HX$. By definition of the coproduct topology, $U_G = i_{GX}^{-1}(U)$ and $U_H = i_{HX}^{-1}(U)$ are open, and thus by definition of the topology on the limit,

$$U_G = \bigcup_i Gp_i^{-1}(V_i) \quad \text{and} \quad U_H = \bigcup_j Gq_j^{-1}(W_j)$$

where each V_i (resp. W_j) is open in GX_i (resp. HX_j). Since I is cofiltered, for every $i, j \in I$ there exists a $k \in I$ and morphisms $p_{k,i} : X_k \to X_i$ and $p_{k,j} : X_k \to X_j$. Let us denote by $i \wedge j$ the choice of such a k for the pair i, j and note that $Gp_i^{-1}(W) = Gp_{i\wedge j}^{-1}(Gp_{i\wedge j,i}^{-1}(W))$. Consider now the set

$$V = \bigcup_{i,j} q_{i\wedge j}^{-1}(Gp_{i\wedge j,i}^{-1}(U_i) \uplus Hp_{i\wedge j,i}^{-1}(U_j))$$

where q_i is the canonical projection $(\lim_i HX_i + GX_i) \to HX_i + GX_i$. By construction it is open in $\lim_i(GX_i + HX_i)$ since it is a union of inverse images of sets which are open in $GX_{i\wedge j} + HX_{i\wedge j}$ by definition of the coproduct topology. We claim that $\phi[U] = V$. For any thread $(x_i) \in U$, if the thread is in the HX component then it must belong to U_G and thus there must exist an i such that $x_i \in V_i$, since we can assume that the connecting morphisms are surjective there exists for each j an element $x_{i\wedge j} \in Gp_{i\wedge j,i}^{-1}(U_i)$ in the thread and it follows that the thread belongs to V, and similarly if the thread (x_i) is in the GX component. Similarly, starting from (x_i) in V, it is clear that (x_i) belongs to U and it thus ϕ is open. ◀

Finite products preserve the model condition

▶ **Proposition 21.** *(i) If G, H preserve monos, then so does $G \times H$. (ii) If G, H preserve embeddings, then so does $G \times H$. (iii) If G, H preserve intersections, then so does $G \times H$. (iv) If G, H are $\mathbb{A}$-continuous, then so is $G \times H$.*

Proof. (i) Straightforward. (ii) Since embeddings are equalizers in **Pol**, this result is simply a case of limits commuting with limits (Mac Lane [15] IX). (iii) Similarly, since intersections are finite limits and they commute with finite limits in **Pol**. (iv) Finally, since ccds are cofiltered limits, they commute with finite products. ◀

Composition

We now consider the operation of functor composition. The following is trivial:

▶ **Proposition 22.** *The conditions 1. to 3. of Definition 11 are preserved under functor composition.*

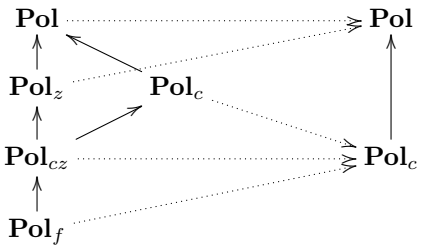


Figure 1 Signature for G and V.

The condition of $\mathbf{Pol}_f$-continuity (Definition 11, 4.) does not behave as well: if F, G are $\mathbf{Pol}_f$-continuous endofunctors and F maps finite spaces to non-finite spaces, GF has no reason to be $\mathbf{Pol}_f$-continuous. On the other hand, if F maps finite spaces to a subcategory with respect to which G is continuous, then the composition GF will be $\mathbf{Pol}_f$-continuous. In order to make this intuition formal, we need to capture the global behaviour of functors on the subcategories of **Pol**. To do so, we propose to abstract functors as monotonic functions on the poset of subcategories of **Pol**.

▶ **Definition 23** (Partial order on subcategories)**.** We denote by $(\mathbb{P}, \leq)$ the lattice over the subcategories of **Pol** displayed in Equation 1 and generated by the subcategory relation, i.e. $\mathbb{A} \leq \mathbb{B}$ iff $\mathbb{A}$ is a subcategory of $\mathbb{B}$. We will denote by $\wedge$ and $\vee$ the infinimum and the supremum.

Note that $\mathbb{P}$ has as maximal element **Pol** and as minimal element $\mathbf{Pol}_f$. The known behaviour of a endofunctor over **Pol** can be presented as a monotonic function from $\mathbb{P}$ to itself. We call such a function a *signature* assigned to the functor.

▶ **Definition 24** (Signatures and signature assignments)**.** We denote by $Sign(\mathbb{P})$ the set of order-preserving functions from $\mathbb{P}$ to itself. We say that an endofunctor G admits $f \in Sign(\mathbb{P})$ as a signature if for all $\mathbb{A} \in \mathbb{P}$, there exists a functor $G' : \mathbb{A} \to f(\mathbb{A})$ such that the following diagram commutes in **Cat**:

$$\begin{array}{ccc} \mathbf{Pol} & \xrightarrow{G} & \mathbf{Pol} \\ {\scriptstyle I_{\mathbb{A}\mathbf{Pol}}}\uparrow & & \uparrow{\scriptstyle I_{f(\mathbb{A})\mathbf{Pol}}} \\ \mathbb{A} & \xrightarrow[G']{} & f(\mathbb{A}) \end{array} \tag{2}$$

where $I_{\mathbb{A}\mathbb{B}}$ denotes the obvious inclusion functor. If G admits f as a signature, we call the pair (G, f) a *signature assignment*.

▶ **Example 25.** It is known that the Giry functor G and the Vietoris functor V preserve compactness (see resp. Parthasarathy [17], Th. 6.4 and Kechris [13], Th. 4.26). Therefore, both G and V admit the signature (in dotted arrows) in Figure 1. However, the fact that V maps finite spaces to finite spaces implies that it admits a finer signature (Figure 2). Note also that the functor M^+ does *not* preserve compactness.

The exact signature of a functor might be unknown. However, it is always possible to assign to a functor the signature corresponding to the constant function $\mathbb{A} \in \mathbb{P} \mapsto \mathbf{Pol}$. In fact, the lattice structure on $\mathbb{P}$ lifts to signatures:

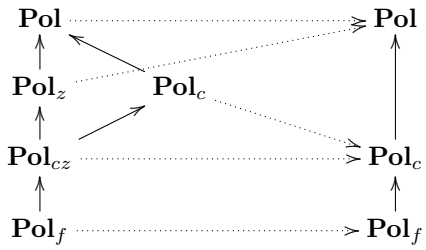


Figure 2 A finer signature for V.

▶ **Lemma 26.** *Define the relation on* $Sign(\mathbb{P})$ $f \leq^* g \Leftrightarrow \forall \mathbb{A} \in \mathbb{P}, f(\mathbb{A}) \leq g(\mathbb{A})$. *Set* $f \wedge g = \mathbb{A} \mapsto f(\mathbb{A}) \wedge g(\mathbb{A})$ *and similarly for* $\vee$. *Then* $(Sign(P), \leq^*)$ *is a lattice and the constant function* $\mathbb{A} \mapsto \mathbf{Pol}$ *is its maximal element.*

Proof. Reflexivity and transitivity are trivial. Assume $f \leq^* g$ and $g \leq^* f$, then by antisymmetry of $(P, \leq)$ we have $f = g$. Maximality of the constant **Pol** function is trivial. ◀

Let us now define a criterion for functor composition ensuring preservation of $\mathbf{Pol}_f$-continuity.

▶ **Proposition 27.** *Let* F *and* G *be respectively a* $\mathbb{A}$*-continuous and a* $\mathbb{B}$*-continuous functor such that* F *admits signature* f *and* G *admits signature* g. *If* $f(\mathbf{Pol}_f) \leq \mathbb{B}$, *then* GF *is* $\mathbb{C}$*-continuous, where* $\mathbb{C} = \sup\{\mathbb{C}' \mid \mathbb{C}' \leq \mathbb{A} \wedge f(\mathbb{C}') \leq \mathbb{B}\}$.

Proof. Since $f(\mathbf{Pol}_f) \leq \mathbb{B}$, we know that $\mathbb{C}$ exists and verifies $\mathbf{Pol}_f \leq \mathbb{C}$. Let $D : I^{op} \to \mathbb{C}$ be a $\mathbb{C}$ *ccd*. By assumption of $\mathbb{A}$-continuity and using hat $\mathbb{C} \leq \mathbb{A}$, $F(\lim D) \cong \lim FD$. Since F admits f as a signature, FD is a $f(\mathbb{C})$-*ccd* and since $f(\mathbb{C}) \leq \mathbb{B}$, $G(\lim FD) \cong \lim GFD$. ◀

In the next section, we will leverage Proposition 27 by defining a type system for polynomial composites of endofunctors.

4.3 Syntax for parameterisations and models

We capture the results of this section into grammars for parameterisations and models.

A grammar for parameterisations

In what follows, we let $\mathcal{G} = \{\mathsf{G}, \mathsf{M}^+\}$ be the Giry-like functors (respectively Giry, the finite positive measure and finite nonzero measure functors). We recall that Δ is the diagonal functor.

▶ **Definition 28** (Parameterisations generated by a family of functors)**.** Parameterisations, denoted by $\mathcal{P}$, are defined by the following grammar:

$$\begin{array}{lll} \mathcal{P} & ::= & F \mid G\Delta \mid G \times G \mid \mathcal{P} + \mathcal{P} \\ G & ::= & \mathsf{G} \mid \mathsf{M}^+ \end{array}$$

where F ranges over functors satisfying the parameter condition.

We have the following expected result:

▶ **Theorem 29.** *All* $P \in \mathcal{P}$ *verify the parameter condition.*

Proof. By induction, using the results of Section 4.1. ◀

$$\textsc{Axiom}\quad \frac{F \in \mathcal{M}_0 \qquad F \text{ is } \mathbb{A}\text{-continuous} \qquad F \text{ admits signature } f}{F :: (\mathbb{A}, f)}$$

$$\textsc{Sum}\quad \frac{F :: (\mathbb{A}, f) \qquad G :: (\mathbb{B}, g)}{F + G :: (\mathbb{A} \wedge \mathbb{B}, f \vee g)} \qquad \textsc{Product}\quad \frac{F :: (\mathbb{A}, f) \qquad G :: (\mathbb{B}, g)}{F \times G :: (\mathbb{A} \wedge \mathbb{B}, f \vee g)}$$

$$\textsc{Composition}\quad \frac{F :: (\mathbb{A}, f) \qquad G :: (\mathbb{B}, g) \qquad g(\mathbf{Pol}_f) \leq \mathbb{A}}{F \circ G :: (\mathbb{C}, f \circ g) \qquad \text{where } \mathbb{C} = \sup\{\mathbb{C}' \mid \mathbb{C}' \leq \mathbb{A} \wedge f(\mathbb{C}') \leq \mathbb{B}\}}$$

Figure 3 Inferences rules for the type system on models.

A grammar for models

Proposition 27 gives a sufficient condition ensuring that composition of functors satisfying the model condition still satisfies the model condition. We integrate this result in a type system for polynomial composites of functors satisfying the model condition.

▶ **Definition 30** (Functor types). A *functor type* is a pair $(\mathbb{A}, f)$ with $\mathbb{A} \in \mathbb{P}$ and $f \in Sign(\mathbb{P})$. The set of types is defined by $Types(\mathbb{P}) \triangleq \mathbb{P} \times Sign(P)$.

Functor types are assigned to elements of the polynomial closure of the set of functors that satisfy the model condition.

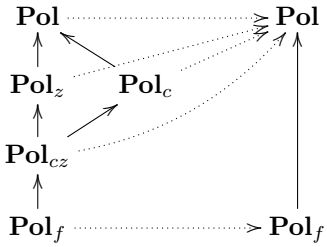
▶ **Definition 31** (Typing judgments). We inductively define a relation between functors satisfying the model condition and functor types through the set of inference rules in Figure 3. The fact that a functor F admits the type $(\mathbb{A}, f)$ will be denoted by $F :: (\mathbb{A}, f)$.

Our type system is sound with respect to the model condition.

▶ **Theorem 32.** *If $M :: (\mathbb{A}, f)$ then M is $\mathbb{A}$-continuous, m admits signature f and M satisfies the model condition.*

Proof. The proof is by induction. The properties of preservation of monos, preservation of embeddings and preservation of intersections are treated in Section 4.2. **Sum rule**. Both F and G are $\mathbb{A} \wedge \mathbb{B}$-continuous, therefore by Proposition 20, $F + G$ is $\mathbb{A} \wedge \mathbb{B}$-continuous (and therefore are least $\mathbf{Pol}_f$-continuous). It is clear that a coproduct of finite spaces is finite and similarly for compact zero-dimensional spaces, compact spaces and zero-dimensional spaces. Therefore, $F + G$ admits $f \vee g$ as a signature. The case of the product rule is similar. **Composition**. $\mathbb{C}$-continuity is by Proposition 27. That FG admits $f \circ g$ as a signature is trivial. ◀

▶ **Example 33.** It is instructive to consider the the multiset functor B. It maps finite spaces to finite spaces but we ignore its behaviour on other subcategories, hence the signature:



It is only known to be $\mathbf{Pol}_f$-continuous ([4]). Therefore, GB is a valid model functor but BG breaks the third premise of the composition rule in Figure 3: indeed, G maps finite spaces to compact spaces (Figure 1).

▶ **Definition 34.** The set of models is defined to be that of typeable functors and will be denoted by $\mathcal{M}$.

Natural parameterised models

Theorem 29 and 32 delineate a class of parameters and models to which the Machine (Theorem 13) applies. These combined results can be reframed concisely as follows:

▶ **Theorem 35.** *For all parameterisation $P \in \mathcal{P}$ and all model $M \in \mathcal{M}$,*

$$Nat(P, M) \cong Nat(P|_{\mathbf{Pol}_f}, M|_{\mathbf{Pol}_f}).$$

5 Applications

It is hard to overstate the importance of independently and identically distributed (*i.i.d.*) sequences of random of variables and their generalisation to exchangeable processes to probability and statistics, as witnessed by the wealth of powerful asymptotic results which apply to them – to name a few, the law of large number, the central limit theorem and the de Finetti theorem [12]. We illustrate the usefulness of our framework by recasting i.i.d. processes and the de Finetti theorem as instances of our parameterised models.

5.1 The *iid* natural transformation

Let X be finite Polish. For all integer $n > 0$, we construct an arrow $iid^n_X : \mathsf{G}(X) \to \mathsf{G}(X^n)$ as follows: $iid^n_X(p) = (B_1, B_2, \ldots, B_n) \mapsto p(B_1) \cdot p(B_2) \cdots p(B_n)$. One easily verifies that this map is well-typed and continuous. We have the following result:

▶ **Proposition 36.** *For all all positive integer n, the family iid^n_X defines a natural transformation $iid^n : \mathsf{G} \Rightarrow \mathsf{G}(-^n)$.*

Proof. Let X, Y be finite spaces and let $f : X \to Y$ be a function. Let $p \in \mathsf{G}(X)$ be given and let $(y_1, \ldots, y_n)$ be a sequence in Y^n. We have:

$$\begin{aligned}(\mathsf{G}(f^n) \circ iid^n_X)(p)(y_1, \ldots, y_n) &= (iid^n_X(p) \circ (f^n)^{-1})(y_1, \ldots, y_n) \\ &= p(f^{-1}(y_1)) \cdot p(f^{-1}(y_2)) \cdots p(f^{-1}(y_n)) \\ &= iid^n_Y(\mathsf{G}(f)(p))(y_1, \ldots, y_n)\end{aligned}$$

We have proved that the iid^n transformation is well defined on all finite spaces. One easily checks that G is a $\mathcal{P}$-functor and $\mathsf{G}(-^n)$ is a $\mathcal{M}$-functor. Applying Theorem 35, we conclude that iid^n admits an unique extension to the whole of $\mathbf{Pol}$. ◀

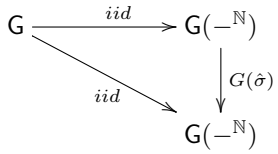


Figure 4 *iid* is exchangeable.

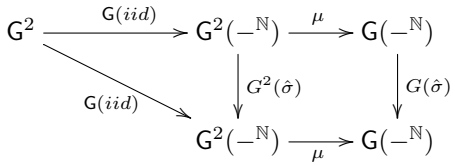


Figure 5 Mixtures of *iid* are exchangeable.

The family of natural transformations $\{iid^n\}_{n>0}$ can in turn be extended to a natural transformation $iid : \mathsf{G} \Rightarrow \mathsf{G}(-^{\mathbb{N}})$. The following result relies on the Bochner extension theorem ([5], Th. 2.5) along with the naturality of the canonical projections $\pi_n : -^{\mathbb{N}} \Rightarrow -^n$ and $\pi_{nm} : -^n \Rightarrow -^m$.

▶ **Proposition 37.** *There exists a unique natural transformation* $iid : \mathsf{G} \Rightarrow \mathsf{G}(-^{\mathbb{N}})$ *such that for all* n, $iid^n = \mathsf{G}(\pi_n) \circ iid$.

5.2 Exchangeable measures and the de Finetti theorem

For all $n > 0$, we denote by S_n the symmetric group on $\{1, \ldots, n\}$. Each $\sigma \in S_n$ induces a natural transformation $\hat{\sigma} : -^{\mathbb{N}} \Rightarrow -^{\mathbb{N}}$ whose component at X is defined by $\hat{\sigma}_X(x_1, \ldots, x_n, \ldots) = (x_{\sigma(1)}, \ldots, x_{\sigma(n)}, x_{n+1}, \ldots)$. As illustrated by the commutative diagram in Figure 4, the distribution of *iid* is invariant by the action of such permutations. Elements of $\mathsf{G}(X^{\mathbb{N}})$ invariant by the action of $\mathsf{G}(\hat{\sigma})$ for all n and all $\sigma \in S_n$ are called *exchangeable measures*. The diagram in Figure 4 indicates that *iid* is a natural family of exchangeable measures. An easy computation shows that the same property is verified by *mixtures of iids* (see Figure 5). The de Finetti theorem states that *all* exchangeable measures can be represented as such mixtures of *iid*s. We will prove that this representation is *natural*. We introduce a functor mapping any Polish space X to the space of exchangeable measures $\mathsf{G}_{ex}(X^{\mathbb{N}})$. Exchangeable measures form a closed convex subset of $\mathsf{G}(X^{\mathbb{N}})$, therefore they form a Polish space when given the subspace topology. We have the following result:

▶ **Proposition 38.** $\mathsf{G}_{ex}(-^{\mathbb{N}})$ *is a* $\mathcal{P}$*-functor.*

Proof. Note that $\mathsf{G}_{ex}(-^{\mathbb{N}})$ is a subfunctor of $\mathsf{G}(-^{\mathbb{N}})$ which is a $\mathcal{P}$-functor (Proposition 18). Z-cocontinuity is easily seen to be preserved by subfunctors. Preservation of epis follow from the measurable selection theorem and naturality of $\hat{\sigma}$ for all $\sigma \in S_n$. ◀

Let us introduce another natural transformation: for all $n > 0$, the *empirical measure* $\mathcal{E}_n : -^n \Rightarrow \mathsf{G}$ computes the relative frequencies of elements of a sequence and is defined by $\mathcal{E}_{n,X}(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}$. For all n, we define the *random empirical measure at time* n $\mathsf{G}(\mathcal{E}_n \circ \pi_n) : \mathsf{G}_{ex}(-^{\mathbb{N}}) \to \mathsf{G}^2$. This defines a sequence of natural transformations indexed by n. The de Finetti theorem gives us the following:

▶ **Theorem 39** (de Finetti [12])**.** *Let* X *be Polish.*

- *For all* $P \in \mathsf{G}_{ex}(X^{\mathbb{N}})$*, the limit* $deF_X(P) \triangleq \lim_n \mathsf{G}(\mathcal{E}_n \circ \pi_n)(P)$ *exists in* $\mathsf{G}^2(X)$*;*
- *the associated map* $P \mapsto deF_X(P)$ *is continuous from* $\mathsf{G}_{ex}(X^{\mathbb{N}})$ *to* $\mathsf{G}^2(X)$*;*
- *exchangeable probabilities are mixtures of iids:* $\mu_X \circ \mathsf{G}(iid_X) \circ deF_X = id_{\mathsf{G}_{ex}(X^{\mathbb{N}})}$.

Given this, we can easily prove the following:

▶ **Theorem 40.** *The family of maps deF constructed in Theorem 39 is a natural transformation from* $\mathsf{G}_{ex}(-^{\mathbb{N}})$ *to* G^2.

Proof. Let $f : X \to Y$ be a continuous map. We have:

$$\begin{array}{rcll}
(\mathsf{G}^2(f) \circ deF_X)(P) & = & \mathsf{G}^2(f)(\lim_n \mathsf{G}(\mathcal{E}_{n,X} \circ \pi_n)(P)) & \\
& = & \lim_n \mathsf{G}^2(f)(\mathsf{G}(\mathcal{E}_{n,X} \circ \pi_n)(P)) & \text{(Continuity)} \\
& = & \lim_n \mathsf{G}(\mathcal{E}_{n,X}\pi_n \circ f^{\mathbb{N}})(P) & \text{(Naturality)} \\
& = & \lim_n \mathsf{G}(\mathcal{E}_{n,X}\pi_n)(\mathsf{G}_{ex}(f^{\mathbb{N}})(P)) & \\
& = & deF_Y(\mathsf{G}_{ex}(f^{\mathbb{N}})(P)) &
\end{array}$$

◀

This result together with Proposition 38 and Theorem 35 implies that the de Finetti transformation is entirely characterised by its finite components. Concretely, it is enough to prove the de Finetti theorem on finite spaces for our framework to extend it to arbitrary Polish spaces.

6 Conclusion

We have proposed a type theory for Polish spaces with the essential functors of the modeling trade: Vietoris, Giry, and Skorokhod. Thus one can re-construct all classical models of mixed probabilistic and non-determinism within our grammar, guaranteeing an adequate level of expressivity for parameterised models. Not only do we subsume classical constructions studied in concurrency theory, but the compass of our type theory also includes probabilities on functions spaces which are hot pursuits in probabilistic modeling - e.g. solutions of stochastic differential equations. For this fledgling type theory, we provide a "normalisation theorem" showing that existence and equality between our structure arrows is completely determined by the finite case.

However, in many ways this type theory is still a draft. An axiomatic or syntactic treatment is not yet in order, as one needs first to decide a certain number of questions which this contribution has left unanswered. For instance, we do not know if parameter functors are closed under products. Further progress might hint at natural such treatments.

References

1. I. Beylin and P. Dybjer. Extracting a proof of coherence for monoidal categories from a proof of normalization for monoids. In *Types for Proofs and Programs*, pages 47–61. Springer Berlin Heidelberg, 1995.
2. P. Billingsley. *Convergence of Probability Measures*. Wiley, 1968.
3. N. Bourbaki. *Elements de mathématique. Topologie Générale*. Springer, 1971.
4. F. Dahlqvist, V. Danos, and I. Garnier. Giry and the machine. Feb 2016. To appear in the proceedings of MFPS XXXII.
5. V. Danos and I. Garnier. Dirichlet is natural. *Electronic Notes in Theoretical Computer Science*, 319:137 – 164, 2015. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

6 J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

7 S.N. Ethier and T.G. Kurtz. *Markov processes: characterization and convergence*. Probability and mathematical statistics. Wiley, 1986.

8 B. A. Frigyik, A. Kapila, and M.R. Gupta. Introduction to the Dirichlet distribution and related processes. Technical report, University of Washington., 2010.

9 M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, number 915 in Lecture Notes In Math., pages 68–85. Springer-Verlag, 1981.

10 J. Goubault-Larrecq and K. Keimel. Choquet–Kendall–Matheron theorems for non-Hausdorff spaces. *Mathematical Structures in Computer Science*, 21(03):511–561, 2011.

11 P.T. Johnstone. *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

12 O. Kallenberg. *Probabilistic Symmetries and Invariance Principles*. Number v. 10 in Applied probability. Springer, 2005.

13 A. S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Text in Mathematics*. Springer, 1995.

14 K. Keimel and G. Plotkin. Predicate transformers for extended probability and non-determinism. *Mathematical Structures in Computer Science*, 19(03):501–539, 2009.

15 S. Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 1998.

16 R. Mardare, P. Panangaden, and G. Plotkin. Quantitative algebraic reasoning. *To appear in LICS 2016*, 2016.

17 K.R. Parthasarathy. *Probability Measures on Metric Spaces*. AMS Chelsea Publishing Series. Academic Press, 1972.

18 R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

19 R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and non-determinism. *Electronic Notes in Th. Comp. Sc.*, 222:3–99, 2009.

20 F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331(1):115–142, 2005.

Coalgebraic Trace Semantics for Büchi and Parity Automata*

Natsuki Urabe[†1], Shunsuke Shimizu[2], and Ichiro Hasuo[3]

1 **Department of Computer Science, The University of Tokyo, Japan and JSPS Research Fellow**
urabenatsuki@is.s.u-tokyo.ac.jp
2 **Department of Computer Science, The University of Tokyo, Japan**
shunsuke@is.s.u-tokyo.ac.jp
3 **Department of Computer Science, The University of Tokyo, Japan**
ichiro@is.s.u-tokyo.ac.jp

Abstract

Despite its success in producing numerous general results on state-based dynamics, the theory of *coalgebra* has struggled to accommodate the *Büchi acceptance condition*—a basic notion in the theory of automata for infinite words or trees. In this paper we present a clean answer to the question that builds on the "maximality" characterization of infinite traces (by Jacobs and Cîrstea): the accepted language of a Büchi automaton is characterized by two commuting diagrams, one for a *least* homomorphism and the other for a *greatest*, much like in a system of (least and greatest) fixed-point equations. This characterization works uniformly for the nondeterministic branching and the probabilistic one; and for words and trees alike. We present our results in terms of the *parity* acceptance condition that generalizes Büchi's.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases coalgebra, Büchi/parity/probabilistic/tree automaton

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.24

1 Introduction

Büchi Automata. *Automata* are central to theoretical computer science. Besides their significance in formal language theory and as models of computation, many *formal verification* techniques rely on them, exploiting their balance between expressivity and tractable complexity of operations on them. See e.g. [30, 12]. Many current problems in verification are about *nonterminating* systems (like servers); for their analyses, naturally, automata that classify *infinite* objects—such as infinite words and infinite trees—are employed.

The *Büchi acceptance condition* is the simplest nontrivial acceptance condition for automata for infinite objects. Instead of requiring finally reaching an accepting state ◎—which makes little sense for infinite words/trees—it requires accepting states visited *infinitely often*. This simple condition, too, has proved both expressive and computationally tractable: for the word case the Büchi condition can express any ω-regular properties; and the emptiness problem for Büchi automata can be solved efficiently by searching for a *lasso* computation.

* The authors are supported by Grants-in-Aid No. 24680001 & 15KT0012, JSPS.
† N.U. is supported by Grant-in-Aid for JSPS Fellows.

Coalgebras. Studies of automata and state-based transition systems in general have been shed a fresh *categorical* light in 1990's, by the theory of *coalgebra*. Its simple modeling of state-based dynamics—as a *coalgebra*, i.e. an arrow $c\colon X \to FX$ in a category $\mathbb{C}$—has produced numerous results that capture mathematical essences and provide general techniques. Among its basic results are: *behavior-preserving maps* as homomorphisms; a *final coalgebra* as a fully abstract domain of behaviors; *coinduction* (by finality) as definition and proof principles; a general span-based definition of *bisimulation*; etc. See e.g. [16, 22]. More advanced results are on: coalgebraic modal logic (see e.g. [7]); process algebras and congruence formats (see e.g. [18]); generalization of Kleene's theorem (see e.g. [24]); etc.

Büchi Automata, Coalgebraically. In the coalgebra community, however, two important phenomena in automata and/or concurrency have been known to be hard to model—many previous attempts have seen only limited success. One is *internal (τ-)transitions* and *weak (bi)similarity*; see e.g. recent [11]. The other one is the Büchi acceptance condition.

Here is a (sketchy) explanation why these two phenomena should be hard to model coalgebraically. The theory of coalgebra is centered around *homomorphisms* as behavior-preserving maps; see the diagram on the right. Deep rooted in it is the idea of *local matching* between one-step transitions in c and those in d. This is what fails in the two phenomena: in weak bisimilarity a one-step transition in c is matched by a possibly multi-step transition in d; and the Büchi acceptance condition—stipulating that accepting states are visited *infinitely often*, in the long run—is utterly *nonlocal*.

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle d} \\ X & \xrightarrow[f]{} & Y \end{array}$$

There have been some works that study Büchi acceptance conditions (or more general *parity* or *Muller* conditions) in coalgebraic settings. One is [5], where they rely on the lasso characterization of nonemptiness and use $\mathbf{Sets}^2$ as a base category. Another line is on *coalgebra automata* (see e.g. [31]), where however Büchi/parity/Muller acceptance conditions reside outside the realm of coalgebras.[1] Inspired by these works, and also by our work [14] on alternating fixed points and coalgebraic model checking, the current paper introduces a coalgebraic modeling of Büchi and parity automata based on *systems of fixed-point equations*.

Contributions. We present a clean answer to the question of "Büchi automata, coalgebraically," relying on the previous work on coalgebraic infinitary trace semantics [15, 6] and fixed-point equations [14]. Our modeling, hinted in (1), features: 1) accepting states as a *partition* of a state space; and 2) explicit use of μ and ν—for least/greatest fixed points—in diagrams. We state our results for the *parity* condition (that generalizes the Büchi one).

$$\boxed{\begin{array}{c} \overline{F}X \longrightarrow\!\!\!\!\!\rightarrow \overline{F}Z \\ {\scriptstyle c}\uparrow \quad =_{\nu} \quad \cong\uparrow J\zeta \\ X \xrightarrow[\mathsf{tr}^{\infty}(c)]{} Z \end{array} \quad \text{in a Kleisli category } \mathcal{K}\ell(T)}$$

Characterization of languages under no (i.e. the trivial) acceptance condition [15, 6]

$$\Longrightarrow \quad \boxed{\begin{array}{ccc} \overline{F}X \longrightarrow\!\!\!\!\!\rightarrow \overline{F}Z & & \overline{F}X \longrightarrow\!\!\!\!\!\rightarrow \overline{F}Z \\ {\scriptstyle c_1}\uparrow \quad =_{\mu} \quad \cong\uparrow J\zeta & & {\scriptstyle c_2}\uparrow \quad =_{\nu} \quad \cong\uparrow J\zeta \\ X_1 \xrightarrow[\mathsf{tr}^{\mathrm{p}}(c_1)]{} Z & & X_2 \xrightarrow[\mathsf{tr}^{\mathrm{p}}(c_2)]{} Z \end{array}}$$

Under the Büchi acceptance condition, with $X_1 = \{\bigcirc\text{'s}\}$ and $X_2 = \{\textcircled{\scriptsize◎}\text{'s}\}$

(1)

Our framework is generic: its leading examples are *nondeterministic* and (generative) *probabilistic* tree automata, with the Büchi/parity acceptance condition.

Our contributions are: 1) coalgebraic modeling of automata with the Büchi/parity conditions; 2) characterizing their accepted languages by diagrams with μ's and ν's (tr^{p} in (1));

[1] More precisely: a coalgebra automaton is an automaton (with Büchi/parity/Muller acceptance conditions) that *classifies* coalgebras (as generalization of words and trees). A coalgebra automaton itself is *not* described as a coalgebra; nor is its acceptance condition.

and 3) proving that the characterization indeed captures the conventional definitions. The last "sanity-check" proves to be intricate in the probabilistic case, and our proof—relying on previous [6, 23]—identifies the role of *final sequences* [32] in probabilistic processes.

With explicit μ's and ν's—that specify in *which* homomorphism, among *many* that exist, we are interested—we depart from the powerful reasoning principle of *finality* (existence of a unique homomorphism). We believe this is a necessary step forward, for the theory of coalgebra to take up long-standing challenges like the Büchi condition and weak bisimilarity. Our characterization (1)—although it is not so simple as the uniqueness argument by finality—seems useful, too: we have obtained some results on *fair simulation* notions between Büchi automata [28], following the current work.

Organization of the Paper. In Section 2 we provide backgrounds on: the coalgebraic theory of trace in a *Kleisli category* [15, 6] (where we explain the diagram on the left in (1)); and systems of fixed-point equations. In Section 3 we present a coalgebraic modeling of Büchi/parity automata and their languages. Coincidence with the conventional definitions is shown in Section 4 for the nondeterministic setting, and in Section 5 for the probabilistic one.

Most proofs are deferred to the appendices, that are found in [27].

Future Work. Here we are based on the coalgebraic theory of trace and simulation [21, 15, 13, 25]; it has been developed under the *trivial* acceptance condition (any run that does not diverge, i.e. that does not come to a deadend, is accepted). The current paper is about accommodating the Büchi/parity conditions in the *trace* part of the theory; for the *simulation* part we also have exploited the current results to obtain sound *fair simulation* notions for nondeterministic Büchi tree automata and probabilistic Büchi word automata [28].

On the practical side our future work mainly consists of proof methods for *trace/language inclusion*, a problem omnipresent in formal verification. *Simulations*—as one-step, local witnesses for trace inclusion—have been often used as a sound (but not necessarily complete) proof method that is computationally more tractable; with the observations in [28] we are naturally interested in them. Possible directions are: synthesis of *simulation matrices* between finite systems by linear programming, like in [26]; synthesis of simulations by other optimization techniques for program verification (where problem instances are infinite due to the integer type); and simulations as a proof method in interactive theorem proving.

2 Preliminaries

2.1 Coalgebras in a Kleisli Category

We assume some basic category theory, most of which is covered in [16].

The conventional coalgebraic modeling of systems—as a function $X \to FX$—is known to capture *branching-time* semantics (such as bisimilarity) [16, 22]. In contrast accepted languages of Büchi automata (with nondeterministic or probabilistic branching) constitute *linear-time* semantics; see [29] for the so-called *linear time-branching time spectrum*.

For the coalgebraic modeling of such linear-time semantics we follow the "Kleisli modeling" tradition [21, 15, 13]. Here a system is parametrized by a monad T and an endofunctor F on **Sets**: the former represents the *branching type* while the latter represents the *(linear-*

time) transition type; and a system is modeled as a function of the type $X \to TFX$.[2]

A function $X \to TFX$ is nothing but an $\overline{F}$*-coalgebra* $X \nrightarrow \overline{F}X$ in the *Kleisli category* $\mathcal{K}\ell(T)$—where $\overline{F}$ is a suitable lifting of F. This means we can apply the standard coalgebraic machinery to linear-time behaviors, by changing the base category from **Sets** to $\mathcal{K}\ell(T)$.

A monad $T = (T, \eta, \mu)$ on a category $\mathbb{C}$ induces the *Kleisli category* $\mathcal{K}\ell(T)$. The objects of $\mathcal{K}\ell(T)$ are the same as $\mathbb{C}$'s; and for each pair X, Y of objects, the homset $\mathcal{K}\ell(T)(X, Y)$ is given by $\mathbb{C}(X, TY)$. An arrow $f \in \mathcal{K}\ell(T)(X, Y)$—that is $X \to TY$ in $\mathbb{C}$—is called a *Kleisli arrow* and is denoted by $f : X \nrightarrow Y$ for distinction. Given two successive Kleisli arrows $f : X \nrightarrow Y$ and $g : Y \nrightarrow Z$, their *Kleisli composition* is given by $\mu_Z \circ Tg \circ f : X \nrightarrow Z$ (where $\circ$ is composition in $\mathbb{C}$). This composition in $\mathcal{K}\ell(T)$ is denoted by $g \odot f$ for distinction. The *Kleisli inclusion* $J : \mathbb{C} \to \mathcal{K}\ell(T)$ is defined by $J(X) = X$ and $J(f) = \eta_Y \circ f : X \nrightarrow Y$.

In this paper we mainly use two combinations of T and F. The first is the *powerset monad* $\mathcal{P}$ and a polynomial functor on **Sets**; the second is the *(sub-)Giry monad* [10] $\mathcal{G}$ and a polynomial functor on **Meas**, the category of measurable spaces and measurable functions. The *Giry monad* [10] is commonly used for modeling (not necessarily discrete) probabilistic processes. We shall use its "sub" variant; a *subprobability measure* over $(X, \mathfrak{F}_X)$ is a measure μ such that $0 \leq \mu(X) \leq 1$ (we do not require $\mu(X) = 1$).

▶ **Definition 2.1** $(\mathcal{P}, \mathcal{G})$**.** The *powerset monad* $\mathcal{P}$ on **Sets** is: $\mathcal{P}X = \{A \subseteq X\}$; $(\mathcal{P}f)(A) = \{f(x) \mid x \in A\}$; its unit is $\eta^{\mathcal{P}}_X(x) = \{x\}$; and its multiplication is $\mu^{\mathcal{P}}_X(M) = \bigcup_{A \in M} A$.

The *sub-Giry monad* is a monad $\mathcal{G} = (\mathcal{G}, \eta^{\mathcal{G}}, \mu^{\mathcal{G}})$ on **Meas** such that $\mathcal{G}(X, \mathfrak{F}_X) = (\mathcal{G}X, \mathfrak{F}_{\mathcal{G}X})$, where $\mathcal{G}X$ is the set of all *subprobability measures* on $(X, \mathfrak{F}_X)$, and $\mathfrak{F}_{\mathcal{G}X}$ is the smallest σ-algebra such that, for each $S \in \mathfrak{F}_X$, the function $\mathrm{ev}_S : \mathcal{G}X \to [0, 1]$ defined by $\mathrm{ev}_S(P) = P(S)$ is measurable. Moreover, $\eta^{\mathcal{G}}_{(X,\mathfrak{F}_X)}(x)(S)$ is 1 if $x \in S$ and 0 otherwise (the *Dirac* distribution), and $\mu^{\mathcal{G}}_{(X,\mathfrak{F}_X)}(\Psi)(S) = \int_{\mathcal{G}(X,\mathfrak{F}_X)} \mathrm{ev}_S \, d\Psi$.

▶ **Definition 2.2** (polynomial functors on **Sets** and **Meas**)**.** A *polynomial functor* F on **Sets** is defined by the BNF notation $F ::= \mathrm{id} \mid A \mid F_1 \times F_2 \mid \coprod_{i \in I} F_i$. Here $A \in$ **Sets**.

A *(standard Borel) polynomial functor* F on **Meas** is defined by the BNF notation $F ::= \mathrm{id} \mid (A, \mathfrak{F}_A) \mid F_1 \times F_2 \mid \coprod_{i \in I} F_i$. Here I is countable, and we require each constant $(A, \mathfrak{F}_A) \in$ **Meas** be a *standard Borel space* (see e.g. [9]). The σ-algebra $\mathfrak{F}_{FX}$ associated to FX is defined as usual, with (co)product σ-algebras, etc. F's action on arrows is obvious.

A standard Borel polynomial functor shall often be called simply a *polynomial functor*.

The technical requirement of being standard Borel—meaning that it arises from a *Polish space* [9]—will be used in the probabilistic setting of Section 5; we follow [6, 23] in its use.

There is a well-known correspondence between a polynomial functor and a *ranked alphabet*—a set Σ with an *arity map* $|\_| : \Sigma \to \mathbb{N}$. In this paper a functor F (for the linear-time behavior type) is restricted to be polynomial; this essentially means that we are dealing with systems that generate *trees* over some ranked alphabet (with additional T-branching).

▶ **Definition 2.3** (Tree_Σ)**.** An *(infinitary)* Σ*-tree*, as in the standard definition, is a possibly infinite tree whose nodes are labeled with the ranked alphabet Σ and whose branching degrees are consistent with the arity of labels. The set of Σ-trees is denoted by Tree_Σ.

[2] Another eminent approach to coalgebraic linear-time semantics is the *Eilenberg-Moore* one (see e.g. [17, 1]): notably in the latter a system is expressed as $X \to FTX$. The Eilenberg-Moore approach can be seen as a categorical generalization of *determinization* or the *powerset construction*. It is however not clear how determinization serves our current goal (namely a coalgebraic modeling of the Büchi/parity acceptance conditions).

Table 1 Overview of existing results on coalgebraic trace semantics.

Semantics	Finite trace	Infinitary trace
Coalgebraic modeling	$\begin{array}{ccc} \overline{F}X & \xrightarrow{\overline{F}(\mathsf{tr}(c))} & \overline{F}A \\ {\scriptstyle c}\uparrow & = & \cong\uparrow{\scriptstyle J\alpha^{-1}} \\ X & \xrightarrow[\mathsf{tr}(c)]{} & A \end{array}$ final (3)	$\begin{array}{ccc} \overline{F}X & \xrightarrow{\overline{F}(\mathsf{tr}^\infty(c))} & \overline{F}Z \\ {\scriptstyle c}\uparrow & =_\nu & \cong\uparrow{\scriptstyle J\zeta} \\ X & \xrightarrow[\mathsf{tr}^\infty(c)]{} & Z \end{array}$ weakly final (4)
	Finality in $\mathcal{K}\ell(T)$ (Theorem 2.7)	(Weak finality + maximality) in $\mathcal{K}\ell(T)$ (Theorem 2.8)

▶ **Lemma 2.4.** *Let Σ be a ranked alphabet, and $F_\Sigma = \coprod_{\sigma\in\Sigma}(\_)^{|\sigma|}$ be the corresponding polynomial functor on* **Sets**. *The set Tree_Σ of (infinitary) Σ-trees carries a final F_Σ-coalgebra. The same holds in* **Meas**, *for countable Σ and the corresponding polynomial functor F_Σ.* ◀

We collect some standard notions and notations for such trees in Appendix A in [27].

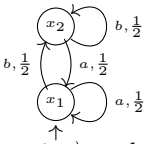
It is known [13, 25] that for $(\mathbb{C}, T) \in \{(\mathbf{Sets}, \mathcal{P}), (\mathbf{Meas}, \mathcal{G})\}$ and polynomial F on $\mathbb{C}$, there is a canonical *distributive law* [20] $\lambda\colon FT \Rightarrow TF$—a natural transformation compatible with T's monad structure. Such λ induces a functor $\overline{F}\colon \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ that makes the diagram (2) commute.

$$\begin{array}{ccc} \mathcal{K}\ell(T) & \xrightarrow{\overline{F}} & \mathcal{K}\ell(T) \\ {\scriptstyle J}\uparrow & & \uparrow{\scriptstyle J} \\ \mathbb{C} & \xrightarrow{F} & \mathbb{C} \end{array} \quad (2)$$

Using this *lifting* $\overline{F}$ of F from $\mathbb{C}$ to $\mathcal{K}\ell(T)$, an arrow $c : X \to TFX$ in $\mathbb{C}$—that is how we model an automaton—can be regarded as an $\overline{F}$*-coalgebra* $c\colon X \nrightarrow \overline{F}X$ in $\mathcal{K}\ell(T)$.

Then the dynamics of $\mathcal{A}$—ignoring its initial and accepting states—is modeled as an $\overline{F}$-coalgebra $c : X \nrightarrow \overline{F}X$ in $\mathcal{K}\ell(\mathcal{P})$ where: $F = \{a, b\} \times (\_)$, $X = \{x_1, x_2\}$ and $c : X \to \mathcal{P}FX$ is the function $c(x_1) = c(x_2) = \{(a, x_1), (b, x_2)\}$. The information on initial and accepting states is redeemed later in

▶ Example 2.5. Let $\mathcal{M}$ be the Markov chain on the right. The dynamics of $\mathcal{M}$ is modeled as an $\overline{F}$-coalgebra $c : X \nrightarrow \overline{F}X$ in $\mathcal{K}\ell(\mathcal{G})$ where: $F = \{a, b\} \times (\_)$, $X = \{x_1, x_2\}$ with the discrete measurable structure, and $c\colon X \to \mathcal{G}FX$ is the (measurable) function defined by $c(x)\big\{(a, x_1)\big\} = c(x)\big\{(b, x_2)\big\} = 1/2$, and $c(x)\big\{(d, x')\big\} = 0$ for the other $(d, x') \in \{a, b\} \times X$.



Later we will equip Markov chains with accepting states and obtain *(generative) probabilistic Büchi automata*. Their probabilistic accepted languages will be our subject of study.

▶ Remark 2.6. Due to the use of the *sub*-Giry monad is that, in $f\colon X \nrightarrow Y$ in $\mathcal{K}\ell(\mathcal{G})$, the probability $f(x)(Y)$ can be smaller than 1. The missing $1 - f(x)(Y)$ is understood as that for *divergence*. In the nondeterministic case $f\colon X \nrightarrow Y$ in $\mathcal{K}\ell(\mathcal{P})$ *diverges* at x if $f(x) = \emptyset$.

This is in contrast with a system coming to halt generating a 0-ary symbol (such as ✓ in (5) later); this is deemed as *successful termination*.

2.2 Coalgebraic Theory of Trace

The above "Kleisli" coalgebraic modeling has produced some general results on: linear-time process semantics (called *trace semantics*); and *simulations* as witnesses of trace inclusion, generalizing the theory in [19]. Here we review the former; it underpins our developments later. A rough summary is in Table 1: typically the results apply to $T \in \{\mathcal{P}, \mathcal{D}, \mathcal{G}\}$—where $\mathcal{D}$ is the *subdistribution monad* on **Sets**, a discrete variant of $\mathcal{G}$—and polynomial F. In what follows we present these previous results in precise terms, sometimes strengthening the assumptions for the sake of presentation. The current paper's goal is to incorporate the Büchi acceptance condition in (the right column of) Table 1.

Firstly, *finite trace semantics*—linear-time behaviors that eventually terminate, such as the accepted languages of *finite* words for NFAs—is captured by finality in $\mathcal{K}\ell(T)$.

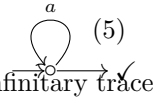
▶ **Theorem 2.7** ([13]). *Let $T \in \{\mathcal{P}, \mathcal{D}\}$ and F be a polynomial functor on* **Sets**. *An initial F-algebra $\alpha\colon FA \xrightarrow{\cong} A$ in* **Sets** *yields a final $\overline{F}$-coalgebra in $\mathcal{K}\ell(T)$, as in (3) in Table 1.* ◀

The carrier A of an initial F-algebra in **Sets** is given by finite words/trees (over the alphabet that corresponds to F). The significance of Theorem 2.7 is that: for many examples, the unique homomorphism $\mathsf{tr}(c)$ induced by finality (3) captures the finite trace semantics of the system c. Here the word "finite" means that we collect only behaviors that eventually terminate.

What if we are also interested in *nonterminating* behaviors, like the infinite word $b^\omega = bbb\dots$ accepted by the automaton in Example 2.5? There is a categorical characterization of such *infinitary trace semantics* too, although proper finality is now lost.

▶ **Theorem 2.8** ([15, 6, 25]). *Let $(\mathbb{C}, T) \in \{(\mathbf{Sets}, \mathcal{P}), (\mathbf{Meas}, \mathcal{G})\}$ and F be a polynomial functor on $\mathbb{C}$. A final F-coalgebra $\zeta\colon Z \xrightarrow{\cong} FZ$ in $\mathbb{C}$ gives rise to a weakly final $\overline{F}$-coalgebra in $\mathcal{K}\ell(T)$, as in (4) in Table 1. Moreover, the coalgebra $J\zeta$ additionally admits the* greatest *homomorphism $\mathsf{tr}^\infty(c)$ with respect to the pointwise order $\sqsubseteq$ in the homsets of $\mathcal{K}\ell(T)$ (given by inclusion for $T = \mathcal{P}$, and by pointwise $\leq$ on subprobability measures for $T = \mathcal{G}$). That is: for each homomorphism f from c to $J\zeta$ we have $f \sqsubseteq \mathsf{tr}^\infty(c)$.* ◀

In many examples the greatest homomorphism $\mathsf{tr}^\infty(c)$ captures the infinitary trace semantics of the system c. (Here by *infinitary* we mean *both finite and infinite* behaviors.) For example, for the system (5) where ✓ denotes successful termination, its finite trace semantics is $\{\varepsilon, a, aa, \dots\}$ whereas its infinitary trace semantics is $\{\varepsilon, a, aa, \dots\} \cup \{a^\omega\}$. The latter is captured by the diagram (4), with $T = \mathcal{P}$ and $F = \{\checkmark\} + \{a\} \times (\_)$.



(5)

2.3 Equational Systems for Alternating Fixed Points

Nested, alternating greatest and least fixed points—as in a μ-calculus formula $\nu u_2.\mu u_1.(p \wedge u_2) \vee \Box u_1$—are omnipresent in specification and verification. For their relevance to the Büchi/parity acceptance condition one can recall the well-known translation of LTL formulas to Büchi automata and vice versa (see e.g. [30]). To express such fixed points we follow [8, 2] and use *equational systems*—we prefer them to the textual μ-calculus-like presentations.

▶ **Definition 2.9** (equational system). Let $L_1, \dots, L_n$ be posets. An *equational system E* over $L_1, \dots, L_n$ is an expression

$$u_1 =_{\eta_1} f_1(u_1, \dots, u_n) \ , \quad \dots, \quad u_n =_{\eta_n} f_n(u_1, \dots, u_n) \tag{6}$$

where: $u_1, \dots, u_n$ are *variables*, $\eta_1, \dots, \eta_n \in \{\mu, \nu\}$, and $f_i\colon L_1 \times \dots \times L_n \to L_i$ is a monotone function. A variable u_j is a *μ-variable* if $\eta_j = \mu$; it is a *ν-variable* if $\eta_j = \nu$.

The *solution* of the equational system E is defined as follows, under the assumption that L_i's have enough supremums and infimums. It proceeds as: 1) we solve the first equation to obtain an interim solution $u_1 = l_1^{(1)}(u_2, \dots, u_n)$; 2) it is used in the second equation to eliminate u_1 and yield a new equation $u_2 =_{\eta_2} f_2^\sharp(u_2, \dots, u_n)$; 3) solving it again gives an interim solution $u_2 = l_2^{(2)}(u_3, \dots, u_n)$; 4) continuing this way from left to right eventually eliminates all variables and leads to a closed solution $u_n = l_n^{(n)} \in L_n$; and 5) by propagating these closed solutions back from right to left, we obtain closed solutions for all of $u_1, \dots, u_n$. A precise definition is found in Appendix B in [27].

It is important that the order of equations *matters*: for $(u =_\mu v, v =_\nu u)$ the solution is $u = v = \top$ while for $(v =_\nu u, u =_\mu v)$ the solution is $u = v = \bot$.

Whether a solution is well-defined depends on how "complete" the posets $L_1, \dots, L_n$ are. It suffices if they are *complete lattices*, in which case every monotone function $L_i \to L_i$ has greatest/least fixed points (the *Knaster-Tarski* theorem). This is used in the nondeterministic setting: note that $\mathcal{P}Y$, hence the homset $\mathcal{K}\ell(\mathcal{P})(X,Y)$, are complete lattices.

▶ **Lemma 2.10.** *The system E (6) has a solution if each L_i is a complete lattice.* ◀

This does not work in the probabilistic case, since the homsets $\mathcal{K}\ell(\mathcal{G})(X,Y) = \mathbf{Meas}(X, \mathcal{G}Y)$ with the pointwise order—on which we consider equational systems—are not complete lattices. For example $\mathcal{G}Y$ lacks the greatest element in general; even if $Y = 1$ (when $\mathcal{G}1 \cong [0,1]$), the homset $\mathcal{K}\ell(\mathcal{G})(X,1)$ can fail to be a complete lattice. See Example B.2 in [27]. Our strategy is: 1) to apply the following *Kleene*-like result to the homset $\mathcal{K}\ell(\mathcal{G})(X,1)$; and 2) to "extend" fixed points in $\mathcal{K}\ell(\mathcal{G})(X,1)$ along a final F-sequence. See Section 5.1 later.

▶ **Lemma 2.11.** *The equational system E (6) has a solution if: each L_i is both a pointed ω-cpo and a pointed ω^{op}-cpo; and each f_i is both ω-continuous and ω^{op}-continuous.* ◀

In Appendix B in [27] we have additional lemmas on "homomorphisms" of equational systems and preservation of solutions. They play important roles in the proofs of the later results.

3 Coalgebraic Modeling of Parity Automata and Its Trace Semantics

Here we present our modeling of Büchi/parity automata. We shall do so axiomatically with parameters $\mathbb{C}$, T and F—much like in Section 2.1–2.2. Our examples cover: both nondeterministic and probabilistic branching; and automata for trees (hence words as a special case).

▶ Assumptions 3.1. In what follows a monad T and an endofunctor F, both on $\mathbb{C}$, satisfy:

- The base category $\mathbb{C}$ has a final object 1 and finite coproducts.
- The functor F has a final coalgebra $\zeta\colon Z \to FZ$ in $\mathbb{C}$.
- There is a *distributive law* $\lambda\colon FT \Rightarrow TF$ [20], hence $F : \mathbb{C} \to \mathbb{C}$ is lifted to $\overline{F}\colon \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$. See (2).
- For each $X, Y \in \mathcal{K}\ell(T)$, the homset $\mathcal{K}\ell(T)(X,Y)$ carries an order $\sqsubseteq_{X,Y}$ (or simply $\sqsubseteq$).
- Kleisli composition $\odot$ and cotupling $[\_,\_]$ are monotone with respect to the order $\sqsubseteq$. The latter gives rise to an order isomorphism $\mathcal{K}\ell(T)(X_1 + X_2, Y) \cong \mathcal{K}\ell(T)(X_1, Y) \times \mathcal{K}\ell(T)(X_2, Y)$, where $+$ is inherited along a left adjoint $J\colon \mathbb{C} \to \mathcal{K}\ell(T)$.
- $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is *locally monotone*: for $f, g \in \mathcal{K}\ell(T)(X,Y)$, $f \sqsubseteq g$ implies $\overline{F}f \sqsubseteq \overline{F}g$.

▶ Example 3.2. The category **Sets**, the powerset monad $\mathcal{P}$ (Definition 2.1) and a polynomial functor F on **Sets** (Definition 2.2) satisfy Assumption 3.1. Here for $X, Y \in \mathcal{K}\ell(\mathcal{P})$, an order $\sqsubseteq_{X,Y}$ is defined by: $f \sqsubseteq g$ if $f(x) \subseteq g(x)$ for each $x \in X$.

▶ Example 3.3. The category **Meas**, the sub-Giry monad $\mathcal{G}$ (Definition 2.1) and a polynomial functor F on **Meas** (Definition 2.2) satisfy Assumption 3.1. For $X, Y \in \mathcal{K}\ell(\mathcal{G})$, a natural order $\sqsubseteq_{(X,\mathfrak{F}_X),(Y,\mathfrak{F}_Y)}$ is defined by: $f \sqsubseteq g$ iff $f(x)(A) \leq g(x)(A)$ (in $[0,1]$) for each $x \in X$ and $A \in \mathfrak{F}_Y$.

3.1 Coalgebraic Modeling of Büchi/Parity Automata

The Büchi and parity acceptance conditions have been big challenges to the coalgebra community, because of their *nonlocal* and *asymptotic* nature (see Section 1). One possible

modeling is to take the distinction between ◯ vs. ◎—or different priorities in the parity case—as *state labels*. This is much like in the established coalgebraic modeling of deterministic automata as $2 \times (\_)^\Sigma$-coalgebras (see e.g. [22, 16]). Here the set 2 tells if a state is accepting or not.

A key to our current modeling, however, is that accepting states should rather be specified by a *partition* $X = X_1 + X_2$ of a state space, with $X_1 = \{\text{◯'s}\}$ and $X_2 = \{\text{◎'s}\}$. This idea smoothly generalizes to *parity* conditions, too, by $X_i = \{\text{states of priority } i\}$. Equipping such partitions to coalgebras (with explicit initial states, as in Section 2.2) leads to the following.

Henceforth we state results for the parity condition, with Büchi being a special case.

▶ **Definition 3.4** (parity (T, F)-system)**.** A *parity* (T, F)*-system* is given by a triple $\mathcal{X} = \big((X_1, \dots, X_n), c\colon X \nrightarrow \overline{F}X, s\colon 1 \nrightarrow X\big)$ where n is a positive integer, and:

- $(X_1, \dots, X_n)$ is an n-tuple of objects in $\mathbb{C}$ for *states* (with their *priorities*), and we define $X = X_1 + \dots + X_n$ (a coproduct in $\mathbb{C}$);
- $c\colon X \nrightarrow \overline{F}X$ is an arrow in $\mathcal{K}\ell(T)$ for *dynamics*; and
- $s\colon 1 \nrightarrow X$ is an arrow in $\mathcal{K}\ell(T)$ for *initial states*.

For each $i \in [1, n]$ we define $c_i\colon X_i \nrightarrow \overline{F}X$ to be the restriction $c \circ \kappa_i\colon X_i \nrightarrow \overline{F}X$ along the coprojection $\kappa_i\colon X_i \hookrightarrow X$, in case the maximum priority is $n = 2$, a parity (T, F)-system is referred to as a *Büchi* (T, F)*-system*.

3.2 Coalgebraic Trace Semantics under the Parity Acceptance Condition

On top of the modeling in Definition 3.4 we characterize *accepted languages*—henceforth referred to as *trace semantics*—of parity (T, F)-systems. We use systems of fixed-point equations; this naturally extends the previous characterization of infinitary traces (i.e. under the trivial acceptance conditions) by maximality (Theorem 2.8; see also (1)).

▶ **Definition 3.5** (trace semantics of parity (T, F)-systems)**.** Let $\mathcal{X} = \big((X_1, \dots, X_n), c, s\big)$ be a parity (T, F)-system. It induces the following equational system $E_\mathcal{X}$, where $\zeta\colon Z \xrightarrow{\cong} FZ$ is a final coalgebra in $\mathbb{C}$ (see Assumption 3.1). The variable u_i ranges over the poset $\mathcal{K}\ell(T)(X_i, Z)$.

$$E_\mathcal{X} \;\coloneqq\; \left[\begin{array}{llll} u_1 & =_\mu & (J\zeta)^{-1} \odot \overline{F}[u_1, \dots, u_n] \odot c_1 & \in \mathcal{K}\ell(T)(X_1, Z) \\ u_2 & =_\nu & (J\zeta)^{-1} \odot \overline{F}[u_1, \dots, u_n] \odot c_2 & \in \mathcal{K}\ell(T)(X_2, Z) \\ & \vdots & & \\ u_n & =_{\eta_n} & (J\zeta)^{-1} \odot \overline{F}[u_1, \dots, u_n] \odot c_n & \in \mathcal{K}\ell(T)(X_n, Z) \end{array}\right]$$

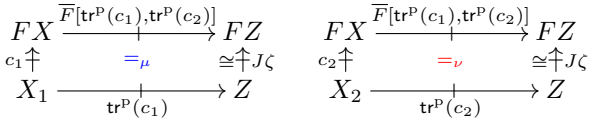
Here $\eta_i = \mu$ if i is odd and $\eta_i = \nu$ if i is even. The functions in the equations are seen to be monotone, thanks to the monotonicity assumptions on cotupling, $\overline{F}$ and $\odot$ (Assumption 3.1).

We say that (T, F) constitutes a *parity trace situation*, if $E_\mathcal{X}$ has a solution for any parity (T, F)-system $\mathcal{X}$, denoted by $\mathsf{tr}^{\mathrm{p}}_1(\mathcal{X})\colon X_1 \nrightarrow Z, \dots, \mathsf{tr}^{\mathrm{p}}_n(\mathcal{X})\colon X_n \nrightarrow Z$. The composite

$$\mathsf{tr}^{\mathrm{p}}(\mathcal{X}) \;\coloneqq\; \big(1 \xrightarrow{s} X = X_1 + X_2 + \dots + X_n \xrightarrow{[\mathsf{tr}^{\mathrm{p}}_1(\mathcal{X}), \mathsf{tr}^{\mathrm{p}}_2(\mathcal{X}), \dots, \mathsf{tr}^{\mathrm{p}}_n(\mathcal{X})]} Z\big)$$

is called the *trace semantics* of the parity (T, F)-system $\mathcal{X}$.

If $\mathcal{X}$ is a Büchi (T, F)-system, the equational system $E_\mathcal{X}$—with their solutions $\mathsf{tr}^{\mathrm{p}}_1(\mathcal{X})$ and $\mathsf{tr}^{\mathrm{p}}_2(\mathcal{X})$ in place—can be expressed as the following diagrams (with explicit μ and ν). See (1).



(7)

4 Coincidence with the Conventional Definition: Nondeterministic

The rest of the paper is devoted to showing that our coalgebraic characterization (Definition 3.5) indeed captures the conventional definition of accepted languages. In this section we study the nondeterministic case; we let $\mathbb{C} = \mathbf{Sets}$, $T = \mathcal{P}$, and F be a polynomial functor.

We first have to check that Definition 3.5 makes sense. Existence of enough fixed points is obvious because $\mathcal{K}\ell(\mathcal{P})(X_i, Z)$ is a complete lattice (Lemma 2.10). See also Example 3.2.

▶ **Theorem 4.1.** *$T = \mathcal{P}$ and a polynomial F constitute a parity trace situation (Definition 3.5).* ◀

Here is the conventional definition of automata [12].

▶ **Definition 4.2** (NPTA)**.** A *nondeterministic parity tree automaton* (NPTA) is a quadruple

$$\mathcal{X} \;=\; \big((X_1, \dots, X_n),\, \Sigma,\, \delta\colon X \to \mathcal{P}\big(\textstyle\coprod_{\sigma\in\Sigma} X^{|\sigma|}\big),\, s \in \mathcal{P}X\big)\,,$$

where $X = X_1 + \cdots + X_n$, each X_i is the set of *states* with the *priority* i, Σ is a ranked alphabet (with the arity map $|\_|\colon \Sigma \to \mathbb{N}$), δ is a *transition function* and s is the set of *initial states*.

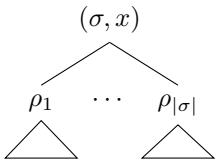
The accepted language of an NPTA $\mathcal{X}$ is conventionally defined in the following way. Here we are sketchy due to the lack of space; precise definitions are in Appendix A in [27].

A (possibly infinite) $(\Sigma \times X)$-labeled tree ρ is a *run* of an NPTA $\mathcal{X} = (\vec{X}, \Sigma, \delta, s)$ if: for each node with a label (σ, x), it has $|\sigma|$ children and we have $\big(\sigma, (x_1, \dots, x_{|\sigma|})\big) \in \delta(x)$ where $x_1, \dots, x_{|\sigma|}$ are the X-labels of its children. For a pedagogical reason we do not require the root X-label to be an initial state. A run ρ of an NPTA $\mathcal{X}$ is *accepting* if any infinite branch π of the tree ρ satisfies the parity acceptance condition (i.e. $\max\{i \mid \pi$ visits states in X_i infinitely often$\}$ is even). The sets of runs and accepting runs of $\mathcal{X}$ are denoted by $\mathrm{Run}_{\mathcal{X}}$ and $\mathrm{AccRun}_{\mathcal{X}}$, respectively.

The function $\mathrm{rt}\colon \mathrm{Run}_{\mathcal{X}} \to X$ is defined to return the root X-label of a run. For each $X' \subseteq X$, we define $\mathrm{Run}_{\mathcal{X},X'}$ by $\{\rho \in \mathrm{Run}_{\mathcal{X}} \mid \mathrm{rt}(\rho) \in X'\}$; the set $\mathrm{AccRun}_{\mathcal{X},X'}$ is similar. The map $\mathrm{DelSt}\colon \mathrm{Run}_{\mathcal{X}} \to \mathrm{Tree}_{\Sigma}$ takes a run, removes all X-labels and returns a Σ-tree.

▶ **Definition 4.3** ($\mathrm{Lang}(\mathcal{X})$ for NPTAs)**.** Let $\mathcal{X}$ be an NPTA. Its *accepted language* $\mathrm{Lang}(\mathcal{X})$ is defined by $\mathrm{DelSt}(\mathrm{AccRun}_{\mathcal{X},s})$.

The following coincidence result for the nondeterministic setting is fairly straightforward. A key is the fact that accepting runs are characterized—among all possible runs—using an equational system that is parallel to the one in Definition 3.5.



(8)

▶ **Lemma 4.4.** *Let $\mathcal{X} = (\vec{X}, \Sigma, \delta, s)$ be an NPTA, and $l_1^{\mathrm{sol}}, \dots, l_n^{\mathrm{sol}}$ be the solution of the following equational system, whose variables $u_1, \dots, u_n$ range over $\mathcal{P}(\mathrm{Run}_{\mathcal{X}})$.*

$$u_1 \;=_{\eta_1}\; \Diamond_{\mathcal{X}}(u_1 \cup \cdots \cup u_n) \cap \mathrm{Run}_{\mathcal{X},X_1}\,,\quad \dots\,,\quad u_n \;=_{\eta_n}\; \Diamond_{\mathcal{X}}(u_1 \cup \cdots \cup u_n) \cap \mathrm{Run}_{\mathcal{X},X_n} \tag{9}$$

Here: $\Diamond_{\mathcal{X}} : \mathcal{P}(\mathrm{Run}_{\mathcal{X}}) \to \mathcal{P}(\mathrm{Run}_{\mathcal{X}})$ *is given by* $\Diamond_{\mathcal{X}} R \coloneqq \big\{\big((\sigma, x), (\rho_1, \dots, \rho_{|\sigma|})\big) \in \mathrm{Run}_{\mathcal{X}} \,\big|\, \sigma \in \Sigma, x \in X, \rho_i \in R\big\}$ *(see the figure (8) above);* $X = X_1 + \dots + X_n$*; and* η_i *is* μ *(for odd* i*) or* ν *(for even* i*). Then the* i*-th solution* l_i^{sol} *coincides with* $\mathrm{AccRun}_{\mathcal{X}, X_i}$*.* ◀

We shall translate the above result to the characterization of accepted trees (Lemma 4.5). In its proof (that is deferred to the appendix in [27]) Lem. B.3—on homomorphisms of equational systems—plays an important role.

▶ **Lemma 4.5.** *Let* $\mathcal{X} = (\vec{X}, \Sigma, \delta, s)$ *be an NPTA, and let* $l_1'^{\mathrm{sol}}, \dots, l_n'^{\mathrm{sol}}$ *be the solution of the following equational system, where* u_i' *ranges over the complete lattice* $\big(\mathcal{P}(\mathrm{Tree}_\Sigma)\big)^{X_i}$*:*

$$u_1' =_{\eta_1} \Diamond_\delta([u_1', \dots, u_n']) \restriction X_1 \;, \quad \dots \;, \quad u_n' =_{\eta_n} \Diamond_\delta([u_1', \dots, u_n']) \restriction X_n \;. \tag{10}$$

Here η_i *is* μ *(for odd* i*) or* ν *(for even* i*);* $(\_) \restriction X_i \colon \big(\mathcal{P}(\mathrm{Tree}_\Sigma)\big)^X \to \big(\mathcal{P}(\mathrm{Tree}_\Sigma)\big)^{X_i}$ *denotes domain restriction; and the function* $\Diamond_\delta \colon \big(\mathcal{P}(\mathrm{Tree}_\Sigma)\big)^X \to \big(\mathcal{P}(\mathrm{Tree}_\Sigma)\big)^X$ *is given by*

$$(\Diamond_\delta T)(x) \;\coloneqq\; \big\{\big(\sigma, (\tau_1, \dots, \tau_{|\sigma|})\big) \,\big|\, \big(\sigma, (x_1, \dots, x_{|\sigma|})\big) \in \delta(x), \tau_i \in T(x_i)\big\} \;.$$

Then we have a coincidence $l_i'^{\mathrm{sol}} = \mathrm{DelSt}'(\mathrm{AccRun}_{\mathcal{X}, X_i})$*, where the function* $\mathrm{DelSt}' \colon \mathcal{P}(\mathrm{Run}_{\mathcal{X}}) \to (\mathcal{P}(\mathrm{Tree}_\Sigma))^X$ *is given by* $\mathrm{DelSt}'(R)(x) \coloneqq \mathrm{DelSt}(\{\rho \in R \mid \mathrm{rt}(\rho) = x\})$*. Recall that* rt *returns a run's root* X*-label.* ◀

▶ **Theorem 4.6** (coincidence, in the nondeterministic setting)**.** *Let* $\mathcal{X} = ((X_1, \dots, X_n), \Sigma, \delta, s)$ *be an NPTA, and* $F_\Sigma = \coprod_{\sigma \in \Sigma} (\_)^{|\sigma|}$ *be the polynomial functor on* **Sets** *that corresponds to* Σ*. Then* $\mathcal{X}$ *is identified with a parity* $(\mathcal{P}, F_\Sigma)$*-system; moreover* $\mathrm{Lang}(\mathcal{X})$ *(in the conventional sense of Definition 4.3) coincides with the coalgebraic trace semantics* $\mathsf{tr}^{\mathrm{p}}(\mathcal{X})$ *(Definition 3.5). Note here that* Tree_Σ *carries a final* F_Σ*-coalgebra (Lemma 2.4).*

Proof. We identify $\mathcal{X}$ with the $(\mathcal{P}, F_\Sigma)$-system $\big((X_1, \dots, X_n), \delta \colon X \nrightarrow \overline{F_\Sigma} X, s \colon 1 \nrightarrow X\big)$, and let $1 = \{\bullet\}$. The equational system $E_{\mathcal{X}}$ in Definition 3.5 is easily seen to coincide with (9) in Lemma 4.5. The claim is then shown as follows, exploiting the last coincidence.

$$\begin{aligned} \mathsf{tr}^{\mathrm{p}}(\mathcal{X}) &= [\mathsf{tr}_1^{\mathrm{p}}(\mathcal{X}), \dots, \mathsf{tr}_n^{\mathrm{p}}(\mathcal{X})] \odot s(\bullet) \qquad \text{by Definition 3.5} \\ &= [\mathrm{DelSt}'(\mathrm{AccRun}_{\mathcal{X}, X_1}), \dots, \mathrm{DelSt}'(\mathrm{AccRun}_{\mathcal{X}, X_n})](s) \\ &= \mathrm{DelSt}(\mathrm{AccRun}_{\mathcal{X}, s}) = \mathrm{Lang}(\mathcal{X}) \qquad \text{by Definition 4.3.} \end{aligned}$$ ◀

5 Coincidence with the Conventional Definition: Probabilistic

In the probabilistic setting the coincidence result is much more intricate. Even the well-definedness of parity trace semantics (Definition 3.5) is nontrivial: the posets $\mathcal{K}\ell(\mathcal{G})(X_i, Z)$ of our interest are not complete lattices, and they even lack the greatest element $\top$. Therefore neither of Lemma 2.10–2.11 ensures a solution of $E_{\mathcal{X}}$ in Definition 3.5. As we hinted in Section 2.3 our strategy is: 1) to apply the Lemma 2.11 to the homset $\mathcal{K}\ell(\mathcal{G})(X, 1)$; and 2) to "extend" fixed points in $\mathcal{K}\ell(\mathcal{G})(X, 1)$ along a final F-sequence. Implicit in the proof details below, in fact, is a correspondence between: abstract categorical arguments along a final sequence; and concrete operational intuitions on probabilistic parity automata.

In this section we let $\mathbb{C} = \mathbf{Meas}$, $T = \mathcal{G}$ (Definition 2.1), and F be a polynomial functor.

▶ Remark 5.1. The class of probabilistic systems of our interest are *generative* (as opposed to *reactive*) ones. Their difference is eminent in the types of transition functions:

$X \longrightarrow \mathcal{G}(A \times X)$	(word)	$X \longrightarrow \mathcal{G}(\coprod_{\sigma \in \Sigma} X^{\lvert\sigma\rvert})$	(tree)	for generative;
$X \longrightarrow (\mathcal{G}X)^A$	(word)	$X \longrightarrow \prod_{\sigma \in \Sigma} \mathcal{G}(X^{\lvert\sigma\rvert})$	(tree)	for reactive.

A generative system (probabilistically) chooses which character to *generate*; while a reactive one *receives* a character from the environment. Reactive variants of probabilistic tree automata have been studied e.g. in [4], following earlier works like [3] on reactive probabilistic word automata. Further discussion is in Appendix C.1 in [27].

5.1 Trace Semantics of Parity $(\mathcal{G}, F)$-Systems is Well-Defined

In the following key lemma—that is inspired by the observations in [6, 23, 25]—a typical usage is for $X_A = X_1 + \cdots + X_i$ and $X_B = X_{i+1} + \cdots + X_n$.

▶ **Lemma 5.2.** *Let $\mathcal{X} = ((X_1, \dots, X_n), s, c)$ be a parity $(\mathcal{G}, F)$-system, and suppose that we are given a partition $X = X_A + X_B$ of $X := X_1 + \cdots + X_n$.*

We define a function $\Gamma \colon \mathcal{K}\ell(\mathcal{G})(X, Z) \to \mathcal{K}\ell(\mathcal{G})(X, 1)$ by $\Gamma(g) = J!_Z \odot g$, where $! \colon Z \to 1$ is the unique function of the type. Its variants $\Gamma_A : \mathcal{K}\ell(\mathcal{G})(X_A, Z) \to \mathcal{K}\ell(\mathcal{G})(X_A, 1)$ and $\Gamma_B : \mathcal{K}\ell(\mathcal{G})(X_B, Z) \to \mathcal{K}\ell(\mathcal{G})(X_B, 1)$ are defined similarly.

For arbitrary $g_B \colon X_B \nrightarrow Z$, we define $\mathfrak{G}^{g_B}$ and $\mathfrak{H}^{g_B}$ as the following sets of "fixed points":

$$\mathfrak{G}^{g_B} := \left\{ g_A \colon X_A \nrightarrow Z \;\middle|\; \begin{array}{ccc} \overline{F}X & \xrightarrow{\overline{F}[g_A, g_B]} & \overline{F}Z \\ {\scriptstyle c_A}\uparrow\!\uparrow & = & \downarrow\!\downarrow{\scriptstyle J\zeta^{-1}} \\ X_A & \xrightarrow[g_A]{} & Z \end{array} \right\} \text{ and } \mathfrak{H}^{g_B} := \left\{ h_A \colon X_A \nrightarrow 1 \;\middle|\; \begin{array}{ccc} \overline{F}X & \xrightarrow{\overline{F}[h_A, \Gamma_B(g_B)]} & \overline{F}1 \\ {\scriptstyle c_A}\uparrow\!\uparrow & = & \downarrow\!\downarrow{\scriptstyle J!_{F1}} \\ X_A & \xrightarrow[h_A]{} & 1 \end{array} \right\} \tag{11}$$

Then Γ_A restricts to a function $\mathfrak{G}^{g_B} \to \mathfrak{H}^{g_B}$. Moreover, the restriction is an order isomorphism, with its inverse denoted by $\Delta^{g_B} \colon \mathfrak{H}^{g_B} \xrightarrow{\cong} \mathfrak{G}^{g_B}$. ◀

In the proof of the last lemma (deferred to the appendix in [27]), the inverse Δ^{g_B} is defined by "extending" $h_A \colon X_A \nrightarrow 1$ to $X_A \nrightarrow Z$, along the final F-sequence $1 \leftarrow F1 \leftarrow \cdots$ (more precisely: the image of the sequence under the Kleisli inclusion $J \colon \mathbf{Meas} \to \mathcal{K}\ell(\mathcal{G})$).

We are ready to prove existence of $E_{\mathcal{X}}$'s solution (Definition 3.5).

▶ **Lemma 5.3.** *Assume the same setting as in Lemma 5.2. We define $\Phi_{\mathcal{X}} \colon \mathcal{K}\ell(\mathcal{G})(X, Z) \nrightarrow \mathcal{K}\ell(\mathcal{G})(X, Z)$ and $\Psi_{\mathcal{X}} \colon \mathcal{K}\ell(\mathcal{G})(X, 1) \nrightarrow \mathcal{K}\ell(\mathcal{G})(X, 1)$, respectively, by*

$$\Phi_{\mathcal{X}}(g) \;:=\; J\zeta^{-1} \odot \overline{F}g \odot c \quad \text{and} \quad \Psi_{\mathcal{X}}(h) \;:=\; J!_{F1} \odot \overline{F}h \odot c\,;$$

these are like the diagrams in (11), except that the latter are parametrized by X_A, X_B, g_B. Now consider the following equational systems, where: $\eta_i = \mu$ if i is odd and $\eta_i = \nu$ if i is even; u_i ranges over $\mathcal{K}\ell(\mathcal{G})(X_i, Z)$; and u'_i ranges over $\mathcal{K}\ell(\mathcal{G})(X_i, 1)$.

$$E = \begin{bmatrix} u_1 =_{\eta_1} \Phi_{\mathcal{X}}([u_1, \dots, u_n]) \odot \kappa_1 \\ \vdots \\ u_n =_{\eta_n} \Phi_{\mathcal{X}}([u_1, \dots, u_n]) \odot \kappa_n \end{bmatrix} \quad E' = \begin{bmatrix} u'_1 =_{\eta_1} \Psi_{\mathcal{X}}([u'_1, \dots, u'_n]) \odot \kappa_1 \\ \vdots \\ u'_n =_{\eta_n} \Psi_{\mathcal{X}}([u'_1, \dots, u'_n]) \odot \kappa_n \end{bmatrix} \tag{12}$$

We claim that the equational systems have solutions $(l_1^{\mathrm{sol}}, \dots, l_n^{\mathrm{sol}})$ and $(l_1'^{\mathrm{sol}}, \dots, l_n'^{\mathrm{sol}})$; and moreover, we have $\Gamma(\mathsf{tr}^{\mathrm{p}}(\mathcal{X})) = \Gamma([l_1^{\mathrm{sol}}, \dots, l_n^{\mathrm{sol}}]) = [l_1'^{\mathrm{sol}}, \dots, l_n'^{\mathrm{sol}}]$. ◀

▶ **Theorem 5.4.** *$T = \mathcal{G}$ and a polynomial F constitute a parity trace situation (Definition 3.5).* ◀

▶ Remark 5.5. The process-theoretic interpretation of the isomorphism $\mathfrak{G}^{g_B} \cong \mathfrak{H}^{g_B}$ is interesting. Let us set $X_A = X$ and $X_B = \emptyset$ for simplicity. The greatest element on the left is

the *infinitary trace semantics* (i.e. accepted languages under the trivial acceptance condition), as in Theorem 2.8 (cf. Table 1). The corresponding greatest element on the right—a function $h_A \colon X_A \to \mathcal{G}1 \cong [0,1]$—assigns to each state $x \in X$ the probability with which a run from x *does not diverge* (recall from Remark 2.6 that the *sub*-Giry monad $\mathcal{G}$ allows divergence probabilities). The accepted language under the parity condition is in general an element of $\mathfrak{G}^{g_B}$ that is neither greatest nor least; the corresponding element in $\mathfrak{H}^{g_B}$ assigns to each state the probability with which it generates a *accepting* run (over any Σ-tree).

5.2 Probabilistic Parity Tree Automata and Its Languages

▶ **Definition 5.6** (PPTA)**.** A *(generative) probabilistic parity tree automaton (PPTA)* is

$$\mathcal{X} \;=\; \big((X_1, \dots, X_n),\, \Sigma,\, \delta \colon X \to \mathcal{G}\big(\textstyle\coprod_{\sigma\in\Sigma} X^{|\sigma|}\big),\, s \in \mathcal{G}X \big) \;,$$

where $X = X_1 + \dots + X_n$, each X_i is a countable set and Σ is a countable ranked alphabet. The subdistribution s over X is for the choice of *initial states*.

In Definition 5.6 the size restrictions on X and Σ are not essential: restricting to discrete σ-algebras, however, makes the following arguments much simpler.

We shall concretely define accepted languages of PPTAs, continuing Section 4 and deferring precise definitions to Appendix A in [27]. This is mostly standard; a reactive variant is found in [4].

▶ **Definition 5.7** (Tree_Σ and $\mathrm{Run}_\mathcal{X}$)**.** Let Σ be a ranked alphabet; Tree_Σ is the set of Σ-trees. A finite $(\Sigma \cup \{*\})$-labeled tree λ, with its branching degrees compatible with the label arities, is called a *partial Σ-tree*. Here the new symbol $*$ ("continuation") is deemed to be 0-ary. The *cylinder set* associated to λ, denoted by $\mathrm{Cyl}_\Sigma(\lambda)$, is the set of (non-partial) Σ-trees that have λ as their prefix (in the sense that a subtree is replaced by $*$). The (smallest) σ-algebra on Tree_Σ generated by the family $\{\mathrm{Cyl}_\Sigma(\lambda) \mid \lambda \text{ is a partial } \Sigma\text{-tree}\}$ will be denoted by $\mathfrak{F}_\Sigma$.

A *run* of a PPTA $\mathcal{X}$ with state space X is a (possibly infinite) $(\Sigma \times X)$-labeled tree whose branching degrees are compatible with the arities of Σ-labels. $\mathrm{Run}_\mathcal{X}$ denotes the set of runs. The measurable structure $\mathfrak{F}_\mathcal{X}$ on $\mathrm{Run}_\mathcal{X}$ is defined analogously to $\mathfrak{F}_\Sigma$: a *partial run* ξ of $\mathcal{X}$ is a suitable $(\Sigma \cup \{*\}) \times X$-labeled tree; it generates a *cylinder set* $\mathrm{Cyl}_\mathcal{X}(\xi) \subseteq \mathrm{Run}_\mathcal{X}$; and these cylinder sets generate the σ-algebra $\mathfrak{F}_\mathcal{X}$. Finally, the set $\mathrm{AccRun}_\mathcal{X}$ of *accepting runs* consists of all those runs all branches of which satisfy the (usual) *parity acceptance condition* (namely: $\max\{i \mid \pi \text{ visits states in } X_i \text{ infinitely often}\}$ is even).

The following result is much like [4, Lem. 36] and hardly novel.

▶ **Lemma 5.8.** *The set* $\mathrm{AccRun}_\mathcal{X}$ *of accepting runs is an* $\mathfrak{F}_\mathcal{X}$*-measurable subset of* $\mathrm{Run}_\mathcal{X}$*.* ◀

In the following $\mathrm{NoDiv}_\mathcal{X}(x)$ is the probability with which an execution from x does not diverge: since we use the *sub*-Giry monad (Definition 5.6), a PPTA can exhibit divergence.

▶ **Definition 5.9** ($\mu^{\mathrm{Run}}_\mathcal{X}$ over $\mathrm{Run}^\mathcal{G}_\mathcal{X}$)**.** Let $\mathcal{X} = ((X_1, \dots, X_n), \Sigma, \delta, s)$ be a PPTA.

Firstly, for each $k \in \mathbb{N}$, let $\mathrm{NoDiv}_{\mathcal{X},k} \colon X \to [0,1]$ ("no divergence in k steps") be defined inductively by: $\mathrm{NoDiv}_{\mathcal{X},0}(x) \coloneqq 1$ and

$$\mathrm{NoDiv}_{\mathcal{X},k+1}(x) \coloneqq \sum_{(\sigma,(x_1,\dots,x_{|\sigma|}))\in\coprod_{\sigma\in\Sigma} X^{|\sigma|}} \delta(x)\big(\sigma,(x_1,\dots,x_{|\sigma|})\big) \cdot \textstyle\prod_{i\in[1,|\sigma|]} \mathrm{NoDiv}_{\mathcal{X},k}(x_i)\,.$$

We define $\mathrm{NoDiv}_\mathcal{X}(x) \coloneqq \bigwedge_{k\in\mathbb{N}} \mathrm{NoDiv}_{\mathcal{X},k}(x)$.

Secondly we define a subprobability measure $\mu^{\mathrm{Run}}_{\mathcal{X}}$ over $\mathrm{Run}_{\mathcal{X}}$. It is given by

$$\mu^{\mathrm{Run}}_{\mathcal{X}}(\mathrm{Cyl}_{\mathcal{X}}(\xi)) \coloneqq s\big(\mathrm{rt}(\xi)\big) \cdot P_{\mathcal{X}}(\xi) \quad \text{for each partial run } \xi, \text{ where } P_{\mathcal{X}}(\xi) \text{ is given by}$$

$$P_{\mathcal{X}}(\xi) \coloneqq \begin{cases} \mathrm{NoDiv}_{\mathcal{X}}(x) & \text{if } \xi = \big((*, x)\big); \\ \delta(x)\big(\sigma, \big(\mathrm{rt}(\xi_1), \dots, \mathrm{rt}(\xi_{|\sigma|})\big)\big) \cdot \prod_{i \in [1, |\sigma|]} P_{\mathcal{X}}(\xi_i) & \text{if } \xi = \big((\sigma, x), (\xi_1, \dots, \xi_{|\sigma|})\big). \end{cases} \tag{13}$$

The above extends to a measure thanks to Carathéodory's theorem. See Lem. C.3 in [27].

Thirdly we introduce a measure $\mu^{\mathrm{Tree}}_{\mathcal{X}}$ over Tree_{Σ} ("which trees are generated by what probabilities"). It is a *push-forward measure* of $\mu^{\mathrm{Run}}_{\mathcal{X}}$ along $\mathrm{DelSt} \colon \mathrm{Run}_{\mathcal{X}} \to \mathrm{Tree}_{\Sigma}$:

$$\mu^{\mathrm{Tree}}_{\mathcal{X}}(\mathrm{Cyl}_{\Sigma}(\lambda)) \;\coloneqq\; \mu^{\mathrm{Run}}_{\mathcal{X}}\left(\mathrm{DelSt}^{-1}(\mathrm{Cyl}_{\Sigma}(\lambda)) \cap \mathrm{AccRun}_{\mathcal{X}}\right) \quad \text{for each partial } \Sigma\text{-tree } \lambda. \tag{14}$$

Since X is countable DelSt is easily seen to be measurable. Finally, the *accepted language* $\mathrm{Lang}(\mathcal{X}) \in \mathcal{G}(\mathrm{Tree}_{\Sigma})$ of $\mathcal{X}$ is defined by $\mu^{\mathrm{Tree}}_{\mathcal{X}}$ in the above.

5.3 Coincidence between Conventional and Coalgebraic Languages

▶ **Lemma 5.10.** *Let $\mathcal{X} = ((X_1, \cdots, X_n), \Sigma, \delta, s)$ be a PPTA with $X = \coprod_i X_i$, and $\Psi'_{\mathcal{X}}$ be*

$$\Psi'_{\mathcal{X}} \colon [0,1]^X \to [0,1]^X, \;\; \Psi'_{\mathcal{X}}(p)(x) \coloneqq \textstyle\sum_{(\sigma, x_1, \dots, x_{|\sigma|}) \in \coprod_{\sigma} X^{|\sigma|}} \delta(x)(\sigma, (x_1, \dots, x_{|\sigma|})) \cdot \prod_{i \in [1, |\sigma|]} p(x_i).$$

Let us define $\mu^{\mathrm{Tree}}_{\mathcal{X},x} \coloneqq \mu^{\mathrm{Tree}}_{\mathcal{X}(x)}$ where $\mathcal{X}(x)$ is the PPTA obtained from $\mathcal{X}$ by changing its initial distribution s into the Dirac distribution δ_x; $\mu^{\mathrm{Run}}_{\mathcal{X},x}$ is similar. We define $\mathrm{AccProb}_{\mathcal{X}} \colon X \to [0,1]$—it assigns to each state the probability of generating an accepting run—by $\mathrm{AccProb}_{\mathcal{X}}(x) \coloneqq \mu^{\mathrm{Run}}_{\mathcal{X},x}(\mathrm{AccRun}_{\mathcal{X}})$.

Consider the following equational system, where u'_i ranges over $\mathcal{K}\ell(\mathcal{G})(X_i, 1)$, and $(\_) \restriction X_i$ denotes domain restriction.

$$u'_1 \;=_{\eta_1}\; \Psi'_{\mathcal{X}}([u'_1, \cdots, u'_n]) \restriction X_1, \quad \dots, \quad u_n \;=_{\eta_n}\; \Psi'_{\mathcal{X}}([u'_1, \cdots, u'_n]) \restriction X_n$$

We claim: 1) the system has a solution $l'^{\mathrm{sol}}_1, \dots, l'^{\mathrm{sol}}_n$; and 2) $[l'^{\mathrm{sol}}_1, \dots, l'^{\mathrm{sol}}_n] = \mathrm{AccProb}_{\mathcal{X}}$. ◀

Its proof (in [27]) relies on Lem. B.4 on homomorphisms of equational systems.

▶ **Theorem 5.11** (coincidence, in the probabilistic setting)**.** *Let $\mathcal{X} = ((X_1, \dots, X_n), \Sigma, \delta, s)$ be a PPTA, and $X = X_1 + \cdots + X_n$, and F_{Σ} be the polynomial functor on* **Meas** *that corresponds to Σ. Then $\mathcal{X}$ is identified with a parity $(\mathcal{G}, F_{\Sigma})$-system; moreover its coalgebraic trace semantics $\mathsf{tr}^{\mathrm{p}}(\mathcal{X})$ (Definition 3.5) coincides with the (probabilistic) language $\mathrm{Lang}(\mathcal{X})$ concretely defined in Definition 5.9. Precisely: $\mathsf{tr}^{\mathrm{p}}(\mathcal{X})(\bullet)(U) = \mathrm{Lang}(\mathcal{X})(U)$ for any measurable subset U of Tree_{Σ}, where $\bullet$ is the unique element of 1 in $\mathsf{tr}^{\mathrm{p}}(\mathcal{X}) \colon 1 \to \mathcal{G}(\mathrm{Tree}_{\Sigma})$.* ◀

Acknowledgments. Thanks are due to Corina Cîrstea, Kenta Cho, Bartek Klin, Tetsuri Moriya and Shota Nakagawa for useful discussions; and to the anonymous referees for their comments.

References

1 Jirí Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A coalgebraic perspective on minimization and determinization. In *Proc. FoSSaCS'12*, volume 7213 of *LNCS*, pages 58–73. Springer, 2012. doi:10.1007/978-3-642-28729-9_4.

2 André Arnold and Damian Niwiński. *Rudiments of μ-Calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001. doi:10.1016/S0049-237X(01)80001-X.

3 Christel Baier and Marcus Größer. Recognizing omega-regular languages with probabilistic automata. In *Proc. LICS'05*, pages 137–146. IEEE Computer Society, 2005. URL: http://dx.doi.org/10.1109/LICS.2005.41, doi:10.1109/LICS.2005.41.

4 Arnaud Carayol, Axel Haddad, and Olivier Serre. Randomization in automata on infinite trees. *ACM Trans. Comp. Logic*, 15(3):24:1–24:33, 2014. doi:10.1145/2629336.

5 Vincenzo Ciancia and Yde Venema. Stream automata are coalgebras. In *Selected Papers of CMCS'12*, volume 7399 of *LNCS*, pages 90–108. Springer, 2012. doi:10.1007/978-3-642-32784-1_6.

6 Corina Cîrstea. Generic infinite traces and path-based coalgebraic temporal logics. *Electr. Notes in Theor. Comp. Sci.*, 264(2):83–103, 2010. doi:10.1016/j.entcs.2010.07.015.

7 Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comp. Journ.*, 54(1):31–41, 2011. doi:10.1093/comjnl/bxp004.

8 Rance Cleaveland, Marion Klein, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In *Proc. CAV'92*, volume 663 of *LNCS*, pages 410–422. Springer, 1992. doi:10.1007/3-540-56496-9_32.

9 Ernst-Erich Doberkat. *Stochastic Coalgebraic Logic*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2009. doi:10.1007/978-3-642-02995-0.

10 Michèle Giry. Categorical aspects of topology and analysis. In *A categorical approach to probability theory, an Intl. Conference at Carleton University, 1981, Proceedings*, volume 915 of *Lect. Notes in Math.*, pages 68–85. Springer, 1982. doi:10.1007/BFb0092872.

11 Sergey Goncharov and Dirk Pattinson. Coalgebraic weak bisimulation from recursive equations over monads. In *Proc. ICALP'14, Part II*, volume 8573 of *LNCS*, pages 196–207. Springer, 2014. doi:10.1007/978-3-662-43951-7_17.

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

13 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Comp. Sci.*, 3(4):11:1–11:36, 2007. doi:10.2168/LMCS-3(4:11)2007.

14 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. Lattice-theoretic progress measures and coalgebraic model checking. In *Proc. POPL'16*, pages 718–732. ACM, 2016. doi:10.1145/2837614.2837673.

15 Bart Jacobs. Trace semantics for coalgebras. *Electr. Notes in Theor. Comp. Sci.*, 106:167–184, 2004. doi:10.1016/j.entcs.2004.02.031.

16 Bart Jacobs. Introduction to coalgebra. Towards mathematics of states and observations. Draft of a book (ver. 2.0), available online, 2012. URL: http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf.

17 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. *J. Comp. & Syst. Sci.*, 81(5):859–879, 2015. doi:10.1016/j.jcss.2014.12.005.

18 Bartek Klin. Bialgebraic methods and modal logic in structural operational semantics. *Inf. & Comp.*, 207(2):237–257, 2009. doi:10.1016/j.ic.2007.10.006.

19 Nancy Lynch and Frits Vaandrager. Forward and backward simulations. *Inf. & Comp.*, 121(2):214–233, 1995. doi:10.1006/inco.1995.1134.

20 Philip S. Mulry. Lifting theorems for Kleisli categories. In *Proc. MFPS'93*, volume 802 of *LNCS*, pages 304–319. Springer, 1994. doi:10.1007/3-540-58027-1_15.

21 John Power and Hayo Thielecke. Environments, continuation semantics and indexed categories. In *Proc. TACS'97*, volume 1281 of *LNCS*, pages 391–414. Springer, 1997. doi:10.1007/BFb0014560.

22 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

23 Christoph Schubert. Terminal coalgebras for measure-polynomial functors. In *Proc. TAMC'09*, volume 5532 of *LNCS*, pages 325–334. Springer, 2009. doi:10.1007/978-3-642-02017-9_35.

24 Alexandra Silva. A short introduction to the coalgebraic method. *ACM SIGLOG News*, 2(2):16–27, April 2015. doi:10.1145/2766189.2766193.

25 Natsuki Urabe and Ichiro Hasuo. Coalgebraic infinite traces and kleisli simulations. In Lawrence S. Moss and Pawel Sobocinski, editors, *Proc. CALCO'15*, volume 35 of *LIPIcs*, pages 320–335. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.CALCO.2015.320.

26 Natsuki Urabe and Ichiro Hasuo. Quantitative simulations by matrices. *Inf. & Comp.*, 2016. In press. doi:10.1016/j.ic.2016.03.007.

27 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic trace semantics for Büchi and parity automata. arXiv preprint, 2016.

28 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Fair simulation for nondeterministic and probabilistic Büchi automata: a coalgebraic perspective. *CoRR*, abs/1606.04680, 2016. URL: http://arxiv.org/abs/1606.04680.

29 R.J. van Glabbeek. The linear time – branching time spectrum I: The semantics of concrete, sequential processes. In J.A. BergstraA. PonseS.A. Smolka, editor, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001. doi:10.1016/B978-044482830-9/50019-9.

30 Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency, the of 8th Banff Higher Order Workshop, 1995, Proceedings*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995. doi:10.1007/3-540-60915-6_6.

31 Yde Venema. Automata and fixed point logic: A coalgebraic perspective. *Inf. & Comp.*, 204(4):637–678, 2006. doi:10.1016/j.ic.2005.06.003.

32 James Worrell. On the final sequence of a finitary set functor. *Theor. Comp. Sci.*, 338(1-3):184–199, 2005. doi:10.1016/j.tcs.2004.12.009.

Bisimulations and Unfolding in $\mathcal{P}$-Accessible Categorical Models

Jérémy Dubut[1], Eric Goubault[2], and Jean Goubault-Larrecq[3]

1 **LSV, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France and LIX, Ecole Polytechnique, CNRS, Université Paris-Saclay, 91128 Palaiseau, France**
dubut@lsv.ens-cachan.fr

2 **LIX, Ecole Polytechnique, CNRS, Université Paris-Saclay, 91128 Palaiseau, France**
goubault@lix.polytechnique.fr

3 **LSV, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France**
goubault@lsv.ens-cachan.fr

Abstract

We propose a categorical framework for bisimulations and unfoldings that unifies the classical approach from Joyal and al. via open maps and unfoldings. This is based on a notion of categories accessible with respect to a subcategory of path shapes, i.e., for which one can define a nice notion of trees as glueings of paths. We show that transition systems and presheaf models are instances of our framework. We also prove that in our framework, several notions of bisimulation coincide, in particular an "operational one" akin to the standard definition in transition systems. Also, our notion of accessibility is preserved by coreflections. This also leads us to a notion of unfolding that behaves well in the accessible case: it is a right adjoint and is a universal covering, i.e., it is initial among the morphisms that have the unique lifting property with respect to path shapes. As an application, we prove that the universal covering of a groupoid, a standard construction in algebraic topology, is an unfolding, when the category of path shapes is well chosen.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases categorical models, bisimulation, coreflections, unfolding, universal covering

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.25

1 Introduction

Bisimulations were introduced in [12] as a way to express that two concurrent systems are "equivalent", in a way that would reflect not only trace equivalence, but also the branching structure of executions. Later, Joyal, Nielsen and Winskel [6] developed a theory of bisimulations using open maps, which are particular morphisms in some category of models, which satisfy lifting properties with respect to a specified subcategory of execution paths. They made explicit links between this abstract view of bisimulations and the classical relational definition, for some models of concurrency, including transition systems and event structures.

In some other line of work, Nielsen, Plotkin and Winskel [10] introduced a notion of unfolding for 1-safe Petri nets. The unfolding produces an "equivalent" Petri net, which is infinite in general (in the absence of cut-rules) and is non-looping. This is at the basis of numerous verification methods on Petri nets [2]. Later, Winskel [13] developed the categorical framework of Petri nets and unfoldings by relating them to coreflections (special cases of

adjoint functors) between some categories of concurrent models, more particularly occurence nets and event structures. Winskel also developed a classification of concurrent models by coreflections [14].

As open maps, unfoldings are closely related to prominent concepts in algebraic topology. Unfoldings are closely linked to coverings, which are nice fibered spaces which (also) satisfy unique lifting properties (see e.g. [4]). Coverings are closely related to partial unfoldings of the state space (see e.g. [13]). The universal covering can actually be defined, in ordinary algebraic topology, as a complete delooping of paths in topological spaces, which makes it very similar to unfoldings of transition systems. Although the analogy is enlightening, formalizing it stumbles on various difficulties. In this paper, we propose to unify those theories, by putting lifting properties of "paths" at the center of the picture.

In Section 2 we recall the case of transition systems, and the classical notion of bisimilarity [12]. The categorical approaches to bisimulation via open maps and via "strong path-bisimulation" of Joyal, Nielsen and Winskel [6] are equivalent to this classical notion of bisimulation in the particular case of transition systems, but are not equivalent in general, if we do not set up the proper context.

The framework of $\mathcal{P}$-accessible models we are developing is going to define this context, where those two notions of bisimilarities will be equivalent. We introduce accessible models in Section 3 and prove this result in the same section. Somehow, these categories are the right ones in the sense that their objects are closely tied with the glueing of paths within them. It will be trivially the case for transition systems: they will be accessible models in our sense.

In Section 4, we show that the framework of presheaf models of [6] is also a particular case of our framework, yielding another proof of one of the main results of [6].

Many models are related through coreflections (see, e.g. [11]). In Section 5, we show that, when two models are related this way, then accessibility is transferred from one the other. This makes our notion possibly applicable to a great variety of models.

We then turn to unfoldings in accessible models, in Section 6. Indeed, there is a very nice notion of paths and path extensions in accessible models, making the notion of unfolding very natural. In particular, in Section 6.2, we show that the unfolding of a model is bisimilar to the original model. As a bonus, unfoldings are defined in a canonical manner in accessible categories: they are right adjoints (Section 6.3). Finally, we show that unfoldings in accessible models enjoy unique path lifting properties (Section 7.2) making them similar to universal coverings. In the case of groupoids for instance, we show that unfoldings are universal coverings (recapped in Section 7.1).

2 Categorical models and bisimilarities

We first recall, from [6], two notions of bisimilarity in a category with a specified subcategory of path shapes.

2.1 Category of models, subcategory of paths

We consider a category $\mathcal{M}$ (of **models**) together with a small subcategory (of **path-shapes**) $\mathcal{P}$. We assume that $\mathcal{M}$ and $\mathcal{P}$ have a common initial object I, i.e., an object $I \in \mathcal{P}$ such that for every object A of $\mathcal{P}$ (resp. of $\mathcal{M}$), there is a unique morphism in $\mathcal{P}$ (resp. in $\mathcal{M}$) from I to A. We note ι_A this unique morphism. One typical example is the category of transition systems, that we briefly recap below.

Fix an alphabet Σ. A **transition system** $T = (Q, i, \Delta)$ on Σ is the following data: a set Q (of states); a initial state $i \in Q$; a set of transitions $\Delta \subseteq Q \times \Sigma \times Q$.

A **morphism of transition systems on** Σ $f : T_1 = (Q_1, i_1, \Delta_1) \longrightarrow T_2 = (Q_2, i_2, \Delta_2)$ is a function $f : Q_1 \longrightarrow Q_2$ such that $f(i_1) = i_2$ and for every $(p, a, q) \in \Delta_1$, $(f(p), a, f(q)) \in \Delta_2$.

We note $\mathbf{TS}(\Sigma)$, the category of transition systems on Σ and morphisms of transition systems.

The subcategory of path-shapes will be in this case the category of branches: for $n \in \mathbb{N}$, a **n-branch shape on** Σ is a transition system $([n], 0, \Delta)$ where:

- $[n]$ is the set $\{0, \dots, n\}$;
- Δ is of the form $\{(i, a_i, i+1) \mid i \in [n-1]\}$ for some a_0, ..., a_{n-1} in Σ.

$$0 \xrightarrow{a_0} 1 \xrightarrow{a_1} 2 \qquad \cdots \qquad n-1 \xrightarrow{a_{n-1}} n$$

We then take $\mathbf{Br}(\Sigma)$ as the full subcategory of $\mathbf{TS}(\Sigma)$ of n-branch shapes for all $n \in \mathbb{N}$. A common initial object of $\mathbf{TS}(\Sigma)$ and $\mathbf{Br}(\Sigma)$ is then the 0-branch shape $I = ([0], 0, \varnothing)$. We call **$n$-branch of a transition system** T any morphism of transition system from a n-branch form to T.

2.2 A relational bisimilarity of models: path-bisimilarity

Equivalence of transition systems is defined through the notion of **bisimulation**. Classically [12], a bisimulation between $T_1 = (Q_1, i_1, \Delta_1)$ and $T_2 = (Q_2, i_2, \Delta_2)$ is a relation $R \subseteq Q_1 \times Q_2$ such that:

(i) $(i_1, i_2) \in R$;

(ii) if $(q_1, q_2) \in R$ and $(q_1, a, q_1') \in \Delta_1$ then there is $q_2' \in Q_2$ such that $(q_2, a, q_2') \in \Delta_2$ and $(q_1', q_2') \in R$;

(iii) if $(q_1, q_2) \in R$ and $(q_2, a, q_2') \in \Delta_2$ then there is $q_1' \in Q_1$ such that $(q_1, a, q_1') \in \Delta_1$ and $(q_1', q_2') \in R$.

We then say that two transition systems are **bisimilar** if there is a bisimulation between them.

A bisimulation between T_1 and T_2 induces a relation R_n' between n-branches of T_1 and n-branches of T_2 by:

$$R_n' = \{(f_1 : B_1 \longrightarrow T_1, f_2 : B_2 \longrightarrow T_2) \mid \forall i \in [n], (f_1(i), f_2(i)) \in R\}$$

These relations satisfy that:

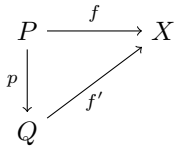
- $(\iota_{T_1}, \iota_{T_2}) \in R_0'$ by (i);
- by (ii), if $(f_1, f_2) \in R_n'$ and if $(f_1(n), a, q_1) \in \Delta_1$ then there is $q_2 \in Q_2$ such that $(f_2(n), a, q_2) \in \Delta_2$ and $(f_1', f_2') \in R_{n+1}'$ where $f_i'(j) = f_i(j)$ if $j \leq n$, q_i otherwise;
- symmetrically with (iii);
- if $(f_1, f_2) \in R_{n+1}'$ then $(f_1', f_2') \in R_n'$ where f_i' is the restriction of f_i to $[n]$.

In fact, bisimilarity of transition systems is equivalent to the existence of such relations on n-branches. This leads us to the general notion of strong path-bisimulation [6].

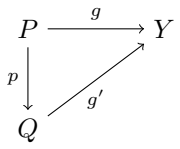
A **strong path-bisimulation** R between X and Y, objects of $\mathcal{M}$ is a set of elements of the form $X \xleftarrow{f} P \xrightarrow{g} Y$ with P object of $\mathcal{P}$ such that:

(a) $X \xleftarrow{\iota_X} I \xrightarrow{\iota_Y} Y$ belongs to R;

(b) if $X \xleftarrow{f} P \xrightarrow{g} Y$ belongs to R then for every **path extension** of X, i.e, every morphism p in $\mathcal{P}$ such that:



commutes then there exists a path extension of Y



such that $X \xleftarrow{f'} Q \xrightarrow{g'} Y$ belongs to R;

(c) symmetrically;

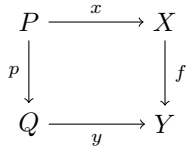
(d) if $X \xleftarrow{f} P \xrightarrow{g} Y$ belongs to R and if we have a morphism $p : Q \longrightarrow P \in \mathcal{P}$ then $X \xleftarrow{f \circ p} Q \xrightarrow{g \circ p} Y$ belongs to R;

We say that X and Y are **strong path bisimilar** iff there exists a strong path bisimulation between them.

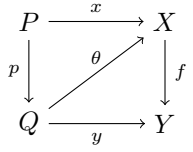
2.3 A fibrational bisimilarity of models: $\mathcal{P}$-bisimilarity

Lifting properties are a useful ingredient in category theory and algebraic topology. In [6], they permit to design an abstract notion of bisimilarity via morphisms which satisfy lifting properties with respect to paths, recovering a large variety of models and motivating the use of presheaf models by the work on pretopoi in [5].

We say that a morphism $f : X \longrightarrow Y$ of $\mathcal{M}$ is **($\mathcal{P}$-)open** iff for all commutative diagrams:



with $p : P \longrightarrow Q \in \mathcal{P}$, there exists a morphism $\theta : Q \longrightarrow X$ such that the following diagram commutes:



We then say that two objects X and Y of $\mathcal{M}$ are **$\mathcal{P}$-bisimilar** iff there exists a span $f : Z \longrightarrow X$ and $g : Z \longrightarrow Y$ where f and g are $\mathcal{P}$-open.

It is known that if X and Y are $\mathcal{P}$-bisimilar then they are strong path bisimilar [6]. The converse also holds in the case of transition systems (both $\mathcal{P}$ and path bisimilarities coincide with the classical bisimilarity), but there is no general result for the converse. The purpose of the following section is to investigate a general framework in which those two notions of bisimilarity will coincide.

3 Accessible models and equivalence of bisimilarities

For the converse, we must build a span of open maps from a strong path-bisimulation. It requires in particular that we construct an object of $\mathcal{M}$, which will be the tip of the span. One way of doing so is to glue the elements of the bisimulation in order to obtain an "object of bisimilar paths". Categorically, a glueing is a colimit, so a natural hypothesis should be the existence of some colimits in $\mathcal{M}$.

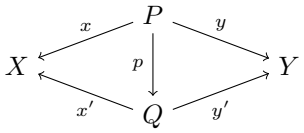
Concretely, a $\mathcal{P}$**-tree** in $\mathcal{M}$ is a colimit in $\mathcal{M}$ of a small diagram with values in $\mathcal{P}$, i.e., of a functor $D : \mathcal{D} \longrightarrow \mathcal{P}$ where $\mathcal{D}$ is a small category. We say that **all $\mathcal{P}$-trees exist in $\mathcal{M}$** if every small diagram with values in $\mathcal{P}$ has a colimit in $\mathcal{M}$. In the category of transition systems, $\mathbf{Br}(\Sigma)$-trees are exactly synchronization trees, i.e., a transition system $T = (Q, i, \Delta)$ such that:

- every state in Q is accessible, i.e., for every $q \in Q$, there is a n-branch $f : B \longrightarrow T$ for some $n \in \mathbb{N}$ such that $f(n) = q$;
- T is acyclic, i.e., for every branch $f : B \longrightarrow T$, there is no $i \neq j$ such that $f(i) = f(j)$;
- T is non-joining, i.e., if (q_1, a, p) and $(q_2, b, p) \in \Delta$ then $a = b$ and $q_1 = q_2$.

In particular, all $\mathbf{Br}(\Sigma)$-trees exists in $\mathbf{TS}(\Sigma)$. We note $\mathbf{Tree}(\mathcal{M}, \mathcal{P})$ for the full subcategory of $\mathcal{M}$ of $\mathcal{P}$-trees.

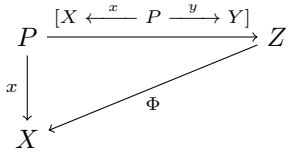
Let R be a strong path bisimulation between X and Y and assume that all $\mathcal{P}$-trees exist. Let us construct a span of maps between X and Y. First, we construct the tip of the span as the colimit of a particular diagram with values in $\mathcal{P}$, defined from R. Let $\mathcal{C}$ be the following category:

- objects of $\mathcal{C}$ are elements of R;
- morphisms from $X \xleftarrow{x} P \xrightarrow{y} Y$ to $X \xleftarrow{x'} Q \xrightarrow{y'} Y$ are morphisms $p : P \longrightarrow Q$ of $\mathcal{P}$ such that the following diagram commutes:



Then define the diagram $F : \mathcal{C} \longrightarrow \mathcal{P}$ which maps every $X \xleftarrow{x} P \xrightarrow{y} Y \in R$ to P and every p to itself. Since $\mathcal{P}$-trees exist (F is small because R is a set), let $(Z, ([\alpha])_{\alpha \in R})$ be the colimit of F, where the $[X \xleftarrow{x} P \xrightarrow{y} Y] : P = F(X \xleftarrow{x} P \xrightarrow{y} Y) \longrightarrow Z$ are the maps from the colimit.

Z will be the tip of our span. Now we need to construct maps $\Phi : Z \longrightarrow X$ and $\Psi : Z \longrightarrow Y$. Let us do it for Φ: since $(X, (F(X \xleftarrow{x} P \xrightarrow{y} Y) \xrightarrow{x} X))$ is a cocone of F, there exists a unique morphism $\Phi : Z \longrightarrow X$ such that for all $X \xleftarrow{x} P \xrightarrow{y} Y \in R$ the following diagram commutes:



To prove that strong path-bisimilarity implies $\mathcal{P}$-bisimilarity, we just need to prove that Φ is open. But it does not hold in general. We will need that we do not create more paths in a tree than the ones we used in the glueing. In the case of transition systems, this says that every path in a tree seen as the colimit of a certain diagram D with values in $\mathcal{P}$ is a subbranch of some $D(i)$. More generally, we will say that $\mathcal{M}$ is $\mathcal{P}$**-accessible** if :

- all $\mathcal{P}$-trees exist;
- every morphism $f : P \longrightarrow Z$ where $P \in \mathcal{P}$ and $(Z, (\eta_d)_{d\in\mathcal{D}})$ is the colimit of a non-empty small diagram $D : \mathcal{D} \longrightarrow \mathcal{P}$ factorizes as $f = \eta_d \circ p$ for some $d \in \mathcal{D}$ with $p : P \longrightarrow D(d) \in \mathcal{P}$.

In particular, $\mathbf{TS}(\Sigma)$ is $\mathbf{Br}(\Sigma)$-accessible.

▶ Remark. The name "accessible" is a reference to κ-accessible categories [8] where κ is a cardinal, which is a very similar property of a category, requiring the existence of some colimits (in this case, filtered colimits) and the same kind of factorizations for morphisms whose codomain is such a colimit.

Assuming that $\mathcal{M}$ is $\mathcal{P}$-accessible, we can now prove that Φ is open. Consider a commutative diagram of the form:

$$\begin{array}{ccc} P & \xrightarrow{z} & Z \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle \Phi} \\ Q & \xrightarrow[x]{} & X \end{array}$$

with p in $\mathcal{P}$. As Z is a colimit of a non-empty (because R is non-empty) small diagram, then by $\mathcal{P}$-accessibility, $z : P \longrightarrow Z$ factorizes as $[X \xleftarrow{x'} P' \xrightarrow{y'} Y] \circ p'$ for some $X \xleftarrow{x'} P' \xrightarrow{y'} Y \in R$ and $p' : P \longrightarrow P' \in \mathcal{P}$. Then, by condition (d) of a strong path bisimulation, $X \xleftarrow{x'\circ p'} P \xrightarrow{y'\circ p'} Y$ belongs to R. Moreover, the following diagram commutes:

$$\begin{array}{ccccc} & & P & & \\ & {\scriptstyle x'\circ p'}\swarrow & \downarrow{\scriptstyle p'} & \searrow{\scriptstyle y'\circ p'} & \\ X & \nwarrow{\scriptstyle x'} & P' & \nearrow{\scriptstyle y'} & Y \end{array}$$

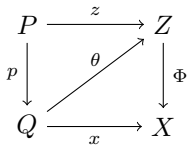
Then, $z = [X \xleftarrow{x'} P' \xrightarrow{y'} Y] \circ p' = [X \xleftarrow{x'\circ p'} P \xrightarrow{y'\circ p'} Y]$.
So, $x \circ p = \Phi \circ z = \Phi \circ [X \xleftarrow{x'\circ p'} P \xrightarrow{y'\circ p'} Y] = x' \circ p'$ by definition of Φ. This means that we have the following commutative diagram:

$$\begin{array}{ccc} P & \xrightarrow{x'\circ p'} & X \\ {\scriptstyle p}\downarrow & \nearrow{\scriptstyle x} & \\ Q & & \end{array}$$

Then, by condition (b) of a strong path bisimulation, there is a path extension of Y:

$$\begin{array}{ccc} P & \xrightarrow{y'\circ p'} & Y \\ {\scriptstyle p}\downarrow & \nearrow{\scriptstyle y} & \\ Q & & \end{array}$$

such that $X \xleftarrow{x} Q \xrightarrow{y} Y$ belongs to R.
Then the morphism $\theta = [X \xleftarrow{x} Q \xrightarrow{y} Y] : Q \longrightarrow Z$ is the lifting we were looking for:



So we deduce:

▶ **Theorem 1.** *If $\mathcal{M}$ is $\mathcal{P}$-accessible and if X and Y are strong path bisimilar then they are $\mathcal{P}$-bisimilar.*

4 Presheaf models

Presheaf models were introduced in [6], motivated by the work on pretopoi in [5]. We prove in this section that presheaf models are a particular case of accessible models.

Assume given a small category Δ with an initial object J. A **rooted presheaf on** Δ is a functor F from Δ^{op} to **Set** such that $F(J)$ is a singleton. Let $[\Delta^{op}, Set]_*$ be the category of rooted presheaves on Δ and natural transformations. We have a functor (called the **Yoneda embedding**) $\mathfrak{Y} : \Delta \longrightarrow [\Delta^{op}, Set]_*$:

- we associate an object P of Δ with the rooted presheaf $\mathfrak{Y}(P)$ which maps:
  - every object Q of Δ to $\Delta(Q, P)$
  - every morphism $p : Q \longrightarrow Q'$ of Δ to the function $\mathfrak{Y}(P)(p) : \Delta(Q', P) \longrightarrow \Delta(Q, P)$ $f \mapsto f \circ p$
- we associate a morphism $p : P \longrightarrow P'$ with the natural transformation $\mathfrak{Y}(p) : \mathfrak{Y}(P) \longrightarrow \mathfrak{Y}(P')$ defined by

$$\mathfrak{Y}(p)_Q : \Delta(Q, P) \longrightarrow \Delta(Q, P') \qquad f \mapsto p \circ f$$

▶ **Theorem 2.** *Let $\mathcal{P}$ be the image of $\mathfrak{Y}$ and $\mathcal{M} = [\Delta^{op}, Set]_*$. Then $\mathcal{M}$ is $\mathcal{P}$-accessible.*

Proof.

- **$\mathcal{P}$ is a full embedding of $\mathcal{M}$:** by the Yoneda lemma.
- **computation of colimits in $\mathcal{M}$:** consider a small diagram $D : U \longrightarrow \mathcal{M}$. The colimit in $[\Delta^{op}, Set]_*$ of D is the colimit in $[\Delta^{op}, Set]$ (which is cocomplete [1]) of the small (non-empty) diagram $D_\perp : U_\perp \longrightarrow M$ where:
  - $U_\perp$ is the category obtained by adding an object $\perp$ to U with a unique morphism from $\perp$ to any object of U or $\perp$ and no morphism from an object of U to $\perp$
  - $D_\perp$ maps $\perp$ to $\mathfrak{Y}(J)$ (which is the initial object of $\mathcal{M}$ and $\mathcal{P}$ by the Yoneda lemma), any object u of U to $D(u)$, the morphism from $\perp$ to u object of $U_\perp$ to the unique natural transformation η_u from $\mathfrak{Y}(J)$ to $D_\perp(u)$ and any morphism ν of U to $D(\nu)$
- **all trees exist:** consequence of the previous point
- **$\mathcal{P}$-accessibility:** let $D : U \longrightarrow \mathcal{P}$ be a non-empty small diagram and $f : \mathfrak{Y}(P) \longrightarrow colim\ D$ a morphism of $\mathcal{M}$ with P in Δ and $colim\ D$ the colimit of D in $\mathcal{M}$. $(colim\ D)(P)$ is computed as the quotient:

$$(\bigsqcup_{u \in U} D(u)(P) \sqcup \Delta(P, J))/\sim$$

  where $\sim$ is the equivalence relation on $\bigsqcup_{u \in U} D(u)(P) \sqcup \Delta(P, J)$ generated by:
  - for every $\nu : u \longrightarrow u'$ of U, for every $x \in D(u)(P)$, $x \sim D(\nu)_P(x)$
  - for every $x \in \Delta(P, J)$ and every u in U, $x \sim \eta_u(x)$

Since U is non-empty, every x in $\Delta(P, J)$ is equivalent to some element of $\coprod_{u \in U} D(u)(P)$. So, every element of $(colim\ D)(P)$ is the image of one of the projections of an element of some $D(u)(P)$. Let v be an object of U and $x \in D(v)(P)$ such that $f_P(id_P) \in (colim\ D)(P)$ is the image of x by the projection from $D(v)(P)$ to $(colim\ D)(P)$. By the Yoneda lemma, there exists a unique natural transformation $\theta : \mathfrak{Y}(P) \longrightarrow D(v)$ such that $\theta_P(id_P) = x$. θ belongs to $\mathcal{P}$ because $\mathcal{P}$ is a full embedding of $\mathcal{M}$. If $\pi_v : D(v) \longrightarrow colim D$ is the morphism from the universal cocone, then by the Yoneda lemma, $f = \pi_v \circ \theta$.

◀

5 Relationships with coreflections

Coreflections are a nice categorical way to express the fact that a computational model can be simulated by another one. This view was initiated in [13], where it was shown in particular that there is a coreflection from event structures to occurrence nets and so to 1-safe Petri nets. Note that the right adjoints of those coreflections give interesting constructions : in the case of occurrence nets in Petri nets, the right adjoint gives what is called the unfolding of a 1-safe Petri net. In this section, we prove that accessibility is preserved by coreflections.

In fact we can prove the even more general following theorem:

▶ **Theorem 3.** *Let $\mathcal{P}$ (resp. $\mathcal{P}'$) be a subcategory of $\mathcal{M}$ (resp. $\mathcal{M}'$). Assume that:*

- *$\mathcal{M}$ is $\mathcal{P}$-accessible*
- *there is a functor $F : \mathcal{M} \longrightarrow \mathcal{M}'$ such that:*
  - *F preserves trees i.e. for every small diagram $D : U \longrightarrow \mathcal{P}$, the colimit of $F \circ D$ in $\mathcal{M}'$ exists and is equal to $F(colim\ D)$*
  - *F induces an functor from $\mathcal{P}$ to $\mathcal{P}'$*
  - *there is a functor $G : \mathcal{P}' \longrightarrow \mathcal{P}$ and a natural isomorphism $\nu : F \circ G \longrightarrow id_{\mathcal{P}'}$*

Then $\mathcal{M}'$ is $\mathcal{P}'$-accessible.

The preservation of trees holds for example when F is a left adjoint. The other two conditions hold for example when F induces a equivalence between $\mathcal{P}$ and $\mathcal{P}'$. So, we deduce:

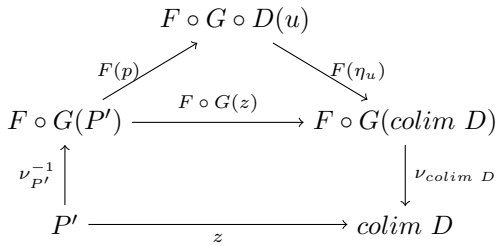
▶ **Corollary 4.** *If $F : \mathcal{M} \longrightarrow \mathcal{M}'$ is a coreflection, if $\mathcal{P}'$ is the image of $\mathcal{P}$ by F and if $\mathcal{M}$ is $\mathcal{P}$-accessible then $\mathcal{M}'$ is $\mathcal{P}'$-accessible.*

Proof of Theorem 3. Let $G : \mathcal{P}' \longrightarrow \mathcal{P}$ and $\nu : F \circ G \longrightarrow id_{\mathcal{P}'}$ a natural isomorphism.

- **existence of trees:** let $D : U \longrightarrow \mathcal{P}'$ be a small diagram. By preservation of trees and existence of trees in $\mathcal{M}$, the colimit of $F \circ G \circ D$ in $\mathcal{M}'$ exists and is equal to $F(colim\ G \circ D)$. But ν induces a natural isomorphism between D and $F \circ G \circ D$. Then the colimit of D in $\mathcal{M}'$ exists.
- **$\mathcal{P}'$-accessibility:** Let $z : P' \longrightarrow Z$ morphism of $\mathcal{M}'$ with $P' \in \mathcal{P}'$ and $(Z, (\eta_u)_{u \in U})$ is the colimit of a non-empty small diagram $D : U \longrightarrow \mathcal{P}'$.
  By naturality of ν, the following diagram commutes:

$$\begin{array}{ccc} F \circ G(P') & \xrightarrow{F \circ G(z)} & F \circ G(Z) \\ {\scriptstyle \nu_{P'}^{-1}}\uparrow & & \downarrow{\scriptstyle \nu_Z} \\ P' & \xrightarrow[z]{} & Z \end{array}$$

  By $\mathcal{P}$-accessibility, $G(z) : G(P') \longrightarrow G(colim\ D) = colim\ (G \circ D)$ factorizes as $G(z) = \eta_u \circ p$ with $p : G(P') \longrightarrow G \circ D(u)$ morphism of $\mathcal{P}$ and $\eta_u : G \circ D(u) \longrightarrow colim(G \circ D)$ is from the universal cocone. Then the following diagram commutes:



Then z factorizes as $\eta'_u \circ (\nu_{D(u)} \circ F(p) \circ \nu^{-1}_{P'})$ with $\eta'_u : D(u) \longrightarrow colim\ D$ coming from the universal cocone and $\nu_{D(u)} \circ F(p) \circ \nu^{-1}_{P'} : P' \longrightarrow D(u)$ morphism of $\mathcal{P}'$.

◀

6 Unfoldings in accessible models

6.1 The case of TS(Σ)

The unfolding of a transition system is an equivalent system without loops, obtained by "unfolding" the loops. More precisely, it is a tree which will be bisimilar to the transition system. Given a transition system $T = (Q, i, \Delta)$, the unfolding Unfold(T) of T is the synchronization tree (P, j, Γ) where:

- $P = \{(q_0, a_1, q_1, \ldots, a_n, q_n) \mid q_i \in Q,\ a_i \in \Sigma,\ (q_i, a_{i+1}, q_{i+1}) \in \Delta \wedge q_0 = i\}$
- $j = (i)$
- $\Gamma = \{((q_0, a_1, q_1, \ldots, a_n, q_n), b, (q_0, a_1, q_1, \ldots, a_n, q_n, b, q)) \mid (q_n, b, q) \in \Delta\}$

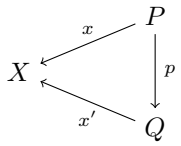
It is easy to check that $\{(q_n, (q_0, a_1, q_1, \ldots, a_n, q_n)) \mid (q_0, a_1, q_1, \ldots, a_n, q_n) \in P\}$ is a bisimulation between T and Unfold(T).

Equivalently, the unfolding of T can be defined as a glueing of all branches of T, this is the way we will define more generally the unfolding in a categorical model.

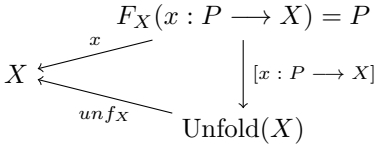
6.2 $\mathcal{P}$-unfolding and bisimilarity

Let $\mathcal{M}$ be a category where all $\mathcal{P}$-trees exist and X an object of $\mathcal{M}$. Let $\mathcal{P} \downarrow X$ be the small comma category whose:

- objects are morphisms $x : P \longrightarrow X$ of $\mathcal{M}$ with P in $\mathcal{P}$
- morphisms from $x : P \longrightarrow X$ to $x' : Q \longrightarrow X$ are morphisms $p : P \longrightarrow Q$ of $\mathcal{P}$ such that the following diagram commutes:



We then define the small diagram $F_X : \mathcal{P} \downarrow X \longrightarrow \mathcal{P}$ which maps every $x : P \longrightarrow X$ to P and every p to itself. Let Unfold(X) be the colimit of F_X in $\mathcal{M}$. We call it the **($\mathcal{P}$-) unfolding of** X. Since $(X, (x : P \longrightarrow X)_x)$ is a cocone of F_X, there is a unique morphism $unf_X : \text{Unfold}(X) \longrightarrow X$ such that for every $x : P \longrightarrow X$ with $P \in \mathcal{P}$, the following diagram commutes:



where $[x : P \longrightarrow X]$ is the morphism coming from the colimit.

Using a similar argument as in Theorem 1, we have the following:

▶ **Theorem 5.** *When $\mathcal{M}$ is $\mathcal{P}$-accessible, unf_X is $\mathcal{P}$-open and so X and* Unfold(X) *are $\mathcal{P}$-bisimilar (strong path bisimilar).*

6.3 Unfolding is a right adjoint

The following lemma implies that the unfolding of a tree (and so of an unfolding) is isomorphic to the tree itself:

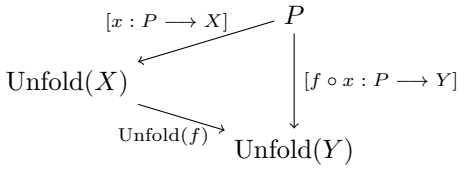
▶ **Lemma 6.**

(i) *When all trees exist in $\mathcal{M}$,* Unfold *extends to a functor* Unfold $: \mathcal{M} \longrightarrow Tree(\mathcal{M}, \mathcal{P})$.

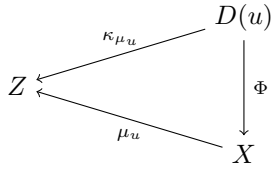
(ii) *When $\mathcal{M}$ is $\mathcal{P}$-accessible, $\mathcal{P}$ is dense in $Tree(\mathcal{M}, \mathcal{P})$ i.e. for all $X \in Tree(\mathcal{M}, \mathcal{P})$, $(X, (x)_{x:P \longrightarrow X})$ is a colimit of F_X.*

Proof.

(i) Let $f : X \longrightarrow Y$ be a morphism of $\mathcal{M}$. Then $(\text{Unfold}(Y), ([f \circ x : P \longrightarrow Y])_{x:P \longrightarrow X})$ is a cocone of F_X. So there is a unique morphism $\text{Unfold}(f) : \text{Unfold}(X) \longrightarrow \text{Unfold}(Y)$ such that for every path $x : P \longrightarrow X$ of X, the following diagram commutes:



(ii) Assume given another cocone $(Z, (\kappa_x : P \longrightarrow Z)_{x:P \longrightarrow X})$ of F_X. We construct a morphism $\Phi : X \longrightarrow Z$ like this: as X is in $Tree(\mathcal{M}, \mathcal{P})$, there is a small non-empty diagram $G : U \longrightarrow \mathcal{P}$ such that $(X, (\mu_u)_{u \in U})$ is a colimit of G for some μ_u. So, for all u, $\mu_u : D(u) \longrightarrow X$ is an object of $\mathcal{P} \downarrow X$. Since $(Z, (\kappa_{\mu_u} : D(u) \longrightarrow Z)_{u \in U})$ is a cocone of D, there is a unique morphism $\Phi : X \longrightarrow Z$ such that for all $u \in U$, the following diagram commutes:



Then, we can check that Φ is a morphism of cocones from $(X, (x)_{x:P \longrightarrow X})$ to $(Z, (\kappa_x : P \longrightarrow Z)_{x:P \longrightarrow X})$ and that it is the unique such morphism.

◀

From this sort of density property, we deduce that the unfolding is a right adjoint of the inclusion of trees in $\mathcal{M}$. This result is similar to the one from [13] stating that the unfolding is the right adjoint of the inclusion of occurrence nets in 1-safe Petri nets.

▶ **Theorem 7.** *When $\mathcal{M}$ is $\mathcal{P}$-accessible,* Unfold *is a right adjoint of inj : **Tree**$(\mathcal{M}, \mathcal{P}) \longrightarrow \mathcal{M}$, the embedding of **Tree**$(\mathcal{M}, \mathcal{P})$ in $\mathcal{M}$. In particular, the injection of **Tree**$(\mathcal{M}, \mathcal{P})$ in $\mathcal{M}$ is a coreflection.*

Proof.

- **definition of the counit $\epsilon : inj \circ \text{Unfold} \longrightarrow id_{\mathcal{M}}$: $\epsilon_X = unf_X$.**

- **definition of the unit** $\eta : id_{Tree(\mathcal{M},\mathcal{P})} \longrightarrow \text{Unfold} \circ inj$**:** by density of $\mathcal{P}$ in $Tree(\mathcal{M},\mathcal{P})$, for all $X \in Tree(\mathcal{M},\mathcal{P})$ there is a unique (iso)morphism $\eta_X : X \longrightarrow \text{Unfold}(X)$ such that for all $x : P \longrightarrow X$, $\eta_X \circ x = [x : P \longrightarrow X]$.

◀

7 Unfoldings and universal coverings

Unfoldings and coverings of spaces [9] are very similar in the sense that they both "unfold" loops (or "kill" the first homotopy group). But it seems that there were no general formal links in the literature between those two structures. We present here a view toward this.

7.1 Coverings of groupoids

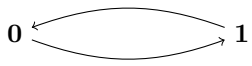
Coverings of groupoids are more natural than coverings of spaces as they are defined by lifting properties and their existence does not assume any hypothesis on the groupoid. They are very close to coverings of spaces since a covering of a space induces a covering of its fundamental groupoid and lots of properties of coverings of spaces can be expressed on the induced coverings of groupoids [9].

A **small pointed connected groupoid** (spc groupoids for short) is a pair $(\mathcal{C}, c)$ of a small connected groupoid $\mathcal{C}$ and an object c of $\mathcal{C}$. A **pointed functor** is a functor $F : (\mathcal{C}, c) \longrightarrow (\mathcal{D}, d)$ between spc groupoids such that $F(c) = d$. We note $\mathbf{Grpd}_\star$ the category of spc groupoids and pointed functors.

A **covering of a spc groupoids** $(\mathcal{C}, c)$ is a pointed functor $F : (\tilde{\mathcal{C}}, \tilde{c}) \longrightarrow (\mathcal{C}, c)$ such that for every morphism $f : c \longrightarrow c'$ of $\mathcal{C}$ there exists a unique object $\tilde{c'}$ of $\tilde{\mathcal{C}}$ and an unique morphism $\tilde{f} : \tilde{c} \longrightarrow \tilde{c'}$ such that $F(\tilde{f}) = f$. We say that a covering is **universal** if $\tilde{\mathcal{C}}(\tilde{c}, \tilde{c}) = \{id_{\tilde{c}}\}$.

Covering are similar to open maps since they satisfy a lifting property. In fact, they are open maps when we consider the following subcategory of paths. Let $\mathcal{I}$ be the full subcategory of $\mathbf{Grpd}_\star$ whose objects are the following to spc groupoids:

- **0**, the spc groupoid with one object and only the identity as morphism
- **1**, the spc groupoid with two objects:



pointed on 0.

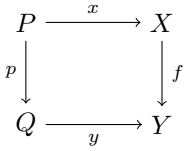
It is easy to check that $\mathbf{Grpd}_\star$ is $\mathcal{I}$-accessible.

Coverings are exactly the open maps whose lifts are unique. Universal coverings are universal in the category of coverings in the following sense [9]: given a universal covering $F : (\tilde{\mathcal{C}}, \tilde{c}) \longrightarrow (\mathcal{C}, c)$ and a covering $G : (\mathcal{D}, d) \longrightarrow (\mathcal{C}, c)$, then there is a unique pointed functor $H : (\tilde{\mathcal{C}}, \tilde{c}) \longrightarrow (\mathcal{D}, d)$ such that $G \circ H = F$. Moreover, H is a covering. This means that universal covering is initial in the category of coverings. In particular, universal coverings are unique up to isomorphism. Contrary to universal coverings of spaces, universal coverings of groupoids always exist [9].

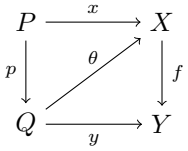
7.2 Unfoldings and unique path lifting property

We have just seen that (universal) coverings are defined by unique lifting property. Now let us see the link between unfoldings and unique liftings.

We say that a morphism $f : X \longrightarrow Y$ is a ($\mathcal{P}$**-) covering** if it has the **unique path lifting property**, i.e., if for all commutative diagram:



with $p : P \longrightarrow Q \in \mathcal{P}$, there exists a unique morphism $\theta : Q \longrightarrow X$ such that the following diagram commutes:



▶ Remark. This is the same as $\mathcal{P}$-open but with the unicity of the lift.

The following result states that unfolding is a covering and that moreover it is initial among coverings.

▶ **Theorem 8.** *When $\mathcal{M}$ is $\mathcal{P}$-accessible:*

(i) *unf_X has the unique path lifting property*

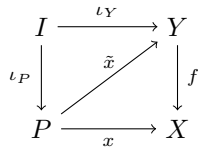
(ii) *for every morphism $f : Y \longrightarrow X$ which has the unique lifting property, there is a unique morphism $\tilde{f} : \mathrm{Unfold}(X) \longrightarrow Y$ such that $f \circ \tilde{f} = unf_X$. Moreover, $\tilde{f}$ has the unique path lifting property.*

Proof.

(i) This is a consequence of ii) because id_X has the unique path lifting property and $id_X \circ unf_X = unf_X$ and so $unf_X = \widetilde{id_X}$.

(ii)

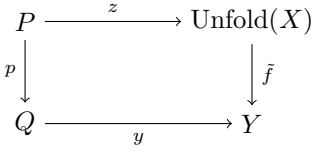
- **construction of $\tilde{f}$:** For every $x : P \longrightarrow X$, by the unique path lifting property, there is a unique $\tilde{x} : P \longrightarrow Y$ such that



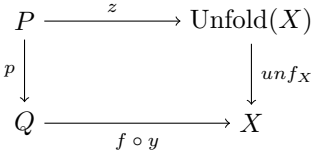
i.e. a unique $\tilde{x}$ such that $f \circ \tilde{x} = x$. Since $(Y, (\tilde{x})_{x:P \longrightarrow X})$ is a cocone of F_X and since $(\mathrm{Unfold}(X), ([x])_x)$ is a colimit of F_X, there is a unique $\tilde{f} : \mathrm{Unfold}(X) \longrightarrow Y$ such that for every $x : P \longrightarrow X$, $\tilde{f} \circ \iota_x = \tilde{x}$ and so, $f \circ \tilde{f} \circ \iota_x = f \circ \tilde{x} = x = unf_X \circ \iota_x$ and by unicity of the definition of unf_X, $f \circ \tilde{f} = unf_X$.

- **unicity of $\tilde{f}$:** consequence of the unique path lifting property of f.

- **existence of the lift:** The lift of a diagram of the form:



with $p \in \mathcal{P}$, is obtained as a lift of the following diagram:



coming from the fact that unf_X is $\mathcal{P}$-open.
- **unicity of the lift:** consequence of $\mathcal{P}$-accessibility.

◀

In the case of $\mathbf{Grpd}_\star$ and $\mathcal{I}$, this implies that the unfolding is a covering and is initial in the category of coverings. So we deduce:

▶ **Corollary 9.** *The universal covering of a spc groupoid coincides with its $\mathcal{I}$-unfolding.*

8 Conclusion

We have generalized Joyal, Nielsen and Winskel's approach of [6] to what we called accessible models. We have shown in particular that presheaf models and transitions systems are particular cases of accessible models. In these models, not only do we have a faithful formulation of bisimulation in the form of open maps, but also, we have a nice characterization of unfoldings, as form of generalized universal covering.

In the future, we would like to exploit this framework on a variety of models. As coreflections produce accessible categories from accessible categories, this is already the case for some interesting models. On top of this, we would like to study the case of 1-safe Petri nets in more detail and also, hybrid and stochastic hybrid models for which notions of bisimulations have been defined in the literature, see e.g. [7, 3].

References

1. F. Borceux. *Handbook of Categorical Algebra 2 : Categories and Structures*. Cambridge University Press, 1994.
2. J. Esparza and K. Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer Publishing Company, Incorporated, 2008.
3. A. Girard, A. A. Julius, and G. J. Pappas. Approximate bisimulation for a class of stochastic hybrid systems. In *2006 American Control Conference*, June 2006.
4. A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.
5. A. Joyal and I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.
6. A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from Open Maps. *Information and Computation*, 127(2):164–185, 1996.
7. G. Lafferriere, G. J. Pappas, and S. Sastry. *Hybrid Systems V*, chapter Hybrid Systems with Finite Bisimulations, pages 186–203. Springer Berlin Heidelberg, 1999.
8. M. Makkai and R. Paré. *Accessible categories: The foundations of categorical model theory Contemporary Mathematics*. 104. American Mathematical Society, 1989.
9. J. P. May. *A Concise Course in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, 1999.
10. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part I. *Theor. Comput. Sci.*, 13:85–108, 1981.
11. M. Nielsen and G. Winskel. *Models for Concurrency*. Oxford University Press, 1995.
12. D. Park. Concurrency and Automata on Infinite Sequences. *Lecture Notes in Computer Science*, 154:167–183, 1981.
13. G. Winskel. A New Definition of Morphism on Petri Nets. In *STACS 84, Symposium of Theoretical Aspects of Computer Science, Paris, France, 11-13 April, 1984, Proceedings*, pages 140–150, 1984.

14 G. Winskel. Event structures. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, pages 325–392, 1986.

A Uniform Framework for Timed Automata

Tomasz Brengos[1] **and Marco Peressotti**[2]

1 **Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland**
t.brengos@mini.pw.edu.pl

2 **Department of Mathematics, Computer Science, and Physics, University of Udine, Italy**
marco.peressotti@uniud.it

Abstract

Timed automata, and machines alike, currently lack a general mathematical characterisation. In this paper we provide a uniform coalgebraic understanding of these devices. This framework encompasses known behavioural equivalences for timed automata and paves the way for the extension of these notions to new timed behaviours and for the instantiation of established results from the coalgebraic theory as well. Key to this work is the use of *lax functors* for they allow us to model time flow as a context property and hence offer a general and expressive setting where to study timed systems: the index category encodes "how step sequences form executions" (e.g. whether steps duration get accumulated or kept distinct) whereas the base category encodes "step nature and composition" (e.g. non-determinism and labels). Finally, we develop the notion of *general saturation* for lax functors and show how equivalences of interest for timed behaviours are instances of this notion. This characterisation allows us to reason about the expressiveness of said notions within a uniform framework and organise them in a spectrum independent from the behavioural aspects encoded in the base category.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Coalgebras, lax functors, general saturation, timed behavioural equivalence, timed language equivalence

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.26

1 Introduction

Timed automata, and machines alike, are abstract devices used in the modelling and verification of real-time dynamical systems and recipients of much attention, both in terms of theoretical and practical developments [2, 17, 9, 19]. Despite decades long efforts, a general and mathematical characterisation of all these devices is missing. A uniform account of these formalisms would provide guidelines for developing new kinds of timed systems starting from existing notions of computations and, from a more foundational point of view, would allow a cross-fertilizing exchange of definitions, notions, and techniques.

The theory of coalgebras has been recognized as a good context for the study of concurrent and reactive systems [22]: systems are represented as maps of the form $X \to BX$ for some suitable *behavioural functor* B. By changing the underlying category and functor a wide range of cases are covered, from traditional LTSs to systems with I/O, quantitative aspects, probabilistic distribution, and even systems with continuous state. Frameworks of this kind provide great returns from a theoretical and a practical point of view, since they prepare the ground for general results and tools which can be readily instantiated to various cases, and moreover they help us to discover connections and similarities between apparently different

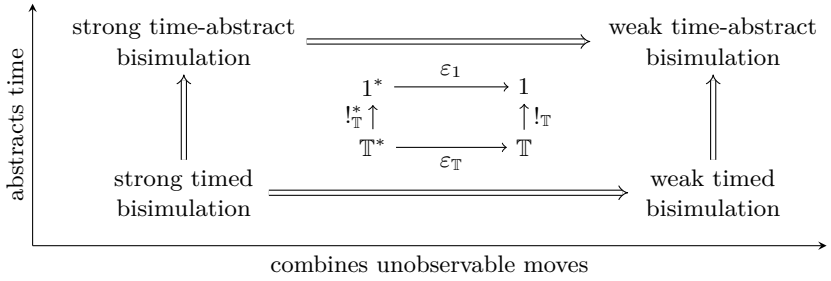


Figure 1 Monoid morphisms and the corresponding spectrum of behavioural equivalences.

notions. Among the several valuable results offered by the coalgebraic approach we mention general accounts of bisimulation [1, 26], structural operational semantics [27, 21], trace equivalence [12], minimization [3], determinisation [23] and up-to techniques [4].

Recent works [8, 6, 7] extended the theory of coalgebras with a general understanding of coalgebras with unobservable moves and their weak bisimulations. As shown in loc. cit. systems with internal steps should be modelled as coalgebras whose type is a monad. This allows us to view coalgebras as endomorphisms in suitable Kleisli categories, hence, allowing mutual composition. At the heart of weak bisimulation of such systems lies the notion of *saturation*. Intuitively, it can be understood as a reflexive and transitive closure of a system. Already, at this coalgebraic level, the so-called *lax functors* start to emerge. Indeed, the reflexive and transitive morphisms can be seen as lax functors whose domain category is the terminal category [7]. Additionally, the results of [6] demonstrate that the coalgebraic saturation arises as a consequence of an adjoint situation between certain two categories of lax functors. All these observations suggest that lax functors and adjunctions between lax functor categories lie at the heart of different behavioural equivalences that take into account accumulation and abstraction of certain portions of data.

In this paper we embrace and extend the lax functorial setting as a general environment where to model systems dynamics. In the approach we propose:

- the index category models those aspects of the computations under scrutiny associated with the computation *context*, like time flow;
- the base category models *effects* like non-determinism or unobservable moves.

Because systems considered in this work have a flat[1] state space we can safely restrict to index categories with one object i.e. monoids.

We extend the theory of saturation by allowing for monoid homomorphisms whose codomain is not the final one-object monoid thus introducing the so called *general saturation* (for lax functors on a monoid category). This theory extends the uniform definition of strong and weak behavioural equivalences given by (simple) saturation offering a greater control of which computational aspects are abstracted away. The resulting notion of behavioural equivalence is parameterised by monoid congruences and naturally forms a *spectrum* reflecting their *discriminating power*. This spectrum reflects the inclusion ordering on congruences with coarser congruences yielding coarser notions of behavioural equivalence. This remarkable correspondence allows for reasoning about the expressiveness of these definitions by means of simple diagrams of monoid homomorphisms (cf. Figure 1). Besides from finer to coarser, this spectrum can be organised along two orthogonal dimensions which intuitively correspond to:

[1] We say that a class of systems has flat state spaces when there are no archetypal rôles associated to states that prescribe restrictions on their possible behaviours, as opposed i.e., to alternating games.

- collecting effects modelled in the base category (e.g. τ-actions) and
- abstracting from aspects modelled in the index category (e.g. time).

The former is determined by the counit $\varepsilon\colon (-)^* \to \mathcal{I}d$ of the free-monoid adjunction whereas the latter by arbitrary monoid morphisms and their extension to free monoids as exemplified by the inner diagram in Figure 1.

Synopsis and related work Our paper is closely related to the research presented in [5, 7, 8, 6] with the emphasis laid on [6]. Indeed, Brengos' [6], which is highly motivated by Sobociński's work on relational presheaves (i.e. lax functors whose codomain category is the category of sets and relations) and their saturation [25], presents the lax functorial framework as a natural extension of coalgebraic weak bisimulation and saturation studied in [7, 8]. The main focus of [6] is on lax functorial weak bisimulation and reflexive and transitive saturation. It is worth mentioning that in loc. cit. the author already pointed out that timed transition systems and their weak behavioural equivalence can be modelled in the lax functorial setting. Our paper extends these results in a systematic way by:

- describing the categorical framework which pinpoints the relation between timed automata and their semantics (Section 2.2),
- presenting the concept of general saturation and the family of behavioural equivalences associated with it (Section 2.3),
- capturing a much wider spectrum of language and behavioural equivalences (Section 3).

2 A coalgebraic account of timed automata and their semantics

In this section we introduce a general framework for timed automata and systems alike. The key ingredient is to separate time flow from other aspects of the computation (like non-determinism) and model the dependency of the latter from the former by means of (lax) functors. We remark that, although we use timed automata as a reference example, this framework can accommodate any notion of resources and effects as long as they can be modelled by some monoid M and by a (suitably enriched) category K, respectively.

We assume the reader to be familiar with basic 1-category and 2-category theory notions and refer to [20, 18] for a thorough introduction.

2.1 Unobservable moves, acceptance and non-determinism

The behaviour associated to the classical notion of timed automata arise from non-determinism, unobservable moves, and acceptance with the latter being relevant when language equivalence is considered. All these computational effects are described briefly below and are captured by Kleisli categories of well-known monads we will use as running examples.

The first example we discuss is the powerset monad $\mathcal{P}$ whose Kleisli category $\mathcal{K}l(\mathcal{P})$ consists of sets as objects and maps of the form $X \to \mathcal{P}Y$ as morphisms between X and Y and, for each $f\colon X \to \mathcal{P}Y$ and $g\colon Y \to \mathcal{P}Z$, the composite $g \circ f\colon X \to \mathcal{P}Z$ is:

$$g \circ f(x) = \bigcup g(f(x)) = \{z \mid z \in g(y) \text{ and } y \in f(x)\}.$$

Let $\Sigma_\tau \triangleq \Sigma + \{\tau\}$. The Set-endofunctor $\mathcal{P}^\Sigma \triangleq \mathcal{P}(\Sigma_\tau \times \mathcal{I}d)$ whose coalgebras model labelled transition systems with unobservable moves [22] is a monad [7], herein the *LTS monad*. The Kleisli category $\mathcal{K}l(\mathcal{P}^\Sigma)$ has sets as objects and maps $X \to \mathcal{P}(\Sigma_\tau \times Y)$ as morphisms from X to Y. For $f\colon X \to \mathcal{P}^\Sigma(Y)$ and $g\colon Y \to \mathcal{P}^\Sigma(Z)$ the composite $g \circ f \in \mathcal{K}l(\mathcal{P}^\Sigma)$ is

$$g \circ f(x) = \{(\sigma, z) \mid x \xrightarrow{\sigma}_f y \xrightarrow{\tau}_g z \text{ or } x \xrightarrow{\tau}_f y \xrightarrow{\sigma}_g z\},$$

where $x \xrightarrow{\sigma}_f y$ denotes $(\sigma, y) \in f(x)$. The above Kleisli category (see [7, 8] for a discussion) lets us consider non-deterministic systems with unobservable moves (a.k.a. τ-actions) as endomorphisms and allows their mutual composition. We refer the interested reader to [5, 7] for further details on how to extend other computational effects besides non-determinism with unobservable moves.

Finally, the endofunctor $\mathcal{P}^{\Sigma}_{\checkmark} \triangleq \mathcal{P}(\Sigma_\tau \times \mathcal{I}d + \{\checkmark\})$ whose coalgebras model the behaviour of non-deterministic automata with ε-moves (or ε-NA in short) [11, 24] also carries a monadic structure [7, Example 4.5] (here and in loc. cit., the label ε is represented by the label τ). The composition in its Kleisli category is given as follows. For two morphisms $f\colon X \to \mathcal{P}^{\Sigma}_{\checkmark}(Y)$ and $g\colon Y \to \mathcal{P}^{\Sigma}_{\checkmark}(Z)$ the value of their composition $g \circ f$ in $\mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$ on $x \in X$ is:

$$\{(\sigma, z) \mid x \xrightarrow{\sigma}_f y \xrightarrow{\tau}_g z \text{ or } x \xrightarrow{\tau}_f y \xrightarrow{\sigma}_g z\} \cup \{\checkmark \mid \checkmark \in f(x) \text{ or } x \xrightarrow{\tau}_f y \text{ and } \checkmark \in g(y)\}.$$

Morphisms of this category model non-deterministic computations with unobservables and accepting states (for a morphism $f\colon X \to \mathcal{P}^{\Sigma}_{\checkmark}Y$ any such state $x \in X$ is specified by $\checkmark \in f(x)$, which we will often denote as $x\downarrow$). It is important to note that $\mathcal{P}^{\Sigma}$ is a submonad of $\mathcal{P}^{\Sigma}_{\checkmark}$. However, since the terminating states in classical timed automata and their semantics are an extra feature [2] we decide to consider these two examples of monads separately. Indeed, the Kleisli category for the LTS monad will be our main example throughout Section 2.2, 2.3 and 3.1. Our focus will move onto $\mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$ in Section 3.2, where we discuss finite trace equivalence and sets of accepted words.

The essence of non-determinism of these three examples lies in the fact that their Kleisli categories are suitably order enriched in a natural point-wise manner:

$$f \leq g \overset{\triangle}{\iff} f(x) \subseteq g(x) \quad \forall x \in X$$

with their hom-posets admitting arbitrary joins. Composition in $\mathcal{K}l(\mathcal{P})$ and $\mathcal{K}l(\mathcal{P}^{\Sigma})$ preserves them, whereas in $\mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$ it only preserves non-empty ones [7]. Joins of morphisms, which abstractly model non-deterministic choice, will play a key rôle in saturation.

▶ Remark. It is an easy exercise to prove that these three Set-based monads are commutative strong monads [16]. It means that the monoidal structure $(\mathsf{Set}, \times, 1)$ extends to all three Kleisli categories above in a natural manner.

2.2 Timed automata in K and their semantics

Recall from [2] that a timed automaton, a.k.a. *time transition table*, over an alphabet Σ and with access to C-many clocks is essentially a set of locations L and a relation on source and target locations, symbols, and expressions for describing clock constraints and resets. However, the actual semantics of *clock expressions* is not part of the definition of timed transition tables but rather of their semantics as *timed transition systems* (herein TTS) which is covered below. Hence, a timed automaton is essentially a morphism $L \to \mathcal{E} \otimes L$ in $\mathsf{K} = \mathcal{K}l(T)$ (where T is $\mathcal{P}^{\Sigma}_{\checkmark}$ or $\mathcal{P}^{\Sigma}$, depending whether acceptance is necessary, and $\otimes$ is $\times$) or, more generally, a $(\mathcal{E} \otimes \mathcal{I}d)$-coalgebra for some given object of (clock) *expressions* $\mathcal{E} \in \mathsf{K}$ for a monoidal category $(\mathsf{K}, \otimes, I)$.

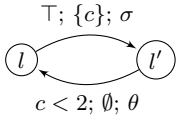
The interaction between timed automata and time flow is mediated by (a fixed and finite set of) clocks which an automaton can interact with by means of the clock expressions associated to its transitions. Clocks can be abstracted as an object $\mathcal{V} \in \mathsf{K}$ representing *clock configurations* (a.k.a. clock valuations) together with:

- a morphism $\mathsf{eval}\colon \mathcal{V} \otimes \mathcal{E} \to \mathcal{V}$ specifying the effect of expressions and

- a $\mathbb{T}$-indexed family $\{\mathsf{flow}_t \colon \mathcal{V} \to \mathcal{V}\}_{t\in\mathbb{T}}$ describing the effect of time flow and such that $\mathsf{flow}_0 = id_{\mathcal{V}}$ and, for all $t, t' \in \mathbb{T}$, $\mathsf{flow}_{t+t'} = \mathsf{flow}_t \circ \mathsf{flow}_{t'}$ with $\mathbb{T} = (\mathbb{T}, +, 0)$ being the monoid of time, e.g. $\mathbb{T} = ([0, \infty), +, 0)$.

In the case of timed automata (with clocks in C) the object of configurations is the set $\mathcal{V}$ of all functions $v \colon C \to \mathbb{T}$ and the object of expressions is the set $\mathcal{E}$ of pairs (γ, δ) where γ is a subset of C and δ is a syntactic expression generated by the grammar $\delta ::= c \leq r \mid r \leq c \mid \neg\delta \mid \delta \wedge \delta$ where c is a clock and the r is a non-negative rational number (cf. [2, Def. 3.6]).

▶ **Example 2.1.** Consider the timed transition table from [2, Ex. 3.4] depicted below: where $\top$ can be seen as a short hand for $0 \leq c$ which is always satisfied. Let $C = \{c\}$ and $\Sigma = \{\sigma, \theta\}$. Intuitively, the edge from l to l' describes a transition that can be performed provided the input character is σ and resets c as its side effect. The other transition assumes $c < 2$ input θ and does not reset c. This automaton is equivalent to the $\mathcal{E} \otimes \mathcal{I}d$-coalgebra α on $\{l, l'\}$ such that:



$$\alpha(l) = \{(\sigma, (\{c\}, \top), l'))\} \qquad \alpha(l') = \{(\theta, ((\emptyset, c < 2), l))\}.$$

Time flow is given, on $t \in \mathbb{T}$, as $\mathsf{flow}_t(v) = \{(\tau, \lambda c : C.v(c) + t)\}$ and expression evaluation as

$$\mathsf{eval}(v, (\gamma, \delta)) = \begin{cases} \{(\tau, v[\gamma := 0])\} & \text{if } v \vDash \delta \\ \emptyset & \text{otherwise} \end{cases} \quad \text{where} \quad v[\gamma := 0](c) = \begin{cases} 0 & \text{if } c \in \gamma \\ v(c) & \text{otherwise} \end{cases}$$

and $v \vDash \delta$ is the obvious interpretation of the boolean expression obtained by replacing every c in δ with $v(c)$.

▶ Remark. At a first glance τs and singletons may sound strident, especially given how the semantics of timed automata is presented in [2]. However, their meaning is precise and plays a crucial rôle in capturing how eval and flow interact with timed automata in defining their semantics as timed transition systems—as will be made clear below. In fact, they arise from the unit of $\mathcal{P}^{\Sigma}$ meaning that when composed the only computational effect (modelled in K and) carried by eval and flow is to prevent the firing of any transition not enabled by the current clock configuration.

Given the semantics for expressions and (time) flow, every timed automaton yields a *timed transition system* (TTS) that is a transition system with labels[2] in $\Sigma_\tau \times \mathbb{T}$ or, equivalently, a $\mathbb{T}$-indexed family of LTSs i.e. endomorphisms in $\mathsf{K} = \mathcal{K}l(\mathcal{P}^{\Sigma})$:

$$\frac{X \to \mathcal{P}(\Sigma_\tau \times \mathbb{T} \times X)}{\mathbb{T} \to \mathcal{P}(\Sigma_\tau \times X)^X}$$

Albeit the two presentations are equivalent for TTSs, we adopt the latter since:

- As noted in [17], the correspondence does not hold for arbitrary timed behaviours e.g. the convex-set semantics of Segala systems assume probability distribution supports to be (finitely) bounded (cf. [5, 13]) whereas a single entry of a timed transition table can easily yield uncountably many transitions in the associated TTS.
- Time flow is an aspect of the computational context instead of an observation and the latter representation allows us to separate the rôle of time from non-deterministic computations modelled in the base category K—along the lines of the lax functor approach.

[2] The definition of TTS may vary: some authors (e.g. [2]) consider transitions to be labelled by pairs (σ, t) of consumed symbols and time (durations) whereas others (e.g. [19]) consider "duration-less" *discrete transitions* (i.e. labelled by $\sigma \in \Sigma$) and "symbol-less" *time transitions* (i.e. labelled by $t \in (0, \infty)$). By introducing a distinguished symbol τ the two approaches are uniformly covered by a single model where labels are pairs in $\Sigma_\tau \times \mathbb{T}$; duration-less and symbol-less transitions become $(\sigma, 0)$ and (τ, t), respectively.

In general, given eval and flow, every $\mathcal{E}\otimes\mathcal{I}d$-coalgebra, i.e. every morphism $\alpha\colon L \to \mathcal{E}\otimes L$ in K induces its *semantics*, i.e. a $\mathbb{T}$-indexed family $\{\alpha_t\colon \mathcal{V}\otimes L \to \mathcal{V}\otimes L\}_{t\in\mathbb{T}}$ where each endomorphism $\alpha_t \in \mathsf{K}$ is defined as:

$$\mathcal{V}\otimes L \xrightarrow[\mathsf{flow}_t\otimes\alpha]{} \mathcal{V}\otimes\mathcal{E}\otimes L \xrightarrow[\mathsf{eval}\otimes id_L]{} \mathcal{V}\otimes L \qquad \mathcal{V}\otimes L \xrightarrow{\ \alpha_t\ } \mathcal{V}\otimes L$$

From the definition of α_t it is clear how any computational effect described by eval and flow interact with α. In particular, when $\mathsf{K} = \mathcal{K}l(\mathcal{P}^{\Sigma})$ and α models a timed automaton, the above definition readily expands as follows:

$$\alpha_t(v,l) = \{(\sigma,v',l') \mid (\sigma,(\gamma,\delta),l') \in \alpha(l),\ v_t = \lambda c.v(c)+t,\ v_t \vDash \delta, \text{ and } v' = v_t[\gamma := 0]\} \quad (1)$$

highlighting the contribution of effects in the definition of expression and flow semantics for timed automata. Clearly, (1) precisely defines the timed transition system associated to the given automaton α. This correspondence can be easily checked when the presentation from [2] is used: consider the LTS with labels in $\Sigma_\tau \times \mathbb{T}$ given by

$$(v,l) \xrightarrow{(\sigma,t)} (v',l') \iff^{\triangle} (\sigma,v',l') \in \alpha_t(v,l)$$

then recall from [2, Def. 3.8] that

$$(l,v) \xrightarrow{(\sigma,t)} (l',v') \iff^{\triangle} \exists (l,\delta,\gamma,\sigma,l') \in T_\alpha \text{ s.t. } v+t \vDash \delta \text{ and } v' = (v+t)[\gamma := 0]$$

where $T_\alpha \subseteq (L,\mathcal{E},\Sigma,L)$ is the automaton transition table (cf. [2, Def. 3.7]) and

$$(l,\delta,\gamma,\sigma,l') \in T_\alpha \iff (\sigma,(\gamma,\delta),l') \in \alpha(l).$$

▶ **Example 2.2.** Let α be the coalgebra from Example 2.1, then:

$$\alpha_t(v,l) = \{(\sigma,(v+t)[c := 0],l')\} \qquad \alpha_t(v,l') = \{(\theta,(v+t),l) \mid (v+t) < 2\}.$$

Although $\mathsf{flow}_{t+t'} = \mathsf{flow}_t \circ \mathsf{flow}_{t'}$ we can derive neither $\alpha_{t+t'} \leq \alpha_t \circ \alpha_{t'}$ nor $\alpha_{t+t'} \geq \alpha_t \circ \alpha_{t'}$. This can be routed to clock constraints being as fine grained as being able to single out exact time instants (e.g. $c = 42$) and hence the semantics cannot preserve (not even up-to laxness) the action of time described by flow. Indeed, the notion of TTS does not assume any condition on the time component of transitions (cf. [2, 17]) and the associated notion of execution (or run) maintains transition duration distinct.

There is a 1-1 correspondence between families $\{\alpha_t\}_{t\in\mathbb{T}}$ and (strict) functors $\underline{\alpha}\colon \mathbb{T}^* \to \mathsf{K}$:

$$\underline{\alpha}_{t_1\dots t_n} = \alpha_{t_1} \circ \dots \circ \alpha_{t_n} \text{ for } t_1 \dots t_n \in \mathbb{T}^*$$

where the free monoid $\mathbb{T}^*$ is seen as a single object category: each α_t is $\underline{\alpha}(t)$ for $t\colon * \to *$ and the shared carrier is $\underline{\alpha}(*)$. In order to better illustrate the rôle of the free monoid let $\mathbb{T}$ be the trivial monoid $1 = (\{0\},+,0)$ and recall that $1^* \cong \mathbb{N}$: a functor $\underline{\alpha}\colon 1^* \to \mathsf{K}$ is an endomorphism in K together with all its finite self-compositions (i.e. $\underline{\alpha}(n) = \alpha^n$). As observed in [6] this precisely puts coalgebras with unobservable moves into the (lax) functorial picture as they are endomorphisms in the Kleisli categories for their type monads.

So far we modelled the semantics of timed automata as families of transition systems indexed over time durations i.e. strict functors from $\mathbb{T}^*$ to K. This setting is enough to define "timed" behavioural equivalences by componentwise extension of the "untimed" notion

but, however intuitive it may be, its limited applicability become clear as soon as weak or time-abstract bisimulations are considered. In such cases *lax functors* have to be considered.

In fact, all these notions share a certain pattern: they combine computations abstracting time or unobservable moves (i.e. effects). Since computations of a timed automaton α are described by a functor $\underline{\alpha}\colon \mathbb{T}^* \to \mathsf{K}$, these transformations can be intuitively understood as "turning a functor $\mathbb{T}^* \to \mathsf{K}$ into some functor-like assignment with a different base category". In the remaining of the section we develop a general theory of such transformations we will refer to as *saturations*.

2.3 General saturation for lax functors

In this section we extend the theory of saturation developed in [7, 8, 6] for modelling weak behavioural equivalences for systems with unobservable moves to cope with time in the abstract sense described above. Intuitively, saturation can be understood as a closure of a given system w.r.t. a certain pattern (e.g. reflexive and transitive closure). Akin to loc. cit., the theory is developed in an order enriched setting where the order stems from suitable notion of simulation and non-determinism between systems and provides a notion of approximation between the intermediate steps of the aforementioned closure operation. For timed automata, the ordering is given by pointwise extension of the inclusion order defined by $\mathcal{P}$ (cf. loc. cit.).

Below we assume J is a wide subcategory of an order enriched category K (i.e. a subcategory with all objects from K).

The category of lax functors

Here we focus on recalling the main notions from the theory of order enriched categories and lax functors needed in our paper.

A functor-like assignment π from a category $\mathbb{D}$ to K is called *lax functor* if:

- $id_{\pi D} \leq \pi(id_D)$ for any object $D \in \mathbb{D}$,
- $\pi(d_1) \circ \pi(d_2) \leq \pi(d_1 \circ d_2)$ for any two composable morphisms $d_1, d_2 \in \mathbb{D}$.

Let $\pi, \pi'\colon \mathbb{D} \to \mathsf{K}$ be two lax functors. A family $f = \{f_D\colon \pi D \to \pi' D\}_{D\in\mathbb{D}}$ of morphisms in K is called *lax natural transformation* from π to π' if for any $d\colon D \to D'$ in $\mathbb{D}$ we have $f_{D'} \circ \pi(d) \geq \pi'(d) \circ f_D$. *Oplax functors* and *oplax transformations* are defined by reversing the order in the above. Note that in the more general 2-categorical setting an (op)lax functor and an (op)lax natural transformation are assumed to additionally satisfy extra coherence conditions [18]. In our setting of order enriched categories these conditions are vacuously true, hence we do not list them here.

Let $\mathbb{D}$ be a small category. By $[\mathbb{D}, \mathsf{K}]^J$ we denote the category whose objects are lax functors from $\mathbb{D}$ to K and whose morphisms are oplax transformations with components from J. The category $[\mathbb{D}, \mathsf{K}]^J$ is order enriched with the order on hom-sets given as follows. For $\pi, \pi' \in [\mathbb{D}, \mathsf{K}]^J$ and two oplax transformations $f, f'\colon \pi \to \pi'$ whose components are morphisms in J we define:

$$f \leq f' \iff f_D \leq f'_D \text{ in } \mathsf{K} \text{ for any } D \in \mathbb{D}.$$

Any functor $q\colon \mathbb{D} \to \mathbb{E}$ between small categories induces the *change-of-base functor* $[q, \mathsf{K}]^J\colon [\mathbb{E}, \mathsf{K}]^J \to [\mathbb{D}, \mathsf{K}]^J$ which is given for any object $\pi \in [\mathbb{E}, \mathsf{K}]^J$ and for any oplax transformation $f = \{f_E\colon \pi(E) \to \pi'(E)\}_{E\in\mathbb{E}}$ between $\pi, \pi' \in [\mathbb{E}, \mathsf{K}]^J$ by $[q, \mathsf{K}]^J(\pi) = \pi \circ q$ and $[q, \mathsf{K}]^J(f)_D = f_{q(D)}$.

To keep the paper more succinct, we only focus on $\mathbb{D}$ being a monoid category (i.e. a one-object category). In this case, a monoid $M = (M, \cdot, 1)$ will be often associated with the one-object category it induces. The only object of the category M will be denoted by $*$ and the composition $\circ$ of morphisms $m_1, m_2 \colon * \to *$ for $m_1, m_2 \in M$ is given by: $m_1 \circ m_2 = m_1 \cdot m_2$. In this case, any lax functor $\pi \in [M, \mathsf{K}]^J$ is a family $\pi = \{\pi(m) = \pi_m \colon X \to X\}_{m \in M}$ of endomorphisms in K with a common carrier $X = \pi(*)$ which additionally satisfies [6]:

$$id_X \leq \pi_1 \text{ and } \pi_m \circ \pi_n \leq \pi_{m \cdot n}.$$

An oplax transformation between two lax functors $\pi = \{\pi_m \colon X \to X\}_{m \in M}$ and $\pi' = \{\pi'_m \colon Y \to Y\}_{m \in M}$ in $[M, \mathsf{K}]^J$ is given in terms of an arrow $f \colon X \to Y$ in J which satisfies:

$$f \circ \pi_m \leq \pi'_m \circ f \text{ for any } m \in M.$$

The curious reader is referred to [6] for several examples of these categories.

General saturation

For a functor $q \colon M \to N$ between monoid categories for $(M, \cdot, 1)$ and $(N, \cdot, 1)$ (i.e. a monoid homomorphism) and an order enriched category K we say that K admits *q-saturation* provided that $[q, \mathsf{K}]^J \colon [N, \mathsf{K}]^J \to [M, \mathsf{K}]^J$ admits a left strict 2-adjoint. In order to prove q-saturation admittance below, we work with stronger types of order enrichment on K. We will introduce them now. We say that K is $\mathsf{DCpo}^{\vee}$*-enriched* if it is DCpo-enriched (i.e. each hom-set is a complete order with directed suprema being preserved by the composition) and, additionally, each hom-set admits binary joins (which are *not* assumed to be preserved by the composition). A $\mathsf{DCpo}^{\vee}$-enriched category K is *J-left distributive* if $f \circ (g \vee h) = f \circ g \vee f \circ h$ for any maps $f \in J$ and $g, h \in \mathsf{K}$ with suitable domains and codomains. We define *J-right distributivity* dually. Finally, we say that K is *J-distributive* if it is both, J-left and J-right distributive.

▶ **Theorem 2.3.** *Let $q \colon M \to N$ be a surjective homomorphism and K be J-left distributive (hence $\mathsf{DCpo}^{\vee}$-enriched). Then K admits q-saturation with the left strict 2-adjoint $\Sigma_q : [M, \mathsf{K}]^J \to [N, \mathsf{K}]^J$ given as the identity on morphisms and on any lax functor $\pi \in [M, \mathsf{K}]^J$ as*

$$\Sigma_q(\pi)(*) = \pi(*) \quad \text{and} \quad \Sigma_q(\pi)_w = \bigvee \Pi_w$$

where $w \in N$, $\Pi_w = \bigcup_{n \in \mathbb{N}} \Pi_{w,n}$, $\Pi_{w,0} = \{\pi_{m_1} \vee \ldots \vee \pi_{m_k} \mid q(m_i) = w\}$, and $\Pi_{w,n} = \{t_1 \vee t_2 \vee \ldots \vee t_k \mid t_i \in \Pi_{w_1, n-1} \circ \ldots \circ \Pi_{w_l, n-1}$ and $w_1 \ldots w_l = w\}$.

Proof. Firstly we show that $\Sigma_q(\pi)$ is a well defined lax functor in $[N, \mathsf{K}]^J$. Note that the family $\{\Pi_{w,n}\}_{n \in \mathbb{N}}$ is an ascending family of sets and Π_w is, by surjectivity of q, a non-empty directed set. Moreover, $\Pi_w \circ \Pi_{w'} \subseteq \Pi_{w \cdot w'}$ for any $w, w' \in N$. We have $id \leq \bigvee \Pi_1 \leq \Sigma_q(\pi)_1$ and

$$\Sigma_q(\pi)_w \circ \Sigma_q(\pi)_{w'} = \bigvee \Pi_w \circ \bigvee \Pi_{w'} = \bigvee \Pi_w \circ \Pi_{w'} \leq \bigvee \Pi_{w \cdot w'} = \Sigma_q(\pi)_{w \cdot w'}.$$

This proves the first statement. Now, we will verify that any oplax transformation in $[M, \mathsf{K}]^J$ is mapped onto an oplax transformation in $[N, \mathsf{K}]^J$. Assume $f \circ \pi_m \leq \pi'_m \circ f$ for any $m \in M$. We have $f \circ \Sigma_q(\pi)_w = f \circ \bigvee \Pi_w = \bigvee f \circ \Pi_w \leq \bigvee \Pi'_w \circ f = \Sigma_q(\pi')_w \circ f$, which follows by J-left distributivity, induction and Scott continuity of the composition.

To complete the proof we have to show that for any lax functor $\pi \in [M, \mathsf{K}]^J$ and $\pi' \in [N, \mathsf{K}]^J$ the poset $[M, \mathsf{K}]^J(\pi, [q, \mathsf{K}](\pi'))$ is isomorphic to $[N, \mathsf{K}]^J(\Sigma_q(\pi), \pi')$. Indeed:

$$
\begin{aligned}
&f \circ \pi_m \leq \pi'_{q(m)} \circ f \text{ for any } m \in M \overset{(\dagger)}{\iff} \\
&f \circ (\pi_{m_1} \vee \ldots \vee \pi_{m_k}) \leq \pi'_{q(m)} \circ f \text{ for any } m, m_1 \ldots m_k \in M \text{ and } q(m_i) = q(m) \iff \\
&\forall m \in M, f \circ \bigvee \Pi_{q(m),0} \leq \pi'_{q(m)} \circ f \overset{(\dagger\dagger)}{\iff} \forall m \in M, \forall n \in \mathbb{N}, f \circ \bigvee \Pi_{q(m),n} \leq \pi'_{q(m)} \circ f \\
&\iff \forall m \in M, f \circ \bigvee \Pi_{q(m)} \leq \pi'_{q(m)} \circ f \iff \forall m \in M, f \circ \Sigma_q(\pi)_{q(m)} \leq \pi'_{q(m)} \circ f.
\end{aligned}
$$

The equivalence (†) follows by J-left distributivity of K and (††) by induction on $n \in \mathbb{N}$. ◀

If K comes equipped with an even stronger type of order enrichment then the formula for Σ_q simplifies considerably. Indeed, we have the following (see also [6, Th. 3.14]).

▶ **Theorem 2.4.** *If the hom-posets of K admit arbitrary non-empty suprema which are preserved by the composition then for any $\pi \in [M, \mathsf{K}]^J$ we have:*

$$\Sigma_q(\pi)_w = \bigvee_{m:q(m)=w} \pi_m.$$

Proof. It follows by the fact that in this case we have $\Pi_w = \Pi_{w,0}$. Hence, $\Sigma_q(\pi)_w = \bigvee \Pi_w = \bigvee \Pi_{w,0} = \bigvee_{m:q(m)=w} \pi_m$. ◀

▶ Remark. The notion of lax functorial saturation has already been considered in [6] whenever $N = 1$ is the one-element monoid and $q \colon M \to 1$ the unique homomorphism sending all elements of M to the neutral element of 1. In that case it was simply called saturation and was shown to be directly linked to the notion of coalgebraic weak bisimulation.

In the remaining of this subsection we take $q_1 \colon M \to N_1$, $q_2 \colon N_1 \to N_2$ and $q \colon M \to N$ to be monoid homomorphisms and K to be J-left distributive (hence $\mathsf{DCpo}^\vee$-enriched).

▶ **Theorem 2.5.** *The functor $\Sigma_{q_2} \circ \Sigma_{q_1}$ is (naturally isomorphic to) $\Sigma_{q_2 \circ q_1}$.*

Proof. This statement follows directly by the fact that Σ_{q_1} and Σ_{q_2} are left adjoints and that $[q_1, \mathsf{K}]^J \circ [q_2, \mathsf{K}]^J = [q_2 \circ q_1, \mathsf{K}]^J$. ◀

▶ **Theorem 2.6.** *The functor $\Sigma_q \circ [q, \mathsf{K}]^J$ is the identity on $[N, \mathsf{K}]^J$.*

Proof. It is enough to show that for any $\pi' \in [N, \mathsf{K}]^J$ we have $\Sigma_q(\pi' \circ q) = \pi'$. Take $\pi = \pi' \circ q$ and note that $\Pi_{w,0} = \{\pi_{m_1} \vee \ldots \vee \pi_{m_k} \mid q(m_i) = w\} = \{\pi'_w \vee \ldots \vee \pi'_w \mid q(m_i) = w\} = \{\pi'_w\}$ for $w \in N$. Moreover, since π' is a lax functor we can easily prove by induction that $t \leq \pi'_w$ holds for any $t \in \Pi_{w,n}$. Hence, $\Sigma_q(\pi' \circ q)_w = \bigvee \Pi_w = \pi'_w$ which proves the assertion. ◀

▶ **Theorem 2.7.** *If K is J-distributive then, for any $\pi, \pi' \in [M, \mathsf{K}]^J$ and $f \colon \pi(*) \to \pi'(*) \in J$:*

$$f \circ \pi_m = \pi'_m \circ f \text{ for any } m \in M \implies f \circ \Sigma_q(\pi)_w = \Sigma_q(\pi')_w \circ f \text{ for any } w \in N.$$

Proof. We have:

$$f \circ (\pi_{m_1} \vee \ldots \vee \pi_{m_k}) \overset{J\text{-LD}}{=} f \circ \pi_{m_1} \vee \ldots \vee f \circ \pi_{m_k} = \pi'_{m_1} \circ f \vee \ldots \vee \pi'_{m_k} \circ f \overset{J\text{-RD}}{=} (\pi'_{m_1} \vee \ldots \vee \pi'_{m_k}) \circ f.$$

Hence, for any $f \circ \Pi_{w,0} = \Pi'_{w,0} \circ f$. By induction it follows that $f \circ \Pi_{w,n} = \Pi'_{w,n} \circ f$ for any n. Hence, $f \circ \Pi_w = \Pi'_w \circ f$ which proves the assertion. ◀

2.4 Saturation-based behavioural equivalences

Notions of coalgebraic strong bisimulation have been well captured in the literature [1, 22, 26]. This paper builds on a variant called *kernel bisimulation* i.e. "a relation which is the kernel of a compatible refinement of a system" [26]. However, since coalgebras with internal moves are endomorphisms in suitable Kleisli categories[7], kernel bisimulation has been reformulated in [8] in the language of *behavioural morphisms* for endomorphisms in an arbitrary category K. This choice allows us to streamline the exposition while working at the abstract level of morphism between endomorphisms and accommodate general saturation along the lines of [8]. Let us quickly recall this notion here. We say that $f\colon X \to Y \in J$ is a *behavioural morphism* on an endomorphism $\alpha\colon X \to X \in \mathsf{K}$ provided that there is $\beta\colon Y \to Y \in \mathsf{K}$ such that $f \circ \alpha = \beta \circ f$. If $\mathsf{K} = \mathcal{K}l(T)$ for a monad T on C and J is the category whose objects are those of C and whose morphisms are $\eta_Y \circ f$, where η is the unit of the monad and $f\colon X \to Y$ a morphism of C, then the behavioural morphisms coincide with homomorphisms of T-coalgebras [8]. This approach is adopted in this paper taking lax functors and their saturation into account.

▶ **Definition 2.8.** Let $\pi \in [M, \mathsf{K}]^J$ and $X = \pi(*)$. A morphism $f\colon X \to Y$ in J is called *q-behavioural morphism* on π provided that there is a lax functor $\pi' \in [N, \mathsf{K}]^J$ such that $\pi'(*) = Y$ for any $n \in N$ the following holds:

$$f \circ \Sigma_q(\pi)_n = \pi'_n \circ f.$$

A *q-bisimulation* on π is a relation $R \rightrightarrows \pi(*)$ (i.e. a jointly monic span) in J which is the kernel pair of a q-behavioural morphism on π.

▶ **Theorem 2.9.** *A morphism $f\colon X \to Y$ in J is a q-behavioural morphism on $\pi \in [M, \mathsf{K}]^J$ if and only if there is $\pi' \in [M, \mathsf{K}]^J$ such that $f \circ \Sigma_q(\pi)_n = \Sigma_q(\pi')_n \circ f$ for any $n \in N$.*

Proof. By Theorem 2.6. ◀

Definition 2.8 is a conservative extension of kernel bisimulation. In fact, $\underline{\alpha} \in [M^*, \mathsf{K}]$ describes a family of endomorphisms in K, i.e., a timed system, and all its finite self-composition and since $\Sigma_{id_{M^*}} \cong \mathcal{I}d$, id_{M^*}-bisimulation coincides with kernel bisimulation.

Among the several choices of monoid homomorphisms we focus on two main families: those abstracting effects (e.g. non-determinism) and those abstracting aspects modelled by the context (e.g. time). The first case corresponds to $\varepsilon_M\colon M^* \to M$ where $\varepsilon\colon (-)^* \to \mathcal{I}d$ is the counit forming the free monoid adjunction. By Theorem 2.4, if K admits and preserves arbitrary non-empty joins the functor $\Sigma_{\varepsilon_M}\colon [M^*, \mathsf{K}]^J \to [M, \mathsf{K}]^J$ is given on a system $\underline{\alpha}$:

$$\Sigma_{\varepsilon_M}(\underline{\alpha})_m = \bigvee_{\vec{m} \in \varepsilon_M^{-1}(m)} \underline{\alpha}_{\vec{m}} = \bigvee_{m_1 \cdot \ldots \cdot m_k = m} \alpha_{m_1} \circ \cdots \circ \alpha_{m_k}.$$

Steps in the saturated system are combinations of all sequences in the original one associated with any decomposition of the stage m. In particular, when K models unobservable moves, ε_M-bisimulation can be seen as the weak counterpart of strong behavioural equivalence for systems modelled in $[M^*, \mathsf{K}]^J$, i.e., id_{M^*}-bisimulation. The second case corresponds to $q^*\colon M^* \to N^*$ for some $q\colon M \to N$. The functor $\Sigma_{q^*}\colon [M^*, \mathsf{K}]^J \to [N^*, \mathsf{K}]^J$ is given on a system $\underline{\alpha}$ as:

$$\Sigma_{q^*}(\underline{\alpha})_{n_1 \ldots n_k} = \bigvee_{n_i = q(m_i)} \underline{\alpha}_{m_1 \ldots m_k} = \bigvee_{n_i = q(m_i)} \alpha_{m_1} \circ \cdots \circ \alpha_{m_k}.$$

Steps in the saturated system are combinations of all steps associated to a pre-image through q; in particular, $\Sigma_{q^*}(\underline{\alpha})_n = \bigvee_{n = q(m)} \alpha_m$. When $!_M\colon M \to 1$ is considered, all the information

in M is erased whereas computational effects in K are preserved. From this perspective $!^*_M$-bisimulation can be seen as a conservative generalisation of time-abstract bisimulations (cf. Section 3.1). These two orthogonal directions can be combined as $\varepsilon_N \circ q^*$ or as $q \circ \varepsilon_M$; by naturality of ε the notions of saturation coincide:

$$\Sigma_{\varepsilon_N \circ q^*}(\underline{\alpha})_n = \Sigma_{q \circ \varepsilon_M}(\underline{\alpha})_n = \bigvee_{\substack{n_1 \cdot \ldots \cdot n_k = n \\ n_i = q(m_i)}} \alpha_{m_1} \circ \cdots \circ \alpha_{m_k} = \bigvee_{\substack{n = q(m) \\ m_1 \cdot \ldots \cdot m_k = m}} \alpha_{m_1} \circ \cdots \circ \alpha_{m_k}.$$

When K models unobservable moves and N is 1, $!_{M^*}$-bisimulation can be thought as the weak counterpart of $!^*_M$-bisimulation.

The notion of q-bisimulation for systems modelled in $[M, \mathsf{K}]^J$ arise from some congruence for M and, *vice versa*, each congruence defines a notion of q-bisimulation. The following theorem states that that coarser congruences define coarser notions of bisimulations. In the remainder of this section we assume that $q_1 \colon M \to N_1$ and $q_2 \colon N_1 \to N_2$ are monoid morphisms and K is a J-distributive $\mathsf{DCpo}^\vee$-enriched category.

▶ **Theorem 2.10.** *A q_1-behavioural morphism on π is a $(q_2 \circ q_1)$-behavioural morphism on π.*

Proof. Follows directly by Theorems 2.5 and 2.7. ◀

▶ **Corollary 2.11.** *Let $q_1 \colon M \to N_1$ and $q_2 \colon N_1 \to N_2$ be monoid homomorphisms. For any $\pi \in [M, \mathsf{K}]^J$, if $R \rightrightarrows \pi(*)$ is a q_1-bisimulation on π then it is a $(q_2 \circ q_1)$-bisimulation on π.*

Thus, q-bisimulations for systems modelled in the context of $[M, \mathsf{K}]^J$ form a spectrum where the finest and coarsest notions are defined by $id_M \colon M \to M$ and $!_M \colon M \to 1$, respectively.

3 Application: timed behavioural and language equivalences

This section instantiates the general framework developed in Section 2.4 to timed automata covering semantic equivalences of interest such as (weak) timed and time-abstract bisimulations and (finite) timed and untimed language equivalences.

3.1 Behavioural equivalences

This subsection recovers known notions of behavioural equivalences for timed automata as instances of q-bisimulation thus deriving the spectrum illustrated in Figure 1. Before stating these results, recall from [2, 19] the following definitions of the main bisimulations of interest.

▶ **Definition 3.1** ([2, 19])**.** For a TTS α and an equivalence relation R on its carrier:

- R is a *(strong) timed bisimulation* for α if $(x, y) \in R$ and $x \xrightarrow{(\sigma,t)} x'$ implies that there is $y' \in X$ such that $y \xrightarrow{(\sigma,t)} y'$ and $(x', y') \in R$;
- R is a *(strong) time-abstract bisimulation* for α if $(x, y) \in R$ and $x \xrightarrow{(\sigma,t)} x'$ implies that there are $y' \in X$ and $t' \in [0, \infty)$ such that $y \xrightarrow{(\sigma,t')} y'$ and $(x', y') \in R$;
- R is a *weak timed bisimulation* for α if $(x, y) \in R$ and $x \xRightarrow{(\sigma,t)} x'$ implies that there is $y' \in X$ such that $y \xRightarrow{(\sigma,t)} y'$ and $(x', y') \in R$;
- R is a *weak time-abstract bisimulation* for α if $(x, y) \in R$ and $x \xRightarrow{(\sigma,t)} x'$ implies that there are $y' \in X$ and $t' \in [0, \infty)$ such that $y \xRightarrow{(\sigma,t')} y'$ and $(x', y') \in R$;

where $\to$ and $\Rightarrow$ denote the transition relation of α and the smallest relation such that:

$$\frac{}{x \overset{(\tau,0)}{\Longrightarrow} x} \qquad \frac{x \xrightarrow{(\tau,t)} y}{x \overset{(\tau,t)}{\Longrightarrow} y} \qquad \frac{x \overset{(\tau,t_0)}{\Longrightarrow} x' \quad x' \xrightarrow{(\sigma,t_1)} y' \quad y' \overset{(\tau,t_2)}{\Longrightarrow} y \quad t = t_0 + t_1 + t_2}{x \overset{(\sigma,t)}{\Longrightarrow} y}$$

▶ **Theorem 3.2.** *For a TTS α, an equivalence relation R on its carrier is a timed/weak timed/time-abstract/weak time-abstract bisimulation for α if, and only if, it is a $id_{\mathbb{T}^*}$-/$\varepsilon_{\mathbb{T}}$-/$!^*_{\mathbb{T}}$-/$!_{\mathbb{T}^*}$-bisimulation for $\underline{\alpha}$, respectively.*

▶ **Corollary 3.3.** *For a TTS α and an equivalence relation R on its carrier, the implications illustrated in Figure 1 hold true.*

Proof. By the (inner) commuting diagram in Figure 1, Corollary 2.11 and Theorem 3.2. ◀

Finally, since neither $!^*_{\mathbb{T}}$ factors $\varepsilon_{\mathbb{T}}$ nor *vice versa* it is not possible to derive that, in general, strong time-abstract bisimulations are weak timed ones nor *vice versa*.

3.2 Finite trace equivalence

This subsection focuses on putting the language equivalence of timed systems into the lax functorial setting. Here, we work with $\mathsf{K} = \mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$- the Kleisli category for the ε-NA monad together with the monoidal structure $(\times, 1)$. A timed coalgebra $\alpha\colon L \to \mathcal{E} \otimes L$ in K is a set map $\alpha\colon L \to \mathcal{P}(\Sigma_\tau \times \mathcal{E} \times L + \{\checkmark\})$ and it represents an ordinary timed automaton with a specified set of terminal states (here, this set is given by $\{l \in L \mid \checkmark \in \alpha(l)\}$). Classically, its semantics is a transition system $\mathcal{V} \times L \to \mathcal{P}(\Sigma_\tau \times \mathbb{T} \times \mathcal{V} \times L + \{\checkmark\})$ over the state-space $\mathcal{V} \times L$, which is obtained from α analogously to (1) taking into account the terminals [2]. Systems of type $X \to \mathcal{P}^{\Sigma}_{\checkmark}(\mathbb{T} \times X)$ will be referred to as *timed transition systems with accepting states* or $\checkmark$*-TTS* in short. However, from our perspective, and following the guidelines of the setting from Section 2.2, the semantics of α is given in terms of a functor $\underline{\alpha}\colon \mathbb{T}^* \to \mathsf{K}$ with $\underline{\alpha}_{t_1,\ldots,t_n} = \alpha_{t_1} \circ \ldots \circ \alpha_{t_n}$, where $\alpha_t\colon \mathcal{V} \times L \to \mathcal{P}^{\Sigma}_{\checkmark}(\mathcal{V} \times L)$ for (v, l) is defined by:

$$\begin{aligned}\alpha_t(v,l) = &\{(\sigma, v', l') \mid (\sigma, l', (\gamma, \delta)) \in \alpha(l),\ v_t = \lambda c.v(c) + t,\ v_t \vDash \delta, \text{ and } v' = v_t[\gamma := 0]\} \\ &\cup \textbf{ if } \checkmark \in \alpha(l) \textbf{ then } \{\checkmark\} \textbf{ else } \emptyset.\end{aligned}$$

Now, in order to obtain finite (weak) (un)timed trace equivalence, we will play with the second parameter that the q-bisimulation takes implicitly into account, namely the base category J. Here, we consider J to be different from Set. We put J to be the Kleisli category $\mathcal{K}l(\mathcal{P})$ for the powerset monad, as it is a subcategory of $\mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$ with the inclusion functor $(-)^\sharp\colon \mathcal{K}l(\mathcal{P}) \to \mathcal{K}l(\mathcal{P}^{\Sigma}_{\checkmark})$ given, for any X and $f\colon X \to \mathcal{P}Y$ by $X^\sharp = X$ and $f^\sharp(x) = \{(\tau, y) \mid y \in f(x)\}$. Taking the source category of behavioural morphisms to be different from Set is akin to the classical approach towards modelling coalgebraic finite trace equivalence in terms of bisimulation for base categories given by the Kleisli categories for a monadic part of the type functor [12, 14].

Given a state x of a $\checkmark$-TTS we define the following four sets of languages it accepts:

$$\begin{aligned}&T(x) = \{(\sigma_1, t_1)\ldots(\sigma_n, t_n) \in (\Sigma_\tau \times \mathbb{T})^* \mid x \xrightarrow{(\sigma_1,t_1)} \ldots \xrightarrow{(\sigma_n,t_n)} x_n \text{ and } x_n \downarrow\},\\ &UT(x) = \{\sigma_1 \ldots \sigma_n \in \Sigma^*_\tau \mid \exists t_1, \ldots t_n \text{ s.t.} (\sigma_1, t_1)\ldots(\sigma_n, t_n) \in T(x)\},\\ &WT(x) = \{(\sigma_1, t_1)\ldots(\sigma_n, t_n) \in (\Sigma \times \mathbb{T})^* \mid x \overset{(\sigma_1,t_1)}{\Longrightarrow} \ldots \overset{(\sigma_n,t_n)}{\Longrightarrow} x_n \text{ and } x_n \downarrow\},\\ &WUT(x) = \{\sigma_1 \ldots \sigma_n \in \Sigma^* \mid \exists t_1, \ldots t_n \text{ s.t.} (\sigma_1, t_1)\ldots(\sigma_n, t_n) \in WT(x)\}.\end{aligned}$$

▶ **Theorem 3.4.** *Let* $\alpha\colon X \to \mathcal{P}(\Sigma_\tau \times \mathbb{T} \times X + \{\checkmark\})$ *be a* $\checkmark$*-TTS with its induced functor denoted by* $\underline{\alpha}\colon \mathbb{T}^* \to \mathsf{K}$*. A morphism* $f\colon X \to \mathcal{P}Y$ *in* $\mathcal{K}l(\mathcal{P})$ *is a* $id_{\mathbb{T}^*}$*-/*$\varepsilon_{\mathbb{T}}$*-/*$!^*_{\mathbb{T}}$*-/*$\Lambda_{\mathbb{T}^*}$*-behavioural morphism on* $\underline{\alpha}$ *making* $f(x) = f(x')$ *for* $x, x' \in X$ *if, and only if we have respectively:*

$$T(x) = T(x') \;/\; WT(x) = WT(x') \;/\; UT(x) = UT(x') \;/\; WUT(x) = WUT(x').$$

4 Conclusions

In this paper we presented a framework for modelling timed automata and showed how behavioural equivalences of interest can be recovered in a uniform and general way. Moreover, we were able to organise, by means of simple diagrams of monoid homomorphisms, these equivalences in a spectrum on the base of their discriminating power—as Figure 1 exemplifies.

Lax functors are the cornerstone these results build on for they allow us to separate aspects of the computations under scrutiny into those arising from effects (base category) and from the environment (index category). We extended the theory of saturation [6, 8, 7] developing the notion of *general saturation* for lax functors on a monoid category. Akin to saturation cf. loc. cit., this construction offers a uniform definition of behavioural equivalences and a fine control on the computational aspects they abstract away, being these time or unobservable moves. Thanks to its categorical development, this framework can accommodate any computational aspect modelled by some monoid M together with a $\mathsf{DCpo}^{\vee}$-enriched, J-distributive category K. Besides non-deterministic systems we should mention Segala, weighted, probabilistic, nominal, and continuous systems as possible instances for K and refer the curious reader to [6, 8]. Although the categories formed by these systems do not necessarily satisfy left distributivity, in [6, 8] we presented a method to obtain it by moving to a richer category and performing saturation there. This suggests that results described here should readily generalise q-behavioural morphisms to systems which do not necessarily satisfy this paper assumptions.

Although timed processes present some fundamental differences w.r.t. timed automata, we mention the interesting account of these systems and their strong bisimulation by Kick [15]. In loc. cit., time-dependent computations are modelled by a suitable comonad over Set and then combined with other behavioural aspects by means of comonad products. Because of technical difficulties associated with comonad products, this approach appears less flexible when behavioural equivalences beside strong timed bisimulation are considered.

Perhaps the closest work to ours is [10] where a categorical account of timed weak bisimulation is proposed. We remark that loc. cit. relies on open-map bisimulation and is limited to weak timed bisimulation for timed transition systems only.

The categorical characterization of timed automata paves the way for further interesting lines of research. One is to explore the framework expressiveness providing new instances besides timed. Another is to extend the framework with results from the rich theory of coalgebras such as minimization [3], determinisation [23], and up-to techniques [4]. Finally, we believe this, and recent works like [6, 8] suggest the rich and malleable setting of lax functors as a context where to study complex interactions of several computational aspects. To this end, we plan to further develop the general theory of lax functors and saturation.

References

1 Peter Aczel and Nax Mendler. A final coalgebra theorem. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Proc. CTCS*, volume 389 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 1989.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

3 Filippo Bonchi, Marcello M. Bonsangue, Helle Hvid Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva. Algebra-coalgebra duality in brzozowski's minimization algorithm. *ACM Trans. Comput. Log.*, 15(1):3, 2014.

4 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In *CSL-LICS*, pages 20:1–20:9. ACM, 2014.

5 Tomasz Brengos. On coalgebras with internal moves. In Marcello M. Bonsangue, editor, *Proc. CMCS*, Lecture Notes in Computer Science, pages 75–97. Springer, 2014.

6 Tomasz Brengos. Lax functors and coalgebraic weak bisimulation. *CoRR*, abs/1404.5267, 2015.

7 Tomasz Brengos. Weak bisimulation for coalgebras over order enriched monads. *Logical Methods in Computer Science*, 11(2:14):1–44, 2015.

8 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.

9 Taolue Chen, Tingting Han, and Joost-Pieter Katoen. Time-abstracting bisimulation for probabilistic timed automata. In *TASE*, pages 177–184. IEEE Computer Society, 2008.

10 Natalya Gribovskaya and Irina Virbitskaite. A categorical view of timed weak bisimulation. In *TAMC*, volume 6108 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2010.

11 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic forward and backward simulations. In *Proc. JSSST Annual Meeting*, 2006.

12 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.

13 Bart Jacobs. Coalgebraic trace semantics for combined possibilistic and probabilistic systems. In *Proc. CMCS*, Electronic Notes in Theoretical Computer Science, pages 131–152, 2008.

14 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. In Dirk Pattinson and Lutz Schröder, editors, *Proc. CMCS*, volume 7399 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2012.

15 Marco Kick. *A Mathematical Model of Timed Processes*. Ph.D. dissertation, University of Edinburgh, 2003.

16 Anders Kock. Strong functors and monoidal monads. *Arc. Math.*, 23(1):113–120, 1972.

17 Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

18 Stephen Lack. *A 2-categories companion*. Springer, 2010.

19 Kim G. Larsen and Yi Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75 – 101, 1997.

20 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

21 Marino Miculan and Marco Peressotti. Structural operational semantics for non-deterministic processes with quantitative aspects. *Theoretical Computer Science*, 2016.

22 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

23 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013.

24 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.

25 Paweł Sobociński. Relational presheaves, change of base and weak simulation. *Journal of Computer and System Sciences*, 81(5):901–910, 2015.

26 Sam Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1), 2011.

27 Daniele Turi and Gordon Plotkin. Towards a mathematical operational semantics. In *Proc. LICS*, pages 280–291. IEEE Computer Society Press, 1997.

Analyzing Timed Systems Using Tree Automata*

S. Akshay[1], Paul Gastin[2], and Shankara Narayanan Krishna[1]

1 **Dept. of CSE, IIT Bombay, Powai, Mumbai 400076, India**
akshayss@cse.iitb.ac.in
2 **LSV, ENS-Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France**
paul.gastin@lsv.ens-cachan.fr
1 **Dept. of CSE, IIT Bombay, Powai, Mumbai 400076, India**
krishnas@cse.iitb.ac.in

Abstract

Timed systems, such as timed automata, are usually analyzed using their operational semantics on timed words. The classical region abstraction for timed automata reduces them to (untimed) finite state automata with the same time-abstract properties, such as state reachability. We propose a new technique to analyze such timed systems using finite tree automata instead of finite word automata. The main idea is to consider timed behaviors as graphs with matching edges capturing timing constraints. Such graphs can be interpreted in trees opening the way to tree automata based techniques which are more powerful than analysis based on word automata. The technique is quite general and applies to many timed systems. In this paper, as an example, we develop the technique on timed pushdown systems, which have recently received considerable attention. Further, we also demonstrate how we can use it on timed automata and timed multi-stack pushdown systems (with boundedness restrictions).

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Timed automata, tree automata, pushdown systems, tree-width

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.27

1 Introduction

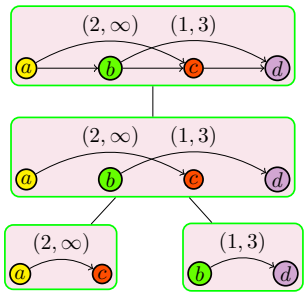
The advent of timed automata [4] marked the beginning of an era in the verification of real-time systems. Today, timed automata form one of the well accepted real-time modelling formalisms, using real-valued variables called clocks to capture time constraints. The decidability of the emptiness problem for timed automata is achieved using the notion of region abstraction. This gives a sound and finite abstraction of an infinite state system, and has paved the way for state-of-the-art tools like UPPAAL, which have successfully been used in the verification of several complex timed systems. In recent times [1, 6, 13] there has been a lot of interest in the theory of verification of more complex timed systems enriched with features such as concurrency, communication between components and recursion with single or multiple threads. In most of these approaches, decidability has been obtained by cleverly extending the fundamental idea of region or zone abstractions.

In this paper, we give a technique for analyzing timed systems, inspired from a different approach based on graphs and tree automata. This approach has been exploited for analyzing various types of *untimed systems*, e.g., [17, 10]. The basic template of this approach has three steps: (1) capture the behaviors of the system as graphs, (2) show that the class of

* This work was partly supported by LIA InForMeL, Indo-French CEFIPRA project AVERTS and DST-INSPIRE faculty award [IFA12-MA-17].

graphs that are actual behaviors of the system is MSO-definable, and (3) show that this class of graphs has bounded tree-width (or clique-width or split-width), or restrict the analysis to such bounded behaviors. Then, non-emptiness of the given system boils down to the satisfiability of an MSO sentence on graphs of bounded tree-width, which is decidable by Courcelle's theorem. Since, graphs of bounded tree-width can be interpreted in binary trees, the problem reduces to non-emptiness of a tree automaton whose existence follows from Courcelle's theorem. But, by providing a direct construction of the tree automaton, it is possible to obtain a good complexity for the decision procedure.

Our technique starts similarly, by replacing timed word behaviors of timed systems with graphs consisting of untimed words with additional time-constraint edges, called words with timing constraints (TCWs). However, the main complication here is that a TCW describes a run of the timed system, where the constraints are recorded but not checked. The TCW corresponds to an actual concrete run iff it is *realizable*. So, we are interested in the class of graphs which are *realizable* TCWs. The structural property that a graph must be a TCW is MSO-definable. However, it is unclear whether realizability is MSO definable over words with timing constraints. Given this, we cannot directly apply the approach of [17, 10]. Instead, we work on decomposition trees and construct a finite tree automaton checking realizability, which is the most involved part of the paper. More precisely, we show that words with timing constraints (TCWs) which are behaviors of certain classes of timed systems (like timed pushdown systems) are graphs of bounded split/tree-width. Hence, these graphs admit binary tree decompositions as depicted in the adjoining figure. Each node of the tree depicts an incomplete behavior/graph of the system, and by combining these behaviors as we go up the tree, we obtain a full or complete behavior (run) of the system. We construct a tree automaton that checks if the generated graph encoded as a tree satisfies the ***ValCoRe*** property (1) *Validity*: The root node depicts a syntactically correct labeled graph (TCW); (2) *Correctness of run*: The graph is indeed a correct run of the underlying timed system and; (3) *Realizability*: The root node depicts a realizable graph, i.e., we can find timestamps that realize all timing-constraints. To check realizability, the tree automaton needs to maintain a *finite* abstraction for each subtree encoding a TCW. Thanks to the bound on split/tree-width, our abstraction keeps a bounded number of positions, called end-points, in the (arbitrarily large) TCW. It subsumes (arbitrarily long) paths of timing constraints in the TCW by new timing constraints between these end-points. The constants in these new constraints are sums of original constants and may grow unboundedly. Hence, a key difficulty is to introduce suitable abstractions which aid in bounding the constants, while at the same time preserving realizability. Using tree decompositions of graph behaviors of bounded split/tree-width and tree automata proved to be a very successful technique for the analysis of *untimed* infinite state systems [17, 11, 10, 2]. This paper opens up this powerful technique for analysis of *timed* systems.



To illustrate the technique, we have reproved the decidability of non-emptiness of timed automata and timed pushdown automata (TPDA), by showing that both these models have a split-width ($|X|+3$ and $4|X|+6$) that is linear in the number of clocks X of the underlying system. This bound directly tells us the amount of information that we need to maintain in the construction of the tree automata. For TPDA we obtain an ExpTime algorithm, matching the known lower-bound for the emptiness problem of TPDA. For timed automata, since the split-trees are word-like (at each binary node, one subtree is small) we may use

word automata instead of tree automata, reducing the complexity from ExpTime to PSpace, again matching the lower-bound. Interestingly, if one considers TPDA with no explicit clocks, but the stack is timed, then the split-width is a constant, 2. In this case, we have a polynomial time procedure to decide emptiness, assuming a unary encoding of constants in the system. To further demonstrate the power of our technique, we derive a new decidability result for non-emptiness of timed multi-stack pushdown automata under bounded rounds, by showing that the split-width of this model is again linear in the number of clocks, stacks and rounds. Exploring decidable subclasses of untimed multi-stack pushdown systems is an active research area [5, 12, 14, 16, 15], and our technique can extend these to handle time.

It should be noticed that the tree automata for validity and realizability (the most involved construction of this paper) are independent of the timed system under study. Hence, to apply the technique to other systems, one only needs to prove the bound on split-width and to show that their runs can be captured by tree automata. This is a major difference compared to many existing techniques for timed systems which are highly system dependent. Finally, we mention an orthogonal approach to deal with timed systems given in [6], where the authors show the decidability of the non-emptiness problem for a class of timed pushdown automata by reasoning about sets with timed-atoms. Detailed proofs and illustrative examples, omitted due to lack of space, can be found in [3].

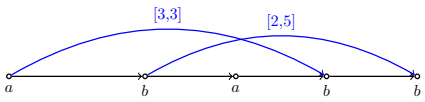
2 Graphs for behaviors of timed systems

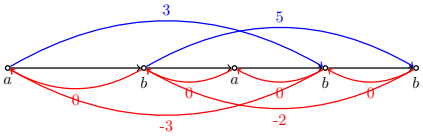
We fix an alphabet Σ and use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$ where ε is the silent action. We also fix a finite set of closed intervals $\mathcal{I}$ which contains the special interval $[0,0]$. For a set S, we use $\leq \subseteq S \times S$ to denote a partial or total order on S. For any $x, y \in S$, we write $x < y$ if $x \leq y$ and $x \neq y$, and $x \lessdot y$ if $x < y$ and there does not exist $z \in S$ such that $x < z < y$.

2.1 Abstractions of timed behaviors

▶ **Definition 1.** A *word with timing constraints* (TCW) over $\Sigma, \mathcal{I}$ is a structure $\mathcal{V} = (P, \to, \lambda, \rhd, \theta)$ where P is a finite set of positions or points, $\lambda\colon P \to \Sigma_\varepsilon$ labels each position, the reflexive transitive closure $\leq = \to^*$ is a total order on P and $\to = \lessdot$ is the successor relation, $\rhd \subseteq < = \to^+$ gives the pairs of positions carrying a timing constraint, whose interval is given by $\theta\colon \rhd \to \mathcal{I}$.

For any position $i \in P$, the *indegree* (resp. *outdegree*) of i is the number of positions j such that $(j,i) \in \rhd$ (resp. $(i,j) \in \rhd$). A TCW is *simple* (denoted STCW) if each position has at most one timing constraint (incoming or outgoing) attached to it, i.e., for all $i \in P$, *indegree(i) + outdegree(i)* ≤ 1. A TCW is depicted below (left) with positions $1, 2, \ldots, 5$ labelled over $\{a, b\}$. *indegree*(4)=1, *outdegree*(1)=1 and *indegree*(3)=0. The curved edges decorated with intervals connect the positions related by $\rhd$, while straight edges are the successor relation $\to$. Note that this TCW is *simple*.





An *ε-timed word* is a sequence $w = (a_1, t_1) \ldots (a_n, t_n)$ with $a_1 \ldots a_n \in \Sigma_\varepsilon^+$ and $(t_i)_{1 \leq i \leq n}$ a non-decreasing sequence of real time values. If $a_i \neq \varepsilon$ for all $1 \leq i \leq n$, then w is a *timed word*. The projection on Σ of an ε-timed word is the timed word obtained by removing ε-labelled positions. Consider a TCW $W = (P, \to, \lambda, \rhd, \theta)$ with $P = \{1, \ldots, n\}$.

A timed word w is a *realization* of W if it is the projection on Σ of an ε-timed word $w' = (\lambda(1), t_1) \dots (\lambda(n), t_n)$ such that $t_j - t_i \in \theta(i,j)$ for all $(i,j) \in \rhd$. In other words, a TCW is realizable if there exists a timed word w which is a realization of W. For example, the timed word $(a, 0.9)(b, 2.1)(a, 2.1)(b, 3.9)(b, 5)$ is a realization of the TCW depicted above (left), while $(a, 1.2)(b, 2.1)$ $(a, 2.1)(b, 3.9)(b, 5)$ is not.

We can (and often will) view a TCW W as a *directed* weighted graph with edges $E = \rhd \cup \rhd^{-1} \cup \rightarrow^{-1}$ and weights induced by θ as follows: if $(i,j) \in \rhd$ and $\theta(i,j) = [I_\ell, I_r]$ then the weight of the *forward edge* is the upper constraint $\mathsf{wt}(i,j) = I_r$ and the weight of the *back edge* is the negative value of the lower constraint $\mathsf{wt}(j,i) = -I_\ell$. Further, to ensure that time is non-decreasing we add 0-weight back edges between consecutive positions that are not already constrained, i.e., if $(i,j) \in \lessdot \setminus \rhd$ then $\mathsf{wt}(j,i) = 0$. The directed weighted graph depicted above (right) corresponds to the TCW on its left. A *directed* path in W is a sequence of positions $\rho = p_1, p_2, \dots, p_n$ $(n > 1)$ linked with edges: $(p_i, p_{i+1}) \in E$ for all $1 \leq i < n$. It is a cycle or loop if $p_n = p_1$. Its weight is $\mathsf{wt}(\rho) = \sum_{1 \leq i < n} \mathsf{wt}(p_i, p_{i+1})$. Then, we have the following standard result:

▶ Proposition 2 ([7]). A TCW W is realizable iff it has no negative cycles.

Thus, to check if a TCW is realizable, we check for absence of negative weight cycles, which can be done in polynomial time, e.g., using the Bellman Ford algorithm (see [7] for details).

2.2 TPDA and their semantics as simple TCWs

Dense-timed pushdown automata (TPDA), introduced in [1], are an extension of timed automata, and operate on a finite set of real-valued clocks and a stack which holds symbols with their ages. The age of a symbol in the stack represents time elapsed since it was pushed on to the stack. Formally, a TPDA $\mathcal{S}$ is a tuple $(S, s_0, \Sigma, \Gamma, \Delta, X, F)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ, Γ, are respectively a finite set of input, stack symbols, Δ is a finite set of transitions, X is a finite set of real-valued variables called clocks, $F \subseteq S$ are final states. A transition $t \in \Delta$ is a tuple $(s, \gamma, a, \mathsf{op}, R, s')$ where $s, s' \in S$, $a \in \Sigma$, γ is a finite conjunction of atomic formulae of the kind $x \in I$ for $x \in X$ and $I \in \mathcal{I}$, $R \subseteq X$ are clocks reset, op is one of the following stack operations:

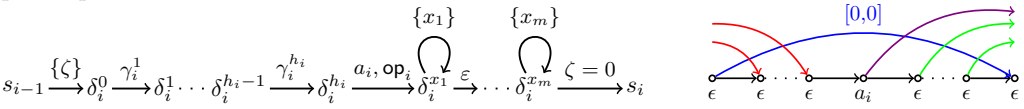
1. nop does not change the contents of the stack,
2. $\downarrow_c$ where $c \in \Gamma$ is a push operation that adds c on top of the stack, with age 0.
3. $\uparrow_c^I$ where $c \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$ is an interval, is a pop operation that removes the top most symbol of the stack provided it is a c with age in the interval I.

Timed automata (TA) can be seen as TPDA using nop operations only. This definition of TPDA is equivalent to the one in [1], but allows checking conjunctive constraints and stack operations together. In [6], it is shown that TPDA of [1] are expressively equivalent to timed automata with an untimed stack. Nevertheless, our technique is oblivious to whether the stack is timed or not, hence we focus on the syntactically more succinct model TPDA with timed stack and get good complexity bounds.

We define the semantics in terms of simple TCWs. An STCW $\mathcal{V} = (P, \rightarrow, \lambda, \rhd, \theta)$ is *generated* or *accepted* by a TPDA $\mathcal{S}$ if there is an accepting abstract run $\rho = (s_0, \gamma_1, a_1, \mathsf{op}_1, R_1, s_1)$ $(s_1, \gamma_2, a_2, \mathsf{op}_2, R_2, s_2) \cdots$ $(s_{n-1}, \gamma_n, a_n, \mathsf{op}_n, R_n, s_n)$ of $\mathcal{S}$ such that, $s_n \in F$ and

- the sequence of push-pop operations is well-nested: in each prefix $\mathsf{op}_1 \cdots \mathsf{op}_k$, number of pops is at most number of pushes, and in the full sequence $\mathsf{op}_1 \cdots \mathsf{op}_n$, they are equal.

- We have $P = P_0 \uplus P_1 \uplus \cdots \uplus P_n$ with $P_i \times P_j \subseteq \rightarrow^+$ for $0 \le i < j \le n$. Each transition $\delta_i = (s_{i-1}, \gamma_i, a_i, \mathsf{op}_i, R_i, s_i)$ gives rise to a sequence of consecutive points P_i in the STCW. The transition δ_i is simulated by a sequence of "micro-transitions" as depicted below (left) and it represents an STCW shown below (right). Incoming red edges check guards from γ_i (wrt different clocks) while outgoing green edges depict resets from R_i that will be checked later. Further, the outgoing edge on the central node labeled a_i represents a push operation on stack.



  where $\gamma_i = \gamma_i^1 \wedge \cdots \wedge \gamma_i^{h_i}$ and $R_i = \{x_1, \ldots, x_m\}$. The first and last micro-transitions, corresponding to the reset of a new clock ζ and checking of constraint $\zeta = 0$ ensure that all micro-transitions in the sequence occur simultaneously. We have a point in P_i for each micro-transition (excluding the ε-micro-transitions between $\delta_i^{x_j}$). Hence, P_i consists of a sequence $\ell_i \to \ell_i^1 \to \cdots \to \ell_i^{h_i} \to p_i \to r_i^1 \to \cdots \to r_i^{g_i} \to r_i$ where g_i is the number of timing constraints which are checked later, corresponding to clocks reset during transition i. Thus, the reset-loop on a clock is fired $k \ge 0$ times if k constraints are checked on this clock until its next reset. This ensures that the STCW remains *simple*. Similarly, h_i is the number of timing constraints conjuncted in γ_i. We have $\lambda(p_i) = a_i$ and all other points are labelled ε. The set P_0 encodes the initial resets of clocks that will be checked before their first reset. So we let $R_0 = X$ and P_0 is $\ell_0 \to r_0^1 \to \ldots \to r_0^{g_0} \to r_0$.
- The relation for timing constraints can be partitioned as $\rhd = \rhd^s \uplus \biguplus_{x \in X \cup \{\zeta\}} \rhd^x$ where
  - $\rhd^\zeta = \{(\ell_i, r_i) \mid 0 \le i \le n\}$ and we set $\theta(\ell_i, r_i) = [0, 0]$ for all $0 \le i \le n$.
  - We have $p_i \rhd^s p_j$ if $\mathsf{op}_i = \downarrow_b$ is a push and $\mathsf{op}_j = \uparrow_b^I$ is the matching pop (same number of pushes and pops in $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1}$), and we set $\theta(p_i, p_j) = I$.
  - for each $0 \le i < j \le n$ such that the t-th conjunct of γ_j is $x \in I$ and $x \in R_i$ and $x \notin R_k$ for $i < k < j$, we have $r_i^s \rhd^x \ell_j^t$ for some $1 \le s \le g_i$ and $\theta(r_i^s, \ell_j^t) = I$. Therefore, every point ℓ_i^t with $1 \le t \le h_i$ is the target of a timing constraint. Moreover, every reset point r_i^s for $1 \le s \le g_i$ should be the source of a timing constraint: $r_i^s \in \mathsf{dom}(\rhd^x)$ for some $x \in R_i$. Also, for each i, the reset points $r_i^1, \ldots, r_i^{g_i}$ are grouped by clocks (as suggested by the sequence of micro-transitions simulating δ_i): if $1 \le s < u < t \le g_i$ and $r_i^s, r_i^t \in \mathsf{dom}(\rhd^x)$ for some $x \in R_i$ then $r_i^u \in \mathsf{dom}(\rhd^x)$. Finally, for each clock, we require that the timing constraints are well-nested: for all $u \rhd^x v$ and $u' \rhd^x v'$, with $u, u' \in P_i$, if $u < u'$ then $u' < v' < v$.

We denote by $\mathsf{STCW}(\mathcal{S})$ the set of simple TCWs generated by $\mathcal{S}$ and define the language of $\mathcal{S}$ as the set of *realizable* STCWs, i.e., $\mathcal{L}(\mathcal{S}) = \mathrm{Real}(\mathsf{STCW}(\mathcal{S}))$. Indeed, this is equivalent to defining the language as the set of timed words accepted by $\mathcal{S}$, according to a usual operational semantics [1]. The STCW semantics of timed automata (TA) can be obtained from the above discussion by just ignoring the stack components (using nop operations only).

We now identify some important properties satisfied by STCWs generated from a TPDA. Let $\mathcal{V} = (P, \rightarrow, \lambda, \rhd, \theta)$ be a STCW. We say that $\mathcal{V}$ is *well timed* w.r.t. a set of clocks Y and a stack s if the $\rhd$ relation can be partitioned as $\rhd = \rhd^s \uplus \biguplus_{x \in Y} \rhd^x$ where

(T_1) the relation $\rhd^s$ corresponds to the matching push-pop events, hence it is well-nested: for all $i \rhd^s j$ and $i' \rhd^s j'$, if $i < i' < j$ then $i' < j' < j$.

(T_2) For each $x \in Y$, the relation $\rhd^x$ corresponds to the timing constraints for clock x and is well-nested: for all $i \rhd^x j$ and $i' \rhd^x j'$, if $i < i'$ are in the same *x-reset block* (i.e.,

a maximal consecutive sequence $i_1 \lessdot \cdots \lessdot i_n$ of positions in the domain of $\rhd^x$), and $i < i' < j$, then $i' < j' < j$. Each guard should be matched with the closest reset block on its left: for all $i \rhd^x j$ and $i' \rhd^x j'$, if $i < i'$ are not in the same x-reset block then $j < i'$.

It is then easy to check that STCWs defined by a TPDA with set of clocks X are well-timed for the set of clocks $Y = X \cup \{\zeta\}$, i.e., satisfy the properties above. We obtain the same for TA by just ignoring the stack edges, i.e., (T_1) above.

3 Bounding the width of graph behaviors of timed systems

In this section, we check if the graphs (STCWs) introduced in the previous section have a bounded tree-width. As a first step towards that, we introduce *special tree terms* (STTs) from Courcelle [8] and their semantics as labeled graphs. It is known [8] that special tree terms using at most K colors (K-STTs) define graphs of "special" tree-width at most $K-1$. Formally, a (Σ,Γ)-labeled graph is a tuple $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ where $\lambda\colon V \to \Sigma$ is the vertex labeling and $E_\gamma \subseteq V^2$ is the set of edges for each label $\gamma \in \Gamma$. Special tree terms form an algebra to define labeled graphs. The syntax of K-STTs over (Σ,Γ) is given by $\tau ::= (i,a) \mid \mathsf{Add}^\gamma_{i,j}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau \oplus \tau$, where $a \in \Sigma$, $\gamma \in \Gamma$ and $i, j \in [K] = \{1,\dots,K\}$ are colors. The semantics of each K-STT is a colored graph $[\![\tau]\!] = (G_\tau, \chi_\tau)$ where G_τ is a (Σ,Γ)-labeled graph and $\chi_\tau\colon [K] \to V$ is a partial injective function assigning a vertex of G_τ to atmost one color.

- $[\![(i,a)]\!]$ consists of a single a-labeled vertex with color i.
- $\mathsf{Add}^\gamma_{i,j}$ adds a γ-labeled edge to the vertices colored i and j (if such vertices exist).
- Forget_i removes color i from the domain of the color map.
- $\mathsf{Rename}_{i,j}$ exchanges the colors i and j.
- $\oplus$ is the disjoint union of two graphs if they use different colors and is undefined otherwise.

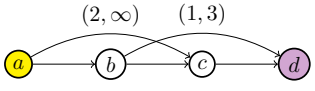
The *special tree-width* of a graph G is defined as the least K such that $G = G_\tau$ for some $(K+1)$-STT τ. See [8] for more details and its relation to tree-width. For TCWs, we have successor edges and $\rhd$-edges carrying timing constraints, so we take $\Gamma = \{\to\} \cup \{(x,y) \mid x \in \mathbb{N}, y \in \overline{\mathbb{N}}\}$ with $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. In this paper, we will actually make use of STTs with the following restricted syntax, which are sufficient and make our proofs simpler:

$$\mathsf{atomicSTT} ::= (1,a) \mid \mathsf{Add}^{x,y}_{1,2}((1,a) \oplus (2,b))$$
$$\tau ::= \mathsf{atomicSTT} \mid \mathsf{Add}^{\to}_{i,j}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau \oplus \tau$$

with $a, b \in \Sigma_\varepsilon$, $0 \le x < M$, $0 \le y < M$ or $y = +\infty$ for some $M \in \mathbb{N}$ and $i, j \in [2K] = \{1,\dots,2K\}$. The terms defined by this grammar are called (K,M)-STTs. Here, timing constraints are added directly between leaves in atomic STTs which are then combined using disjoint unions and adding successor edges. For instance, consider the 4-STT given below

$$\tau = \mathsf{Forget}_3\,\mathsf{Add}^{\to}_{1,3}\,\mathsf{Forget}_2\,\mathsf{Add}^{\to}_{2,4}\,\mathsf{Add}^{\to}_{3,2}(\mathsf{Add}^{2,\infty}_{1,2}((1,a) \oplus (2,c)) \oplus \mathsf{Add}^{1,3}_{3,4}((3,b) \oplus (4,d)))$$

where $\mathsf{Add}^\gamma_{i,j}((i,\alpha) \oplus (j,\beta))$, $i, j \in \mathbb{N}, \alpha, \beta \in \Sigma_\varepsilon$ is the same as $\mathsf{Rename}_{1,i}\,\mathsf{Rename}_{2,j}\,\mathsf{Add}^\gamma_{1,2}((1,\alpha) \oplus (2,\beta))$. Its semantics $[\![\tau]\!]$ is the adjoining STCW where only endpoints labelled a and d are colored, as the other two colors were "forgotten" by τ.Abusing notation, we will also use $[\![\tau]\!]$ for the graph G_τ ignoring the coloring χ_t.



Split-TCWs and split-game. We find it convenient to prove that a STCW has bounded special tree-width by playing a split-game, whose game positions are STCWs in which some successor edges have been cut, i.e., are missing. Formally, a *split*-TCW is a structure $\mathcal{V} = (P, \to, \dashrightarrow, \lambda, \rhd, \theta)$ where $\to$ and $\dashrightarrow$ are the present and absent successor edges (also called *holes*), respectively, such that $\to \cap \dashrightarrow = \emptyset$ and $(P, \to \cup \dashrightarrow, \lambda, \rhd, \theta)$ is a TCW. Notice that, for a split-TCW, $\lessdot = \to \cup \dashrightarrow$ and $< = \lessdot^+$. A *block* or *factor* of a split-TCW is a maximal set of points of P connected by $\to$. We denote by $\mathsf{EP}(\mathcal{V}) \subseteq P$ the set of left and right endpoints of blocks of $\mathcal{V}$. A left endpoint e is one for which there is no f with $f \to e$. Right endpoints are defined similarly. Points in $P \setminus \mathsf{EP}(\mathcal{V})$ are called internal. The number of blocks is the *width* of $\mathcal{V}$: $\mathsf{width}(\mathcal{V}) = 1 + |\dashrightarrow|$. TCWs may be identified with split-TCWs of width 1, i.e., with $\dashrightarrow = \emptyset$. A split-TCW is *atomic* if it consists of a single point ($|P| = 1$) or a single timing constraint with a hole ($P = \{p_1, p_2\}$, $p_1 \dashrightarrow p_2$, $p_1 \rhd p_2$). The directed weighted graph for a split-TCW is defined on the associated TCW under $\to \cup \dashrightarrow$ and hence has back edges with $\mathsf{wt} = 0$ across a hole as well.

The split-game is a two player turn based game $\mathcal{G} = (V_\exists \uplus V_\forall, E)$ where Eve's set of game positions $V_\exists$ consists of all connected (wrt. $\to \cup \rhd$) split-TCWs and Adam's set of game positions $V_\forall$ consists of non-connected split-TCWs. The edges E of $\mathcal{G}$ reflect the moves of the players. Eve's moves consist of splitting a factor in two, i.e., removing one successor edge in the graph. Adam's moves amount to choosing a connected component of the split-TCW. Atomic split-TCWs are terminal positions in the game: neither Eve nor Adam can move from an atomic split-TCW. A play on a split-TCW $\mathcal{V}$ is a path in $\mathcal{G}$ starting from $\mathcal{V}$ and leading to an atomic split-TCW. The cost of the play is the maximum width of any split-TCW encountered in the path. Eve's objective is to minimize the cost, while Adam's objective is to maximize it. Notice that Eve has a strategy to decompose a TCW $\mathcal{V}$ into *atomic* split-TCWs if and only if $\mathcal{V}$ is *simple*, i.e, at most one timing constraint is attached to each point. The *cost* of a strategy σ for Eve from a split-TCW $\mathcal{V}$ is the maximal cost of the plays starting from $\mathcal{V}$ and following strategy σ.

The *split-width* of a simple (split-)TCW $\mathcal{V}$ is the minimal cost of Eve's (positional) strategies starting from $\mathcal{V}$. Let STCW^K (resp. $\mathsf{STCW}^{K,M}$) denote the set of simple TCWs with split-width bounded by K (resp. and using constants at most M) over the fixed alphabet Σ. The crucial link between special tree-width and split-width is given below.

▶ **Lemma 3.** STCWs *of split-width at most K have special tree-width at most $2K-1$.*

Intuitively, we only need to keep colors for end-points of blocks. Hence, each block of an STCW $\mathcal{V}$ needs at most two colors and if the width of $\mathcal{V}$ is at most K then we need at most $2K$ colors. From this it can be shown that a strategy of Eve of cost at most K can be encoded by a $2K$-STT, which gives a special tree-width of at most $2K-1$.

Split-width for timed systems. Viewing special tree terms as trees, our goal in the next section is to construct tree automata to recognize sets of (K, M)-STTs, and thus capture (K split-width) bounded behaviors of a given system. To show that these capture *all* behaviors of the given system, we show that we can find K such that all (graph) behaviors of the given system have K-bounded split-width. We do this now for TPDA and timed automata.

▶ **Theorem 4.** *Given a timed system $\mathcal{S}$ using a set of clocks X, all words in its* STCW *language have split-width bounded by K, i.e.,* $\mathsf{STCW}(\mathcal{S}) \subseteq \mathsf{STCW}^K$*, where*

1. *$K = |X| + 4$ if $\mathcal{S}$ is a timed automaton,*
2. *$K = 4|X| + 6$ if $\mathcal{S}$ is a timed pushdown automaton,*

We prove a slightly more general result, by showing that all well-timed split-STCWs for the set of clocks $Y = X \cup \{\zeta\}$ have bounded split-width (lifting the definition of well-timed to split-STCWs). As noted earlier, the STCWs defined by a TPDA with set of clocks X are well-timed for the set of clocks $Y = X \cup \{\zeta\}$ and hence we obtain a bound on the split-width as required above. The following lemma completes the proof of Theorem 4 (2).

▶ **Lemma 5.** *The split-width of a well-timed* STCW *is bounded by* $4|Y| + 2$.

Proof (sketch). We prove this by playing the "split-width game" between *Adam* and *Eve* in which *Eve* has a strategy to disconnect the word without introducing more than $4|Y| + 2$ blocks. *Eve*'s strategy processes the word from right to left. We have three cases as follows.

Case (1) is when the last/right-most event, say j, is an internal point, i.e., it is not the target of a $\triangleright$ relation. In this case, Eve will just split the process-edge before the last point with a single cut.

Case (2) is when the last event is the target of $\triangleright^x$ for some clock $x \in Y$. In this case, she will detach the last timing constraint $i \triangleright^x j$ where j is the last point of the split-TCW. By (T_2) we deduce that i is the *first* point of the *last reset block* for clock x. *Eve* splits three process-edges to detach the matching pair $i \triangleright^x j$: these three edges are those connected to i and j. Since the matching pair $i \triangleright^x j$ is atomic, to prolong the game *Adam* should choose the remaining split-TCW $\mathcal{V}'$. Note that we now have a hole instead of position i. We call this a reset-hole for clock x. During the inductive process, we have at most one such reset hole for each $x \in Y$, since the hole only widens in the reset block for each clock.

Note that the last event cannot be a push or the source of a timing constraint. So, the remaining Case (3) is a stack edge $i \triangleright^s j$ where the pop event j is the last event of the split-TCW, details of which are in [3]. ◀

Now, if the STCW is from a timed automaton then, $\triangleright^s$ is empty and Eve's strategy only has the first two cases above. Doing, this we obtain a bound of $|Y| + 3$ on split-width, which proves Theorem 4 (1).

4 The tree automata technique illustrated via TPDA and TA

We now describe our proof technique of using tree automata to analyze timed systems. At a high level, given a timed system $\mathcal{S}$ using constants less than M (say a timed automaton or a TPDA), we want to construct a tree automaton that accepts all (K, M)-STTs whose semantics are STCWs of split-width at most K which are realizable and accepted by $\mathcal{S}$. We break this into three parts. First, recall that STCWs of bounded split-width are graphs of bounded STTs (Lemma 3). However, not all graphs defined by bounded STTs are STCWs. We construct a tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ which accepts only *valid* (K, M)-STTs, i.e., those representing STCWs of split-width at most K.

▶ Proposition 6. We can build a tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ of size $\mathcal{O}(M) \cdot 2^{\mathcal{O}(K^2)}$ which accepts only (K, M)-STTs and such that $\mathsf{STCW}^{K,M} = \{[\![\tau]\!] \mid \tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}})\}$.

Our next step is to define a tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ which accepts all valid STTs whose semantics are *realizable* STCWs.

▶ Proposition 7. We can build a tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ of size $M^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2 \lg K)}$ such that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{real}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$.

Note that $\mathcal{A}^{K,M}_{\mathsf{real}}$ may not accept *all* (K, M)-STTs which denote realizable STCWs, but it will accept all such *valid* STTs. Once we have this, our third and final step is to build a tree automaton which accepts the valid STTs denoting STCWs accepted by the timed system.

▶ Proposition 8. Let $\mathcal{S}$ be a TPDA of size $|\mathcal{S}|$ (constants encoded in unary) with set of clocks X and using constants less than M. Then, we can build a tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ of size $|\mathcal{S}|^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2(|X|+1))}$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{S}}^{K,M}) = \{\tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M}) \mid [\![\tau]\!] \in \mathsf{STCW}(\mathcal{S})\}$.

In Section 5, we detail the most complex tree automaton construction, $\mathcal{A}_{\mathsf{real}}^{K,M}$ for realizability, thus proving Proposition 7. The construction of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ (Proposition 6) is somewhat similar (and easier) and we refer the reader to [3] for its details as well as the proof of (Proposition 8). We remark that for $\mathcal{A}_{\mathsf{valid}}^{K,M}, \mathcal{A}_{\mathcal{S}}^{K,M}$ we can also define an MSO formula and use Courcelle's theorem [9], but the direct tree automata construction gives us better control on complexity bounds and helps for $\mathcal{A}_{\mathsf{real}}^{K,M}$.

Thus, the tree automaton $\mathcal{A}$ checking ***ValCoRe*** (i.e., validity, correctness and realizability) is $\mathcal{A} = \mathcal{A}_{\mathsf{real}}^{K,M} \cap \mathcal{A}_{\mathcal{S}}^{K,M}$. We have $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exist some *realizable* STCWs in $\mathsf{STCW}(\mathcal{S}) \cap \mathsf{STCW}^{K,M}$. Since checking emptiness of a finite tree automaton is decidable in PTIME, we obtain that emptiness is decidable for the corresponding timed system restricted to STCWs of split-width at most K.

▶ **Theorem 9.** *Checking whether the timed system $\mathcal{S}$ accepts a* realizable STCW *of split-width at most K is decidable.*

By Theorem 4, all STCWs in the semantics of a TPDA $\mathcal{S}$ have split-width bounded by some fixed K and Theorem 9 gives a complete decision procedure for checking emptiness of TPDA. From these bounds on split-width and the size of the tree automata for validity, realizability and the system given in the above propositions, we obtain ExpTime decision procedures for checking emptiness of TPDA.

In the above technique, the only system-specific component is the automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ for the timed system $\mathcal{S}$. However, Proposition 8 can easily be adapted for timed automata and for several other timed systems, which are discussed in Section 6. Hence, this technique is generic and can be used for several other timed systems.

Moreover, for timed automata, it can be seen, for instance, from the analysis of Cases (1) and (2) of proof of Lemma 5 that one of the connected components (the pair $i \rhd^x j$) is always atomic. Therefore the split-tree is "word-like", i.e., for each binary node, one subtree is small, in our case atomic. Therefore, we can encode the subtree in the label of the binary node itself and use word automata instead of tree automata to check for emptiness (in NLogSpace instead of PTime), yielding the complexity stated below.

▶ **Corollary 10.** *Emptiness of TPDA and TA are decidable in* ExpTime *and* PSpace *respectively.*

5 Tree automata for realizable valid (K, M)-STTs

Our goal in this section is to define a finite bottom-up tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ that runs on (K, M)-STTs and accepts only *valid* (K, M)-STTs whose semantics are realizable STCWs. Let us first give a high-level picture. A state of the tree automaton will be a split-TCW with at most K blocks and $2K$ points. At any stage of the run, while processing a subtree τ of the (K, M)-STT, the state, i.e., split-TCW q reached will be a finite abstraction of the split-TCW $[\![\tau]\!]$ generated by τ, such that q is valid and realizable iff the TCW $[\![\tau]\!]$ is. At a leaf, the state of an atomic-STT is just a single matching edge with a hole. At each subsequent step going up, the tree automaton simulates the operations of τ: at a $\oplus$ move, it combines two split-TCW q_1 and q_2 to form a new valid split-TCW q by guessing an ordering between the blocks such that no new negative cycle is introduced (i.e., q continues to be realizable), and at an $\mathsf{Add}_{i,j}^{\rightarrow}$ node, it adds a process edge to fill up the corresponding hole in

the split-TCW. At a Forget_i node, it removes an internal point, but to maintain realizability, the constraints on internal positions must be propagated to the end-points of the block and this process is continued. Finally, at the root, we obtain a TCW which is a finite abstraction of the semantics $[\![\tau]\!]$ of a valid (K, M)-STT τ such that $[\![\tau]\!]$ is a realizable TCW. Then, we show that the tree automaton accepts all such (K, M)-STTs, which concludes the proof of Proposition 7. There are two key difficulties that we have glossed over in this sketch:

- first, the propagation of constraints can increase the bounds arbitrarily, along an arbitrarily long (even if finite) run. Fixing this is the hardest part and we carefully define abstractions that bound the constraints by a constant $M' = \mathcal{O}(M)$, while preserving realizability.
- This leads to another subtle issue: while checking that realizability is preserved under our operations (of combining split-TCW and adding process edges), it is no longer sufficient to just check whether this combination is "safe". It may be that currently no negative cycle is formed, but at a later stage, some other operation ($\oplus$) gives rise to a negative cycle, which we do not observe since we capped the value of timing constraints. So, we need to show that all operations are safe no matter what happens in the future. For this we start by defining the notion of preserving realizability "under all contexts" as well as the formal notion of a "shuffle" used at $\oplus$ nodes.

Shuffle and Realizability under contexts. Let $\mathcal{V}_1 = (P_1, \rightarrow_1, \dashrightarrow_1, \lambda_1, \rhd_1, \theta_1)$ and $\mathcal{V}_2 = (P_2, \rightarrow_2, \dashrightarrow_2, \lambda_2, \rhd_2, \theta_2)$ be two split-TCWs such that their respective set of positions P_1 and P_2 are disjoint. Further, let $\leq$ be a total order on $P = P_1 \cup P_2$ such that $\dashrightarrow_1 \cup \dashrightarrow_2 \subseteq <$ and $\rightarrow_1 \cup \rightarrow_2 \subseteq \lessdot$. Such orders are called *admissible*. Then, we define the split-TCW $\mathcal{V} = (P, \rightarrow, \dashrightarrow, \lambda, \rhd, \theta)$ by $P = P_1 \uplus P_2$, $\lambda = \lambda_1 \cup \lambda_2$, $\rightarrow = \rightarrow_1 \cup \rightarrow_2$, $\dashrightarrow = \lessdot \setminus \rightarrow$, $\rhd = \rhd_1 \cup \rhd_2$, and $\theta = \theta_1 \cup \theta_2$. Indeed, this corresponds to *shuffling* the blocks $\mathcal{V}_1$ and $\mathcal{V}_2$ with respect to the admissible order $\leq$ and is called a shuffle, denoted by $\mathcal{V} = \mathcal{V}_1 \sqcup\!\sqcup_{\leq} \mathcal{V}_2$.

Let M be a positive integer. An *M-context* C is a split-TCW such that the maximal constant in the intervals is strictly smaller than the fixed constant M. Given a context C and a split-TCW $\mathcal{V}$, we define an operation $C \circ \mathcal{V}$ if $\mathsf{width}(C) = \mathsf{width}(\mathcal{V}) + 1$. $C \circ \mathcal{V}$ is the split-TCW obtained by *shuffling* the blocks of C and $\mathcal{V}$ in strict alternation. Two split-TCWs U and V are *equivalent*, denoted $U \sim_M V$, iff they have the same number of blocks and preserve realizability under all M-contexts. That is, there exists $k \in \mathbb{N}$ such that $\mathsf{width}(U) = \mathsf{width}(V) = k$ and for all M-contexts $C \in \mathsf{STCW}$ with $\mathsf{width}(C) = k + 1$, $C \circ U$ is realizable iff $C \circ V$ is realizable. It is easy to see that $\sim_M$ is an equivalence relation. A function $f : \mathsf{STCW} \to \mathsf{STCW}$ is said to be *sound* if it preserves realizability under all M-contexts, i.e., for all $W \in \mathsf{STCW}$ we have $W \sim_M f(W)$. The idea is to define a sound abstraction of finite index, so that a finite tree automaton can work only on the abstractions.

▶ **Lemma 11.** (Congruence lemma) *Let U_1, U_2, U_1' and U_2' be split-*TCWs *such that $U_1 \sim_M U_1'$ and $U_2 \sim_M U_2'$. Then, $U_1 \sqcup\!\sqcup_{\leq} U_2 \sim_M U_1' \sqcup\!\sqcup_{\leq} U_2'$ for all admissible orders $\leq$ on the blocks.*

A (possibly infinite) tree automaton for realizability. We now build the tree automaton for realizability in two steps. First, we detail a construction which is correct and sound (i.e., preserves realizability under all contexts), but in which constants can grow unboundedly. Subsequently, we show conditions under which it has finitely many states and additional abstractions to ensure finiteness.

▶ Proposition 12. We can build a tree automaton $\mathcal{A}_{\mathsf{inf}}^{K,M}$ such that $\mathcal{L}(\mathcal{A}_{\mathsf{inf}}^{K,M}) = \{\tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M}) \mid [\![\tau]\!] \text{ is realizable}\}$.

Proof. The construction builds on the construction of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, which is detailed in [3]. The states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ are pairs (q, wt) where $q = (P, <, \to)$ is a state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, i.e., $P \subseteq [2K]$, $<$ is a total order on P, $\to \subseteq \lessdot$ is the successor relation between points in the same block, q has at most K blocks; and $\mathsf{wt}\colon P^2 \to \overline{\mathbb{Z}} = \mathbb{Z} \cup \{+\infty\}$ gives the timing constraints. The first component is finite but weights can grow unboundedly. We assume $\mathsf{wt}(k,k) = 0$ for all $k \in P$ and if $i < j$ then $\mathsf{wt}(j,i) \le 0 \le \mathsf{wt}(i,j)$. We identify (q, wt) with a split-TCW (ignoring $\triangleright, \Sigma$, as these are irrelevant for realizability).

We first give the invariant that will be maintained by the automaton. Let τ be a (K,M)-STT with $[\![\tau]\!] = (V, \to, \lambda, \triangleright, \theta, \chi)$. If a (bottom-up) run of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ reads τ and reaches state (q, wt) with $q = (P, <, \to)$, it induces a total order on blocks of $[\![\tau]\!]$ and turns it into a split-TCW $([\![\tau]\!], \dashrightarrow)$. We say that the abstraction (q, wt) of τ computed by $\mathcal{A}^{K,M}_{\mathsf{inf}}$ is *sound* if it preserves realizability under contexts, i.e., $([\![\tau]\!], \dashrightarrow) \sim_M (q, \mathsf{wt})$. *The key invariant is that $\mathcal{A}^{K,M}_{\mathsf{inf}}$ always computes a sound abstraction of the given* STT. We now formalize the definition of the tree automaton.

- AtomicSTTs: When reading the atomic STT $\tau = (1, a)$ with $a \in \Sigma$, $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1\}, \emptyset, \emptyset)$ and $\mathsf{wt}(1,1) = 0$. Similarly, when reading an atomic STT $\tau = \mathsf{Add}^{c,d}_{1,2}((1,a) \oplus (2,b))$, $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1,2\}, 1 < 2, \emptyset)$, $\mathsf{wt}(1,1) = 0 = \mathsf{wt}(2,2)$, $\mathsf{wt}(1,2) = d$ and $\mathsf{wt}(2,1) = -c$. In both cases, it is easy to check that (q, wt) is a sound abstraction of τ.
- $\mathsf{Rename}_{i,j}$: We define transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Rename}_{i,j}} (q', \mathsf{wt}')$ where (q', wt') is obtained by exchanging colors i and j in (q, wt), which clearly preserves soundness.
- $\mathsf{Add}^{\to}_{i,j}$: We define $(q, \mathsf{wt}) \xrightarrow{\mathsf{Add}^{\to}_{i,j}} (q', \mathsf{wt})$, when q' is obtained from $q = (P, <, \to)$ by adding a successor edge $(i,j) \in \lessdot \setminus \to$. Then, if $\tau' = \mathsf{Add}^{\to}_{i,j}\, \tau$ and (q, wt) is a sound abstraction of τ, it follows that (q', wt') is a sound abstraction of τ' (adding edges only reduces number of contexts to be considered to show equivalence of realizability under contexts.)
- $\oplus$: We define transitions $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} (q, \mathsf{wt})$ when $q = (P, <, \to)$ is a shuffle of q_1 and q_2 and for all $i, j \in P = P_1 \uplus P_2$, $\mathsf{wt}(i,j)$ is $\mathsf{wt}_1(i,j)$ if $i, j \in P_1$ and $\mathsf{wt}_2(i,j)$ if $i, j \in P_2$. If they do not come from the same state, i.e., if $(i,j) \in (P_1 \times P_2) \cup (P_2 \times P_1)$, then $\mathsf{wt}(i,j)$ is ∞ if $i < j$ and 0 otherwise, i.e., $i \ge j$. Now, if $\tau = \tau_1 \oplus \tau_2$ and $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2)$ are sound abstractions of τ_1, τ_2 then (q, wt) is a sound abstraction of τ. The total ordering $<$ of q indicates how blocks of q_1 and q_2 are shuffled. Hence $(q, \mathsf{wt}) = (q_1, \mathsf{wt}_1) \sqcup\!\sqcup_{\le} (q_2, \mathsf{wt}_2)$. Now, the induced ordering on the blocks of $[\![\tau]\!]$ corresponds to the same shuffle of blocks, i.e., $([\![\tau]\!], \dashrightarrow) = ([\![\tau_1]\!], \dashrightarrow_1) \sqcup\!\sqcup_{\le} ([\![\tau_2]\!], \dashrightarrow_2)$. Now, applying the congruence Lemma 11, we obtain that (q, wt) is a sound abstraction of τ.
- Forget_i: We define transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q', \mathsf{wt}')$ when the following hold
  - i is not an endpoint, q' is obtained from $q = (P, <, \to)$ by removing internal point i,
  - i is not part of a negative cycle of length 2: for all $j \ne i$ we have $\mathsf{wt}(j,i) + \mathsf{wt}(i,j) \ge 0$,
  - for all $j, k \in P' = P \setminus \{i\}$, we define $\mathsf{wt}'(j,k) = \min(\mathsf{wt}(j,k), \mathsf{wt}(j,i) + \mathsf{wt}(i,k))$, i.e., wt' is obtained by eliminating i.

  If the second condition above is not satisfied then the tree automaton $\mathcal{A}^{K,M}_{\mathsf{inf}}$ has no transitions from (q, wt) reading Forget_i. With this we can prove that if $\tau' = \mathsf{Forget}_i\, \tau$ and (q, wt) is a sound abstraction of τ, then (q', wt') is a sound abstraction of τ'.
- Accepting condition: Finally, we define a state (q, wt) to be accepting if q consists of a single block with no internal points, left endpoint i and right endpoint j (possibly $i = j$), and the pair (q, wt) is realizable, i.e., $\mathsf{wt}(i,j) + \mathsf{wt}(j,i) \ge 0$.

We can now check that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{inf}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$. ◀

Observe that the constants in wt' increase only at *forget* transitions, where a back edge $j > k$ with $j > i > k$ grows in absolute value with the update $\mathsf{wt}'(j,k) = \min(\mathsf{wt}(j,k), \mathsf{wt}(j,i) + \mathsf{wt}(i,k))$. A forward edge $j < k$ may get a big value only if $\mathsf{wt}(j,k) = \infty$, else it can only decrease due to the min operation. A first question is if there are classes where they will not grow *unboundedly*. A simple solution is to consider time-bounded classes where all behaviors must occur within some global time bound T: if some back edge grows $> T$ in absolute value after a forget move we reject the STT; while if the same happens with a forward edge, then replace it with ∞. Thus, we obtain,

▶ **Corollary 13.** *If the system is time-bounded by some constant T, then there exists a finite tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ of size at most $T^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2 \lg K)}$ for checking realizability.*

However, when we do not assume a global time bound the constants in the states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ may grow unboundedly. We next show how to modify the above construction so that the constants are always bounded. This generalizes the above corollary with a better complexity.

Bounding the constants. The finite tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ will work on a finite subset of the states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$. More precisely, a state (q, wt) of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \to)$ is a state of $\mathcal{A}^{K,M}_{\mathsf{real}}$ if for all $i, j \in P$ we have $\mathsf{wt}(i,j) = +\infty$ or $|\mathsf{wt}(i,j)| \leq 8KM$.

Now, to bound back edges we define a transformation β which reduces the weight of a back edge when it goes above a certain constant, while *preserving realizability under all contexts*. In fact, we define it on back edges across a block. Let (q, wt) be a state of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \to)$. A pair of points $(j,i) \in P^2$ is said to be a *block back edge* (denoted BBE) if $i < j$ are the end points of a block in q, i.e., $i \to^+ j$ and this $\to$-path cannot be extended (on the left or on the right). A *big block back edge (*BBBE*)* is block back edge e such that $M + \mathsf{wt}(e) \leq 0$. For any two positions $i < j$, we define $\mathsf{BBE}(i,j)$ to be the set of block back edges between i and j. That is, $\mathsf{BBE}(i,j) = \{(\ell,k) \mid (\ell,k) \text{ is a } \mathsf{BBE} \text{ and } i \leq k < \ell \leq j\}$. We also define $\mathcal{B}(i,j)$ to be the set of big block back edges between i and j: $\mathcal{B}(i,j) = \{e \in \mathsf{BBE}(i,j) \mid e \text{ is big}\}$. We now define $\beta(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where, for any $i < j$,

$$\mathsf{wt}'(i,j) = \mathsf{wt}(i,j) + \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e)) \qquad \mathsf{wt}'(j,i) = \mathsf{wt}(j,i) - \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e))$$

The idea is to change the weight of *big* BBE to $-M$ by adding an offset to all the other edges (backward and forward) crossing this block. Note that this does not increase the absolute value of *any* constant. Further, after the backward abstraction, the absolute value of weights of block back edges is bounded by M, i.e., for all BBE $i \curvearrowleft j$, we have $\mathsf{wt}'(j,i) \geq -M$. Indeed, either the edge was big and we get $\mathsf{wt}'(j,i) = -M$ or it was not big and $\mathsf{wt}'(j,i) = \mathsf{wt}(j,i) > -M$. Notice also that a BBE is big in (q, wt) iff it is big in $\beta(q, \mathsf{wt})$. The crucial property is that we leave the weights of all cycles unchanged (under all contexts).

▶ **Lemma 14.** *For all states $W = (q, \mathsf{wt})$ of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \to)$ such that all points are endpoints $P = \mathsf{EP}(W)$, we have $W \sim_M \beta(W)$.*

While block back edges are now bounded (and back edges across holes can also be bounded by $-M$), this does not suffice to bound all back edges. To obtain such a bound on all back edges, we need to relate large back edges to edges contained within them.

▶ **Definition 15.** A split-TCW W is said to satisfy the *back edge property* (BEP) if for all $i \leq j \leq k \leq \ell$ with either $j \dashrightarrow k$ or $j = k$, we have $\mathsf{wt}(\ell, i) > \mathsf{wt}(\ell, k) - M + \mathsf{wt}(j, i)$.

With this, we have our second and crucial invariant, that we maintain inductively in the tree automaton, (I2): $\mathcal{A}^{K,M}_{\mathsf{real}}$ always satisfies BEP. Preserving this invariant requires a slight transformation of the shuffle operation (at a $\oplus$ node). Namely, after every shuffle we must *strengthen* the constraints of the back edges. Formally, we define a map σ, $\sigma(q,\mathsf{wt}) = (q,\mathsf{wt}')$ where for all $i<j$, $\mathsf{wt}'(i,j) = \mathsf{wt}(i,j)$ and $\mathsf{wt}'(j,i) = \min\{\mathsf{wt}(j',i') \mid i \le i' \le j' \le j\}$ and perform this after every $\oplus$ move of the tree automaton. It is easy to check that σ preserves realizability under contexts and this allows us to show that the invariant (I2) is preserved. Now, under the BEP assumption, we can show that all back edges are bounded,

▶ **Lemma 16.** *Let $W = (q,\mathsf{wt})$ be a state of $\mathcal{A}^{K,M}_{inf}$ with $q = (P,<,\to)$ such that $P = \mathsf{EP}(W)$. If $\beta(W)$ satisfies* BEP*, then the weight of all back edges in $\beta(W)$ are bounded by $2KM$.*

Finally, *forward abstraction* γ removes all forward edges (i.e., changes their weight to ∞) that are too large to be useful for creating negative cycles. Let $W = (q,\mathsf{wt})$ be a state of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P,<,\to)$. A forward edge $(i,j) \in P^2$ with $i<j$ is called *big* if $\mathsf{wt}(i,j) + \sum_{e\in \mathsf{BBE}(i,j)} \mathsf{wt}(e) \ge (3K-1)M$. Note, $\mathsf{wt}(e) \le 0$ as it is a (block) back edge. Then, we define $\gamma(q,\mathsf{wt}) = (q,\mathsf{wt}')$ where, for any $i<j$, $\mathsf{wt}'(j,i) = \mathsf{wt}(j,i)$ and $\mathsf{wt}'(i,j) = \infty$ if (i,j) is *big* and unchanged otherwise. While the definition of this abstraction is simple, showing that it is sound (i.e., preserves realizability under all contexts) is rather tricky. We have,

▶ **Lemma 17.** *If $W = (q,\mathsf{wt})$ is a state of $\mathcal{A}^{K,M}_{inf}$ which satisfies BEP, then we have $W \sim_M \gamma(W)$.*

Thus, $\mathcal{A}^{K,M}_{\mathsf{real}}$ is derived from $\mathcal{A}^{K,M}_{\mathsf{inf}}$ by applying the abstractions at $\oplus$ nodes and at Forget_i nodes. More precisely, $(q_1,\mathsf{wt}_1),(q_2,\mathsf{wt}_2) \xrightarrow{\oplus} \sigma(q,\mathsf{wt})$ is in $\mathcal{A}^{K,M}_{\mathsf{real}}$ if $(q_1,\mathsf{wt}_1),(q_2,\mathsf{wt}_2) \xrightarrow{\oplus} (q,\mathsf{wt})$ is in $\mathcal{A}^{K,M}_{\mathsf{inf}}$. Similarly, if $(q,\mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q',\mathsf{wt}')$ is a transition in $\mathcal{A}^{K,M}_{\mathsf{inf}}$ then $(q,\mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q'',\mathsf{wt}'')$ is in $\mathcal{A}^{K,M}_{\mathsf{real}}$ where $(q'',\mathsf{wt}'') = \gamma(\beta(q',\mathsf{wt}'))$ if q' has no internal points and $(q'',\mathsf{wt}'') = (q',\mathsf{wt}')$ otherwise. The reason for assuming that q' has no internal points before applying the abstractions is that it is a precondition for Lemmas 14 and 16. Note that reachable states of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ (and hence $\mathcal{A}^{K,M}_{\mathsf{real}}$) can have at most two internal points. Thus, along a run, if a state (q,wt) has no internal points, then the constants are bounded by $4KM$, otherwise, the constants are bounded by $8KM$. Thus the constants never exceed $8KM$ in states of $\mathcal{A}^{K,M}_{\mathsf{real}}$, which bounds our state space.

Since the transformations σ,β,γ preserve realizability under contexts (Lemma 14 and Lemma 17) we conclude that the key invariant holds, i.e., $\mathcal{A}^{K,M}_{\mathsf{real}}$ always computes a sound abstraction of the given STT. The acceptance condition of $\mathcal{A}^{K,M}_{\mathsf{real}}$ is the same as for $\mathcal{A}^{K,M}_{\mathsf{inf}}$, and the correctness follows as for $\mathcal{A}^{K,M}_{\mathsf{inf}}$. This completes the proof of Proposition 7.

6 Discussion and Future work

The main contribution of this paper is the technique for analyzing timed systems via tree automata. For simplicity, we only considered closed intervals in this paper, but our technique can be easily adapted to work for all kinds of intervals, i.e., open, half-open etc. Similarly, diagonal constraints of the form $x - y \in I$ can be handled easily by adding matching edges and changing $\mathcal{A}^{K,M}_{\mathcal{S}}$ appropriately.

As another application of our technique, we now consider the model of dense-timed multi-stack pushdown automata (dtMPDA), which have several stacks. The reachability problem for untimed multi-stack pushdown automata (MPDA) is already undecidable, but several restrictions have been studied on (untimed) MPDA, like bounded rounds [14], bounded phase, bounded scope and so on to regain decidability. We look at dtMPDA with the restriction of

"bounded rounds". To the best of our knowledge, this timed model has not been investigated until now. Under this restriction, any run of a dtMPDA can be broken into a finite number of rounds, such that in each round only a single stack is used. As before, the sequence of push-pop operations of any stack must be well-nested. Now, lifting the definition of well-timed STCWs to k-round well-timed STCWs, we can show that such STCWs have split-width at most $(4nk + 4)(|X| + 2)$, where n is the number of stacks. Thus all STCWs generated by runs of dtMPDA using at most k rounds have a bounded split-width. Now, by modifying $\mathcal{A}_{\mathcal{S}}^{K,M}$ appropriately (see [3] for details), we obtain

▶ **Theorem 18.** *Checking emptiness for k-round* dtMPDA *is decidable in* ExpTime.

We believe that our techniques can be extended to other restrictions for dtMPDA such as bounded scope and phase and to the more general model [13] of recursive hybrid automata. Another interesting direction is to use our technique to go beyond reachability and show results on model checking for timed systems. While model-checking against untimed specifications is easy to obtain with our approach, the challenge is to extend it to timed specifications.

References

1. P. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *LICS Proceedings*, pages 35–44, 2012.
2. C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG (invited talk). In *FSTTCS Proceedings*, pages 11–30, 2014.
3. S. Akshay, P. Gastin, and S. N. Krishna. Analyzing timed systems using tree automata. *CoRR*, abs/1604.08443, 2016. URL: `http://arxiv.org/abs/1604.08443`.
4. R. Alur and D. Dill. A theory of timed automata. *In TCS*, 126(2):183–235, 1994.
5. M. F. Atig. Model-checking of ordered multi-pushdown automata. *LMCS*, 8(3), 2012.
6. L. Clemente and S. Lasota. Timed pushdown automata revisited. In *LICS Proceedings*, pages 738–749, 2015.
7. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.
8. B. Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS Proceedings*, pages 13–29, 2010.
9. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. CUP, 2012.
10. A. Cyriac. *Verification of Communicating Recursive Programs via Split-width*. Thèse de doctorat, LSV, ENS Cachan, January 2014.
11. A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR Proceedings*, pages 547–561, 2012.
12. W. Czerwinski, P. Hofman, and S. Lasota. Reachability problem for weak multi-pushdown automata. In *CONCUR Proceedings*, pages 53–68. Springer, 2012.
13. S. N. Krishna, L. Manasa, and A. Trivedi. What's decidable about recursive hybrid automata? In *HSCC Proceedings*, pages 31–40, 2015.
14. S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN Proceedings*, pages 96–107, 2010.
15. S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT Proceedings*, pages 116–128. Springer, 2014.
16. S. La Torre, M. Napoli, and G. Parlato. A unifying approach for multistack pushdown automata. In *MFCS Proceeedings*, pages 377–389. Springer, 2014.
17. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL Proceedings*, pages 283–294, 2011.

On the Expressiveness of QCTL*

Amélie David[1], François Laroussinie[2], and Nicolas Markey[3]

1 **LSV – CNRS, ENS Cachan & University Paris-Saclay – France**
2 **IRIF – University Paris Diderot & CNRS – France**
3 **LSV – CNRS, ENS Cachan & University Paris-Saclay – France**

Abstract

QCTL extends the temporal logic CTL with quantification over atomic propositions. While the algorithmic questions for QCTL and its fragments with limited quantification depth are well-understood (e.g. satisfiability of $\mathsf{Q}^k\mathsf{CTL}$, with at most k nested blocks of quantifiers, is $(k+1)$-EXPTIME-complete), very few results are known about the expressiveness of this logic. We address such expressiveness questions in this paper. We first consider the *distinguishing power* of these logics (i.e., their ability to separate models), their relationship with behavioural equivalences, and their ability to capture the behaviours of finite Kripke structures with so-called characteristic formulas. We then consider their *expressive power* (i.e., their ability to express a property), showing that in terms of expressiveness the hierarchy $\mathsf{Q}^k\mathsf{CTL}$ collapses at level 2 (in other terms, any QCTL formula can be expressed using at most two nested blocks of quantifiers).

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Specification, Verification, Temporal Logic, Expressiveness, Tree automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.28

1 Introduction

Temporal logics have been introduced in computer science in the late 1970's [18]; they provide a powerful formalism for specifying and verifying (e.g. model checking [19, 3]) correctness properties of (finite-state models representing) evolving systems. Various kinds of temporal logics have been defined, with different expressiveness, succinctness and algorithmic properties. For instance, the *Computation Tree Logic* (CTL) expresses properties of the computation tree of the system under study; it has rather limited expressive power (it cannot express fairness), but enjoys PTIME-complete model-checking. The *Linear-time Temporal Logic* (LTL) expresses properties of a single execution at a time. It can express fairness along that single execution, but cannot express properties of other possible executions; LTL model checking is PSPACE-complete. The logic CTL* combines CTL and LTL, offering better expressiveness than CTL with the same (theoretical) complexity as LTL.

In terms of expressiveness, CTL* still has some limitations: in particular, it lacks the ability of *counting*. For instance, it cannot express that an event occurs (at least) at every even position along a path, or that a state has two successors. In order to cope with this, temporal logics have been extended with *propositional quantifiers* [20]: those quantifiers allow for adding fresh atomic propositions in the model before evaluating the truth value of a temporal-logic formula. That a state has at least two successors can then be expressed

* Work supported by STREP project Cassting (FP7-601148) and ERC Stg Grant EQualIS (FP7-308087).

(in *quantified CTL*, hereafter written QCTL) by saying that it is possible to label the model with atomic proposition p in such a way there is a successor that is labelled with p and one that is not.

The algorithmic questions about QCTL have been extensively studied [14, 7, 17, 8, 4, 15], with various semantic assumptions (in particular, depending on whether the labellings refer to the finite-state model or to its execution tree). In the latter case, model-checking QCTL with at most k nested propositional quantifiers (written $\mathsf{Q}^k\mathsf{CTL}$) has been shown k-EXPTIME-complete [15], which tends to indicate that propositional quantification substantially increases the expressiveness of the logic.

However, the expressiveness of QCTL has remained mostly unexplored, except for a few (rather straightforward) results: QCTL is as expressive as[1] MSO; QCTL and QCTL^* are equally expressive; QCTL formulas can be written in prenex normal form. To the best of our knowledge, no results are known about the relative expressiveness of QCTL and its fragments $\mathsf{Q}^k\mathsf{CTL}$ with limited quantification height.

In this paper, we focus on the so-called *tree semantics*, where quantification refers to the execution tree. The main contributions presented in this paper are the following:

- all logics from $\mathsf{Q}^1\mathsf{CTL}$ to full QCTL^* have the same *distinguishing* power. We define a bisimulation equivalence that precisely corresponds to the distinguishing power of these logics.
- given a regular tree $\mathcal{T}$, one can build a *characteristic formula* $\Phi_{\mathcal{T}}$ in $\mathsf{Q}^2\mathsf{CTL}$ such that any tree $\mathcal{T}'$ satisfying $\Phi_{\mathcal{T}}$ is isomorphic to T. This completes the result of [2], where a construction of characteristic formulas in CTL was presented for the bisimulation equivalence.
- all logics from $\mathsf{Q}^2\mathsf{CTL}$ to QCTL^* have the same expressiveness, but $\mathsf{Q}^1\mathsf{CTL}$ and $\mathsf{Q}^1\mathsf{CTL}^*$ are less expressive. In particular, any QCTL or QCTL^* formula can be translated into a formula in $\mathsf{Q}^2\mathsf{CTL}$ (i.e., with at most two nested blocks of propositional quantifiers)[2].

The outline of the paper is as follows: we begin with setting up the necessary formalism in order to define QCTL^* and its fragments. We then devote Section 3 to the study of the *distinguishing power* of $\mathsf{Q}^k\mathsf{CTL}$, showing in particular that if QCTL can distinguish between two finite-state models, then already $\mathsf{Q}^1\mathsf{CTL}$ can. We also develop *characteristic formulas* in this section. Finally, Section 4 focuses on expressiveness, with as main result the fact that any QCTL^* formula has an equivalent formula in $\mathsf{Q}^2\mathsf{CTL}$, but $\mathsf{Q}^2\mathsf{CTL}$ is strictly more expressive than $\mathsf{Q}^1\mathsf{CTL}^*$.

2 Definitions

2.1 Words and trees

Let Σ be a finite alphabet. A finite word over Σ is a finite sequence $w = (w_i)_{1 \leq i \leq k}$. The integer k is the length of w, usually denoted by $|w|$. We write ε for the empty word, which is the unique word of size zero, and identify the alphabet Σ with the set of words of length 1 as long

[1] This requires adequate definitions, since a temporal logic formula may only deal with the reachable part of the model, while MSO has a more *global* point of view.

[2] Notice that a similar result exists for MSO over trees: one alternation of *second-order quantifiers* is enough to express any MSO property. But while it relies on similar tree-automata techniques, our result does not directly follow from the result for MSO: the translated MSO formula may contain *first-order quantifiers*, which involves extra propositional quantifiers when translated to QCTL.

as it raises no ambiguity. For a non-empty word w, we let $\mathsf{last}(w) = w_{|w|}$. The concatenation of two words w and w', denoted $w \cdot w'$, is the word z of size $|w| + |w'|$ defined as

$$z_i = w_i \quad \text{when } 1 \leq i \leq |w| \qquad\qquad z_i = w'_{i-|w|} \quad \text{when } |w|+1 \leq i \leq |w| + |w'|.$$

For a finite word w and a finite set of words S, we let $w \cdot S = \{w \cdot w' \mid w' \in S\}$. A word w is a prefix of a word z if there exists a word w' such that $z = w \cdot w'$. This defines a partial order $\leq$ over words. An infinite word is the limit of an infinite increasing sequence of finite words; infinite words can equivalently be seen as infinite sequences of letters of Σ. The size of an infinite word w is $|w| = +\infty$. The notions of concatenation of a finite word with an infinite word, and of prefix of an infinite word, are easily obtained from the definitions for finite words. We write Σ^* for the set of finite words, and Σ^ω for the set of infinite words. Given an infinite word w, we write $\mathsf{Inf}(w) \subseteq \Sigma$ for the set of letters that appear infinitely many times in w.

▶ **Definition 1.** Let D be a set and Σ be a finite alphabet. A Σ-*labelled* D-*tree* is a pair $\mathcal{T} = \langle T, l\rangle$, where

- $T \subseteq D^*$ is a non-empty set of finite words on D satisfying the following constraints: for any non-empty word $x \in T$, which can be written unequivocally as $x = y \cdot c$ with $y \in D^*$ and $c \in D$, the word y is in T. Moreover, we require that for every word $y \in T$, there exists $c \in D$ such that $y \cdot c \in T$.
- $l \colon T \to \Sigma$ is a labelling function.

Let $\mathcal{T} = \langle T, l\rangle$ be a Σ-labelled tree. The elements of T are the *nodes* of $\mathcal{T}$ and the empty word ε, which is easily shown to necessarily belong to T, is the *root* of $\mathcal{T}$. Given a node $x \in T$, we use $\mathsf{Succ}_{\mathcal{T}}(x)$ (or $\mathsf{Succ}(x)$ when the underlying tree is clear) to denote the set of successors of x, defined as $x \cdot \Sigma \cap T$. The *degree* of $x \in T$, denoted $\mathsf{d}_{\mathcal{T}}(x)$ (or $\mathsf{d}(x)$), is the cardinality of $\mathsf{Succ}(x)$. A tree has *bounded branching* if the degree of all its nodes is bounded. Given a node $x \in T$, we denote with $\mathcal{T}_x$ the *subtree* $\langle T_x, l_x\rangle$ rooted at x, defined by $T_x = \{y \in D^* \mid x \cdot y \in T\}$.

An (infinite) *branch* in T is an infinite increasing (for the prefix relation) sequence of nodes. A branch can be identified with an infinite word $\rho = (x_i)_{i\in\mathbb{N}}$ over D; it can be associated with the infinite word $l(\rho) = (l(x_i))_{i\in\mathbb{N}}$ over Σ. A branch ρ contains a node x whenever x is a prefix of ρ.

2.2 Kripke structures

Fix a finite set AP of atomic propositions.

▶ **Definition 2.** A *Kripke structure* is a tuple $\mathcal{K} = \langle V, E, \ell\rangle$ where V is a finite set of vertices, $E \subseteq V \times V$ is a set of edges (requiring that for any $v \in V$, there exists $v' \in V$ s.t. $(v, v') \in E$), and $\ell \colon V \to 2^{\mathsf{AP}}$ is a labelling function.

A path in a Kripke structure is a finite or infinite word w over V such that $(w_i, w_{i+1}) \in E$ for all $i < |w|$. We write $\mathsf{Path}^*_{\mathcal{K}}$ and $\mathsf{Path}^\omega_{\mathcal{K}}$ for the sets of finite and infinite paths of $\mathcal{K}$, respectively. Given a vertex $v \in V$, the execution tree of $\mathcal{K}$ from v is the 2^{AP}-labelled V-tree $\mathcal{T}_{\mathcal{K},v} = \langle T_{\mathcal{K},v}, \hat{\ell}\rangle$ with $T_{\mathcal{K},x} = \{w \in V^* \mid v \cdot w \in \mathsf{Path}^*_{\mathcal{K}}\}$ and $\hat{\ell}(v \cdot w) = \ell(\mathsf{last}(v \cdot w))$. Notice that two nodes w and w' of $\mathcal{T}_{\mathcal{K},v}$ for which $\mathsf{last}(w) = \mathsf{last}(w')$ give rise to the same subtrees. A tree is said *regular* when it corresponds to the execution tree of some finite Kripke structure.

It will be convenient in some situations to allow Kripke structures to have infinitely many states. For instance, a tree can be seen as an infinite-state Kripke structure.

2.3 QCTL* and its fragments

This section is devoted to the definition of the logic QCTL* and its fragments, and to the semantics of these logics.

2.3.1 Syntax and (tree) semantics

▶ **Definition 3.** The syntax of QCTL* over a finite set AP of atomic propositions is defined by the following grammar:

$$\begin{aligned}\mathsf{QCTL}^* \ni \varphi_s, \psi_s &::= q \mid \neg\varphi_s \mid \varphi_s \vee \psi_s \mid \mathbf{E}\varphi_p \mid \mathbf{A}\varphi_p \mid \exists p.\ \varphi_s \\ \varphi_p, \psi_p &::= \varphi_s \mid \neg\varphi_p \mid \varphi_p \vee \psi_p \mid \mathbf{X}\,\varphi_p \mid \varphi_p\,\mathbf{U}\,\psi_p\end{aligned}$$

where q and p range over AP. Formulas φ_s and ψ_s are called *state formulas*, while φ_p and ψ_p are *path formulas*. The size of a formula $\varphi \in \mathsf{QCTL}^*$, denoted $|\varphi|$, is the number of steps needed to built φ. The logic CTL* is obtained from QCTL* by disallowing rule $\exists p.\ \varphi_s$. The logics QCTL and CTL are obtained by using path formulas built with the following grammar:

$$\varphi_p, \psi_p ::= \mathbf{X}\,\varphi_s \mid \varphi_s\,\mathbf{U}\,\psi_s.$$

Finally, LTL is the fragment of CTL* containing exactly one path quantifier[3] ($\mathbf{E}$ or $\mathbf{A}$); it is easily seen that any LTL formula can be written as $\mathbf{E}\varphi$ or $\mathbf{A}\varphi$, where φ contains no path quantifier.

QCTL* formulas are evaluated over (execution trees of) finite Kripke structures. We begin with defining the semantics of CTL* (and CTL). Given a CTL* formula φ, an infinite 2^{AP}-labelled D-tree $\mathcal{T} = \langle T, l\rangle$, a branch ρ and a node (*i.e.*, a prefix) x of ρ, we write $\mathcal{T}, \rho, x \models \varphi$ to denote that φ holds at x along ρ. This is defined inductively as follows:

$$\begin{aligned}\mathcal{T}, \rho, x &\models p \text{ iff } p \in l(x)\\ \mathcal{T}, \rho, x &\models \neg\varphi \text{ iff } \mathcal{T}, \rho, x \not\models \varphi\\ \mathcal{T}, \rho, x &\models \varphi \vee \psi \text{ iff } \mathcal{T}, \rho, x \models \varphi \text{ or } \mathcal{T}, \rho, x \models \psi\\ \mathcal{T}, \rho, x &\models \mathbf{E}\varphi_p \text{ iff } \exists \rho' \text{ containing } x.\ \mathcal{T}, \rho', x \models \varphi_p\\ \mathcal{T}, \rho, x &\models \mathbf{A}\varphi_p \text{ iff } \forall \rho' \text{ containing } x.\ \mathcal{T}, \rho', x \models \varphi_p\\ \mathcal{T}, \rho, x &\models \mathbf{X}\,\varphi_p \text{ iff } \exists a \in D.\ x \cdot a \leq \rho \text{ and } \mathcal{T}, \rho, x \cdot a \models \varphi_p\\ \mathcal{T}, \rho, x &\models \varphi_p\,\mathbf{U}\,\psi_p \text{ iff } \exists w \in D^*.\ x \cdot w \leq \rho \text{ and}\\ &\qquad \mathcal{T}, \rho, x \cdot w \models \psi_p \text{ and } \forall w' \lneq w.\ \mathcal{T}, \rho, x \cdot w' \models \varphi_p\end{aligned}$$

In order to extend this definition to QCTL*, we first introduce some extra definitions. For a function $l\colon T \to 2^{\mathsf{AP}}$ and $P \subseteq \mathsf{AP}$, we write $l{\cap}P$ for the function defined as $(l{\cap}P)(q) = l(q){\cap}P$ for all $x \in T$. Now, for $P \subseteq \mathsf{AP}$, two trees $\mathcal{T} = \langle T, l\rangle$ and $\mathcal{T}' = \langle T', l'\rangle$ are said *P-equivalent* (denoted by $\mathcal{T} \equiv_P \mathcal{T}'$) if $T = T'$, and $l \cap P = l' \cap P$. Then:

$$\mathcal{T}, \rho, x \models \exists p.\ \varphi_s \text{ iff } \exists \mathcal{T}' \equiv_{\mathsf{AP}\setminus\{p\}} \mathcal{T} \text{ s.t. } \mathcal{T}', \rho, x \models \varphi_s.$$

It is easily noticed that for any state formula φ_s of QCTL* and any two paths ρ and ρ' containing node x, we have $\mathcal{T}, \rho, x \models \varphi_s$ if, and only if, $\mathcal{T}, \rho', x \models \varphi_s$. In view of this,

[3] It is more usual to assume that LTL formulas contain no path quantifier at all; our definition allows for a more uniform presentation, and fits better in our branching-time setting, making it clear how LTL formulas are to be evaluated (existentially or universally) in a tree.

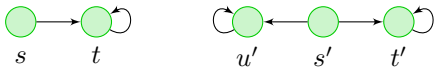


Figure 1 Two Kripke structures (with empty labelling functions) with s and s' being bisimilar.

we define $\mathcal{T}, x \models \varphi_s$ in the natural way. Finally, for a Kripke structure $\mathcal{K}$ and one of its states v, we write $\mathcal{K}, v \models \varphi_s$ whenever $\mathcal{T}_{\mathcal{K},v}, \varepsilon \models \varphi_s$.

In the sequel, we use standard abbreviations such as $\top = p \vee \neg p$, $\bot = \neg\top$, $\mathbf{F}\,\varphi = \top\,\mathbf{U}\,\varphi$, $\mathbf{G}\,\varphi = \neg\mathbf{F}\,\neg\varphi$ and $\forall p.\ \varphi_s = \neg\exists p.\ \neg\varphi_s$. Also note that $\mathbf{A}\varphi_p = \neg\,\mathbf{E}\neg\varphi_p$.

2.3.2 Discussion on the semantics.

Several other natural semantics coexist in the literature for propositional quantifiers. Above we have introduced the *tree semantics*: $\exists p.\ \varphi$ holds true when there exists a p-labelling of the *execution tree* of the Kripke structure under which φ holds. Therefore two nodes (in the tree) corresponding to the same state of the Kripke structure may be labelled differently with the newly-quantified propositions. For example, in the Kripke structure depicted at Fig. 1, we have: $s \models \exists p.(\,\mathbf{EX}\,p \wedge \mathbf{EX}\ \mathbf{EX}\ \neg\,p)$, because it is possible to label only the first occurrence of t with p in the execution tree of this Kripke structure.

Another classical semantics (called the *structure semantics*) consists in labelling the Kripke structure directly. With this semantics, $\exists p.(\,\mathbf{EX}\,p \wedge \mathbf{EX}\ \mathbf{EX}\ \neg\,p)$ would not hold true in state s of Fig. 1: all the occurrences of state t in the execution tree would be labelled the same way.

These two semantics have very different properties (see [15] for a deeper study of these semantics). But none of them make QCTL bisimulation-invariant[4]: as we exemplify in the next section, under both semantics, QCTL can *count* the number of successors of a given state (and thus distinguish between states s and s' in Fig. 1, even though they are bisimilar).

Finally, let us mention the *amorphous semantics* [7], where $\exists p.\ \varphi$ holds true at a state s in some Kripke structure $\mathcal{K}$ if, and only if, there exists some Kripke structure $\mathcal{K}'$ with a state s' such that s and s' are bisimilar and for which there exists a p-labeling making φ hold true at s'. With this semantics, the logic is insensitive to unwinding, and more generally it is bisimulation-invariant (for example, states s and s' of Fig. 1 satisfy the same formulas). This semantics corresponds to bisimulation quantification as studied in [5, 9].

2.3.3 Fragments of QCTL*.

The central topic of this paper is the hierarchy of temporal logics defined by restricting quantifications in QCTL formulas. We define this hierarchy here. Given QCTL* (state) formulas φ and $(\psi_i)_i$, and atomic propositions $(p_i)_i$ that appear free in φ (*i.e.*, not as quantified propositions), we write $\varphi[(p_i \to \psi_i)_i]$ (or $\varphi[(\psi_i)_i]$ when $(p_i)_i$ are understood from the context) for the formula obtained from φ by replacing each occurrence of p_i with ψ_i. Given two sublogics L_1 and L_2 of QCTL*, we write $L_1[L_2] = \{\varphi[(\psi_i)_i] \mid \varphi \in L_1,\ (\psi_i)_i \in L_2\}$.

For a set $P = \{p_i \mid 1 \le i \le k\} \subseteq \mathsf{AP}$, we define *blocks of existential quantifiers* $\exists P.\ \varphi$ as a shorthand for $\exists p_1.\ \exists p_2 \ldots \exists p_k.\ \varphi$. We write EQ^{1}CTL for the set of formulas of the form $\exists P.\ \varphi$ for $\varphi \in$ CTL, and define Q^{1}CTL = CTL[Q^{1}CTL] and Q^{k+1}CTL = Q^{1}CTL[Q^{k}CTL].

[4] The notion of bisimulation is formally defined in Section 2.4.

▶ **Example 4.** *Consider formula*

$$\mathbf{E}_1\mathbf{X}\,\varphi = \mathbf{EX}\,\varphi \wedge \neg\,\exists p.\ \big(\mathbf{EX}\,(p \wedge \varphi) \wedge \mathbf{EX}\,(\neg\, p \wedge \varphi)\big)$$

(where we assume that p does not appear in φ). This $\mathsf{Q}^1\mathsf{CTL}$ formula states that there is exactly one successor satisfying φ: if there were two of them, then labelling only one of them with p would falsify the formula. Now, for $P = \{p_i \mid 1 \leq i \leq k\} \subseteq \mathsf{AP}$, consider the following formula

$$\varphi_{\mathsf{dupl}(P)} = \exists\{q_1, q_2\}.\ \mathbf{E}_1\mathbf{X}\,(q_1) \wedge \mathbf{E}_1\mathbf{X}\,(q_2) \wedge \mathbf{AX}\,(\neg\, q_1 \vee \neg\, q_2) \wedge \bigwedge_{p \in P} \big(\mathbf{EX}\,(q_1 \wedge p) \Leftrightarrow \mathbf{EX}\,(q_2 \wedge p)\big).$$

where $P \cap \{q_1, q_2\} = \varnothing$. This formula in $\mathsf{Q}^2\mathsf{CTL}$ holds true whenever there are two different successor nodes that carry the same atomic propositions of P. Formula $\varphi_{\mathsf{exp}} = \forall P.\ \varphi_{\mathsf{dupl}(P)}$ in $\mathsf{Q}^3\mathsf{CTL}$ is then true in nodes that have at least $2^k + 1$ successors: it states that any labelling with P gives rise to at least two successors with the exact same labelling. Of course, a simpler formula could be obtained for expressing the existence of exactly k successors satisfying a given property; for instance:

$$\mathbf{E}_k\mathbf{X}\,\varphi = \exists P.\ \Big[\bigwedge_{1 \leq i \leq k} \mathbf{E}_1\mathbf{X}\,p_i \wedge \bigwedge_{1 \leq i \neq j \leq k} \mathbf{AX}\,\neg(p_i \wedge p_j) \wedge \mathbf{AX}\,\Big(\big(\bigvee_{1 \leq i \leq k} p_i\big) \Leftrightarrow \varphi\Big)\Big].$$

This formula is in $\mathsf{Q}^2\mathsf{CTL}$. We can express that at least *k successors satisfy φ in $\mathsf{Q}^1\mathsf{CTL}$ as follows:*

$$\mathbf{E}_{\geq k}\mathbf{X}\,\varphi = \exists P.\ \Big(\bigwedge_{1 \leq i \leq k} \mathbf{EX}\,\big(p_i \wedge \bigwedge_{i' \neq i} \neg\, p_{i'}\big) \wedge \mathbf{AX}\,\Big(\big(\bigvee_{1 \leq i \leq k} p_i\big) \Rightarrow \varphi\Big)\Big).$$

2.4 Expressive power and distinguishing power

In the sequel, we compare the relative expressiveness of the fragments $\mathsf{Q}^k\mathsf{CTL}$. Several criteria are classically used to compare the expressiveness of temporal logics: one can compare their ability to distinguish between models (the *distinguishing power*), their ability to express properties (the *expressive power*), or their *succinctness*. In this paper, we only consider the former two notions, which we now formally define.

Distinguishing power. A logic $\mathcal{L}$ is said to be *at least as distinguishing* as another logic $\mathcal{L}'$ over a class $\mathcal{M}$ of models, denoted $\mathcal{L} \geq_{\mathcal{M}} \mathcal{L}'$ (we may omit to mention $\mathcal{M}$ when it is clear from the context), whenever any two states s and s' of any two structures $\mathcal{K}$ and $\mathcal{K}'$ in $\mathcal{M}$ that are $\mathcal{L}$-equivalent (i.e., for all $\varphi \in \mathcal{L}$, it holds $\mathcal{K}, s \models \varphi$ if, and only if, $\mathcal{K}', s' \models \varphi$) are also $\mathcal{L}'$-equivalent. Both logics are said *equally distinguishing*, written $\mathcal{L} \equiv_{\mathcal{M}} \mathcal{L}'$, if $\mathcal{L} \geq_{\mathcal{M}} \mathcal{L}'$, and $\mathcal{L}' \geq_{\mathcal{M}} \mathcal{L}$; finally, $\mathcal{L}$ is *strictly more distinguishing* than $\mathcal{L}'$, denoted $\mathcal{L} >_{\mathcal{M}} \mathcal{L}'$, whenever $\mathcal{L} \geq_{\mathcal{M}} \mathcal{L}'$, and $\mathcal{L}' \ngeq_{\mathcal{M}} \mathcal{L}$. In our setting, $\mathcal{M}$ is the class of all finite Kripke structures.

For classical branching-time temporal logics, it is well known [10] that CTL^*, CTL, and the fragment $\mathsf{B}(\mathbf{X})$ of CTL not involving the *Until* modality, all have the same distinguishing power. Note also that the distinguishing power is often related to some behavioral equivalence. Here we recall the classical notion of (strong) bisimulation: given two Kripke structures $\mathcal{K} = \langle V, E, \ell\rangle$ and $\mathcal{K}' = \langle V', E', \ell'\rangle$, a relation $R \subseteq V \times V'$ is a bisimulation when for any $(v, v') \in R$, the following properties hold:

- $\ell(v) = \ell'(v')$;
- for any transition $(v, w) \in E$, there is a transition $(v', w') \in E$ such that $(w, w') \in R$;

- for any transition $(v', w') \in E$, there is a transition $(v, w) \in E$ such that $(w, w') \in R$.

Two states v and v' are bisimilar (denoted by $v \sim v'$) whenever there exists a bisimulation relation R such that $(v, v') \in R$.

Bisimilarity characterizes the distinguishing power of CTL^*, CTL and $\mathsf{B}(\mathbf{X})$ for finitely-branching Kripke structures: in particular, two bisimilar states cannot be distinguished by CTL [10]. For instance, CTL cannot distinguish between a finite Kripke structure and its execution tree.

Expressive power. A logic $\mathcal{L}$ is said to be *at least as expressive* as a logic $\mathcal{L}'$ over a class $\mathcal{M}$ of models, which we denote by $\mathcal{L} \succeq_{\mathcal{M}} \mathcal{L}'$ (omitting to mention $\mathcal{M}$ if it is clear from the context), whenever for any formula $\varphi' \in \mathcal{L}'$, there is a $\varphi \in \mathcal{L}$ such that φ and φ' are equivalent over $\mathcal{M}$. Both logics $\mathcal{L}$ and $\mathcal{L}'$ are *equally expressive*, denoted $\mathcal{L} \approxeq_{\mathcal{M}} \mathcal{L}'$, when $\mathcal{L} \succeq \mathcal{L}'$ and $\mathcal{L}' \succeq \mathcal{L}$; finally, $\mathcal{L}$ is strictly more expressive than $\mathcal{L}'$, written $\mathcal{L} \succ_{\mathcal{M}} \mathcal{L}'$, if $\mathcal{L} \succeq \mathcal{L}'$ and $\mathcal{L}' \not\succeq \mathcal{L}$.

One easily notices that expressive power is finer than distinguishing power: being more distinguishing implies being more expressive. The converse is not true: for instance, CTL^* is strictly more expressive than CTL [6], and CTL is strictly more expressive than $\mathsf{B}(\mathbf{X})$.

3 Distinguishing power of QCTL

In this section, we prove the following:

▶ **Theorem 5.** *Over finite Kripke structures,* $\mathsf{CTL} < \mathsf{Q}^1\mathsf{CTL} \equiv \mathsf{Q}^k\mathsf{CTL} \equiv \mathsf{Q}^k\mathsf{CTL}^* \equiv \mathsf{QCTL} \equiv \mathsf{QCTL}^*$ *for* $k \geq 1$.

That $\mathsf{CTL} < \mathsf{Q}^1\mathsf{CTL}$ is easily observed, for instance using the Kripke structures of Fig. 1: states s and s' are bisimilar, thus equivalent for CTL, but formula $\mathbf{E}_1\mathbf{X}\top$ ("*there is exactly one successor*") is true in s and false in s'. We now prove the equivalences of Theorem 5.

3.1 Characteristic formulas with QCTL

As said above, CTL has enough power to distinguish between two non-bisimilar states. More precisely, given a finite Kripke structure $\mathcal{K}$ and a state v, one can build a CTL formula $\alpha_{\mathcal{K},v}$ such that for any Kripke structure $\mathcal{K}'$ and any state v', we have: $\mathcal{K}', v' \models \alpha_{\mathcal{K},v}$ if, and only if, $v \sim v'$ [2].

For QCTL and QCTL^*, the appropriate behavioural equivalence is the *isomorphism of the execution tree*. Two trees $\mathcal{T} = \langle T, \ell \rangle$ and $\mathcal{T}' = \langle T', \ell' \rangle$ are said isomorphic if there exists a bijection $\varphi \colon T \to T'$ such that $\ell'(\varphi(t)) = \ell(t)$ for all t, and $\varphi(\varepsilon_{\mathcal{T}}) = \varepsilon_{\mathcal{T}'}$ and $\varphi(u)$ is a successor of $\varphi(t)$ in $\mathcal{T}'$ if, and only if, u is a successor of t in $\mathcal{T}$. As we now explain, QCTL can capture the behaviour of $\mathcal{K}$ up to *tree isomorphism*: there exists a $\mathsf{Q}^2\mathsf{CTL}$ formula $\beta_{\mathcal{K},v}$ such that for any tree $\mathcal{T}'$, it holds $\mathcal{T}' \models \beta_{\mathcal{K},v}$ if, and only if, $\mathcal{T}_{\mathcal{K},v}$ and $\mathcal{T}'$ are isomorphic.

Given a finite Kripke structure $\mathcal{K} = \langle V, E, \ell \rangle$, and a vertex $v \in V = \{v_0, \ldots, v_n\}$, we define the $\mathsf{Q}^2\mathsf{CTL}$ formula $\beta_{\mathcal{K},v}$ as follows:

$$\beta_{\mathcal{K},v} = \exists V. \Big[v \wedge \mathbf{AG} \bigwedge_{i=0}^{n} \Big(v_i \Rightarrow \big(\bigwedge_{j \neq i} \neg v_j \wedge \bigwedge_{p \in \ell(v_i)} p \wedge \bigwedge_{p \in \mathsf{AP} \setminus V \cup \ell(v_i)} \neg p \wedge \neg\, \mathbf{E}_{\geq \mathsf{d}(v_i)+1} \mathbf{X} \top \wedge \bigwedge_{(v_i, v_j) \in E} \mathbf{E}_1 \mathbf{X}\, v_j \big) \Big) \Big]$$

Formula $\beta_{\mathcal{K},v}$ holds true at the root of a tree $\mathcal{T}'$ when it is possible to associate with every node of $\mathcal{T}$ a state v of $\mathcal{K}$ in such a way that this node will behave exactly as v (same labelling and same successors).

▶ **Lemma 6.** *Let $\mathcal{K} = (V, E, \ell)$ be a finite Kripke structure, and $v \in V$. For any tree $\mathcal{T}' = (T', \ell')$, we have: $\mathcal{T}', \varepsilon_{\mathcal{T}'} \models \beta_{\mathcal{K},v}$ if, and only if, $\mathcal{T}_{\mathcal{K},v}$ and $\mathcal{T}'$ are isomorphic.*

Lemma 6 shows that $\mathsf{Q}^2\mathsf{CTL}$ is powerful enough to distinguish between two finite Kripke structures that do not have isomorphic execution trees. Conversely:

▶ **Lemma 7.** *Two finite Kripke structures that have isomorphic execution trees cannot be separated by QCTL*.*

3.2 $\mathsf{Q}^1\mathsf{CTL}$, QCTL and QCTL* have the same distinguishing power

We show in this section that $\mathsf{Q}^1\mathsf{CTL}$ is sufficient to separate non-isomorphic trees.

▶ **Proposition 8.** $\mathsf{Q}^1\mathsf{CTL}$ has the same distinguishing power as QCTL and QCTL* over finite Kripke structures.

Proof. Given a *finite* tree $\mathcal{U}$, we can easily define a $\mathsf{Q}^1\mathsf{CTL}$ formula $\gamma_{\mathcal{U}}$ expressing that $\mathcal{U}$ is embedded as a subtree, in the following sense: any infinite tree $\mathcal{T}$ satisfying $\gamma_{\mathcal{U}}$ contains a subtree that is isomorphic to $\mathcal{U}$. Write $U = \{p_i \mid 0 \leq i \leq k\}$ for the set of nodes of $\mathcal{U}$ (with $p_0 = \varepsilon_{\mathcal{U}}$ being the root of $\mathcal{U}$). Formula $\gamma_{\mathcal{U}}$ first existentially quantifies over $p_0, \ldots, p_k$ (seen here as atomic propositions), labelling nodes of $\mathcal{T}$ with names of nodes of $\mathcal{U}$. Then $\gamma_{\mathcal{U}}$ checks that

- p_0 holds true initially;
- at most one p_i can be true at a time;
- if p_j is a successor of p_i in $\mathcal{U}$, then $\gamma_{\mathcal{U}}$ enforces $\mathbf{AG}\,(p_i \Rightarrow \mathbf{EX}\, p_j)$;
- the labelling function of $\mathcal{T}$ matches that of $\mathcal{U}$.

Notice that we do not prevent any of the p_i to hold true at several places, which would require an extra universal quantification. Obviously, any tree $\mathcal{T}$ containing a subtree isomorphic to $\mathcal{U}$ satisfies $\gamma_{\mathcal{U}}$. The converse also holds: assuming $\mathcal{T}$ has been labelled with $\{p_i \mid 0 \leq i \leq k\}$, we extract a subtree $\mathcal{V}$ as follows: the root of $\mathcal{T}$ (labelled with p_0) is in $\mathcal{V}$, and for each node n in $\mathcal{V}$ labelled with some p_i, for each successor p_j of p_i in $\mathcal{U}$, we insert into $\mathcal{V}$ exactly one successor of n labelled with p_j ($\gamma_{\mathcal{U}}$ enforces the existence of such a node). It is easily seen that $\mathcal{V}$ is isomorphic to $\mathcal{U}$.

Now, if two regular trees are not isomorphic, there must exist a finite subtree $\mathcal{U}$ of one of them that cannot be embedded into the second one. Then $\gamma_{\mathcal{U}}$ will distinguish between these two trees. It follows that $\mathsf{Q}^1\mathsf{CTL}$ and QCTL (and QCTL*) have the same distinguishing power over finite Kripke structures. ◀

3.3 Behavioural equivalences for $\mathsf{Q}^k\mathsf{CTL}$

We conclude our study of the fragments of QCTL by defining intermediary notions of bisimulations which we prove characterize each level of the $\mathsf{Q}^k\mathsf{CTL}$ hierarchy. We begin with defining those refined bisimulations. In this definition, for two labelling functions $\ell\colon T \to 2^{\mathsf{AP}}$ and $\nu\colon T \to 2^P$ with $P \subseteq \mathsf{AP}$, we write $\ell \circ \nu\colon T \to 2^{\mathsf{AP}}$ for the labelling function mapping each $t \in T$ to $[\ell(t) \cap (\mathsf{AP} \setminus P)] \cup \nu(t)$.

▶ **Definition 9.** Consider two regular trees $\mathcal{T} = \langle T, \ell\rangle$ and $\mathcal{T}' = \langle T', \ell'\rangle$. A relation $R \subseteq T \times T'$ is a *k-labelling bisimulation* if R is a bisimulation and either $k = 0$, or $k > 0$ and

for any $(t,t') \in R$, for any $P \subseteq \mathsf{AP}$ and any regular labelling $\nu\colon T \to 2^P$ (resp. $\nu'\colon T \to 2^P$), there exists a regular labelling $\nu'\colon T' \to 2^P$ (resp. $\nu\colon T \to 2^P$) such that there exists a $(k-1)$-labelling bisimulation in the trees $\mathcal{U} = \langle T, \ell \cup \nu\rangle$ and $\mathcal{U}' = \langle T', \ell' \cup \nu'\rangle$ containing (t,t'). We write $t \approx_k t'$ when there exists a k-labelling bisimulation containing (t,t').

Notice that for all k, it holds $\approx_{k+1} \subseteq \approx_k$. The relation $\approx_\infty$ can thus be defined as the limit of these sequences of relations.

In this definition, with *regular labelling* of a regular tree $\langle T, \ell\rangle$, we mean a labelling ν such that $\langle T, \ell \circ \nu\rangle$ is still regular. We let $\exists_r p.\varphi$ be a new QCTL^* modality where the quantification ranges only over regular labellings. Formally:

$$\mathcal{T}, \rho, x \models \exists_r p.\varphi \text{ iff } \exists \nu\colon T \to 2^{\{p\}}.\ \nu \text{ is regular and } \langle T, \ell \circ \nu\rangle, \rho, x \models \varphi.$$

Quantifying over regular labelling does not restrict the expressive power of our logics:

▶ **Lemma 10.** *Given a finite Kripke structure $\mathcal{K}$, a state s, and a QCTL^* formula $\exists p.\varphi$,*

$$\mathcal{K}, s \models \exists p.\varphi \text{ iff } \mathcal{K}, s \models \exists_r p.\varphi$$

Fix two Kripke structures $\mathcal{K}$ and $\mathcal{K}'$, and a logic $\mathcal{L}$ (intended to range over $\mathsf{Q}^k\mathsf{CTL}$ and $\mathsf{Q}^k\mathsf{CTL}^*$). For any two states s and s', in $\mathcal{K}$ and $\mathcal{K}'$ respectively, we write $s \equiv_{\mathcal{L}} s'$ when s and s' are $\mathcal{L}$-equivalent (i.e., when they cannot be distinguished by $\mathcal{L}$). It is easily noticed that if $\mathcal{L} \subseteq \mathcal{L}'$, then $\equiv_{\mathcal{L}'} \subseteq \equiv_{\mathcal{L}}$.

▶ **Lemma 11.** *Over finite Kripke structures, for any $k \geq 0$, the relations $\approx_k$, $\equiv_{\mathsf{Q}^k\mathsf{CTL}}$ and $\equiv_{\mathsf{Q}^k\mathsf{CTL}^*}$ coincide. More precisely, for any two states s and s',*

$$s \approx_k s' \text{ iff } s \equiv_{\mathsf{Q}^k\mathsf{CTL}} s' \text{ iff } s \equiv_{\mathsf{Q}^k\mathsf{CTL}^*} s'.$$

As a corollary of Lemmas 6, 7 and 11, we get:

▶ **Corollary 12.** *For every $k, k', k'' \geq 2$, the relations $\equiv_{\mathsf{Q}^k\mathsf{CTL}^*}$, $\equiv_{\mathsf{Q}^{k'}\mathsf{CTL}}$, $\equiv_{\mathsf{QCTL}}$, $\approx_{k''}$, and $\approx_\infty$ coincide.*

4 Expressive power of QCTL

We now focus on the relative *expressive power* of the $\mathsf{Q}^k\mathsf{CTL}$ hierarchy. Notice that being more distinguishing implies being more expressive. Hence we already have $\mathsf{CTL} \prec \mathsf{Q}^1\mathsf{CTL}$. In this section, we prove the following:

▶ **Theorem 13.** *Over finite Kripke structures, $\mathsf{CTL} \prec \mathsf{Q}^1\mathsf{CTL} \preceq \mathsf{Q}^1\mathsf{CTL}^* \prec \mathsf{Q}^2\mathsf{CTL} \approxeq \mathsf{Q}^k\mathsf{CTL} \approxeq \mathsf{Q}^k\mathsf{CTL}^* \approxeq \mathsf{QCTL} \approxeq \mathsf{QCTL}^*$ for $k \geq 2$.*

4.1 $\mathsf{Q}^2\mathsf{CTL}$ is strictly more expressive than $\mathsf{Q}^1\mathsf{CTL}$ and $\mathsf{Q}^1\mathsf{CTL}^*$

In order to prove this, we have to exhibit a formula of $\mathsf{Q}^2\mathsf{CTL}$ with no equivalent formula in $\mathsf{Q}^1\mathsf{CTL}$. First consider the Kripke structures depicted at Fig. 2. Those structures depend on an integer parameter p. As stated in the following lemma, these structures cannot be distinguished by any $\mathsf{Q}^1\mathsf{CTL}^*$ formula of size less than p:

▶ **Lemma 14.** *For the Kripke structures $\mathcal{K}_p$ and $\mathcal{K}'_p$ of Fig. 2, and for any $\varphi \in \mathsf{Q}^1\mathsf{CTL}^*$ of size less than p it holds $\mathcal{K}_p, s_0 \models \varphi$ if, and only if, $\mathcal{K}'_p, s'_0 \models \varphi$.*

Figure 2 The states s_0 and s'_0 can be distinguished by $\mathbf{E}_1\mathbf{X}\,(\,\mathbf{E}_1\mathbf{X}\,a)$ (we use double arrows labelled with $[k]$ to indicate the presence of k arrows).

▶ **Theorem 15.** *We have $\mathsf{Q}^2\mathsf{CTL} \succ \mathsf{Q}^1\mathsf{CTL}$ and $\mathsf{Q}^2\mathsf{CTL} \succ \mathsf{Q}^1\mathsf{CTL}^*$.*

Proof. $\mathsf{Q}^1\mathsf{CTL}$ is syntactically contained in $\mathsf{Q}^2\mathsf{CTL}$, so that $\mathsf{Q}^2\mathsf{CTL} \succeq \mathsf{Q}^1\mathsf{CTL}$. Moreover, from Theorem 16, every QCTL^* formula can be expressed in $\mathsf{Q}^2\mathsf{CTL}$, i.e. $\mathsf{Q}^2\mathsf{CTL} \succeq \mathsf{QCTL}^*$, which in particular entails $\mathsf{Q}^2\mathsf{CTL} \succeq \mathsf{Q}^1\mathsf{CTL}^*$.

Moreover the $\mathsf{Q}^2\mathsf{CTL}$ formula $\mathbf{E}_1\mathbf{X}\,(\,\mathbf{E}_1\mathbf{X}\,a)$ (which states that there exists a unique path leading to a state where there is a unique successor verifying a) allows us to distinguish the trees of Fig. 2 (for any p): indeed $\mathbf{E}_1\mathbf{X}\,(\,\mathbf{E}_1\mathbf{X}\,a)$ holds true in s_0, but not in s'_0.

Now assume that $\mathbf{E}_1\mathbf{X}\,(\,\mathbf{E}_1\mathbf{X}\,a)$ have an equivalent formula in $\mathsf{Q}^1\mathsf{CTL}^*$. On the one hand, for any p, this formula would holds true in s_0 and not in s'_0; on the other hand, it would have a given size p_0, and according to Lemma 14 it could not distinguish between s_0 and s'_0. Hence $\mathbf{E}_1\mathbf{X}\,(\,\mathbf{E}_1\mathbf{X}\,a)$ has no $\mathsf{Q}^1\mathsf{CTL}^*$ formula. ◀

4.2 QCTL and Q^2CTL are equally expressive

In this section we prove that the hierarchies $\mathsf{Q}^k\mathsf{CTL}$ and $\mathsf{Q}^k\mathsf{CTL}^*$ also collapse in terms of expressive power. We propose an effective translation, using symmetric tree automata, which proves the following result:

▶ **Theorem 16.** *Any QCTL^* formula can be translated into an equivalent formula in $\mathsf{Q}^2\mathsf{CTL}$.*

4.2.1 Symmetric tree automata

We consider here so-called *symmetric* tree automata (*i.e.*, automata over *unranked* trees of arbitrary branching), borrowing formalism from MSO-automata of [12, 13, 24], in order to prove that the expressiveness hierarchy of $\mathsf{Q}^k\mathsf{CTL}$ collapses: the proof consists in first translating any QCTL formula into a (symmetric) tree automaton, and then expressing acceptance of such a tree automaton as a $\mathsf{Q}^2\mathsf{CTL}$ formula. Using "classical" tree automata (as in [16]), the second step would not be possible (at least not easily), as QCTL cannot distinguish the different successors in a ranked tree; moreover, it would only provide an equivalent formula for a limited branching degree.

In the literature, a similar construction is done for MSO [23, 11, 1, 24]: MSO-automata are powerful tree automata whose transition functions are defined with first-order-logic formulas (where the states of the automata are used as unary predicates); this provides a powerful way of describing transition functions for unranked trees with arbitrary branching. Then any MSO formula φ can be turned into a MSO-automaton $\mathcal{A}_\varphi$ recognizing exactly the trees satisfying φ. Here we use an slightly different (but equally expressive) model of tree automata, which correspond to MSO-automata where transition functions are in a so-called *basic form* (see [24] for full details). Formally:

▶ **Definition 17.** Fix an alphabet Σ. A *symmetric parity tree automaton* over Σ is a tuple $\mathcal{A} = \langle Q, q_0, \delta, \Omega \rangle$ where

- Q is the finite set of states of the automaton, and q_0 is the initial state;
- $\delta \colon Q \times \Sigma \to 2^{(\mathbb{N}^{2^Q} \times 2^{2^Q})}$ is the transition function;
- $\Omega \colon Q \to \mathbb{N}$ defines the parity acceptance condition.

An execution of such a tree automaton $\langle Q, q_0, \delta, \Omega \rangle$ over a Σ-labelled D-tree $\mathcal{T} = \langle T, l \rangle$ is a $Q \times T$-labelled $Q \times D$-tree $\mathcal{T}' = \langle T', l' \rangle$ satisfying the following requirements:

- $l'(\varepsilon_{T'}) = (q_0, \varepsilon_T)$, and for any node $n = (q_i, d_i)_{1 \le i \le k}$, it holds $l'(n) = (q_k, (d_i)_{1 \le i \le k})$;
- for any node $n = (q_i, d_i)_{1 \le i \le k}$ of T', there is a tuple $(E, U) \in \delta(l'(n))$ such that, writing $e = \sum_{s \in 2^Q} E(s)$ and viewing E as a multiset $\{E_j \mid 1 \le j \le e\}$, there exists a set $\Delta = \{d'_j \mid 1 \le j \le e\}$ of e distinct directions in D such that
  - for all $1 \le j \le e$, it holds $(d_i)_{1 \le i \le k} \cdot d'_j \in T$,
  - for all $1 \le j \le e$, and for all $q \in E_j$, node $n \cdot (q, d'_j)$ is in T',
  - for all node $(d_i)_{1 \le i \le k+1}$ in T such that $d_{k+1} \notin \Delta$, for there exists $\overline{q} \in U$, such that for all $q \in \overline{q}$, node $n \cdot (q, d_{k+1})$ is in T'.

A branch of an execution tree is accepting if its associated sequence of states of $\mathcal{A}$ satisfies the parity condition (the least priority appearing infinitely often is even). An execution tree is accepting whenever all its branches are. The language $\mathcal{L}(\mathcal{A})$ of such an automaton $\mathcal{A}$ is the set of trees over which $\mathcal{A}$ has an accepting execution tree.

▶ **Example 18.** *Let* $\mathsf{AP} = \{a, b\}$ *and* $\Sigma = 2^{\mathsf{AP}}$. *Let* $\mathcal{A}$ *be the automaton with states* $\{q_0, q_1, q_2\}$, *with* q_0 *being the initial state, and with*

$$\delta(q_0, \sigma) = \begin{cases} (q_1 \mapsto 1; q_0) & \text{if } a \in \sigma \\ (\varnothing; q_0) & \text{otherwise} \end{cases} \qquad \delta(q_1, \sigma) = \begin{cases} (\varnothing; q_2) & \text{if } b \notin \sigma \\ (q_1 \mapsto 1; q_0) & \text{if } a, b \in \sigma \\ (\varnothing; q_0) & \text{otherwise} \end{cases}$$

and $\delta(q_2, \sigma) = (\varnothing; q_2)$ *for any* σ. *The transition* $\delta(q_0, a) = (q_1 \mapsto 1; q_0)$ *means that when the automaton is visiting some node* n *in state* q_0, *one successor of* n *will be visited in state* q_1, *and all the other nodes will be visited in state* q_0. *Similarly,* $\delta(q_2, \sigma) = (\varnothing; q_2)$ *indicates that when the automaton is in state* q_2, *it will remain in state* q_2 *when visiting all the successors of the node being visited; in this case,* $\varnothing$ *represents the empty multiset, or equivalently the constant function* $\mathbf{0}$. *In the end, assuming that* q_0 *and* q_1 *are accepting (Büchi condition), this automaton accepts those tree in which any node labelled with* a *has a successor labelled with* b.

As defined above, symmetric tree automata are alternating, in the sense that they may launch several computations along the same subtree of the input tree. A symmetric tree automaton is said *non-alternating*[5] when the transition function takes values in $2^{(\mathbb{N}^Q \times 2^Q)}$. One may notice that in this case, any execution tree can be seen as a D-tree, instead of a $Q \times D$-tree. The automaton of Example 18 is non-alternating.

It is not difficult to notice that symmetric tree automata are closed under union: given two symmetric tree automata $\mathcal{A}$ and $\mathcal{B}$, there exists a tree automaton $\mathcal{C}$ accepting the union

[5] The classical terminology for this class of automata is *non-deterministic*, for historical reasons. We prefer using *non-alternating*, which better characterizes this class.

of the set of trees accepted by $\mathcal{A}$ and $\mathcal{B}$. Similarly, *non-alternating* symmetric tree automata are easily seen to be closed under projection.

Symmetric tree automata can also be proven to be closed under complement; this however involves dualizing the transition function and writing it back under the expected *basic form*. Finally, any symmetric tree automaton can be transformed into a non-alternating one accepting the same set of trees. This can be achieved by a refined powerset construction, as explained in [24]. In the end:

▶ **Theorem 19** ([24]). *Any symmetric tree automaton can be made non-alternating. Non-alternating symmetric tree automata are closed under union, intersection, projection and complement.*

4.2.2 From QCTL to symmetric tree automata

In this section, we briefly explain how a state formula of QCTL^* can be translated into a (non-alternating) symmetric tree automaton accepting the same tree language. This construction follows the same ideas as explained in [16]: in that paper however, the construction builds a "classical" tree automaton, running on bounded-branching ranked trees. The translation back to QCTL is not possible for such tree automata, as QCTL cannot distinguish between ranked successors of a node.

The construction is inductive, using Theorem 19 for the various rules defining state-formulas of QCTL^*. The basic cases of atomic propositions and boolean operations are easy to handle. Existential quantification over atomic propositions is handled by projection: given a tree automaton $\mathcal{A}_\varphi$ corresponding to a QCTL^* formula φ, the projection of $\mathcal{A}_\varphi$ from alphabet 2^{AP} to alphabet $2^{\mathsf{AP}\setminus\{p\}}$ yields a tree automaton characterizing formula $\exists p.\ \varphi$.

Finally, formulas of the form $\mathbf{E}\varphi_p$ and $\mathbf{A}\varphi_p$ are handled by considering word automata for the path formula φ_p: nested QCTL subformulas can be handled separately, by induction, and replaced by fresh atomic propositions. The resulting formula $\tilde{\varphi}_p$ is a pure LTL formula, and can be turned into a deterministic parity word automaton, which in turn is easily turned into a symmetric tree automaton for $\mathbf{E}\tilde{\varphi}_p$ or $\mathbf{A}\tilde{\varphi}_p$. The nested QCTL subformulas can then be included back by plugging the corresponding symmetric tree automata where needed.

Notice that this construction involves several exponential blowups in the size of the automaton and in the number of priorities. Since model checking $\mathsf{Q}^k\mathsf{CTL}$ formulas is k-EXPTIME-complete, there is no hope of avoiding this non-elementary explosion in the construction of the automaton (because our translation back into $\mathsf{Q}^2\mathsf{CTL}$ is linear in the size of the automaton, as we explain below).

4.2.3 From symmetric tree automata to $\mathsf{Q}^2\mathsf{CTL}$

In this section, we turn a non-alternating symmetric tree automaton $\mathcal{A} = (Q, q_0, \delta, \Omega)$ into a QCTL formula $\Phi_\mathcal{A}$ such that $\mathcal{T} \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \mathcal{T}, \varepsilon \models \Phi_\mathcal{A}$ for any 2^{AP}-labelled tree $\mathcal{T}$.

We begin with a preliminary lemma, which we believe is interesting in itself but will be used in a special case in the sequel.

▶ **Lemma 20.** *For any LTL formula $\mathbf{E}\varphi$, there exists a $\mathsf{Q}^1\mathsf{CTL}$ formula $\Psi_{\mathbf{E}\varphi}$ such that for all tree $\mathcal{T}$, it holds $\mathcal{T}, \varepsilon \models \mathbf{E}\varphi$ if, and only if, $\mathcal{T}, \varepsilon \models \Psi_{\mathbf{E}\varphi}$.*

Proof. Following classical techniques [22, 21], we associate with φ a Büchi (word) automaton $\mathcal{B}_\varphi = \langle Q, q_0, \delta, F\rangle$ accepting exactly the set of infinite words in which φ holds. The automaton $\mathcal{B}_\varphi$ is a non-deterministic word automaton, so that for any $q \in Q$ and any $\sigma \in 2^{\mathsf{AP}}$,

it holds $\delta(q,\sigma) \subseteq Q$. The acceptance condition is defined in terms of a set $F \subseteq Q$ of states: an execution is accepting if some state of F is visited infinitely many times.

We use this automaton in order to write a $\mathsf{Q}^1\mathsf{CTL}$-formula characterizing those trees containing (at least) one branch accepted by $\mathcal{B}_\varphi$. The formula expresses that the tree can be (partially) labelled with states of $\mathcal{B}_\varphi$ in such a way that at least one branch is fully labelled with a sequence of states corresponding to an accepting run of $\mathcal{B}_\varphi$. Following this intuition, we use the states of $Q = \{q_i \mid 0 \leq i \leq n = |Q|-1\}$ as new atomic propositions. For the sake of readability, we define the following two shorthand formulas: for a subset $S \subseteq Q$, formula λ_S is the propositional formula $\bigvee_{q\in S} q$, while for $P, P' \subseteq \mathsf{AP}$, we write $\chi_{P,P'}$ for $\bigwedge_{p\in P} p \wedge \bigwedge_{p'\in P'} \neg p'$. We then let $\Psi_{\mathbf{E}\varphi} = \exists q_0 \ldots \exists q_n.\ \widetilde{\Psi}_{\mathbf{E}\varphi}$, with $\widetilde{\Psi}_{\mathbf{E}\varphi}$ being defined as

$$q_0 \wedge \bigwedge_{i=0}^{n} \mathbf{AG}\left(q_i \Rightarrow \left(\neg\lambda_{Q\setminus\{q_i\}} \wedge \Big[\bigvee_{P\subseteq \mathsf{AP}'} (\chi_{P,\mathsf{AP}'\setminus P} \wedge \mathbf{EX}\,\lambda_{\delta(q_i,P)} \wedge \neg\,\mathbf{EX}\,\lambda_{Q\setminus\delta(q_i,P)})\Big]\right)\right)$$
$$\wedge \Big(\mathbf{AG}\,\mathbf{AF}\,(\neg\lambda_Q \vee \lambda_F)\Big)$$

where AP' stands for $\mathsf{AP}\setminus\{q_0,\ldots,q_n\}$. Formula $\Psi_{\mathbf{E}\varphi}$ reads as follows: it is possible to label the input tree with propositions $(q_i)_{0\leq i\leq n}$ in such a way that the root is labelled with q_0, and any node labelled with some q_i is not labelled with any other state and has a successor node labelled with a possible successor state of $\mathcal{B}_\varphi$. This in particular entails that at least one branch ρ is fully labelled with states of Q. Finally, the second part of the formula asserts that any branch has to fulfill the Büchi acceptance condition or to contain unlabelled nodes. In particular, the branch ρ identified above satisfies the Büchi condition.

It is now easy to prove equivalence of $\mathbf{E}\varphi$ and $\Psi_{\mathbf{E}\varphi}$:

- consider a tree $\mathcal{T} = \langle T, l\rangle$ in which one branch ρ satisfies φ: then $\mathcal{B}_\varphi$ has an accepting run on $l(\rho)$, and this run can be used to label $\mathcal{T}$ with states of $\mathcal{B}_\varphi$ so as to fulfill $\widetilde{\Psi}_{\mathbf{E}\varphi}$;
- conversely, if $\mathcal{T}$ can be labelled with states of Q in order to fulfill $\widetilde{\Psi}_{\mathbf{E}\varphi}$, then one branch has to be fully labelled, and each node along that branch will be labelled with exactly one state of Q. Formula $\Psi_{\mathbf{E}\varphi}$ then enforces that the labelling of consecutive nodes is coherent with the transition function of $\mathcal{B}_\varphi$, and that it satisfies the Büchi condition. ◀

We now describe our main construction: we consider a non-alternating symmetric parity tree automaton $\mathcal{A} = \langle Q, q_0, \delta, \Omega\rangle$, where $Q = \{q_i \mid 0 \leq i \leq n = |Q|-1\}$ and $\delta(q,\sigma) \subseteq \mathbb{N}^Q \times 2^Q$ is a set of pairs (E,U) with $E\colon Q \to \mathbb{N}$ and $U \subseteq Q$. For such a pair (E,U), we write $k(E) = \sum_{q\in Q} E(q)$, and we let $k_{\max}$ be the largest such value appearing in δ. We also see E as a multiset $\{E_i \mid 1 \leq i \leq k(E)\}$.

Reusing ideas (and notations) of the proof of Lemma 20, and with $\mathsf{AP}' = \mathsf{AP}\setminus\{q_0,\ldots,q_n, p_1,\ldots,p_{k_{\max}}\}$, our formula $\Phi_\mathcal{A}$ is written as $\exists q_0 \ldots \exists q_n . \exists p_1 \ldots \exists p_{k_{\max}}.\ \widetilde{\Phi}_\mathcal{A}$, where $\widetilde{\Phi}_\mathcal{A}$ is defined as

$$q_0 \wedge \bigwedge_{i=0}^{n} \mathbf{AG}\left[q_i \Rightarrow \left(\neg\lambda_{Q\setminus\{q_i\}} \wedge \bigvee_{P\subseteq\mathsf{AP}'} \left(\chi_{P,\mathsf{AP}'\setminus P} \wedge \bigvee_{(E,U)\in\delta(q_i,P)} \Psi_{(E,U)}\right)\right)\right] \wedge \neg\,\Psi_{\mathbf{E}\,\neg\,\mathsf{parity}(\Omega)}$$

where $\Psi_{(E,U)}$ encodes the transition (E,U) of $\mathcal{A}$ and $\Psi_{\mathbf{E}\,\neg\,\mathsf{parity}(\Omega)}$ encodes the parity acceptance condition. Using Lemma 20, the latter formula can be expressed as a $\mathsf{Q}^1\mathsf{CTL}$ formula (since parity acceptance condition can be expressed in LTL). Now, we let

$$\Psi_{(E,U)} = \bigwedge_{j=1}^{k(E)} \Big[\mathbf{E}_1\mathbf{X}\,p_j \wedge \mathbf{EX}\Big(p_j \wedge \bigwedge_{\substack{1\leq j'\leq k_{\max}\\ \wedge\, j'\neq j}} \neg p_{j'} \wedge E_j\Big) \wedge \mathbf{AX}\Big((\bigwedge_{j=1}^{k} \neg p_j) \Rightarrow \bigvee_{q\in U} q\Big)\Big]$$

Note that $\Psi_{(E,U)}$ belongs to $\mathsf{Q}^1\mathsf{CTL}$ (because it uses $\mathbf{E}_1\mathbf{X}\,p_j$), so that $\Phi_{\mathcal{A}}$ is in $\mathsf{Q}^2\mathsf{CTL}$. It is not hard to see that $\Phi_{\mathcal{A}}$ characterizes the behaviour of $\mathcal{A}$, since the labelling of a tree $\mathcal{T}$ with $\{q_i \mid 0 \leq i \leq n\}$ corresponds to an execution tree of $\mathcal{A}$ on $\mathcal{T}$. This ends the proof of Theorem 16.

5 Concluding remarks

We see two main directions for future work. First it would be interesting to consider the expressiveness of QCTL^* fragments when the size of block of quantifiers (*i.e.* the number of atomic propositions used in a block) is bounded (several of our proofs use arbitrary many propositions). The second direction is to analyze the expressiveness of these logics in the context of the *structure semantics* (when the labellings apply to Kripke structures instead of execution trees) in order to see whether the hierarchy also collapses for this semantics.

References

1 D. Berwanger and A. Blumensath. The monadic theory of tree-like structures. In *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 285–302. Springer, 2002.

2 M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2):115–131, 1988.

3 E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *LOP'81*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

4 A. Da Costa, F. Laroussinie, and N. Markey. Quantified CTL: Expressiveness and model checking. In *CONCUR'12*, volume 7454 of *LNCS*, pages 177–192. Springer, 2012.

5 G. D'Agostino and M. Hollenberg. Logical questions concerning the μ-calculus: Interpolation, lyndon and łoś-tarski. *Journal of Symbolic Logic*, 65(1):310–332, 2000.

6 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

7 T. French. Decidability of quantified propositional branching time logics. In *AJCAI'01*, volume 2256 of *LNCS*, pages 165–176. Springer, 2001.

8 T. French. Quantified propositional temporal logic with repeating states. In *TIME-ICTL'03*, pages 155–165. IEEE Comp. Soc. Press, 2003.

9 T. French. *Bisimulation Quantifiers for Modal Logics*. Ph.D. thesis, School of Computer Science & Software Engineering, University of Western Australia, 2006.

10 M. C. B. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1), 1985.

11 D. Janin and G. Lenzi. On the relationship between monadic and weak monadic second order logic on arbitrary trees. *Fundamenta Informaticae*, 61(3-4):247–265, 2004.

12 D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *MFCS'95*, volume 969 of *LNCS*, pages 552–562. Springer, 1995.

13 J. Johannsen and M. Lange. CTL^+ is complete for double exponential time. In *ICALP'03*, volume 2719 of *LNCS*, pages 767–775. Springer, 2003.

14 O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In *CAV'95*, volume 939 of *LNCS*, pages 325–338. Springer, 1995.

15 F. Laroussinie and N. Markey. Quantified CTL: expressiveness and complexity. *Logicical Methods in Computer Science*, 10(4), 2014.

16 F. Laroussinie and N. Markey. Augmenting ATL with strategy contexts. *Information and Computation*, 245:98–123, 2015.

17 A. C. Patthak, I. Bhattacharya, A. Dasgupta, P. Dasgupta, and P. P. Chakrabarti. Quantified computation tree logic. *Information Processing Letters*, 82(3):123–129, 2002.

18 A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE Comp. Soc. Press, 1977.

19 J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *SOP'82*, volume 137 of *LNCS*, pages 337–351. Springer, 1982.

20 A. P. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems*. PhD thesis, Harvard University, Cambridge, Massachussets, USA, 1983.

21 M. Y. Vardi. Nontraditional applications of automata theory. In *TACS'94*, volume 789 of *LNCS*, pages 575–597. Springer, 1994.

22 M. Y. Vardi and P. Wolper. Automata theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):183–221, 1986.

23 I. Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Computer Science*, 275(1-2):311–346, 2002.

24 F. Zanasi. *Expressiveness of Monadic Second-Order Logics on Infinite Trees of Arbitrary Branching Degrees*. Master's thesis, Amsterdam University, the Netherlands, 2012.

Model Checking Flat Freeze LTL on One-Counter Automata*

Antonia Lechner[1], Richard Mayr[2], Joël Ouaknine†[3], Amaury Pouly†[4], and James Worrell[5]

1 **Department of Computer Science, University of Oxford, UK**
antonia.lechner@cs.ox.ac.uk
2 **School of Informatics, LFCS, University of Edinburgh, UK**
http://homepages.inf.ed.ac.uk/rmayr/
3 **Department of Computer Science, University of Oxford, UK**
joel.ouaknine@cs.ox.ac.uk
4 **Department of Computer Science, University of Oxford, UK**
amaury.pouly@cs.ox.ac.uk
5 **Department of Computer Science, University of Oxford, UK**
james.worrell@cs.ox.ac.uk

Abstract

Freeze LTL is a temporal logic with registers that is suitable for specifying properties of data words. In this paper we study the model checking problem for Freeze LTL on one-counter automata. This problem is known to be undecidable in full generality and PSPACE-complete for the special case of deterministic one-counter automata. Several years ago, Demri and Sangnier investigated the model checking problem for the flat fragment of Freeze LTL on several classes of counter automata and posed the decidability of model checking flat Freeze LTL on one-counter automata as an open problem. In this paper we resolve this problem positively, utilising a known reduction to a reachability problem on one-counter automata with parameterised equality and disequality tests. Our main technical contribution is to show decidability of the latter problem by translation to Presburger arithmetic.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases One-counter automata, disequality tests, reachability, freeze LTL, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.29

1 Introduction

Runs of infinite-state machines, such as counter automata, can naturally be seen as *data words*, that is, sequences in which each position is labelled by a letter from a finite alphabet and a datum from an infinite domain. Freeze LTL is an extension of Linear Temporal Logic with registers (or variables) and a binding mechanism, which has been introduced to specify properties of data words [3, 4, 8, 11]. The registers allow to compare data at different positions along the same computation.

An example of a freeze LTL formula is

$$\mathsf{F}(v \wedge \downarrow_r \mathsf{XF}(v \wedge \uparrow_r)) \,. \tag{1}$$

† Supported by ERC grant AVS-ISS (648701).
* See full version at https://arxiv.org/abs/1606.02643

Evaluated on a run of a one-counter automaton, this formula is true if and only if there are at least two different positions in the run which both have control state v and the same counter value. Intuitively the operator $\downarrow_r$ binds the current counter value to register r, while the operator $\uparrow_r$ tests whether the current counter value is equal to the content of register r.

This paper concerns the model checking problem for Freeze LTL on one-counter automata. It is known that this problem is undecidable in general, but PSPACE-complete if one restricts to *deterministic* one-counter automata [5]. Rather than restricting the class of one-counter automata, one can seek to identify decidable syntactic fragments of Freeze LTL. This approach was pursued in [6], which studied the *flat* fragment of Freeze LTL. The flatness condition places restrictions on the occurrence of the binding construct $\downarrow_r$ in relation to the until operator (see Section 2.2 for details). For example, in a flat formula in negation normal form the binding operator $\downarrow_r$ can occur within the scope of F but not G. (Thus formula (1) is flat.) The flatness restriction for Freeze LTL has a similar flavour to the flatness restriction for constraint LTL [2] and for Metric Temporal Logic [1].

Demri and Sangnier [6] considered the decidability of model checking flat Freeze LTL across a range of different counter-machine models. For one-counter automata they showed decidability of model checking for a certain fragment of flat Freeze LTL and they left open the problem of model checking flat Freeze LTL in general.

The approach taken in [6] was to reduce the model checking problem for fragments of Freeze LTL on a class of counter automata to a repeated reachability problem in counter automata of the same class with certain kinds of parameterised tests. In particular, under their approach the model checking problem for flat Freeze LTL on one-counter automata reduces to a repeated reachability problem for the class of one-counter automata extended with parameterised equality and disequality tests. This problem considers one-counter automata whose transitions may be guarded by equality or disequality tests that compare the counter value to integer-valued parameters, and it asks whether there exist parameter values such that there is an infinite computation that visits an accepting location infinitely many times. The main technical contribution of this paper is to show decidability of the latter problem by reduction to the decision problem for Presburger arithmetic.

A related work is [9], which considers one-counter automata with parameterised updates and equality tests. It is shown in [9] that reachability in this model is inter-reducible with the satisfiability problem for quantifier-free Presburger arithmetic with divisibility, and therefore decidable. In contrast to [9], in the present paper the counter automata do not have parameterised updates but they do have parameterised disequality tests. The results in this paper do not appear to be straightforwardly reducible to those of [9] nor *vice versa*. Both reachability problems can be seen as special cases of a long-standing open problem identified by Ibarra *et al.* [10] which asks to decide reachability on a class of automata with a single integer-valued counter, sign tests, and parameterised updates.

2 Preliminaries

2.1 One-Counter Automata with Equality and Disequality Tests

We consider automata with an associated single counter, which ranges over the nonnegative integers, and with both equality and disequality tests on counter values. Formally, a *one-counter automaton* (1-CA) is a tuple $\mathcal{C} = (V, E, \lambda, \tau)$, where V is a finite set of *states*, $E \subseteq V \times V$ is a finite set of *edges* between states, $\lambda : E \to Op$ labels each edge with an element from $Op = \{\mathrm{add}(a) : a \in \mathbb{Z}\} \cup \{\mathrm{eq}(a) : a \in \mathbb{N}\}$, and $\tau : V \to 2^{\mathbb{N}}$ maps each state v to a finite set $\tau(v)$ of *invalid counter values* at state v. Intuitively the operation $\mathrm{add}(a)$ adds a

to the counter and $\mathrm{eq}(a)$ tests the counter for equality with a. The association of invalid counter values with each state can be seen as a type of disequality test. This feature is not present in classical presentations of 1-CA, but we include it here to facilitate our treatment of freeze LTL.

For any edge $e = (v, v')$ with $\lambda(e) = \mathrm{op}(a)$, define $\mathrm{start}(e) = v$, $\mathrm{end}(e) = v'$, and $\mathrm{weight}(e) = a$ if $\mathrm{op} = \mathrm{add}$, $\mathrm{weight}(e) = 0$ if $\mathrm{op} = \mathrm{eq}$. A *path* γ is a finite word on the alphabet E: $\gamma = e_1 \cdots e_n$ such that $\mathrm{end}(e_i) = \mathrm{start}(e_{i+1})$ for every i. The *length* of γ, denoted $|\gamma|$, is n. The *state sequence* of γ is $\mathrm{start}(e_1)$, $\mathrm{end}(e_1)$, $\mathrm{end}(e_2)$, …, $\mathrm{end}(e_n)$. The *start* of γ, denoted $\mathrm{start}(\gamma)$, is $\mathrm{start}(e_1)$. The *end* of γ, denoted $\mathrm{end}(\gamma)$, is $\mathrm{end}(e_n)$. A path is *simple* if it contains no repeated vertices. The *weight* of γ, denoted by $\mathrm{weight}(\gamma)$, is $\sum_{i=1}^{n} \mathrm{weight}(e_i)$. A *subpath* γ' of γ is any factor of γ: $\gamma' = e_i e_{i+1} \dots e_j$. If γ and γ' are two paths such that $\mathrm{end}(\gamma) = \mathrm{start}(\gamma')$, $\gamma\gamma'$ is the concatenation of both paths.

A *cycle* ω is a path such that $\mathrm{start}(\omega) = \mathrm{end}(\omega)$. A cycle is *simple* if there are no repeated vertices except for the starting point, which appears twice. A cycle is *positive* if it has positive weight, *negative* if it has negative weight and *zero-weight* if it has weight zero. We denote by $\omega^k = \underbrace{\omega\omega\cdots\omega}_{k \text{ times}}$ the sequence of k iterations of cycle ω.

A *configuration* of a 1-CA $\mathcal{C} = (V, E, \lambda, \tau)$ is a pair (v, c) with $v \in V$ and $c \in \mathbb{Z}$. Intuitively, (v, c) corresponds to the situation where the 1-CA is in state v with counter value c. Since counter values range over the nonnegative integers, configurations (v, c) with $c \geqslant 0$ are called *valid*, otherwise they are *invalid*. The transition relation E between states with guards λ and τ induces an unlabelled transition relation between configurations: for any two configurations (v, c) and (v', c'), there is a transition $(v, c) \longrightarrow (v', c')$ if and only if there is an edge $e \in E$ with $\lambda(e) = \mathrm{op}(a)$ for some a, $\mathrm{start}(e) = v$, $\mathrm{end}(e) = v'$, and $\mathrm{weight}(e) = c' - c$. We will sometimes write $(v, c) \xrightarrow{e} (v', c')$ for such a transition. The transition is *valid* if $c, c' \geqslant 0$ and $c \notin \tau(v)$, and also $c = a$ if $\mathrm{op} = \mathrm{eq}$. Otherwise it is *invalid*.

A *computation* π is a (finite or infinite) sequence of transitions:

$$\pi = (v_1, c_1) \longrightarrow (v_2, c_2) \longrightarrow (v_3, c_3) \longrightarrow \cdots$$

We write $|\pi|$ for the length of π. If $(v_1, c_1) \xrightarrow{e_1} (v_2, c_2) \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} (v_n, c_n)$ is a finite computation, we will also write it as $(v_1, c_1) \xrightarrow{\gamma}{}^{*} (v_n, c_n)$, where $\gamma = e_1 e_2 \cdots e_{n-1}$, or simply $(v_1, c_1) \longrightarrow^{*} (v_n, c_n)$. A computation π is *valid* if all transitions in the sequence are valid, otherwise it is *invalid*. If π is invalid, an *obstruction* is a configuration (v_i, c_i) such that $(v_i, c_i) \longrightarrow (v_{i+1}, c_{i+1})$ is an invalid transition, or, if π is of finite length $n - 1$, $i = n$ and $c_i < 0$.

Given a path γ and a counter value $c \in \mathbb{Z}$, the *path computation* $\gamma(c)$ is the (finite) computation starting at $(\mathrm{start}(\gamma), c)$ and following the sequence of transitions that correspond to the edges in γ.

A *one-counter automaton with parameterised tests* is a tuple (V, E, X, λ, τ), where V, E and λ are defined as before, X is a set of nonnegative integer variables, $Op = \{\mathrm{add}(a) : a \in \mathbb{Z}\} \cup \{\mathrm{eq}(a), \mathrm{eq}(x) : a \in \mathbb{N}, x \in X\}$, and $\tau : V \to 2^{\mathbb{N} \cup X}$. Note that $\tau(v)$ is still required to be finite for each $v \in V$.

For a given 1-CA $\mathcal{C} = (V, E, \lambda, \tau)$, an initial configuration (v, c) and a target configuration (v', c'), the *reachability* problem asks if there is a valid computation from (v, c) to (v', c'). When $\mathcal{C}$ has sets $F_1, \dots, F_n \subseteq V$ of final states and an initial configuration (v, c), the *generalised repeated reachability* problem asks if there is a valid infinite computation from (v, c) which visits at least one state in each F_i infinitely often.

For a 1-CA $\mathcal{C} = (V, E, X, \lambda, \tau)$ with parameterised tests with given initial configuration (v, c) and target configuration (v', c'), the *reachability* problem asks if there exist values for

the parameters such that there is a computation from (v, c) to (v', c'). Similarly, in the case where $\mathcal{C}$ has sets $F_1, \ldots, F_n \subseteq V$ of final states and an initial configuration (v, c), the *generalised repeated reachability* problem asks if there exist values for the parameters such that substituting these values satisfies the generalised repeated reachability condition above.

2.2 Model Checking Freeze LTL on One-Counter Automata

Freeze LTL [5] is an extension of Linear Temporal Logic that can be used to specify properties of data words. Freeze LTL is one of a variety of formalisms that arise by augmenting a temporal or modal logic with variable binding. Given a finite alphabet Σ and set of *registers* R, the formulas of Freeze LTL are given by the following grammar

$$\varphi \quad ::= \quad a \quad | \quad \uparrow_r \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad \mathsf{X}\,\varphi \quad | \quad \varphi\,\mathsf{U}\,\varphi \quad | \quad \downarrow_r \varphi\,,$$

where $a \in \Sigma$ and $r \in R$. We write $\mathrm{LTL}^{\downarrow}$ for the set of formulas of Freeze LTL. A *sentence* is a formula in which each occurrence of a subformula $\uparrow_r$ is in the scope of an operator $\downarrow_r$ (for the same register r).

In general, formulas of $\mathrm{LTL}^{\downarrow}$ are interpreted over data words. In this paper we are interested in a particular kind of data word, namely those arising from valid computations of 1-CA, and we directly define the semantics of $\mathrm{LTL}^{\downarrow}$ over such computations (assuming that the alphabet Σ is the set of control locations of the 1-CA). In this context $\downarrow_r$ can be seen as a binding construct that stores in register r the counter value at the current position in a computation, while $\uparrow_r$ tests whether the counter value at the current position is equal to the content of register r. Formally, define a *register valuation* to be a partial function $f : R \to \mathbb{N}$ and consider a valid infinite computation

$$\pi = (v_1, c_1) \longrightarrow (v_2, c_2) \longrightarrow (v_3, c_3) \longrightarrow \cdots$$

of a 1-CA $\mathcal{C}$. We define a satisfaction relation $\pi, i \vDash_f \varphi$ specifying when an $\mathrm{LTL}^{\downarrow}$ formula φ is satisfied at position i in π under valuation f:

$$\begin{aligned}
\pi, i \vDash_f a &\iff^{\text{def}} v_i = a \\
\pi, i \vDash_f \uparrow_r &\iff^{\text{def}} c_i = f(r) \\
\pi, i \vDash_f \mathsf{X}\,\varphi &\iff^{\text{def}} \pi, i+1 \vDash_f \varphi \\
\pi, i \vDash_f \varphi_1\,\mathsf{U}\,\varphi_2 &\iff^{\text{def}} \pi, j \vDash_f \varphi_2 \text{ for some } j \geqslant i \text{ and } \pi, k \vDash_f \varphi_1 \text{ for all } i \leqslant k < j \\
\pi, i \vDash_f \downarrow_r \varphi &\iff^{\text{def}} \pi, i \vDash_{f[r \mapsto c_i]} \varphi
\end{aligned}$$

where $f[r \mapsto c]$ is the function that maps r to c and is otherwise equal to f. We have omitted the clauses for the Boolean connectives.

An occurrence of a subformula in a $\mathrm{LTL}^{\downarrow}$ formula is *positive* if it lies within the scope of an even number of negations, otherwise it is *negative*. The *flat fragment* of $\mathrm{LTL}^{\downarrow}$ is the set of $\mathrm{LTL}^{\downarrow}$ formulas such that in every positive occurrence of a subformula $\varphi_1\,\mathsf{U}\,\varphi_2$, the binding operator $\downarrow_r$ does not appear in φ_1, and in every negative occurrence of such a subformula, the binding operator does not appear in φ_2.

The negation of many natural $\mathrm{LTL}^{\downarrow}$ specifications yield flat formulas. For example, consider the response property $\mathsf{G}(\downarrow_r (\mathrm{req} \to \mathsf{F}(\mathrm{serve} \wedge \uparrow_r)))$, expressing that every request is followed by a serve with the same associated ticket. Here F and G are the "future" and "globally" modalities, which can be expressed in terms of U in a standard way. The negation of this formula is equivalent to $\mathsf{F}(\downarrow_r (\mathrm{req} \wedge \mathsf{G}(\neg\mathrm{serve} \vee \neg\uparrow_r)))$. The latter is easily seen to be flat after rewriting to the core $\mathrm{LTL}^{\downarrow}$ language with only the U temporal operator.

The main subject of this paper is the decidability of the following model checking problem: given a 1-CA $\mathcal{C}$, a valid configuration (v, c) of $\mathcal{C}$, and a flat sentence $\varphi \in \mathrm{LTL}^{\downarrow}$, does there exist a valid infinite computation π of $\mathcal{C}$, starting at (v, c), such that $\pi, 1 \vDash_{\emptyset} \varphi$? Note that, following [6], we have given an existential formulation of the model checking problem. The problem above is equivalent to asking whether $\neg\varphi$ holds along all infinite computations starting at (v, c).

The model checking problem for flat $\mathrm{LTL}^{\downarrow}$ on 1-CA was reduced to the generalised repeated reachability problem for 1-CA with parameterised tests in [6, Theorem 15]. The idea of the reduction is, given a 1-CA $\mathcal{C}$ and a flat $\mathrm{LTL}^{\downarrow}$ sentence φ, to construct a 1-CA with parameterised tests which is the product of $\mathcal{C}$ and φ. This product automaton includes a parameter x_r for each register r that is mentioned in a subformula of φ of type $\downarrow_r \varphi'$. This is where the restriction to the flat fragment of $\mathrm{LTL}^{\downarrow}$ is crucial, since it allows us to assume, without loss of generality, that the value stored in a register is never overwritten along any computation of $\mathcal{C}$, so that it can be represented by precisely one parameter. An occurrence of the binding operator $\downarrow_r$ in φ is represented in the product automaton by an equality test $\mathrm{eq}(x_r)$. A positive occurrence of a formula of the type $\uparrow_r$ is likewise represented by an equality test $\mathrm{eq}(x_r)$, while a negative occurrence of such a subformula is represented by a disequality test $\tau(v_r) = \{x_r\}$.

Note that the definition of 1-CA with parameterised tests in [6] includes parameterised equality and disequality tests (as in the present paper) together with parameterised inequality tests, i.e., testing whether the counter value is less than or greater than the value of a parameter. However, it is clear from the details of the reduction that only equality and disequality tests are needed, and thus we do not consider inequality tests in this paper. Note also that in the previous section we defined 1-CA to have equality tests on edges and disequality tests on states. On the other hand, the 1-CA considered in [6] have both kinds of tests on edges and allow multiple edges between the same pair of states. It is easy to see that both models are equivalent with respect to reachability, i.e., there are reductions in both directions between reachability problems in the two models.

2.3 Presburger Arithmetic

Presburger arithmetic is the first-order logic over $\langle \mathbb{Z}, +, <, 0, 1 \rangle$, where $+$ and $<$ are the standard addition and ordering of integers. Presburger arithmetic is known to be decidable [12]. Using shorthand notation, we can assume that the atomic formulas of Presburger arithmetic are equalities or inequalities between linear polynomials with integer coefficients.

3 Normal Form for Paths

In this section, we show that any valid finite computation of a 1-CA can be rewritten to a normal form whose shape only depends on the automaton. Informally, any such computation can be described as a sequence of "take this edge" and "take this cycle k times". We show that the maximum length of a description of this kind is independent of the original computation.

First we show that without loss of generality, any computation can be broken down into a small number of segments that do not contain any transitions with equality tests. The idea is that any segment between two identical equality tests can be omitted.

▶ **Lemma 1** (Equality test isolation). *Let π be a valid finite computation from (v, c) to (v', c'). Then there exists a path γ such that $\gamma(c)$ is a valid computation from (v, c) to (v', c') and γ*

is of the form $\gamma = \gamma_0 e_1 \gamma_1 e_2 \cdots e_n \gamma_n$*, where* e_i *is an edge with an equality test,* γ_i *is a path without equality tests and* $n \leqslant |E|$*.*

We give a proof of Lemma 1 in the full version of this paper.

We need to introduce some terminology to formalise our notion of normal form. Given a state v, $\mathrm{SC}(v)$ (resp. $\mathrm{SC}^+(v)$, $\mathrm{SC}^-(v)$) denotes the *set of equality-free simple* (resp. *positive simple, negative simple*) *cycles* starting at v. The set of all equality-free simple (resp. positive simple, negative simple) cycles from all vertices is SC (resp. SC^+, SC^-). Note that each equality-free simple cycle is counted several times in SC: once for each state in the cycle.

The *cycle alphabet*, denoted C, consists of symbols of the form $\underline{\omega^k}$ where $\omega \in \mathrm{SC}$ and $k \in \mathbb{N}$. Note that this alphabet is infinite. Also note that $\underline{\omega^k}$ is a single symbol, underlined to indicate the difference from the cycle ω^k, which consists of $|\omega|k$ symbols from E. For convenience, $\underline{\omega}$ is a shorthand for $\underline{\omega^1}$. We naturally define the start and end of symbol $\underline{\omega^k}$ by the start of ω: $\mathrm{start}(\underline{\omega^k}) = \mathrm{end}(\underline{\omega^k}) = \mathrm{start}(\omega)$.

A *folded path* χ is a word on the alphabet $E \cup C$: $\chi = s_1 \cdots s_n$ such that $\mathrm{end}(s_i) = \mathrm{start}(s_{i+1})$ for every $i < n$. We also define the natural unfolding of a folded path as a monoid homomorphism $\mathrm{unfold} : (E \cup C)^* \to E^*$ such that $\mathrm{unfold}(e) = e$ for $e \in E$ and $\mathrm{unfold}(\underline{\omega^k}) = \omega^k$ for $\underline{\omega^k} \in C$. The weight of a folded path is the weight of its unfolding.

From now on, until Theorem 8 at the end of this section, we fix an initial counter value $c \in \mathbb{N}$ and we only consider computations starting at c which do not include equality tests. We refer to a folded path χ as being valid if $\mathrm{unfold}(\chi)(c)$ is a valid computation.

Define the following nondeterministic rewriting system on folded paths. Each rule of the system has a name, a pattern to match against, a condition which must be satisfied for the rule to apply and the result of the rule. We denote by $\chi \rightsquigarrow \chi'$ the fact that χ rewrites to χ'.

Rule	Pattern	Result	Condition
`fold`	$\psi\omega\phi$	$\psi\underline{\omega}\phi$	ω is a simple cycle of nonzero weight.
`simplify`	$\psi\rho\phi$	$\psi\phi$	Nonempty ρ, $\mathrm{weight}(\mathrm{unfold}(\rho)) = 0$ and $\mathrm{end}(\psi) = \mathrm{start}(\phi)$.
`gather`$^+$	$\psi\underline{\omega^k}\rho\underline{\omega^\ell}\phi$	$\psi\underline{\omega^{k+1}}\rho\underline{\omega^{\ell-1}}\phi$	Result is valid, ω is a positive simple cycle and $\ell > 0$.
`gather`$^-$	$\psi\underline{\omega^k}\rho\underline{\omega^\ell}\phi$	$\psi\underline{\omega^{k-1}}\rho\underline{\omega^{\ell+1}}\phi$	Result is valid, ω is a negative simple cycle and $k > 0$.

▶ **Lemma 2** (Soundness). *If* χ *is valid and rewrites to* χ' *then* χ' *is valid. Furthermore,* χ *and* χ' *start and end at the same state and* $\mathrm{weight}(\mathrm{unfold}(\chi)) = \mathrm{weight}(\mathrm{unfold}(\chi'))$*.*

The proof of Lemma 2 is included in the full version of this paper.

▶ **Lemma 3** (Termination). *There are no infinite chains of rewriting.*

A proof of Lemma 3 can be found in the full version. Here we give an informal explanation. The first thing to notice is that the length of a folded path (over alphabet $E \cup C$) never increases after a rewriting operation. The second thing is that the length of a folded path over E (i.e., ignoring symbols from C) never increases either. Since rule `simplify` decreases the length, it can only be applied finitely many times. Similarly, rule `fold` decreases the length over E because it replaces a symbol from E by one from C. Rules `gather`$^\pm$ are more difficult to analyse because they only reorder the path by replacing symbols from C. But as it can be seen, a symbol $\underline{\omega}$, where ω is a positive cycle, can only move left, and similarly a negative cycle can only move right. Intuitively, this process must be finite because once a positive (negative) cycle reaches the leftmost (rightmost) position, it cannot move anymore.

▶ **Lemma 4** (Size of cycle-free subpaths). *If* $\psi\rho\phi$ *is such that* $\rho \in E^*$ *and no rule applies, then* $|\rho| < |V|$*.*

Proof. Assume the contrary: if ρ only consists of edges and has length $\geqslant |V|$, then some state is repeated in the state sequence of ρ. Thus ρ contains a cycle and thus a simple cycle. So rule `fold` applies if the cycle has nonzero weight, or rule `simplify` applies if it has weight zero. ◀

For $S \subseteq \mathbb{Z}$ and $x \in \mathbb{Z}$, we use $S - x$ to denote $\{y - x \mid y \in S\}$. The idea of the next lemma is to show that given a state v, some counter values prevent reordering of cycles within the folded path. These counter values act as a "barrier" for the `gather` rules and increase the size of the normal form. We call these values *critical* for positive (resp. negative) cycles and denote them by $B^+(v)$ (resp. $B^-(v)$). Formally, $B^+(v)$ contains:

- $\tau(v) - \text{weight}(\omega)$, for every positive (resp. negative) simple cycle ω,
- $\tau(\text{end}(\gamma)) - \text{weight}(\gamma)$ for every prefix[1] γ of every positive (resp. negative) simple cycle ω starting at v.

▶ **Lemma 5** (Obstructions in irreducible paths with cycles). *Let ω be a positive cycle and assume that rule* `gather`$^+$ *(resp.* `gather`$^-$*) does not apply on $\psi\underline{\omega^k}\rho\underline{\omega^\ell}\phi$ (which we assume is valid and $k, \ell > 0$) for this particular pattern. Then there exists a (potentially empty) prefix μ of ρ such that* $\text{unfold}(\psi\underline{\omega^k}\mu)(c)$ *has the form $(v, c) \longrightarrow^* (v', c')$ where c' is critical for v' for positive (resp. negative) cycles,* i.e. *$c' \in B^+(v')$ (resp. $c' \in B^-(v')$). Furthermore $B^+(v')$ and $B^-(v')$ only depends on the automaton and*

$$|B^+(v')| \leqslant |\,\mathrm{SC}^+\,| \sum_{v \in V} |\tau(v)| \quad \text{and} \quad |B^-(v')| \leqslant |\,\mathrm{SC}^-\,| \sum_{v \in V} |\tau(v)|.$$

Proof. We first show the result for positive cycles. Let $\pi = \text{unfold}(\psi\underline{\omega^k}\rho\underline{\omega^\ell}\phi)(c)$ and $\pi' = \text{unfold}(\psi\underline{\omega^{k+1}}\rho\underline{\omega^{\ell-1}}\phi)(c)$. To make things slightly easier to understand, note that:

$$\pi = [\text{unfold}(\psi)\omega^k \,\text{unfold}(\rho)\omega\omega^{\ell-1} \,\text{unfold}(\phi)](c)$$
$$\pi' = [\text{unfold}(\psi)\omega^k\omega \,\text{unfold}(\rho)\omega^{\ell-1} \,\text{unfold}(\phi)](c).$$

Since $\text{unfold}(\rho)\omega$ and $\omega\,\text{unfold}(\rho)$ have the same weight, it is clear that the first ($\text{unfold}(\psi)\omega^k$) and last ($\omega^{\ell-1}\,\text{unfold}(\phi)$) parts of the computation are the same in π and π', i.e., they have the same counter values. Consequently, if they are valid in π, the same parts are also valid in π'. Since by the hypothesis `gather`$^+$ does not apply, π' is invalid. So there must be an obstruction (u, d) in the middle part ($\omega\,\text{unfold}(\rho)$) of π'. There are two possibilities.

The first case is when the obstruction (u, d) is in the $\text{unfold}(\rho)$ part of π'. Note that $d = c^* + \text{weight}(\omega)$, where (u, c^*) is the corresponding configuration in the $\text{unfold}(\rho)$ part of π. Since ω is a positive cycle, $d > c^*$ cannot be negative (since (u, c^*) occurs in π, which is valid). Since we assumed that all computations are free of equality tests, the obstruction must be because of a disequality, i.e., it must be that $d = c^* + \text{weight}(\omega) \in \tau(u)$. Thus $c^* \in \tau(u) - \text{weight}(\omega)$ and c^* is critical for u. Then there exists a prefix μ of ρ such that $\text{unfold}(\psi\underline{\omega^k}\mu)(c) = (v, c) \longrightarrow^* (u, c^*)$ and this shows the result.

The second case is when (u, d) is in the ω part of the middle part ($\omega\,\text{unfold}(\rho)$) of π'. Again, it is impossible that the counter value d be negative. Indeed, remember that ω is a

[1] Other than ω and the empty prefix. Indeed the empty prefix is impossible because $c_2 \notin \tau(v_1)$ as π is valid. And ω correspond to the previous case of the definition.

positive cycle and $k > 0$, thus

$$\begin{aligned}\pi' &= [\mathrm{unfold}(\psi)\omega^{k+1}\,\mathrm{unfold}(\rho)\omega^{\ell-1}\,\mathrm{unfold}(\phi)](c)\\ &= [\mathrm{unfold}(\psi)\omega^{k-1}\omega\omega\,\mathrm{unfold}(\rho)\omega^{\ell-1}\,\mathrm{unfold}(\phi)](c)\\ &= (v,c) \xrightarrow{\mathrm{unfold}(\psi)\omega^{k-1}}{}^{*} (v_1,c_1) \xrightarrow{\omega}{}^{*} (v_1,c_2) \xrightarrow{\omega}{}^{*} (v_1,c_3) \xrightarrow{\mathrm{unfold}(\rho)\omega^{\ell-1}\,\mathrm{unfold}(\phi)}{}^{*} (v'',c'').\end{aligned}$$

We already argued that $(v,c) \longrightarrow^{*} (v_1,c_2)$ is valid, so in particular $(v_1,c_1) \xrightarrow{\omega}{}^{*} (v_1,c_2)$ is valid. Note that the obstruction is in the second iteration of ω: $(v_1,c_2) \xrightarrow{\omega}{}^{*} (v_1,c_3)$. Since ω is a positive cycle, $c_2 > c_1$. Note that initially the cycle ω was feasible (with the counter not going negative) starting with a lower counter value (c_1) so the counter cannot possibly become negative on the second iteration starting with a higher counter value (c_2). Thus, again, the obstruction happens because of a disequality. That is, we can write $\omega = \gamma\gamma'$ such that:

$$\pi' = (v,c) \xrightarrow{\mathrm{unfold}(\psi)\omega^{k}}{}^{*} (v_1,c_2) \xrightarrow{\gamma}{}^{*} (u,d) \xrightarrow{\gamma'}{}^{*} (v_1,c_3) \xrightarrow{\mathrm{unfold}(\rho)\omega^{\ell-1}\,\mathrm{unfold}(\phi)}{}^{*} (v'',c'')$$

and the obstruction happens because $d \in \tau(u)$. Note however that $d = c_2 + \mathrm{weight}(\gamma)$ and thus $c_2 \in \tau(u) - \mathrm{weight}(\gamma)$. In this case, c_2 is critical for v_1. Choose μ to be the empty word, so that $\mathrm{unfold}(\psi\underline{\omega^{k}}\mu)(c) = (v,c) \longrightarrow^{*} (v_1,c_2)$ to show the result.

Observe that the definition of critical values only depend on the automaton itself. Furthermore, the size of $B^{+}(v)$ can easily be bounded. Indeed, there are $|\,\mathrm{SC}^{+}\,|$ positive simple cycles, so in the first case of the definition, there are at most $|\,\mathrm{SC}^{+}\,||\tau(v')|$ values. In the second case, since the cycle ω is simple, each prefix γ of ω ends at a different state. Thus each state is visited at most once, and v is not visited because the prefix is not empty or equal to ω. So the second case includes at most an additional $|\,\mathrm{SC}^{+}\,|\sum_{u\neq v}|\tau(u)|$ values. Finally the total bound is $|\,\mathrm{SC}^{+}\,|\sum_{v\in V}|\tau(v)|$.

The proof is exactly the same in the negative case except for one detail. This time we move negative cycles to the right so that the middle part of π' ($\mathrm{unfold}(\rho)\omega$) can only get higher counter values than the middle part of π ($\omega\,\mathrm{unfold}(\rho)$), as in the positive case. ◀

▶ **Lemma 6** (Length of irreducible paths). *Let χ be a folded path such that no rule applies on χ. Let $Y = \mathrm{SC}^{+}$ or $Y = \mathrm{SC}^{-}$. Then for every $\omega \in Y$, the number of symbols in χ of the form $\underline{\omega^{\cdot}}$ (the exponent does not matter) is bounded by*

$$|V||Y|\Big(1 + \sum_{v\in V}|\tau(v)|\Big).$$

Proof. Without loss of generality, we show the result for $X = \mathrm{SC}^{+}$. First note that if $\underline{\omega^{k}}$ appears in χ and no rule applies, then $k > 0$, otherwise we could apply `simplify` to remove $\underline{\omega^{0}}$. We can thus decompose the path as:

$$\chi = \phi_0\underline{\omega^{k_1}}\phi_1\underline{\omega^{k_2}}\phi_2\cdots\phi_{n-1}\underline{\omega^{k_n}}\phi_n$$

where $k_i > 0$ and ϕ_i does not contain any $\underline{\omega^{\cdot}}$ symbol. Since no rule applies, by Lemma 5, there exist prefixes $\mu_1,\mu_2,\ldots,\mu_{n-1}$ of $\phi_1,\phi_2,\ldots,\phi_{n-1}$ respectively, such that for each i:

$$(v,c) \xrightarrow{\phi_0\underline{\omega^{k_1}}\phi_1\cdots\phi_{i-1}\underline{\omega^{k_i}}\mu_i}{}^{*} (v_i,c_i) \quad \text{where} \quad c_i \in B^{+}(v_i).$$

Assume for a contradiction that there is a repeated configuration among the (v_i, c_i). Then there exists $i < j$ such that $v_i = v_j$ and $c_i = c_j$. Let $\phi_i = \mu_i\rho$ and $\phi_j = \mu_j\rho'$, and observe that:

$$(v,c) \xrightarrow{\phi_0\underline{\omega^{k_1}}\phi_1\cdots\phi_{i-1}\underline{\omega^{k_i}}\mu_i}{}^* (v_i,c_i) \xrightarrow{\rho\underline{\omega^{k_{i+1}}}\phi_{i+1}\cdots\phi_{j-1}\underline{\omega^{k_j}}\mu_j}{}^* (v_i,c_i) \xrightarrow{\rho'\underline{\omega^{k_{j+1}}}\phi_{j+1}\cdots\phi_{n-1}\underline{\omega^{k_n}}\phi_n}{}^* (v',c').$$

Thus the subpath $\rho\underline{\omega^{k_{i+1}}}\phi_{i+1}\cdots\phi_{j-1}\underline{\omega^{k_j}}\mu_j$ has weight 0 and rule `simplify` must apply:

$$\chi \quad\rightsquigarrow\quad \phi_0\underline{\omega^{k_1}}\phi_1\cdots\phi_{i-1}\underline{\omega^{k_i}}\mu_i\rho'\underline{\omega^{k_{j+1}}}\phi_{j+1}\cdots\phi_{n-1}\underline{\omega^{k_n}}\phi_n$$

which is a contradiction because we assumed that no rule can apply on χ.

Consequently, for any $i \neq j$, we have $(v_i, c_i) \neq (v_j, c_j)$. But remember that $c_i \in B^+(v_i)$, thus $(v_i, c_i) \in A$ where:

$$A = \bigcup_{v\in V} \{v\} \times B^+(v).$$

This shows that $n - 1 \leqslant |A|$. Indeed, by the pigeonhole principle, some pair (v_i, c_i) would be repeated if $n - 1 > |A|$. We can easily bound the size of A using the bound on $B^+(v)$ from Lemma 5:

$$|A| \leqslant \sum_{v\in V} |B^+(v)| \leqslant |V|\,|\,\mathrm{SC}^+\,|\sum_{v\in V}|\tau(v)|.$$

Finally we have

$$n \leqslant |V|\,|\,\mathrm{SC}^+\,|\sum_{v\in V}|\tau(v)| + 1 \leqslant |V|\,|\,\mathrm{SC}^+\,|\Big(1 + \sum_{v\in V}|\tau(v)|\Big)$$

because $|V| \geqslant 1$ and $|\,\mathrm{SC}^+\,| \geqslant 1$ unless there are no positive cycles, in which case $n = 0$ anyway. ◀

▶ **Lemma 7** (Length of equality-free computations). *Let π be a valid finite computation (without equality tests) from (v, c) to (v', c'). Then there exists a folded path χ such that* unfold$(\chi(c))$ *is a valid computation from (v, c) to (v', c'), the length of* unfold$(\chi(c))$ *at most that of π and the word length of χ is bounded by:*

$$|V| + |V|^2|\,\mathrm{SC}\,|^2\Big(1 + \sum_{v\in V}|\tau(v)|\Big)$$

Proof. Let χ_0 be the path defined by π: it is a word over alphabet E and is thus a (trivial) folded path. By definition unfold$(\chi_0(c)) = \pi$ is a valid computation from (v, c) to (v', c') and the length of unfold$(\chi_0(c))$ is equal to that of π. Let χ be any rewriting of χ_0 such that no rule applies on χ: it exists because there are no infinite rewriting chains by Lemma 3. By Lemma 2, unfold$(\chi(c))$ is still a valid computation from (v, c) to (v', c'). Let ω be a simple cycle: note that it is either positive or negative, because rule `simplify` removes zero-weight cycles. Then by Lemma 6, the number of symbols of the form $\underline{\omega^{\cdot}}$ appearing in χ is bounded by[2]:

$$|V|\,|\,\mathrm{SC}\,|\Big(1 + \sum_{v\in V}|\tau(v)|\Big) \tag{2}$$

[2] Since obviously $\max(|\,\mathrm{SC}^+\,|, |\,\mathrm{SC}^-\,|) \leqslant |\,\mathrm{SC}\,|$.

and thus the total number of symbols in χ of the form $\underline{\omega^{\cdot}}$ for any ω is bounded by:

$$|V||\operatorname{SC}|^2\Big(1+\sum_{v\in V}|\tau(v)|\Big). \tag{3}$$

Furthermore, inbetween symbols of the form $\underline{\omega^{\cdot}}$, there can be subpaths consisting of symbols in E only, so χ is of the form

$$\chi = \phi_0\underline{\omega_1^{k_1}}\phi_1\underline{\omega_2^{k_2}}\cdots\underline{\omega_n^{k_n}}\phi_n$$

where $\phi_i \in E^*$ and $\omega_i \in \operatorname{SC}$ for all i. By the reasoning above, $n \leqslant (3)$. Furthermore, by Lemma 4, $\phi_i < |V|$ for all i. It follows that the total length of χ is bounded by

$$\begin{aligned}(n+1)(|V|-1)+n &\leqslant |V|+n|V| \\ &\leqslant |V|+|V|^2|\operatorname{SC}|^2\Big(1+\sum_{v\in V}|\tau(v)|\Big).\end{aligned}$$

Finally the length of $\operatorname{unfold}(\chi(c))$ at most that of π because the rewriting system does not increase the length of the path and the length of $\operatorname{unfold}(\chi_0(c))$ is equal to that of π. ◀

The main result of this section shows that any valid computation has an equivalent valid computation given by a folded path whose length only depends on the automaton.

▶ **Theorem 8** (Length of computations). *Let π be a valid finite computation from (v,c) to (v',c'). Then there exists a folded path χ such that $\chi(c)$ is a valid computation from (v,c) to (v',c'), the length of $\operatorname{unfold}(\chi(c))$ is at most that of π and the word length of χ is bounded by:*

$$|E|\left(1+|V|+|V|^2|\operatorname{SC}|^2\Big(1+\sum_{v\in V}|\tau(v)|\Big)\right).$$

Proof. Apply Lemma 1 to isolate the equality tests (at most $|E|$ of them) and apply Lemma 7 to each equality-free subcomputation. We can improve the bound slightly by noticing that there can only be up to $|E|$ equality-free subcomputations (and not $|E|+1$). Indeed, if there are $|E|$ different equality tests in the path, there are no further edges available for equality-free computations, and the word length is at most $|E|$. ◀

4 Reachability with Parameterised Tests

In this section we will show that both the reachability problem and the generalised repeated reachability problem for 1-CA with parameterised tests are decidable, via a symbolic encoding of folded paths, making use of the normal form from the previous section. The result of this encoding is a formula of Presburger arithmetic.

Recall that $C = \{\underline{\omega^k} : \omega \in \operatorname{SC}, k \in \mathbb{N}\}$. Let $C' = \{\underline{\omega^{\cdot}} : \omega \in \operatorname{SC}\}$. We define a *path shape* to be a word over the alphabet $E \cup C'$: $\xi = t_1 \dots t_n$ such that $\operatorname{end}(t_i) = \operatorname{start}(t_{i+1})$, where $\operatorname{start}(\underline{\omega^{\cdot}}) = \operatorname{end}(\underline{\omega^{\cdot}}) = \operatorname{start}(\omega)$. Given a path shape $\xi = \gamma_0\underline{\omega_1^{\cdot}}\gamma_1\dots\underline{\omega_n^{\cdot}}\gamma_n$ with $\gamma_i \in E^*$, we write $\xi(k_1,\dots,k_n)$ for the folded path $\gamma_0\underline{\omega_1^{k_1}}\gamma_1\dots\underline{\omega_n^{k_n}}\gamma_n$. The advantage of working with path shapes rather than folded paths is that the former are words over a finite alphabet.

▶ **Lemma 9** (Encoding computations). *Given a 1-CA $\mathcal{C} = (V,E,X,\lambda,\tau)$ with parameterised tests and configurations (v,c) and (v',c'), and given a path shape $\xi = t_1t_2\dots t_n \in (E \cup C')^*$, there exists a Presburger arithmetic formula $\varphi_{comp}^{(\xi),(v,c),(v',c')}(\boldsymbol{k},\boldsymbol{x})$, with free variables $\boldsymbol{x}$ corresponding to the parameters X and $\boldsymbol{k}$ corresponding to exponents to be substituted in ξ, which evaluates to true if and only if $\operatorname{unfold}(\xi(\boldsymbol{k}))(c)$ is a valid computation from (v,c) to (v',c').*

Proof. Assume first that ξ does not include any equality tests. We define a formula $\varphi^{(t)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, y)$ which, given an equality-free symbol $t \in E \cup C'$ and an integer y, evaluates to true if and only if $\mathrm{unfold}(t(\boldsymbol{k}))(y)$ is a valid computation. There are two cases:

- $t \in E$. Then $\varphi^{(t)}_{valid,noeq}(\boldsymbol{x}, y) \equiv y \geqslant 0 \wedge y + \mathrm{weight}(t) \geqslant 0 \wedge y \notin \tau(\mathrm{start}(t))$.
- $t \in C'$, i.e., $t(\boldsymbol{k}) = \underline{\omega^k}$ for some simple cycle $\omega = e_1 e_2 \dots e_\ell$ and $k \in \boldsymbol{k}$. Then

$$\varphi^{(t)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, y) \equiv \forall k' \, (0 \leqslant k' < k) \Rightarrow \bigwedge_{i=1}^{\ell} \left(y + k' \,\mathrm{weight}(\omega) + \sum_{j=1}^{i-1} \mathrm{weight}(e_j) \geqslant 0 \wedge \right.$$
$$\left. y + k' \,\mathrm{weight}(\omega) + \sum_{j=1}^{i-1} \mathrm{weight}(e_j) \notin \tau(\mathrm{start}(e_i)) \right) \wedge y + k \,\mathrm{weight}(\omega) \geqslant 0.$$

Note that for each edge $e \in E$, $\mathrm{weight}(e)$ is a constant, given by the automaton, and $\mathrm{weight}(\omega)$ is a shorthand for $\sum_{i=1}^{\ell} \mathrm{weight}(e_i)$, which is also a constant. So the only type of multiplication in the formula is by a constant. A formula of the form $a \notin \tau(u)$ is a shorthand for $\bigwedge_{b \in \tau(u)} a \neq b$, which is clearly a Presburger arithmetic formula. Since $\mathcal{C}$ has parameterised tests, in general some of these disequalities include variables from $\boldsymbol{x}$. We can now define a formula with the required property in the case where ξ does not include any equality tests:

$$\varphi^{(\xi),(v,c),(v',c')}_{comp,noeq}(\boldsymbol{k}, \boldsymbol{x}) \equiv \left(\bigwedge_{i=1}^{n-1} \mathrm{end}(t_i) = \mathrm{start}(t_{i+1}) \right) \wedge \mathrm{start}(t_1) = v \wedge \mathrm{end}(t_n) = v' \wedge$$
$$\sum_{i=1}^{n} \mathrm{weight}(t_i(\boldsymbol{k})) = c' - c \wedge \bigwedge_{i=1}^{n} \varphi^{(t_i)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, c + \sum_{j=1}^{i-1} \mathrm{weight}(t_j(\boldsymbol{k}))),$$

where we use the shorthand $\mathrm{weight}(s)$ for $s \in E \cup C$: if $s \in E$ then $\mathrm{weight}(s)$ is a constant as above, and if $s \in C$ then it is of the form $\underline{\omega^k}$ and $\mathrm{weight}(s) = k \sum_{e \in \omega} \mathrm{weight}(e)$. Again, the only multiplications are by constants, so the resulting formula is a formula of Presburger arithmetic.

Finally, in the case where ξ includes equality tests, we split $\mathrm{unfold}(\xi)$ at the t_i which are equality tests, and construct a formula $\varphi_{comp,noeq}$ as above for each equality-free part of ξ. $\varphi^{(\xi),(v,c),(v',c')}_{comp}(\boldsymbol{k}, \boldsymbol{x})$ is the conjunction of these formulas. ◀

▶ Remark 10 (Removing the universal quantification). For simplicity, we have used a universal quantifier in $\varphi^{(t)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, y)$ to express that k iterations of a cycle yield a valid computation. In fact it is possible to rewrite $\varphi^{(t)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, y)$ as a purely existential formula, with a polynomial blowup. Let $\omega = e_1 \cdots e_\ell$ be a cycle and suppose we want to check that $\omega^k(y)$ is a valid computation. Let $u = \mathrm{start}(e_i)$ be a state on the cycle. First we need to express that the counter value at u is never negative along $\omega^k(y)$. Since the counter value at u is monotone during the k iterations of the cycle (it increases if ω is positive and decreases if ω is negative), we only need check that it is nonnegative at the first and last iteration:

$$y + \sum_{j=1}^{i-1} \mathrm{weight}(e_j) \geqslant 0 \wedge y + (k-1) \,\mathrm{weight}(\omega) + \sum_{j=1}^{i-1} \mathrm{weight}(e_j) \geqslant 0.$$

Next, for each $b \in \tau(u)$, we need to check that the cycle avoids b in u. Without loss of generality, assume that ω is positive. Then the counter value at u increases after each iteration. We can now perform a case analysis on the three ways to satisfy a disequality test during the k iterations of ω:

- The value at the first iteration is already bigger than b:

$$y + \sum_{j=1}^{i-1} \text{weight}(e_j) > b.$$

- The value at the last iteration is less than b:

$$y + (k-1)\,\text{weight}(\omega) + \sum_{j=1}^{i-1} \text{weight}(e_j) < b.$$

- There is an iteration k', with $0 \leqslant k' < k-1$, at which the counter value is less than b, but where at the next iteration $k'+1$ the counter value is bigger than b:

$$\exists k' \, (0 \leqslant k' < k-1) \wedge y + k'\,\text{weight}(\omega) + \sum_{j=1}^{i-1} \text{weight}(e_j) < b \\ \wedge\, y + (k'+1)\,\text{weight}(\omega) + \sum_{j=1}^{i-1} \text{weight}(e_j) > b.$$

Finally, we can use a conjunction over all vertices in ω to get a formula which is equivalent to $\varphi^{(t)}_{valid,noeq}(\boldsymbol{k}, \boldsymbol{x}, y)$ but has no universal quantifiers.

▶ **Lemma 11** (Encoding reachability). *Let $\mathcal{C} = (V, E, X, \lambda, \tau)$ be a 1-CA with parameterised tests, and let (v, c) and (v', c') be given configurations of $\mathcal{C}$. Then there exists a Presburger arithmetic formula $\varphi^{(v,c),(v',c')}_{reach}(\boldsymbol{x})$ which evaluates to true if and only if there is a valid computation from (v, c) to (v', c') in $\mathcal{C}$, as well as a formula $\varphi^{(v,c),(v',c')}_{reach_+}$ which is true if and only if there is such a computation of length at least 1.*

Proof. Note that the bounds on the length of computations in 1-CA from the previous section do not depend on the values occurring in equality or disequality tests. That is, if there is a valid computation $(v, c) \xrightarrow{\pi}{}^{*} (v', c')$ for any given values of the parameters, then there is a folded path χ of word length at most $p(\mathcal{C})$ such that $(v, c) \xrightarrow{\text{unfold}(\chi(c))}{}^{*} (v', c')$ is a valid computation, where p is the polynomial function given in Theorem 8. Equivalently, there is a path shape ξ of word length at most $p(\mathcal{C})$ and there exist values $\boldsymbol{k}$ such that $(v, c) \xrightarrow{\text{unfold}(\xi(\boldsymbol{k})(c))}{}^{*} (v', c')$ is a valid computation.

Since path shapes are words over a finite alphabet, we can express this property as a finite disjunction

$$\varphi^{(v,c),(v',c')}_{reach}(\boldsymbol{x}) \equiv \exists \boldsymbol{k} \bigvee_{|\xi| \leqslant p(\mathcal{C})} \varphi^{(\xi),(v,c),(v',c')}_{comp}(\boldsymbol{k}, \boldsymbol{x}).$$

For φ_{reach_+}, we simply change the disjunction to be over all ξ such that $1 \leqslant |\xi| \leqslant p(\mathcal{C})$. ◀

▶ **Lemma 12** (Encoding repeated reachability). *Let $\mathcal{C} = (V, E, X, \lambda, \tau)$ be a 1-CA with parameterised tests, let $F \subseteq V$ be a set of final states, and let (v, c) be the initial configuration of $\mathcal{C}$. Then there exists a Presburger arithmetic formula $\varphi^{(v,c),(F)}_{rep\text{-}reach}(\boldsymbol{x})$ which evaluates to true if and only if there is a valid infinite computation π which starts in (v, c) and visits at least one state in F infinitely often.*

Proof. Suppose there is an infinite computation which starts in (v, c) and visits a state $u \in F$ infinitely often. Equivalently, there is a counter value $d \in \mathbb{N}$ such that $(v, c) \longrightarrow^* (u, d)$ is a valid (finite) computation, and there is a cycle ω with $\mathrm{start}(\omega) = u$ such that $\omega^k(d)$ is a valid computation for all $k \in \mathbb{N}$. There are two possible cases:

- $\mathrm{weight}(\omega) = 0$, so $\omega^k(d)$ is valid for all k if and only if $\omega(d)$ is valid.
- $\mathrm{weight}(\omega) > 0$, so it might be possible to start from (u, d) and follow the edges of ω a finite number of times before an obstruction occurs. However, if ω can be taken an arbitrary number of times, then the counter value will tend towards infinity, so we are free to choose ω to be an equality-free simple cycle, and d to be high enough to guarantee that if ω can be taken once without obstructions, it can be taken infinitely many times.

The resulting formula is then

$$\varphi^{(v,c),(F)}_{\textit{rep-reach}}(\boldsymbol{x}) \equiv \exists d \bigvee_{u \in F} \Bigg(\varphi^{(v,c),(u,d)}_{\textit{reach}}(\boldsymbol{x}) \wedge \Big(\varphi^{(u,d),(u,d)}_{\textit{reach}_+}(\boldsymbol{x}) \vee (d > M(\boldsymbol{x}) \wedge \exists d' \bigvee_{\omega \in \mathrm{SC}^+} \varphi^{(\underline{\omega}),(u,d),(u,d')}_{\textit{comp,noeq}}(1, \boldsymbol{x})) \Big) \Bigg)$$

where $M(\boldsymbol{x}) = \max\left(\bigcup_{v \in V} \tau(v)\right) - \sum\{\mathrm{weight}(e) : e \in E, \mathrm{weight}(e) < 0\}$. The sum over negative edge weights ensures that the counter always stays above $\max\left(\bigcup_{v \in V} \tau(v)\right)$ along the computation $\omega(d)$, since each edge is taken at most once in ω. Since ω is a positive cycle, this implies that the counter always stays above all bad values along $\omega(d^k)$ for each $k \in \mathbb{N}$, so no obstructions can occur. ◀

▶ **Theorem 13** (Decidability of reachability problems). *Both the reachability problem and the generalised repeated reachability problem are decidable for 1-CA with parameterised tests.*

Proof. Given a 1-CA $\mathcal{C} = (V, E, X, \lambda, \tau)$ with parameterised tests and configurations (v, c) and (v', c'), to check if there exist values for the parameters X such that there is a valid computation from (v, c) to (v', c'), we use Lemma 11 to construct the formula $\exists \boldsymbol{x}\, \varphi^{(v,c),(v',c')}_{\textit{reach}}(\boldsymbol{x})$.

To solve the generalised repeated reachability problem for a 1-CA $\mathcal{C} = (V, E, X, \lambda, \tau)$ with sets of final states $F_1, \ldots, F_n \subseteq V$ and initial configuration (v, c), note that this problem can easily be reduced to the simpler case where $n = 1$, using a translation similar to the standard translation from generalised Büchi automata to Büchi automata. In the case where $n = 1$, we can use Lemma 12 to construct the formula $\exists \boldsymbol{x}\, \varphi^{(v,c),(F_1)}_{\textit{rep-reach}}(\boldsymbol{x})$. ◀

▶ **Corollary 14** (Decidability of model checking flat Freeze LTL). *The existential model checking problem for flat Freeze LTL on 1-CA is decidable.*

5 Conclusion

The main result of this paper is that the model checking problem for the flat fragment of Freeze LTL on one-counter automata is decidable. We have concentrated on showing decidability rather than achieving optimal complexity. For example, we have reduced the model checking problem to the decision problem for the class of sentences of Presburger arithmetic with quantifier prefix $\exists^*\forall^*$. We explained in Remark 10 that in fact the reduction can be refined to yield a (polynomially larger) purely existential sentence.

Another important determinant of the complexity of our procedure is the dependence of the symbolic encoding of computations (via path shapes) in Section 4 on the number of

simple cycles in the underlying control graph of the one-counter automaton. The number of such cycles may be exponential in the number of vertices. It remains to be seen whether it is possible to give a more compact symbolic representation, e.g., in terms of the Parikh image of paths. As it stands, our procedure works as follows. From the flat $\mathrm{LTL}^{\downarrow}$ formula, we build a 1-CA with parameterised tests (of exponential size). We then guess the normal form of the path shapes (of exponential size in the size the automaton). We finally check the resulting existential Presburger formula. Since the Presburger formula has size double exponential in the size of the original $\mathrm{LTL}^{\downarrow}$ formula, we get a naive upper bound of 2NEXPTIME for our algorithm. Improving this bound is a subject of ongoing work.

Another interesting complexity question concerns configuration reachability in one-counter automata with non-parameterised equality and disequality tests. For automata with only equality tests and with counter updates in binary, reachability is known to be NP-complete [9]. If inequality tests are allowed then reachability is PSPACE-complete [7]. Now automata with equality and disequality tests are intermediate in expressiveness between these two models and the complexity of reachability in this case is open as far as we know.

References

1 P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. On expressiveness and complexity in real-time model checking. In *Proceedings of ICALP*, volume 5126 of *LNCS*, pages 124–135. Springer, 2008.

2 H. Comon and V. Cortier. Flatness is not a weakness. In *Proceedings of CSL*, volume 1862 of *LNCS*. Springer, 2000.

3 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. In *Proceedings of LICS*, pages 17–26. IEEE Computer Society, 2006.

4 S. Demri, R. Lazić, and D. Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. In *Proceedings of TIME*, pages 113–121, 2005.

5 S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *Proceedings of FOSSACS*, volume 4962 of *LNCS*, pages 490–504, 2008.

6 S. Demri and A. Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In *Proceedings of FOSSACS*, volume 6014 of *LNCS*, pages 176–190, 2010.

7 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015.

8 T. French. Quantified propositional temporal logic with repeating states. In *Proceedings of TIME-ICTL*, pages 155–165. IEEE Computer Society, 2003.

9 C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of CONCUR*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

10 O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines and applications. In *Proceedings of ICALP*, volume 700 of *LNCS*. Springer, 1993.

11 A. Lisitsa and I. Potapov. Temporal logic with predicate lambda-abstraction. In *Proceedings of TIME*, pages 147–155. IEEE Computer Society, 2005.

12 M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congrés de Mathématiciens des Pays Slaves. Warsaw*, pages 92–101, 1929.

Parameterized Systems in BIP: Design and Model Checking*

Igor Konnov[1], Tomer Kotek[2], Qiang Wang[3], Helmut Veith[4], Simon Bliudze[5], and Joseph Sifakis[6]

1 **TU Wien (Vienna University of Technology), Austria**
Konnov@forsyte.at
2 **TU Wien (Vienna University of Technology), Austria**
Kotek@forsyte.at
3 **École polytechnique fédérale de Lausanne, Switzerland**
Qiang.Wang@epfl.ch
4 **TU Wien (Vienna University of Technology), Austria**
Veith@forsyte.at
5 **École polytechnique fédérale de Lausanne, Switzerland**
Simon.Bliudze@epfl.ch
6 **École polytechnique fédérale de Lausanne, Switzerland**
Joseph.Sifakis@epfl.ch

Abstract

BIP is a component-based framework for system design built on three pillars: behavior, interaction, and priority. In this paper, we introduce first-order interaction logic (FOIL) that extends BIP without priorities to systems parameterized in the number of components. We show that FOIL captures classical parameterized architectures such as token-passing rings, cliques of identical components communicating with rendezvous or broadcast, and client-server systems.

Although the BIP framework includes efficient verification tools for statically-defined systems, none are available for parameterized systems with an unbounded number of components. On the other hand, the parameterized model checking literature contains a wealth of techniques for systems of classical architectures. However, application of these results requires a deep understanding of parameterized model checking techniques and their underlying mathematical models. To overcome these difficulties, we introduce a framework that automatically identifies parameterized model checking techniques applicable to a BIP design. To our knowledge, this is the first framework that allows one to apply prominent parameterized model checking results in a systematic way.

1998 ACM Subject Classification [Software Engineering] D.2.2: Design Tools and Techniques, D.2.4 Software/Program Verification

Keywords and phrases Rigorous system design, BIP, verification, parameterized model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.30

* We dedicate this article to the memory of Helmut Veith, who passed away tragically while this manuscript was being prepared. His curiosity and energy ignited our joint effort in this research.
This work was supported by the Austrian National Research Network S11403-N23 (RiSE), the Vienna Science and Technology Fund (WWTF) through the grant APALACHE (ICT15-103), and, partially, by the Swiss National Science Foundation through the National Research Programme "Energy Turnaround" (NRP 70) grant 153997.

1 Introduction

Design, manufacture and verification of large scale complex hardware/software systems (e.g., cyber-physical systems) remains a grand challenge in system design automation [25]. To address this challenge, the rigorous system design methodology [24] and the behaviour-interaction-priority (BIP) framework [4] have been recently proposed. BIP comes with a formal framework and a toolchain. The BIP framework has well-defined semantics for modeling system behavior and architectures. The BIP toolchain supports verification of high-level system designs and automatic system synthesis of low-level implementations from high-level system designs.

The existing BIP tools focus on design and verification of systems with a fixed number of communicating components [5, 22]. However, many distributed systems are designed with parameterization in mind. For instance, the number of components in the system is not typically fixed, but varies depending on the system setup. In this case, one talks about parameterized verification, where the number of components is a parameter.

Model checking is a pragmatic approach to verification that has found many applications in industry, e.g., see [19]. Many efforts were invested into extension of model checking to the parameterized case, which led to numerous parameterized model checking techniques (see [9] for a recent survey). Unfortunately, often parameterized model checking techniques come with their own mathematical models, which makes their practical application difficult. To perform parameterized model checking, the user has to thoroughly understand the research literature. Typically, the user needs to first manually inspect the parameterized models and match them with the mathematical formalisms from the relevant parameterized verification techniques. Using the match, the user would then apply the decidability results (if any) for the parameterized models, e.g., by computing a cutoff or translating the parameterized model into the language of a particular tool for the specific architecture. Thus, there is a gap between the mathematical formalisms and algorithms from the parameterized verification research and the practice of parameterized verification, which is usually done by engineers who are not familiar with the details of the research literature. In this paper, we aim at closing this gap by introducing a framework for design and verification of parameterized systems in BIP. With this framework, we make the following contributions:

1. We extend propositional interaction logic to the parameterized case with arithmetics, which we call *first-order interaction logic* (FOIL). We build on the ideas from configuration logic [21] and dynamic BIP [10]. FOIL is powerful enough to express architectures found in distributed systems, including the classical architectures: token-passing rings, rendezvous cliques, broadcast cliques, and rendezvous stars. We also identify a decidable fragment of FOIL which has important applications in practice. This contribution is covered by Section 3.
2. We provide a framework for integration of mathematical models from the parameterized model checking literature in an automated way: given a parameterized BIP design, our framework detects parameterized model checking techniques applicable to this design. This automation is achieved by the use of SMT solvers and standard (non-parameterized) model checkers. This contribution is covered by Sections 4 and 5.
3. We provide a preliminary prototype implementation of the proposed framework. Our prototype tool takes a parameterized BIP design as its input and detects whether one of the following classical results applies to this BIP design: the cut-off results for token-passing rings by Emerson & Namjoshi [16], the VASS-based algorithms by German & Sistla [18], and the undecidability and decidability results for broadcast systems by Abdulla et al. [1]

and Esparza et al. [17]. More importantly, our framework is not specifically tailored to the mentioned techniques. This contribution is covered by Sections 5 and 6.

We remark that our framework builds on the notions of BIP, which allows us to express complex notions in terminology understood by engineers. Moreover, our framework allows an expert in parameterized model checking to capture seminal mathematical models found in the verification literature, e.g., [18, 17, 16, 13].

This paper is structured as follows. In Section 2, we briefly recall the BIP modeling framework. In Section 3, we introduce our parameterized extension. In Sections 4 and 5, we present our verification framework and the automatic system architecture identification technique. In Section 6, we present the preliminary experiments. Section 7 closes with related work, conclusions, and future work.

2 BIP without priorities

In this section, we review the notions of BIP [4] with the following restrictions: (i) states of the components do not have specific internal structure; (ii) we do not consider interaction priorities. While we believe that our approach can be extended to priorities, we leave this for future work.

As usual, a labeled transition system is a tuple (S, s_0, A, R) with a set of locations S, an initial location $s_0 \in S$, a non-empty set of actions A, and a transition relation $R \subseteq S \times A \times S$.

▶ **Definition 2.1** (Component type)**.** A component type is a transition system $\mathbb{B} = \langle \mathbb{Q}, \ell^0, \mathbb{P}, \mathbb{E} \rangle$ over the finite sets $\mathbb{Q}$ and $\mathbb{P}$. By convention, the set of actions $\mathbb{P}$ is called the set of ports.

Ports form the interface of a component type. We assume that, for each location, no two outgoing transitions from this location are labeled with the same port. We also assume that the ports of each component type, as well as the locations, are disjoint.

Let $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$ be a tuple of component types, where each $\mathbb{B}_i$ is $\langle \mathbb{Q}_i, \ell_i^0, \mathbb{P}_i, \mathbb{E}_i \rangle$ for $i \in [0, k)$. We introduce an infinite set of components $\{\mathbb{B}_i[j] \mid j \geq 0\}$ for $i \in [0, k)$. A *component* $\mathbb{B}_i[j] = \langle \mathbb{Q}_i[j], \ell_i^0[j], \mathbb{P}_i[j], \mathbb{E}_i[j] \rangle$ is obtained from the component type $\mathbb{B}_i$ by renaming the set of ports. Thus, as transition systems, $\mathbb{B}_i[j]$ and $\mathbb{B}_i$ are isomorphic. We postulate $\mathbb{P}_i[j] \cap \mathbb{P}_i[j'] = \emptyset$, for $j \neq j'$.

A BIP model is a composition of finitely many components instantiated from the component types $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$. To denote the number of components of each type, we introduce a size vector $\bar{N} = \langle N_0, \ldots, N_{k-1} \rangle$: there are N_i components of component type $\mathbb{B}_i$, for $i \in [0, k)$.

Coordination of components is specified with interactions. Intuitively, an interaction defines a multi-party synchronization of component transitions. A BIP interaction is a finite set of ports, which defines a possible synchronization among components.

▶ **Definition 2.2** (Interaction)**.** Given a tuple of component types $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$ and a size vector $\bar{N} = \langle N_0, \ldots, N_{k-1} \rangle$, an interaction $\gamma \subseteq \{p \in \mathbb{P}_i[j] \mid i \in [0, k), j \in [0, N_i)\}$ is a set of ports such that $|\gamma \cap \mathbb{P}_i[j]| \leq 1$ for all $i \in [0, k)$ and $j \in [0, N_i)$, i.e., an interaction is a set of ports such that at most one port of each component takes part in an interaction. If $p \in \gamma$, we say that p is *active* in γ.

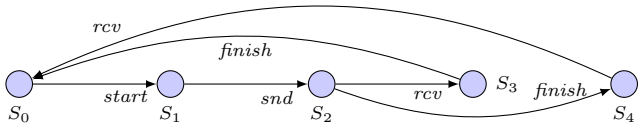
▶ **Definition 2.3** (BIP Model)**.** Given a tuple of component types $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$ and a size vector $\bar{N} = \langle N_0, \ldots, N_{k-1} \rangle$, a BIP model $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma}$ is a tuple $\langle \mathcal{B}, \Gamma \rangle$, where $\mathcal{B}$ is the set $\{\mathbb{B}_i[j] \mid i \in [0, k), j \in [0, N_i)\}$ and Γ is a set of interactions defined w.r.t. $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$ and $\bar{N}$.

▶ **Definition 2.4** (BIP operational semantics)**.** Given a BIP model $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma}$, we define its operational semantics as a transition system $TS(\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma}) = \langle S, s_0, \Gamma, R \rangle$, where:

1. The set of *configurations* S is defined as the Cartesian product of the sets of locations of the components $\mathbb{Q}_0^{N_0} \times \cdots \times \mathbb{Q}_{k-1}^{N_{k-1}}$. Given a configuration $s \in S$, we denote by $s(i,j)$ the j^{th} member of the tuple defined by the i^{th} product $\mathbb{Q}_i^{N_i}$ where $j \in [0, N_i)$.
2. The initial configuration $s_0 \in S$ satisfies that $s_0(i,j) = \ell_i^0[j]$ for all $i \in [0,k)$ and $j \in [0, N_i)$.
3. The transition relation R contains a triple (s, γ, s'), if, for each $i \in [0,k)$ and $j \in [0, N_i)$, the j^{th} component of type i
   - either has an active port $p \in \gamma \cap \mathbb{P}_i[j]$ and $\langle s(i,j), p, s'(i,j) \rangle \in \mathbb{E}_i[j]$,
   - or is not participating in the interaction γ, i.e., $\gamma \cap \mathbb{P}_i[j] = \emptyset$ and $s'(i,j) = s(i,j)$.

Intuitively, the local transitions of components fire simultaneously, provided that their ports are included in the interaction; other components do not move.

▶ **Example 2.5** (Milner's scheduler)**.** We follow the formulation by Emerson & Namjoshi [16]. A scheduler is modeled as a token-passing ring. Only the process that owns the token may start running a new task. The component type $\mathbb{B}_0 = \langle \mathbb{Q}_0, \ell_0^0, \mathbb{P}_0, \mathbb{E}_0 \rangle$ is given by the locations $\mathbb{Q}_0 = \{S_0, \ldots, S_4\}$, the initial location $\ell_0^0 = S_0$, the port types $\mathbb{P}_0 = \{\text{snd}, \text{rcv}, \text{start}, \text{finish}\}$, and the edges $\mathbb{E}_0$ that are shown in the figure below:



A component owns the token when in the location S_0, S_1, or S_3. In S_0, a component initiates its task by interacting on port *start*. The token is then sent to the component's right neighbor on the ring via an interaction on port *snd*. The component then waits until (a) its initiated task has finished, and (b) the component has received the token again. When both (a) and (b) have occurred, the component may initiate a new task. Note that (a) and (b) may occur in either order.

Fix a number $N_0 \in \mathbb{N}$. The following set of interactions represents the ring structure:

$$\Gamma = \{\gamma_{i \to j}, \gamma_{\text{start}(i)}, \gamma_{\text{finish}(i)} \mid 0 \leq i < N_0 \text{ and } j \equiv i + 1 \bmod n_0\}$$

where $\gamma_{i \to j} = \{(\text{snd}, i), (\text{rcv}, j)\}$ is the interaction passing the token from the i^{th} component to the next component on the ring, while the interactions $\gamma_{\text{start}(i)} = \{(\text{start}, i)\}$ and $\gamma_{\text{finish}(i)} = \{(\text{finish}, i)\}$ allow the i^{th} component to take the internal transitions labeled 'start' and 'finish' respectively. The BIP model of the Milner scheduler of size N_0 is $\langle \mathcal{B}, \Gamma \rangle$, where $\mathcal{B}$ is the set of components $\{\mathbb{B}_0[j] \mid j \in [0, N_0)\}$.

3 Parameterized BIP without priorities

Since the number of possible interactions in a parameterized system is unbounded, and each interaction itself may involve an unbounded number of actions, the set of all possible interactions is infinite. Hence, we need a symbolic representation of such a set. To this end, we propose *first order interaction logic*—a uniform and formal language for system topologies and coordination mechanisms in parameterized systems. Using this logic, we introduce a parameterized extension of BIP, and show that this extension naturally captures standard examples.

3.1 FOIL: First order interaction logic

In this section, we fix a tuple of component types $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$. For each port $p \in \mathbb{P}_i$ of an i^{th} component type, we introduce a unary *port predicate* with the same name p. Furthermore, we introduce a tuple of constants $\bar{n} = \langle n_0, \ldots, n_{k-1} \rangle$, which represents the number of components of each type. We also assume the standard vocabulary of Presburger arithmetic, that is, $\langle 0, 1, \leq, + \rangle$.

FOIL syntax. Assume an infinite set of index variables $\mathcal{I}$. We say that ψ is a first order interaction logic formula, if it is constructed according to the following grammar:

$$\psi ::= p(i) \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \exists i :: type_j : \phi.\ \psi \mid \forall i :: type_j : \phi.\ \psi\,,$$

where $p \in \mathbb{P}_0 \cup \cdots \cup \mathbb{P}_{k-1}$, $i \in \mathcal{I}$, and ϕ is a formula in Presburger arithmetic over index variables and the vocabulary $\langle 0, 1, \leq, +, \bar{n} \rangle$.

Informally, the syntax $\mathbf{Q}\ i :: type_j : \phi.\ \psi$, where $\mathbf{Q} \in \{\exists, \forall\}$, restricts the index variable i to be associated with the component type $\mathbb{B}_j$. Notice, however, that this syntax does not enforce type correctness of ports. For instance, one can write a formula $\exists i :: type_j : p(i)$ with some $p \notin \mathbb{P}_j$. While this formula is syntactically correct, it is not in line with Definition 2.2 of interaction given in Section 2. To this end, we say that a FOIL formula is *natural*, if for each of its subformulae $\mathbf{Q}\ i :: type_j : \phi.\ \psi(i)$, for $\mathbf{Q} \in \{\exists, \forall\}$, and every atomic formula $p(i)$ of ψ, it holds that $p \in \mathbb{P}_j$. From here on, we assume FOIL formulae to be natural. We write $\exists i :: type_j.\ \psi$ as a shorthand for $\exists i :: type_j : true.\ \psi$.

FOIL semantics. We give the semantics of a FOIL formula by means of structures. A *first-order interaction logic structure* (FOIL structure) is a pair $\xi = (\mathbb{N}, \alpha_\xi)$: the set of natural numbers $\mathbb{N}$ is the domain of ξ, while α_ξ is the interpretation of all the predicates and of the constants $\bar{n}$. The symbols 0, 1, $\leq$, and $+$ have the natural interpretations over $\mathbb{N}$.

A valuation σ is a function $\sigma : \mathcal{I} \to \mathbb{N}$. We denote by $\sigma[x \mapsto j]$ the valuation obtained from σ by mapping the index variable x to the value j. Assignments are used to give values to free variables in formulae. For a FOIL structure ξ and a valuation σ, the semantics of FOIL is formally given as follows (the semantics of Boolean operators and universal quantifiers is defined in the standard way):

$$\begin{array}{lll} \xi, \sigma \models_{\text{FOIL}} p(i) & \textit{iff} & \alpha_\xi(p) \text{ is true on } \sigma(i) \\ \xi, \sigma \models_{\text{FOIL}} \exists i :: type_j : \phi.\ \psi & \textit{iff} & \text{there is } l \in [0, \alpha_\xi(n_j)) \text{ such that} \\ & & \xi, \sigma[i \mapsto l] \models_{\text{FO}} \phi \text{ and } \xi, \sigma[i \mapsto l] \models_{\text{FOIL}} \psi \end{array}$$

where $\models_{\text{FO}}$ to denotes the standard 'models' relation of first-order logic.

Finally, for a FOIL formula ψ without free variables and a structure ξ, we write $\xi \models_{\text{FOIL}} \psi$, if $\xi, \sigma_0 \models_{\text{FOIL}} \psi$ for the valuation σ_0 that assigns 0 to every index $i \in \mathcal{I}$.[1]

Decidability. It is easy to show that checking validity of a FOIL sentence[2] is undecidable, and that FOIL contains an important decidable fragment:

▶ **Theorem 3.1** (Decidability of FOIL). *The following results about FOIL hold:*

[1] Since ψ has no free variables, our choice of σ_0 is arbitrary: for all σ we have $\xi, \sigma \models_{\text{FOIL}} \psi$ if and only if $\xi, \sigma_0 \models_{\text{FOIL}} \psi$.

[2] A FOIL formula with no free variables is called a *sentence*. A sentence is *valid* if it is satisfied by all structures.

(i) *Validity of FOIL sentences is undecidable.*
(ii) *Validity of FOIL sentences in which all additions are of the form $i + 1$ is decidable.*

Proof. (i) FOIL contains Presburger arithmetic with unary predicates, which is known to be as strong as Peano arithmetic [20]. Hence, satisfiability and validity of FOIL formulae are undecidable.

(ii) The formula $j = i + 1$ is definable in FOIL by $i \leq j \wedge j \neq i \wedge \psi_{consecutive}(i, j)$, where $\psi_{consecutive}(i, j) = \forall \ell :: type_t.\ (j \leq \ell \wedge \ell \leq i) \rightarrow (\ell = i \vee \ell = j)$, where t is the type of i and j. Hence, we can rewrite any FOIL sentence ψ in which all additions are of the form $i + 1$ as an equi-satisfiable first-order logic sentence ψ' without using addition $(+)$. The sentence ψ' belongs to S1S, the monadic second order theory of $(\mathbb{N}, 0, 1, \leq)$, which is decidable, see [27]. ◀

In the following, we restrict addition to the form $i + 1$, and thus stay in the decidable fragment.

3.2 Interactions as FOIL structures

In contrast to Definition 2.2 of a standard interaction, which is represented explicitly as a finite set of ports, we use first order interaction logic formulae to define all the possible interactions in parameterized systems. Our key insight is that each structure of a formula uniquely defines at most one interaction, and the set of all possible interactions is the union of the interactions derived from the structures that satisfy the formula.

Intuitively, if $p(j)$ evaluates to true in a structure ξ, then the j^{th} instance of the respective component type—uniquely identified by the port p—takes part in the interaction identified with ξ. Thus, we can reconstruct a standard BIP interaction from a FOIL structure by taking the set of ports, whose indices are evaluated to true by the unary predicates. Formally, given a FOIL structure $\xi = (\mathbb{N}, \alpha_\xi)$, we define the set $\gamma_\xi = \{(p, j) \mid i \in [0, k),\ p \in \mathbb{P}_i,\ j \in [0, \alpha_\xi(n_j)),\ \alpha_\xi(p)(j) = true\}$. In the following, the notation (p, j) denotes the port p of the j^{th} component of the type $\mathbb{B}_i$ with $p \in \mathbb{P}_i$.

Notice that γ_ξ does not have to be an interaction in the sense of Definition 2.2. Indeed, one can define ξ whose set γ_ξ includes two ports of the same component. We say that ξ *induces an interaction*, if γ_ξ is an interaction in the sense of Definition 2.2.

▶ **Definition 3.2** (Parameterized BIP Model)**.** A parameterized BIP model is a tuple $\langle \mathbb{B}, \bar{n}, \psi, \epsilon \rangle$, where $\mathbb{B} = \langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle$ is a tuple of component types, ψ is a sentence in FOIL over port predicates and a tuple $\bar{n} = \langle n_0, \ldots, n_{k-1} \rangle$ of size parameters, and ϵ is a linear constraint over $\bar{n}$.

The tuple $\bar{n}$ consists of the size parameters for all component types, and the constraint ϵ restricts these parameters. For example, the formula $(n_0 = 1) \wedge (n_1 \geq 10)$ requires every instance of a parameterized BIP model to have only one component of the first type and at least ten components of the second type. The FOIL sentence ψ restricts both the system topology and the communication mechanisms, see Example 3.4.

▶ **Definition 3.3** (PBIP Instance)**.** Given a parameterized BIP model $\langle \mathbb{B}, \bar{n}, \psi, \epsilon \rangle$ and a size vector $\bar{N}$, a *PBIP instance* is a BIP model $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma} = \langle \mathcal{B}, \Gamma \rangle$, where $\mathcal{B}$ and Γ are defined as follows:

1. the numbers $\bar{N}$ satisfy the size constraint ϵ,
2. the set of components $\mathcal{B}$ is $\{\mathbb{B}_i[j] \mid i \in [0, k) \text{ and } j \in [0, N_j)\}$, and

3. the set of interactions Γ consists of all interactions γ_ξ induced by a FOIL structure ξ such that the size parameters $\bar{n}$ are interpreted in ξ as $\bar{N}$, and ξ satisfies ψ, i.e. $\alpha_\xi(\bar{n}) = \bar{N}$ and $\xi \models_{\text{FOIL}} \psi$.

In the rest of this section, we give three examples that show expressiveness of parameterized BIP.

▶ **Example 3.4** (Milner's scheduler revisited)**.** The parameterized BIP model of Milner's scheduler is $\langle\langle \mathbb{B}_0\rangle, \langle n_0\rangle, \psi, \mathit{true}\rangle$, where $\mathbb{B}_0$ is from Example 2.5 and $\psi = \psi_{\mathit{token}} \vee \psi_{\mathit{internal}}$ defined as follows. The formula ψ_{token} defines the token-passing interactions and the formula ψ_{internal} defines the internal interactions of starting or finishing a task:

$$\begin{array}{lcl} \psi_{\mathit{token}} & = & \exists i, j :: \mathit{type}_0 : j = (i+1) \bmod n_0.\ \mathit{snd}(i) \wedge \mathit{rcv}(j) \wedge \psi_{\mathit{only}}(i,j) \\ \psi_{\mathit{only}}(i,j) & = & \forall \ell :: \mathit{type}_0 : \ell \neq i \wedge \ell \neq j.\ \neg \mathit{snd}(\ell) \wedge \neg \mathit{rcv}(\ell) \wedge \neg \mathit{start}(i) \wedge \neg \mathit{finish}(i) \\ \psi_{\mathit{internal}} & = & \exists i :: \mathit{type}_0.\ \psi_{\mathit{only}}(i,i) \wedge (\mathit{start}(i) \vee \mathit{finish}(i)) \end{array}$$

The formula ψ_{token} does not have free variables and holds for a structure ξ, if the induced interaction γ_ξ is a send-receive interaction along some edge $i \to j$ of the ring, where $j = (i+1) \bmod n_0$. In fact, $j = (i+1) \bmod n_0$ is just a shorthand for the formula: $(i+1 < n_0 \wedge j = i+1) \vee (i+1 = n_0 \wedge j = 0)$. The formula $\psi_{\mathit{only}}(i,j)$ excludes any component other than i and j from participating in the interaction. (If $i = j$ then all components other than i are excluded.) The formula ψ_{internal} enables the transitions labeled with 'start' and 'finish', in which only one component changes its location.

Observe that the semantics of FOIL forces the quantified variables i, j, ℓ to be in the range from 0 to $N_0 - 1$. Hence, we omit explicit range constraints. For instance, ψ_{token} is equivalent to the formula:

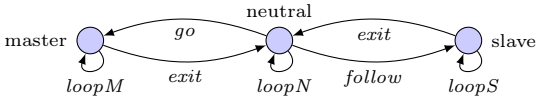
$$\exists i, j :: \mathit{type}_0 : 0 \leq i, j < n_0 \wedge (j = (i+1) \bmod n_0).\ \mathit{snd}(i) \wedge \mathit{rcv}(j) \wedge \psi_{\mathit{only}}(i,j)$$

The set of FOIL structures ξ that satisfy ψ induces the same set of interactions Γ as in Example 2.5. While Example 2.5 defines the set Γ explicitly for any fixed value N_0, in the parameterized setting the interactions are defined uniformly by a single FOIL formula ψ, for all values of N_0.

In this example we do not restrict the initial locations so that exactly one process owns the token in the initial configuration. This delicate issue is resolved in Section 5.4.

▶ **Example 3.5** (Broadcast in a star)**.** Let $\langle\langle \mathbb{B}_0, \mathbb{B}_1\rangle, \langle n_0, n_1\rangle, \psi, \epsilon\rangle$ be a parameterized BIP model with two component types and the size constraint $\epsilon \equiv (n_0 = 1)$. We also assume that component type $\mathbb{B}_0$ (resp. $\mathbb{B}_1$) has only one port *send* (resp. *receive*), i.e., $\mathbb{P}_0 = \{\mathit{send}\}$ and $\mathbb{P}_1 = \{\mathit{receive}\}$. The FOIL formula $\psi = \exists i :: \mathit{type}_0.\ \mathit{send}(i)$ specifies broadcast from the component $\mathbb{B}_0[0]$, the center of the star, to the leaves of type $\mathbb{B}_1$. The set of interactions defined by ψ consists of all sets of ports of the form $\{(\mathit{send}, 0)\} \cup \{(\mathit{receive}, d) \mid d \in D)\}$ for all $D \subseteq [0, n_1)$, including the empty set $D = \emptyset$.

▶ **Example 3.6** (Barrier)**.** Consider a barrier synchronization protocol, cf. [9, Example 6.6]. The component type $\mathbb{B}_0$ is as shown below:



The location *neutral* is the initial location. A synchronization episode consists of three stages:

(i) First, a single component enters the barrier by moving to *master*.
(ii) Then, each of the others components moves to *slave*.
(iii) Finally, the master triggers a broadcast and all components leave the barrier by moving to *neutral*.

The parameterized BIP model of the barrier synchronization protocol is $\langle\langle \mathbb{B}_0\rangle, \langle n_0\rangle, \psi, \mathit{true}\rangle$, where $\psi = \psi_{go} \vee \psi_{follow} \vee \psi_{exit}$, and the following formulae ψ_{go}, ψ_{follow}, and ψ_{exit} describe the interactions of stages (i), (ii), and (iii) respectively:

$$\begin{array}{lcl}
\psi_{go} & = & \exists i :: \mathit{type}_0.\ \mathit{go}(i) \quad \wedge \forall j :: \mathit{type}_0 : i \neq j.\ \mathit{loopN}(j) \\
\psi_{follow} & = & \exists i, j :: \mathit{type}_0.\ \mathit{follow}(i) \wedge \mathit{loopM}(j) \wedge \\
 & & \forall \ell :: \mathit{type}_0 : i \neq \ell.\ \mathit{loopM}(\ell) \vee \mathit{loopN}(\ell) \vee \mathit{loopS}(\ell) \\
\psi_{exit} & = & \forall i :: \mathit{type}_0.\ \mathit{exit}(i)
\end{array}$$

All three formulae enforce progress by requiring at least one process to change its state.

4 Parameterized model checking

In this section, we review the syntax and semantics of the indexed version of CTL^*, called $ICTL^\star$, which is often used to specify the properties of parameterized systems [9]. Though we use indexed temporal logics to define the standard parameterized model checking problem, these logics are not the focus of this paper. Further, we introduce the parameterized model checking problem for parameterized BIP design, and show its undecidability.

Syntax. For a set of index variables $\mathcal{I}$, the $ICTL^\star$ *state* and *path formulae* follow the grammar:

$$\begin{array}{ll}
\theta ::= \mathit{true} \mid \mathit{at}(q, i) \mid \neg\theta \mid \theta_1 \wedge \theta_2 \mid \exists i :: \mathit{type}_j : \phi.\ \theta \mid \forall i :: \mathit{type}_j : \phi.\ \theta \mid \mathbf{E}\,\varphi \mid \mathbf{A}\,\varphi\,, & (\textit{state formulae}) \\
\varphi ::= \theta \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\,\varphi \mid \mathbf{F}\,\varphi \mid \mathbf{G}\,\varphi \mid \varphi_1 \mathbf{U}\,\varphi_2\,. & (\textit{path formulae})
\end{array}$$

where $q \in \bigcup_{0 \leq j < k} \mathbb{Q}_j$ is a location, $i \in \mathcal{I}$ is an index, and ϕ is a formula in Presburger arithmetic over size variables $\bar{n}$ and index variables from the set $\mathcal{I}$.

Semantics. Fix a BIP model $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1}\rangle^{\bar{N}, \Gamma}$ and its transition system $M = \langle S, s_0, \Gamma, R\rangle = \mathit{TS}(\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1}\rangle^{\bar{N}, \Gamma})$ as per Definition 2.4. To evaluate Presburger formulae, we use the first-order structure $PA = \langle \mathbb{N}, 0, 1, \leq, +, \bar{N}\rangle$. The semantics of $ICTL^\star$ formulae is defined inductively using M and PA. We only briefly discuss semantics to highlight the role of quantifiers in indexed temporal logics. For further discussions, we refer the reader to the textbook [12].

State formulae are interpreted over a configuration s and a valuation of index variables $\sigma : \mathcal{I} \to \mathbb{N}$ (the semantics of Boolean operators and universal quantifiers is defined in the standard way):

$$\begin{array}{lll}
M, s, \sigma \models_{\text{ICTL}^\star} \mathit{at}(q, i) & \textit{iff} & q = s(j, \sigma(i)),\ \text{where } q \in \mathbb{Q}_j \\
M, s, \sigma \models_{\text{ICTL}^\star} \exists i :: \mathit{type}_j : \phi.\ \theta & \textit{iff} & PA, \sigma[i \mapsto l] \models_{\text{FO}} \phi \text{ and } M, s, \sigma[i \mapsto l] \models_{\text{ICTL}^\star} \theta \text{ hold,} \\
 & & \text{for some } l \in [0, N_j) \\
M, s, \sigma \models_{\text{ICTL}^\star} \mathbf{E}\,\varphi & \textit{iff} & M, \pi, \sigma \models_{\text{ICTL}^\star} \varphi \text{ for some infinite path } \pi \text{ starting from } s
\end{array}$$

Path formulae are interpreted over an infinite path π, and the valuation function σ as follows (the semantics for Boolean operators and temporal operators $\mathbf{F}$ and $\mathbf{G}$ is defined in the standard way):

$$\begin{array}{lll}
M, \pi, \sigma \models_{\text{ICTL}^\star} \theta & \textit{iff} & M, s, \sigma \models_{\text{ICTL}^\star} \theta,\ \text{where } s \text{ is the first configuration of the path } \pi \\
M, \pi, \sigma \models_{\text{ICTL}^\star} \mathbf{X}\,\varphi & \textit{iff} & M, \pi^1, \sigma \models_{\text{ICTL}^\star} \varphi \\
M, \pi, \sigma \models_{\text{ICTL}^\star} \varphi_1 \mathbf{U}\,\varphi_2 & \textit{iff} & \exists j \geq 0.\ M, \pi^j, \sigma \models_{\text{ICTL}^\star} \varphi_2 \text{ and } \forall i < j.\ M, \pi^i, \sigma \models_{\text{ICTL}^\star} \varphi_1,
\end{array}$$

where π^i is the suffix of the path π starting with the i^{th} configuration.

Finally, given a formula φ without free variables, we say that M satisfies φ, written as $M \models_{\text{ICTL}^\star} \varphi$, if $M, s_0, \sigma_0 \models_{\text{ICTL}^\star} \varphi$ for the valuation σ_0 that assigns zero to each index from the set $\mathcal{I}$. The choice of σ_0 is arbitrary, as for all σ, it holds that $M, s_0, \sigma \models_{\text{ICTL}^\star} \varphi$ if and only if $M, s_0, \sigma_0 \models_{\text{ICTL}^\star} \varphi$.

Now we are in the position to formulate the parameterized model checking problem for BIP:

▶ **Problem 4.1** (Parameterized model checking). *The verification problem for a parameterized* BIP *model* $\langle \mathbb{B}, \bar{n}, \psi, \epsilon \rangle$ *and an ICTL* state formula* θ *without free variables, is whether every instance* $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma}$ *satisfies* θ.

Not surprisingly, Problem 4.1 is undecidable in general. For instance, one can use the proof idea [16] to obtain the following theorem. We do not give a detailed proof here: to a large extent, it repeats the encoding of a unidirectional token ring, which we discuss later in Section 5.4.

▶ **Theorem 4.2** (Undecidability). *Given a two-counter machine* M_2*, one can construct an ICTL*-formula* $\mathbf{G}\,\neg\texttt{halt}$ *and a parameterized BIP model* $\mathcal{B} = \langle \mathbb{B}, \bar{n}, \psi, \epsilon \rangle$ *that simulates* M_2 *and has the property:* M_2 *does not halt if and only if* $\langle \mathbb{B}_0, \ldots, \mathbb{B}_{k-1} \rangle^{\bar{N}, \Gamma} \models \mathbf{G}\,\neg\texttt{halt}$ *for all instances of* $\mathcal{B}$.

5 Identifying the architecture of a parameterized BIP model

In the non-parameterized case, knowing the architecture is not crucial, as there are model checking algorithms that apply in general to arbitrary finite transition systems. However, the architecture dramatically affects decidability of *parameterized* model checking. Architecture identification plays an important step in our verification framework. In this section, we show how to identify system architectures automatically, and present applications to verification.

Our framework. For the sake of exposition, we assume that parameterized BIP models have only one component type. Our identification framework extends easily to the general case.

Given an architecture $\mathcal{A}$, e.g., the token ring architecture, an expert in parameterized model checking creates formula templates in FOIL (*FOIL-templates*) and in temporal logic (*TL-templates*). *FOIL-templates* describe the system topology and communication mechanism for the architecture $\mathcal{A}$. *TL-templates* describe the behaviour of the component type required by the architecture $\mathcal{A}$, e.g., in a token ring, a component which does not have the token cannot send the token. These templates are designed once for all parameterized BIP models compliant with $\mathcal{A}$. In the sequel, *TL-templates* are only used for token rings, thus we omit them from the discussion of other architectures.

Given a parameterized BIP model $\langle \langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon \rangle$—not necessarily compliant with the architecture $\mathcal{A}$—the templates for the architecture $\mathcal{A}$ are instantiated to FOIL formulae $\varphi_1^{FOIL}, \ldots, \varphi_m^{FOIL}$, and temporal logic formulae $\varphi_1^{TL}, \ldots, \varphi_\ell^{TL}$. The FOIL formulae guarantee that the set of interactions expressed by the FOIL formula ψ adheres to $\mathcal{A}$. The temporal logic formulae guarantee that the behaviour of the component type $\mathbb{B}$ adheres to $\mathcal{A}$. The *identification criterion* is as follows: if $\varphi_1^{FOIL} \wedge \cdots \wedge \varphi_m^{FOIL}$ is valid and $\mathbb{B} \models_{\text{TL}} \varphi_1^{TL} \wedge \cdots \wedge \varphi_\ell^{TL}$ holds, then the parameterized model $\langle \langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon \rangle$ is compliant with the architecture $\mathcal{A}$. In practice, we use an SMT solver to check validity of the FOIL formulae and a model checker to check that the component type $\mathbb{B}$ satisfies the temporal formulae.

In the rest of this section we construct FOIL-templates and TL-templates for well-known architectures: cliques of processes communicating via broadcast, cliques of processes communicating via rendezvous, token rings, and server-client systems in which processes are organized in a star and communicate via rendezvous. We show that the provided templates identify the architectures in a sound way.

5.1 The common templates for BIP semantics

As we discussed in Section 3.2, not every FOIL structure induces a BIP interaction. We show that one can write a FOIL-template that restricts FOIL structures to induce BIP interactions. The following template $\eta^{FOIL}_{interaction}(\mathbb{P}_0)$ expresses that there is no component with more than one active port: $\forall j :: type_0.\, \bigwedge_{p,q \in \mathbb{P}_0,\ q \neq p} \neg p(j) \vee \neg q(j)$

As expected, the template $\eta^{FOIL}_{interaction}(\mathbb{P}_0)$ restricts FOIL structures to BIP interactions:

▶ **Proposition 5.1.** *Let $\mathbb{P}_0$ be a set of ports, and η be the instantiation of $\eta^{FOIL}_{interaction}$ with $\mathbb{P}_0$. A FOIL structure ξ satisfies η if and only if ξ induces an interaction.*

To express that a component has at least one active port, we introduce template $active(j) \equiv \bigvee_{p \in \mathbb{P}_0} p(j)$. To simplify notation, parameterization of $active(j)$ by $\mathbb{P}_0$ is omitted.

5.2 Pairwise rendezvous in a clique

In a BIP model, components are said to communicate by binary rendezvous, if all the allowed interactions consist of exactly two ports. The communication is said to be by *pairwise rendezvous*, if there is a binary rendezvous between every two components. Pairwise rendezvous has been widely used as a basic primitive in the parameterized model checking literature, e.g., in [18, 3].

FOIL-templates. We construct a template using two formulae $\eta^{FOIL}_{\leq 2}(\mathbb{P}_0)$ and $\eta^{FOIL}_{\geq 2}(\mathbb{P}_0)$:

- The formula $\eta^{FOIL}_{\leq 2}(\mathbb{P}_0)$ expresses that every interaction has at most two ports: $\forall i, j, \ell :: type_0.\ active(i) \wedge active(j) \wedge active(\ell) \rightarrow i = j \vee j = \ell \vee i = \ell$.
- The formula $\eta^{FOIL}_{\geq 2}(\mathbb{P}_0)$ expresses that every interaction has at least two ports: $\exists i, j :: type_0 : i \neq j.\ active(i) \wedge active(j)$.

We show that the combination of $\eta^{FOIL}_{interaction}$, $\eta^{FOIL}_{\geq 2}$, and $\eta^{FOIL}_{\leq 2}$ defines pairwise rendezvous communication in cliques of all sizes:

▶ **Theorem 5.2.** *Given a one-type parameterized* BIP *model $\langle\langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon\rangle$, if $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2})$ is valid, then for every instance $\mathbb{B}^{N,\Gamma}$, the following holds:*

1. *every interaction is of size 2, that is, $|\gamma| = 2$ for $\gamma \in \Gamma$, and*
2. *for every pair of indices i and j such that $0 \leq i, j < N$ and $i \neq j$ and every pair of ports $p, q \in \mathbb{P}_0$, there is a FOIL structure ξ such that $\xi \models_{\text{FOIL}} \psi \wedge p(i) \wedge q(j)$.*

Proof. Fix an instance $\mathbb{B}^{N,\Gamma}$ of $\langle\langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon\rangle$.

To show Point 1, fix an interaction γ of $\mathbb{B}^{N,\Gamma}$. By Definition 3.3, there is a FOIL structure ξ such that $\xi \models_{\text{FOIL}} \psi$ and $\gamma = \gamma_\xi$. As ξ induces an interaction, by Proposition 5.1, we immediately have that γ_ξ satisfies the instantiation of $\eta^{FOIL}_{interaction}$. Hence, since $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2})$ is valid we conclude that ξ also satisfies $\eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2}$. This immediately gives us the required equality $|\gamma_\xi| = 2$.

To show Point 2, fix a pair of indices i and j such that $0 \leq i, j < N$ and $i \neq j$ and a pair of ports $p, q \in \mathbb{P}_0$. The set $\gamma = \{(p, i), (q, j)\}$ is an interaction. Obviously, one can construct a FOIL structure ξ that induces γ. Since $i \neq j$ and $|\gamma_\xi| = 2$, it holds that $\xi \models_{\text{FOIL}} \eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2}$. Thus, since $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2})$ is valid, it follows that $\xi \models_{\text{FOIL}} \psi$. From this and that ξ induces the interaction γ, we conclude that $\xi \models_{\text{FOIL}} \psi \wedge p(i) \wedge q(j)$. ◀

In Theorem 5.2, the right-hand side of the equivalence does not restrict which pairs of ports may interact, e.g., it does not require the ports to be the same. Thus, if ψ is more restrictive than the right-hand side of the equivalence, validity will not hold. Obviously, one can further restrict the equivalence to reflect additional constraints on the allowed pairs of ports. Moreover, one may restrict which ports are required by the template to communicate via pairwise rendezvous for compositionality, e.g. to allow other ports to participate in other communication primitives and in internal transitions. (One may augment or restrict the templates of all the architectures below similarly.)

Applications. Theorem 5.2 gives us a criterion for identifying parameterized BIP models, where all processes may interact with each other using rendezvous communication. To verify such parameterized BIP models, we can immediately invoke the seminal result by German & Sistla [18, Sec. 4]. Their result applies to specifications written in indexed linear temporal logic without the operator $\mathbf{X}$.

More formally, we say that an $ICTL^\star$ path formula $\chi(i)$ is a 1-$LTL\backslash X$ formula, if χ has only one index variable i and χ does not contain quantifiers $\exists$, $\forall$, $\mathbf{A}$, $\mathbf{E}$, nor temporal operator $\mathbf{X}$. Given a parameterized BIP model $\langle \langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon \rangle$ and a 1-$LTL\backslash X$ formula χ, one can check in polynomial time, whether every instance $\mathbb{B}^{N,\ \Gamma}$ satisfies the formula $\mathbf{E}\ \exists i :: type_0 :\ true.\ \chi(i)$.

5.3 Broadcast in a clique

In BIP, components communicate via broadcast, if there is a "trigger" component whose sending port is active, and the other components either have their receiving port active, or have no active ports. In this section, we denote the sending port with *send* and the receiving port with *receive*. Our results can be easily extended to treat multiple sending and receiving ports. In a broadcast step, all the components with the active ports make their transitions simultaneously. Broadcasts were extensively studied in the parameterized model checking literature [17, 23].

One way to enforce all the processes to receive a broadcast, if they are ready to do so, is to use priorities in BIP: an interaction has priority over any of its subsets. In this paper, we consider BIP without priorities. In this case, one can express broadcast by imposing the following restriction on the structure of the component type $\mathbb{B}$: *every location has a transition labeled with the port receive*. This restriction enforces all interactions to involve all the components, though some of the components may not change their location by firing a self-loop transition. This requirement can be statically checked on the transition system of $\mathbb{B}$, and if the component type does not fulfill the requirement, it is easy to modify the component type's transition system by adding required self-loops.

FOIL-templates. First, we define the formula $\eta^{FOIL}_{bcast}(\mathbb{P}_0)$, which guarantees that every interaction includes one sending port by one component and the receiving ports of the other components:

$$\exists i :: type_0.\ send(i) \wedge \forall j :: type_0 : j \neq i.\ receive(j)$$

We show that the combination of $\eta^{FOIL}_{interaction}$ and η^{FOIL}_{bcast} defines broadcast in cliques of all sizes:

▶ **Theorem 5.3.** *Given a one-type parameterized* BIP *model* $\langle\langle\mathbb{B}\rangle, \langle n\rangle, \psi, \epsilon\rangle$, *if* $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{bcast})$ *is valid, then for every instance* $\mathbb{B}^{N,\Gamma}$, *the following holds:*

1. *every interaction consists of one send port and* $N-1$ *receive ports.*
2. *for every index* c, *such that* $0 \leq c < N$, *there is a FOIL structure* ξ *satisfying the following:*
   $\xi \models_{\text{FOIL}} \psi \wedge \mathit{send}(c) \wedge \forall j :: \mathit{type}_0 : j \neq c.\ \mathit{receive}(j)$.

Proof. The proof follows the same principle as the proof of Theorem 5.2. ◀

Applications. Theorem 5.3 gives a criterion for identifying parameterized BIP models in which all components may send and receive broadcast. Its implications are two-fold. First, it is well-known that parameterized model checking of safety properties is decidable [1] (cf. the discussion in [17]), and there are tools for well-structured transition systems applicable to model checking of parameterized BIP. Second, parameterized model checking of liveness properties is undecidable [17]. From the user perspective, this indicates the need to construct abstractions, or to use semi-decision procedures.

Identifying sending and receiving ports. Now we illustrate how to automatically detect the sending and receiving ports in a parameterized BIP model. We say that a port $p \in \mathbb{P}_0$ in the component type may be a sending port, if in every interaction exactly one component uses this port. Similarly, we say that a port $q \in \mathbb{P}_0$ in the component type may be a receiving port, if in every interaction all but one component use this port. Intuitively, we have to enumerate all port types and check whether they are acting as sending ports or receiving ports. Formally, to find whether p is a potential sending port and q is a potential receiving port, we check whether the following is valid:

$$\psi \ \wedge\ \eta^{FOIL}_{interaction} \ \wedge\ \exists i :: \mathit{type}_0.\ \big(p(i) \vee q(i)\big) \ \rightarrow\ \big(\exists i :: \mathit{type}_0.\ p(i) \wedge \forall j :: \mathit{type}_0 : j \neq i.\ q(j)\big)$$

5.4 Token rings

Token ring is a classical architecture: (i) all processes are arranged in a ring, (ii) the ring size is parameterized but fixed in each run, and (iii) one component owns the token and can pass the token to its neighbor(s). It is easy to express token-passing with rendezvous, so we re-use the templates from Section 5.2. We assume that there is a pair of ports: the port *send* giving away the token and the port *receive* accepting the token. We do not allow the token to change its type, as the parameterized model checking problem is undecidable in this case [26, 16]. Nevertheless, it is easy to extend our results to multiple token types. Here the token is passed in one direction, that is, every component may only receive the token from one neighbor and may only send the token to its other neighbor.

TL-templates. Following the standard assumption [16], we require that every process sends and receives the token infinitely often. We encode this requirement as a local constraint in a form of an LTL formula that is checked against the component type (and not against a BIP instance):

$$\mathbf{G}\Big(\mathit{receive} \rightarrow \mathbf{X}\ (\neg \mathit{receive}\ \mathbf{U}\ \ \mathit{send})\Big) \wedge \mathbf{G}\Big(\mathit{send} \rightarrow \mathbf{X}\ (\neg \mathit{send}\ \mathbf{U}\ \ \mathit{receive})\Big)$$

The left conjunct forces a component that has the token to eventually send it. The right conjunct prevents a component from sending the token twice before receiving it back.

FOIL-templates. We extend the pairwise rendezvous templates with a formula $\eta^{FOIL}_{uniring}(\mathbb{P}_0)$ that restricts the interactions to be performed only among the neighbors in one direction:

$$\exists i, j :: type_0.\ (j = (i+1) \bmod n_0).\ send(i) \wedge receive(j)$$

The modulo notation "$j = (i+1) \bmod n_0$" can be seen as syntactic sugar, as it expands into $(i = n_0 - 1 \to j = 0) \wedge (i < n_0 - 1 \to j = i + 1)$.

▶ **Theorem 5.4.** *Given a one-type parameterized* BIP *model* $\langle \langle \mathbb{B} \rangle, \langle n \rangle, \psi, \epsilon \rangle$, *if* $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2} \wedge \eta^{FOIL}_{uniring})$ *is valid, then every instance* $\mathbb{B}^{N,\Gamma}$ *satisfies:*

1. *every interaction* $\gamma \in \Gamma$ *is of the form* $\{send(c), receive(d)\}$ *for some indices* c *and* d *such that* $0 \leq c, d < N$ *and* $d = (c+1) \bmod N$, *and*
2. *for every index* c *such that* $0 \leq c < N$ *and the index* $d = (c+1) \bmod N$, *there is a FOIL structure* ξ *such that* $\xi \models_{\text{FOIL}} \psi \wedge send(c) \wedge receive(d)$.

Proof. The proof follows the same principle as the proof of Theorem 5.2. ◀

Distributing the token. The token ring architecture assumes that initially only one component has the token. Emerson & Namjoshi [16] assumed that the token was distributed using a "daemon", but this primitive is obviously outside of the token ring architecture. Our framework encompasses token distribution. To this end, we restrict the transition system of the component as follows:

- We assume that the location set $\mathbb{Q}_0$ of the component type $\mathbb{B}_0$ is partitioned into two sets: $\mathbb{Q}_0^{tok}$ is the set of locations possessing the token, and $\mathbb{Q}_0^{ntok}$ is the set of locations without the token. The initial location does not possess the token: $\ell^0 \in \mathbb{Q}_0^{ntok}$.
- We assume that there are two auxiliary ports called *master* and *slave* that are only used in a transition from the initial location ℓ^0. There are only two transitions involving ℓ^0: the transition from ℓ^0 to a location in $\mathbb{Q}_0^{tok}$ that broadcasts via the port *master*, and the transition from ℓ^0 to a location in $\mathbb{Q}_0^{ntok}$ that receives the broadcast via the port *slave*. The broadcast interaction can be checked with the constraints similar to those in Section 5.3.

Applications. Theorem 5.4 gives us a criterion for identifying parameterized BIP models that express a unidirectional token ring. This criterion has a great impact: one can apply non-parameterized BIP tools to verify parameterized BIP designs expressing token rings. As Emerson & Namjoshi showed in their celebrated paper [16], to verify parameterized token rings, it is sufficient to run model checking on rings of small sizes. The bound on the ring size—called a *cut-off*—depends on the specification and typically requires two or three components.

5.5 Pairwise rendezvous in a star

In a star architecture, one component acts as the center, and the other components communicate only with the center. The components communicate via rendezvous (considered in Section 5.2). This architecture is used in client-server applications. Parameterized model checking for the star architecture was investigated by German & Sistla [18]. We assume that a parameterized BIP model contains two component types: $\mathbb{B}_0$ with only one instance, and $\mathbb{B}_1$ that may have many instances.

Table 1 Experimental results of identifying architecture models. The column "Outcome" indicates, whether the benchmark was recognized to have the given architecture (positive), or not (negative). The experiments were performed on a 64-bit Linux machine with 2.8GHz × 4 CPU and 7.8GiB memory.

Benchmark	Architecture model	Outcome	Time (sec.)	Memory (MB)
Milner's scheduler	uni-directional token ring	**positive**	0.068	≤ 10
Milner's scheduler	broadcast in clique	negative	0.016	≤ 10
Semaphore	pairwise rendezvous in star	**positive**	0.096	≤ 10
Semaphore	pairwise rendezvous in clique	negative	0.084	≤ 10
Barrier	broadcast in clique	**positive**	0.028	≤ 10
Barrier	pairwise rendezvous in star	negative	0.008	≤ 10

FOIL-templates. The requirements of rendezvous communication are defined in Section 5.2. We add the restriction η^{FOIL}_{center} that the center is involved in every interaction: $\exists i :: type_0.\ active_0(i)$. By restricting ϵ to have only one instance of type $\mathbb{B}_0$, we arrive at Theorem 5.5, which to a large extent is a consequence of Theorem 5.2.

▶ **Theorem 5.5.** *Given a two-component parameterized* BIP *model* $\langle\langle \mathbb{B}_0, \mathbb{B}_1\rangle, \langle n_0, n_1\rangle, \psi, \epsilon\rangle$, *if* $(\psi \wedge \eta^{FOIL}_{interaction}) \leftrightarrow (\eta^{FOIL}_{interaction} \wedge \eta^{FOIL}_{\geq 2} \wedge \eta^{FOIL}_{\leq 2} \wedge \eta^{FOIL}_{center})$ *and* $\epsilon \leftrightarrow (n_0 = 1)$ *are both valid, then every instance* $\langle \mathbb{B}_0, \mathbb{B}_1\rangle^{\langle N_0, N_1\rangle, \Gamma}$ *admits only the rendezvous interactions with the center, i.e., the only component of type* $\mathbb{B}_0$.

Applications. Theorem 5.5 gives us a criterion for identifying parameterized BIP models, where the user processes communicate with the coordinator via rendezvous. To verify such parameterized BIP models, we can immediately invoke several results by German & Sistla [18, Sec. 3]. First, one can analyze such parameterized BIP models for deadlocks, which is of extreme importance to the practical applications of BIP. Second, the results [18] reduce parameterized model checking to reachability in Petri nets, which allows one to use the existing tools for Petri nets.

6 Prototype implementation and experiments

We have implemented a prototype of the framework introduced in Section 5. This prototype uses the following architecture templates: (a) pairwise rendezvous and broadcast in cliques, (b) token rings, (c) and pairwise rendezvous in stars. As described in Section 5 (see *our framework*), given a parameterized BIP model, the tool constructs a set of FOIL formulae and a set of temporal formulae. The parameterized BIP model follows a predefined architecture, if the FOIL formulae are valid and the component types satisfy the temporal formulae. Our implementation uses nuXmv [11] to check temporal formulae and Z3 [14] to check validity of first-order formulae. FOIL formulae are translated to first-order formulae by guarding the range of quantification explicitly, e.g. $\exists i :: type_0.\ \theta$ is substituted with $\exists i.\ 0 \leq i < n_0 \wedge \theta$. To deal with quantifiers, we run a customized solver with tactic 'qe' (i.e. quantifier elimination). The implementation and benchmarks are available at `http://risd.epfl.ch/parambip`.

Table 1 summarizes our experiments with three benchmarks. We conducted each experiment using two kinds of templates: the expected architecture of the benchmark, and an architecture different from the expected one. In all cases, the architectures were identified as expected. Our preliminary results demonstrate both correctness and efficiency of our approach.

7 Related work and conclusions

We have shown that our framework encompasses several prominent parameterized model checking techniques. To our understanding, the other seminal results can be integrated into our framework: the cut-off results for disjunctive and conjunctive guards [15], network decomposition techniques [13, 3], and techniques based on well-structured transition systems [1] and monotonic abstraction [2].

First-order interaction logic extends propositional interaction logic [6, 7], which was also extended by Dy-BIP [10] and configuration logic [21]. Dy-BIP extends propositional interaction logic with quantification to define interaction topology independent of the number of component instances. It uses dedicated *history variables* to break the symmetry and specify that, throughout the system execution, successive interactions happen among the same components. Dy-BIP does not have a mechanism, such as indexing, to statically distinguish instances of the same component type. Hence, many architectures, e.g., token rings, cannot be expressed. Configuration logic uses higher-order formulae to represent sets of topologies. It does not use indexing either, thereby requiring the second-order extension to express simple architectures such as token rings and linear architectures. Finally, no decidability results or decision procedures have been proposed for the configuration logic yet.

In the future, we will study second-order extensions of FOIL to express more complex architectures such as server-client whose coordinator is chosen non-deterministically. In the long term, we plan to implement a tool that integrates multiple parameterized model checking techniques and uses our framework to guide the verification of parameterized BIP designs. FOIL can also be seen as a specification language for BIP interactions and used for their synthesis similarly to [7]. Finally, it is worth investigating, whether FOIL can be extended to include priorities as in [8].

References

1 P. A. Abdulla, K. Cerans, B. Jonsson, and Y. Tsay. General decidability theorems for infinite-state systems. In *LICS*, 1996.

2 P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Monotonic abstraction: on efficient verification of parameterized systems. *Int. J. Found. Comput. Sci.*, 2009.

3 B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*. Springer, 2014.

4 A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T. Nguyen, and J. Sifakis. Rigorous component-based system design using the BIP framework. *Software, IEEE*, 2011.

5 S. Bliudze, A. Cimatti, M. Jaber, S. Mover, M. Roveri, W. Saab, and Q. Wang. Formal verification of infinite-state BIP models. In *ATVA*, 2015.

6 S. Bliudze and J. Sifakis. The algebra of connectors —structuring interaction in BIP. In *EMSOFT*, 2007.

7 S. Bliudze and J. Sifakis. Causal semantics for the algebra of connectors. *FMSD*, 2010.

8 S. Bliudze and J. Sifakis. Synthesizing glue operators from glue constraints for the construction of component-based systems. In *Software Composition*, 2011.

9 R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. Decidability of parameterized verification. *Synthesis Lectures on Distributed Computing Theory*, 2015.

10 M. Bozga, M. Jaber, N. Maris, and J. Sifakis. Modeling dynamic architectures using Dy-BIP. In *Software Composition*. Springer, 2012.

11 R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *CAV*, 2014.

12 E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
13 E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR*, 2004.
14 L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.
15 E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, 2003.
16 E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL*, 1995.
17 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. *LICS*, 1999.
18 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. ACM*, 1992.
19 O. Grumberg and H. Veith. *25 years of model checking: history, achievements, perspectives*. Springer, 2008.
20 J. Y Halpern. Presburger arithmetic with unary predicates is π_1^1 complete. *J. of Symb. Logic*, 1991.
21 A. Mavridou, E. Baranov, S. Bliudze, and J. Sifakis. Configuration logics: Modelling architecture styles. In *FACS*, 2015.
22 Q.Wang and S. Bliudze. Verification of component-based systems via predicate abstraction and simultaneous set reduction. In *TGC*, 2015.
23 S. Schmitz and P. Schnoebelen. The power of well-structured systems. In *CONCUR*, 2013.
24 J. Sifakis. Rigorous system design. *Foundations and Trends in Electr. Design Automation*, 2013.
25 J. Sifakis. System design automation: Challenges and limitations. *Proc. of the IEEE*, 2015.
26 I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 1988.
27 W. Thomas. *Languages, automata, and logic*. Springer, 1997.

Private Names in Non-Commutative Logic*

Ross Horne[1], Alwen Tiu[2], Bogdan Aman[3], and Gabriel Ciobanu[4]

1 **School of Computer Science and Engineering, Nanyang Technological University, Singapore**
rhorne@ntu.edu.sg
2 **School of Computer Science and Engineering, Nanyang Technological University, Singapore**
atiu@ntu.edu.sg
3 **Romanian Academy, Institute of Computer Science, Blvd. Carol I no.8, 700505 Iaşi, Romania**
bogdan.aman@iit.academiaromana-is.ro
4 **Romanian Academy, Institute of Computer Science, Blvd. Carol I no.8, 700505 Iaşi, Romania**
gabriel@info.uaic.ro

Abstract

We present an expressive but decidable first-order system (named MAV1) defined by using the calculus of structures, a generalisation of the sequent calculus. In addition to first-order universal and existential quantifiers the system incorporates a de Morgan dual pair of nominal quantifiers called 'new' and 'wen', distinct from the self-dual Gabbay-Pitts and Miller-Tiu nominal quantifiers. The novelty of the operators 'new' and 'wen' is they are polarised in the sense that 'new' distributes over positive operators while 'wen' distributes over negative operators. This greater control of bookkeeping enables private names to be modelled in processes embedded as predicates in MAV1. Modelling processes as predicates in MAV1 has the advantage that linear implication defines a precongruence over processes that fully respects causality and branching. The transitivity of this precongruence is established by novel techniques for handling first-order quantifiers in the cut elimination proof.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.3.2 Semantics of Programming Languages; F.1.2 Modes of Computation

Keywords and phrases process calculi, calculus of structures, nominal logic, causal consistency

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.31

1 Introduction

This paper introduces a first-order non-commutative logic, of significance for modelling processes, expressed in a formalism called the calculus of structures [17, 18, 34, 36, 37]. The calculus of structures is effectively a term rewriting system modulo a congruence that can express proof systems combining connectives for sequentiality and parallelism. The calculus of structures was motivated by a desire to understand why pomset logic [30] could not be expressed in the sequent calculus. Pomset logic is inspired by pomsets [29] and linear logic [15], the former being a model of concurrency respecting causality [33], while the

* The first two authors receive support from MOE Tier 2 grant MOE2014-T2-2-076. The second author receives support from NTU Start Up grant M4081190.020. The first, third and fourth authors are supported by ANCS grant number PN-II-ID-PCE-2011-3-0919.

latter can be interpreted in various ways as a logic of resources and concurrency [9, 21, 42]. Acknowledging connections between pomsets, linear logic and concurrency, it is natural to consider the calculus of structures in the context of concurrency theory.

Initial investigations [7] relate a basic process calculus with parallel composition ($P \parallel Q$) and action prefix ($\alpha.P$) to a proof system in the calculus of structures called BV [17]. In that work, a logical characterisation of completed traces is established using provability. As a consequence of this logical characterisation and cut elimination for BV, if P implies Q and P has a completed trace then Q has the same completed trace. Thereby, linear implication is strictly finer than completed trace inclusion. Strictness follows since some processes related by linear implication are not related by trace inclusion. For example, a desirable property of linear implication is that *autoconcurrency* [4, 40, 41] is avoided, since the embedding of $\alpha \parallel \alpha$ does **not** logically imply the embedding of $\alpha.\alpha$. Avoiding autoconcurrency indicates that linear implication fully respects the causal order of events. Preorders that respect causality are significant for various applications, not limited to soundly reasoning about processes deployed on large high-availability distributed systems where a consensus on interleavings is infeasible but causality is respected [11, 22].

Recently, BV was extended with additives to obtain the system MAV [20], enabling choice in processes to be modelled. Results concerning multi-party compatibility and subtyping in a session type system inspired by Scribble [19] have been established using MAV [12]. The current paper continues this line of enquiry by modelling name passing processes in a conservative extension of MAV with first-order quantifiers, named MAV1. The system MAV1 is also a conservative extension of first-order multiplicative additive linear logic MALL1 [23] with mix. A novelty is that MAV1 includes a pair of de Morgan dual nominal quantifiers pronounced *new* and *wen* and written И and Э respectively. In a processes-as-predicates embedding [16], И models new name restriction in the π-calculus. The dual nominal quantifier Э is essential for defining linear implication and models the input of private names in a processes-as-predicates embedding for the πI-calculus [32].

Logically speaking, nominal quantifiers И and Э sit between $\forall$ and $\exists$ such that $\forall x.P \multimap$ И$x.P$ and И$x.P \multimap$ Э$x.P$ and Э$x.P \multimap \exists x.P$, where $\multimap$ is linear implication. The quantifier И is similar in some respects to $\forall$, whereas Э is similar to $\exists$. A crucial difference between $\exists x.P$ and Э$x.P$ is that variable x in the latter cannot be instantiated with arbitrary terms, but only 'fresh' names introduced by И.

The need for new quantifiers. We illustrate why neither universal quantification nor an established self-dual nominal quantifier [14, 25, 28] are capable of soundly modelling name restriction in a processes-as-predicates embedding. We argue that, since trace inclusion is considered to be amongst the coarsest preorders on process [39], it makes sense to impose a minimum requirement that linear implication cannot relate two processes that are unrelated by trace inclusion.

In the following, observe that $R_1 = \nu x.(\overline{a}x \parallel \overline{b}x)$ is a π-calculus process that can output a fresh name twice, once on channel a and once on channel b; but cannot output two perceivably distinct names in any execution. In contrast, observe that $R_2 = \nu x.\overline{a}x \parallel \nu x.\overline{b}x$ is a π-calculus process that outputs two distinct fresh names before terminating, but cannot output the same name twice in any execution. The processes R_1 and R_2 are unrelated by trace inclusion in either direction.

For an encoding using universal quantifiers for name restriction, processes R_1 and R_2 are respectively encoded as predicates $P_1 = \forall x.\big(\overline{ax} \parallel \overline{bx}\big)$ and $P_2 = \forall x.\overline{ax} \parallel \forall x.\overline{bx}$, where operator $\parallel$ is overloaded to connect parallel composition and *par* from linear logic. Unfortunately, the

implication $P_2 \multimap P_1$ is provable. However, R_2 can output two perceivably distinct names but R_1 cannot, so implication would not be sound with respect to trace inclusion. Additionally, we must also avoid the following *diagonalisation* property [25]: $\forall x.\forall y.P(x,y) \multimap \forall z.P(z,z)$.

The self-dual nominal quantifiers of either Gabbay-Pitts [28] or Miller-Tiu [25, 14], as recently investigated in the calculus of structures [31], do successfully avoid the above diagonalisation property. Unfortunately, rather surprisingly, encoding private names using any of these self-dual nominal quantifiers, say ∇, leads to the following problem. Suppose processes R_1 and R_2 are encoded by the respective predicates $Q_1 = \nabla x.(\overline{ax} \parallel \overline{bx})$ and $Q_2 = \nabla x.\overline{ax} \parallel \nabla x.\overline{bx}$. In this case, the linear implication $Q_1 \multimap Q_2$ is provable. This implication is also unsound, since R_1 has a trace that outputs two identical names, whereas R_2 admits no such trace.

Our *new* quantifier И, distinct from the Gabbay-Pitts operator, addresses the above limitations of universal quantification and established self-dual nominal quantifiers. In addition to avoiding diagonalisation, our И quantifier does not distribute over parallel composition in either direction. In MAV1, the predicates $Иx.(\overline{ax} \parallel \overline{bx})$ and $Иx.\overline{ax} \parallel Иx.\overline{bx}$ are, correctly, unrelated by linear implication.

Outline. For a new logical system it is necessary to justify correctness, which we approach in proof theoretic style by cut elimination. Section 2 defines MAV1 and explains cut elimination. Section 3 illustrates a processes-as-predicates embedding in MAV1 and explains that linear implication defines a branching-time preorder that respects causality. Section 4 presents a more detailed explanation of the rules for the nominal quantifiers and the novel strategy of the cut elimination proof.

2 Syntax and Semantics of Predicates in MAV1

In this section we present the syntax and semantics of a first-order system expressed in the calculus of structures, with the technical name MAV1. We assume that the reader has a basic understanding of term-rewriting systems [24].

A term-rewriting system requires an *abstract syntax*, defined in Fig. 1. The *rewrite rules*, in Fig 3, define rules that can be applied to rewrite a predicate of the form on the left of the long right arrow to the predicate on the right. All rewrite rules can be applied in any context, i.e. MAV1 predicates from Fig. 1 with a hole of the following form $\mathcal{C}\{\ \} ::= \{\ \cdot\ \} \mid \mathcal{C}\{\ \} \odot P \mid P \odot \mathcal{C}\{\ \} \mid \mathfrak{O}x.\mathcal{C}\{\ \}$, where $\odot \in \{;, \parallel, \otimes, \&, \oplus\}$ and $\mathfrak{O} \in \{\exists, \forall, И, Э\}$.

Further to rewriting according to rules, the term-rewriting system is defined *modulo a congruence*, where a congruence is an equivalence relation that holds in any context. The congruence, defined in Fig. 2, makes *par* and *times* commutative and *seq* non-commutative in general. The congruence enables α-conversion for quantifiers. In addition, *equivariance* allows names bound by successive nominal quantifiers to be swapped.

As standard, we define a freshness predicate such that a variable x is fresh for a predicate P, written $x \# P$, if and only if x is not a member of the set of free variables of P, such that all quantifiers bind variables in their scope. We also assume the standard notion of capture avoiding substitution of a variable for a term. Terms may be constructed from variables, constants and function symbols. When predicates model process in the π-calculus, atoms are pairs of terms, where the first term represents a channel and the second a message.

We postpone a discussion on the rules until after we introduce the notion of a proof and explain cut elimination in the next section.

$$
\begin{array}{rll}
P ::= & \alpha & \text{(atom)} \\
& \overline{\alpha} & \text{(co-atom)} \\
& \mathrm{I} & \text{(unit)} \\
& \forall x.P & \text{(all)} \\
& \exists x.P & \text{(some)} \\
& Ии x.P & \text{(new)} \\
& Э x.P & \text{(wen)} \\
& P \,\&\, P & \text{(with)} \\
& P \oplus P & \text{(plus)} \\
& P \parallel P & \text{(par)} \\
& P \otimes P & \text{(times)} \\
& P \,;\, P & \text{(seq)}
\end{array}
$$

Figure 1 *MAV1* syntax.

$$
(P, \parallel, \mathrm{I}) \text{ and } (P, \otimes, \mathrm{I}) \text{ are commutative monoids} \quad \mathfrak{d} x.P \equiv \mathfrak{d} y.(P\{{}^{y}/_{x}\}) \text{ if } y \# \mathfrak{d} x.P \quad (\alpha\text{-conversion})
$$

$$
(P, ;, \mathrm{I}) \text{ is a monoid} \quad Ии x.Ии y.P \equiv Ии y.Ии x.P \ \text{(equivariance)} \quad Э x.Э y.P \equiv Э y.Э x.P \ \text{(equivariance)}
$$

Figure 2 Congruence ($\equiv$) for MAV1 predicates. For α-conversion $\mathfrak{d} \in \{\exists, \forall, Ии, Э\}$ is any quantifier.

2.1 Linear Implication and Cut Elimination

This section confirms that MAV1 is a consistent logical system, as established by a cut elimination theorem. Surprisingly, to date, the only direct proof of cut elimination involving quantifiers in the calculus of structures is for a self-dual nominal quantifier [31] distinct from any quantifier in MAV1. Related cut elimination results involving first-order quantifiers in the calculus of structures rely on a correspondence with the sequent calculus [6, 35]. However, due to the presence of the non-commutative operator *seq* there is no sequent calculus presentation [37] for MAV1; hence we pursue here a direct proof.

A derivation is a sequence of zero or more rewrite rules from Fig. 3, where the congruence in Fig. 2 can be applied at any point. We are particularly interested in special derivations, called proofs.

▶ **Definition 1.** A *proof* in MAV1 is a derivation $P \longrightarrow \mathrm{I}$ from a predicate P to the unit I. When such a derivation exists, we say that P is provable, and write $\vdash P$.

To explore the theory of proofs, two auxiliary definitions are introduced: linear negation and linear implication. Notice in the syntax in Fig. 1 linear negation applies only to atoms.

▶ **Definition 2.** *Linear negation* is defined by the following function from predicates to predicates.

$$
\begin{array}{llllll}
\overline{\overline{\alpha}} = \alpha & \overline{P \otimes Q} = \overline{P} \parallel \overline{Q} & \overline{P \parallel Q} = \overline{P} \otimes \overline{Q} & \overline{P \oplus Q} = \overline{P} \,\&\, \overline{Q} & \overline{P \,\&\, Q} = \overline{P} \oplus \overline{Q} & \\
\overline{\mathrm{I}} = \mathrm{I} & \overline{P \,;\, Q} = \overline{P} \,;\, \overline{Q} & \overline{\forall x.P} = \exists x.\overline{P} & \overline{\exists x.P} = \forall x.\overline{P} & \overline{Ии x.P} = Э x.\overline{P} & \overline{Э x.P} = Ии x.\overline{P}
\end{array}
$$

Linear implication, written $P \multimap Q$, is defined as $\overline{P} \parallel Q$.

Linear negation defines de Morgan dualities. As in linear logic, the multiplicatives $\otimes$ and $\parallel$ are de Morgan dual; as are the additives $\&$ and $\oplus$, the first-order quantifiers $\exists$ and $\forall$, and the nominal quantifiers $Ии$ and $Э$. As in BV, sequential composition and the unit are self-dual.

$$\mathcal{C}\{\ \overline{\alpha} \parallel \alpha\ \} \longrightarrow \mathcal{C}\{\ \mathrm{I}\ \} \quad \text{(atomic interaction)} \qquad \mathcal{C}\{\ P \parallel (Q \otimes S)\ \} \longrightarrow \mathcal{C}\{\ (P \parallel Q) \otimes S\ \} \quad \text{(switch)}$$

$$\mathcal{C}\{\ (P\,;Q) \parallel (R\,;S)\ \} \longrightarrow \mathcal{C}\{\ (P \parallel R)\,;(Q \parallel S)\ \} \quad \text{(sequence)}$$

$$\mathcal{C}\{\ (P \,\&\, Q) \parallel R\ \} \longrightarrow \mathcal{C}\{\ (P \parallel R) \,\&\, (Q \parallel R)\ \} \quad \text{(external)} \qquad \mathcal{C}\{\ \mathrm{I} \,\&\, \mathrm{I}\ \} \longrightarrow \mathcal{C}\{\ \mathrm{I}\ \} \quad \text{(tidy)}$$

$$\mathcal{C}\{\ (P\,;Q) \,\&\, (R\,;S)\ \} \longrightarrow \mathcal{C}\{\ (P \,\&\, R)\,;(Q \,\&\, S)\ \} \quad \text{(medial)}$$

$$\mathcal{C}\{\ P \oplus Q\ \} \longrightarrow \mathcal{C}\{\ P\ \} \quad \text{(left choice)} \qquad \mathcal{C}\{\ P \oplus Q\ \} \longrightarrow \mathcal{C}\{\ Q\ \} \quad \text{(right choice)}$$

$$\mathcal{C}\{\ \forall x.P \parallel R\ \} \longrightarrow \mathcal{C}\{\ \forall x.(P \parallel R)\ \} \text{ only if } x \# R \quad \text{(extrude1)} \qquad \mathcal{C}\{\ \forall x.\mathrm{I}\ \} \longrightarrow \mathcal{C}\{\ \mathrm{I}\ \} \quad \text{(tidy1)}$$

$$\mathcal{C}\{\ \forall x.(P\,;S)\ \} \longrightarrow \mathcal{C}\{\ \forall x.P\,;\forall x.S\ \} \quad \text{(medial1)} \qquad \mathcal{C}\{\ \exists x.P\ \} \longrightarrow \mathcal{C}\{\ P\{^{v}/_{x}\}\ \} \quad \text{(select1)}$$

$$\mathcal{C}\{\ И x.P \parallel Э x.Q\ \} \longrightarrow \mathcal{C}\{\ И x.(P \parallel Q)\ \} \quad \text{(close)} \qquad \mathcal{C}\{\ И x.\mathrm{I}\ \} \longrightarrow \mathcal{C}\{\ \mathrm{I}\ \} \quad \text{(tidy name)}$$

$$\mathcal{C}\{\ И x.P \parallel R\ \} \longrightarrow \mathcal{C}\{\ И x.(P \parallel R)\ \} \text{ only if } x \# R \quad \text{(extrude new)}$$

$$\mathcal{C}\{\ Э x.P\ \} \longrightarrow \mathcal{C}\{\ И x.P\ \} \quad \text{(fresh)} \qquad \mathcal{C}\{\ И x.Э y.P\ \} \longrightarrow \mathcal{C}\{\ Э y.И x.P\ \} \quad \text{(new wen)}$$

$$\mathcal{C}\{\ И x.(P\,;S)\ \} \longrightarrow \mathcal{C}\{\ И x.P\,;И x.S\ \} \quad \text{(medial new)}$$

$$\mathcal{C}\{\ Э x.P \odot Э x.S\ \} \longrightarrow \mathcal{C}\{\ Э x.(P \odot S)\ \} \text{ where } \odot \in \{\parallel, ;\} \quad \text{(medial wen)}$$

$$\mathcal{C}\{\ Э x.P \odot R\ \} \longrightarrow \mathcal{C}\{\ Э x.(P \odot R)\ \} \text{ where } \odot \in \{\parallel, ;\} \text{ only if } x \# R \quad \text{(left wen)}$$

$$\mathcal{C}\{\ R \odot Э x.Q\ \} \longrightarrow \mathcal{C}\{\ Э x.(R \odot Q)\ \} \text{ where } \odot \in \{\parallel, ;\} \text{ only if } x \# R \quad \text{(right wen)}$$

$$\mathcal{C}\big\{\ \forall x.\mathfrak{O} y.P\ \big\} \longrightarrow \mathcal{C}\big\{\ \mathfrak{O} y.\forall x.P\ \big\} \text{ for } \mathfrak{O} \in \{И, Э\} \quad \text{(all name)}$$

$$\mathcal{C}\big\{\ \mathfrak{O} x.P \,\&\, \mathfrak{O} x.S\ \big\} \longrightarrow \mathcal{C}\big\{\ \mathfrak{O} x.(P \,\&\, S)\ \big\} \text{ for } \mathfrak{O} \in \{И, Э\} \quad \text{(with name)}$$

$$\mathcal{C}\big\{\ \mathfrak{O} x.P \,\&\, R\ \big\} \longrightarrow \mathcal{C}\big\{\ \mathfrak{O} x.(P \,\&\, R)\ \big\} \text{ only if } x \# R \text{ for } \mathfrak{O} \in \{И, Э\} \quad \text{(left name)}$$

$$\mathcal{C}\big\{\ R \,\&\, \mathfrak{O} x.Q\ \big\} \longrightarrow \mathcal{C}\big\{\ \mathfrak{O} x.(R \,\&\, Q)\ \big\} \text{ only if } x \# R \text{ for } \mathfrak{O} \in \{И, Э\} \quad \text{(right name)}$$

Figure 3 Rewrite rules for predicates in system MAV1. Notice the figure is divided into four parts. The first part defines sub-system BV [17]. The first and second parts together define sub-system MAV [20]. The following restrictions are placed on the rules to ensure the system is analytic [8].

- The *switch*, *sequence*, *medial1*, *medial new* and *extrude new* rules are such that $P \not\equiv \mathrm{I}$ and $S \not\equiv \mathrm{I}$.
- The *medial* rule is such that either $P \not\equiv \mathrm{I}$ or $R \not\equiv \mathrm{I}$ and also either $Q \not\equiv \mathrm{I}$ or $S \not\equiv \mathrm{I}$.
- The rules *external*, *extrude1*, *extrude new*, *left wen* and *right wen* are such that $R \not\equiv \mathrm{I}$.

A *derivation* is a sequence of rewrites, where the congruence in Fig. 2 can be applied at any point in a derivation. The length of a derivation involving only the congruence is zero. The length of a derivation involving one rule from Fig. 3 is one. Given a derivation $P \longrightarrow Q$ of length m and another $Q \longrightarrow R$ of length n, the derivation $P \longrightarrow R$ is of length $m + n$. Unless we make it clear in the context that we refer to a specific rule, $\longrightarrow$ is generally used to represent derivations of any length.

The following proposition is simply a reflexivity property of linear implication in MAV1.

▶ **Proposition 3** (Reflexivity). *For any predicate P, $\vdash \overline{P} \parr P$ holds.*

The main result of this paper is the following, which is a generalisation of *cut elimination* to the setting of the calculus of structures.

▶ **Theorem 4** (Cut elimination). *For any predicate P, if $\vdash \mathcal{C}\{ P \otimes \overline{P} \}$ holds, then $\vdash \mathcal{C}\{ \mathrm{I} \}$ holds.*

The above theorem can be stated alternatively by supposing that there is a proof in MAV1 extended with the extra rewrite rule: $\mathcal{C}\{ \mathrm{I} \} \longrightarrow \mathcal{C}\{ P \otimes \overline{P} \}$ (cut). Given such a proof, a new proof can be constructed that uses only the rules of MAV1. In this formulation, we say that *cut* is *admissible*.

The cut elimination proof for the propositional sub-system MAV appears in a companion journal paper [20]. The current paper advances cut-elimination techniques to tackle first-order system MAV1, as achieved by the lemmas in later sections. Before proceeding with the necessary lemmas, we provide a corollary that demonstrates that one of many consequences of cut elimination is indeed that linear implication defines a precongruence — a reflexive transitive relation that holds in any context.

▶ **Corollary 5.** *Linear implication defines a precongruence.*

3 Linear Implication as a Precongruence for Processes-as-Predicates

We highlight connections between MAV1 and the π-calculus. This illustrates the rationale behind design decisions in MAV1. We assume the reader is familiar with the syntax of the π-calculus. For the π-calculus define a processes-as-predicates embedding as follows.

$$[\![x(y).P]\!]_\pi = \exists y.(xy \,;\, [\![P]\!]_\pi) \qquad [\![\overline{x}y.P]\!]_\pi = \overline{xy} \,;\, [\![P]\!]_\pi \qquad [\![P \parr Q]\!]_\pi = [\![P]\!]_\pi \parr [\![Q]\!]_\pi$$
$$[\![\nu x.P]\!]_\pi = \mathit{И}x.[\![P]\!]_\pi \qquad [\![P + Q]\!]_\pi = [\![P]\!]_\pi \oplus [\![Q]\!]_\pi \qquad [\![1]\!]_\pi = \mathrm{I}$$

Notice action prefixes are captured using atoms consisting of pairs of first-order variables. We consider preorders that do not observe τ actions and are *termination sensitive* [1, 41], hence distinguish between successful termination and deadlock. Successful termination is indicated by a process 1, differing from 0 typical of process calculi. For example $P + 1$ represents the process that may behave like P or may successfully terminate, contrasting to $P + 0$ which only may only proceed as P. This distinction is useful for modelling protocols; for example, we can choose to perform no action in certain executions to avoid deadlocking. Furthermore, 1 as a primitive process matches the unit inherited from BV. Otherwise, the semantics are standard for the π-calculus.

Define labelled transitions for the π-calculus by the following deductive system, plus the symmetric rules for parallel composition and choice. Function $n(.)$ is such that $n(x(y)) = n(\overline{x}(y)) = n(\overline{x}y) = \{x, y\}$, and $x \# P$ is such that x is fresh for P where $z(x).P$ and $\nu x.P$ bind x in P.

$$\frac{}{x(y).P \xrightarrow{x(y)} P} \qquad \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad \frac{P \xrightarrow{A} Q}{P + R \xrightarrow{A} Q} \qquad \frac{}{P + 1 \xrightarrow{\tau} 1}$$

$$\frac{P \xrightarrow{A} Q}{\nu x.P \xrightarrow{A} \nu x.Q}\ x \notin \mathrm{n}(A)\ \ddagger \qquad \frac{P \xrightarrow{A} Q}{P \parr R \xrightarrow{A} Q \parr R}\ \begin{array}{l}\text{if } A = x(y) \text{ or } A = \overline{x}(y)\\ \text{then } y \# R\end{array}$$

$$\frac{P \xrightarrow{\overline{x}y} Q}{\nu y.P \xrightarrow{\overline{x}(y)} Q}\ x \neq y \qquad \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{x(z)} Q'}{P \parr Q \xrightarrow{\tau} \nu z.(P' \parr Q')}\ \dagger \qquad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P \parr Q \xrightarrow{\tau} P' \parr Q'\{{}^{y}/_{z}\}}$$

Define (symbolic weak) completed traces inductively as follows.

- For any process P formed using only the unit 1, name restriction and parallel composition (i.e. with no actions or choice) P has the completed trace I.
- If $P \xrightarrow{x(y)} Q$ and Q has completed trace tr then P has completed trace $\forall y.(\overline{xy}\,;tr)$.
- If $P \xrightarrow{\overline{x}(y)} Q$ and Q has completed trace tr then P has completed trace $\Im y.(xy\,;tr)$.
- If $P \xrightarrow{\overline{x}y} Q$ and Q has completed trace tr then P has completed trace $xy\,;tr$.
- If $P \xrightarrow{\tau} Q$ and Q has completed trace tr then P has completed trace tr.

Observe that deadlocked processes have no completed trace. Contrast for example $\nu x.x(y) \parallel 1$ and $\nu x.x(y) + 1$, where the former has no completed trace but the later has completed trace I.

Interestingly, *equivariance*, in Fig. 2, is a design decision in the sense that cut elimination is still provable for a MAV1 without *equivariance*. However, *equivariance* is a requirement for modelling private names in process calculi. Consider π-calculus process $\nu y.\nu x.(\overline{z}x.\overline{w}y)$ and the completed trace $\Im x.(zx\,;\Im y.wy)$ that outputs a fresh name on channel z then a separate fresh name on channel w. $\vdash \mathit{И}y.\mathit{И}x.(\overline{zx}\,;\overline{wy}) \parallel \Im x.(zx\,;\Im y.wy)$ is provable only with *equivariance*. Hence *equivariance* is necessary for the following proposition.

▶ **Proposition 6.** *If a process P has completed trace tr then $\vdash [\![P]\!]_\pi \parallel tr$.*

Proof. The proof follows by induction over the structure of the derivation for a labelled transition. We present only two inductive cases, for the communication of an input and bound output (†), and the extrusion of a bound output over a distinct private name (‡).

Assume the induction hypotheses, $\vdash [\![P]\!]_\pi \parallel \forall z\left(\overline{xz}\,;\overline{[\![P']\!]_\pi}\right)$ and $\vdash [\![Q]\!]_\pi \parallel \Im z\left(xz\,;\overline{[\![Q']\!]_\pi}\right)$. Implication $\vdash \left([\![P]\!]_\pi \parallel \forall z.\left(\overline{xz}\,;\overline{[\![P']\!]_\pi}\right)\right) \otimes \left([\![Q]\!]_\pi \parallel \Im z.\left(xz\,;\overline{[\![Q']\!]_\pi}\right)\right) \multimap [\![P]\!]_\pi \parallel [\![Q]\!]_\pi \parallel \Im z.\left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right)$ is provable as follows, using Proposition 3.

$$
\begin{array}{ll}
\left(\overline{[\![P]\!]_\pi} \otimes \exists z.\left(xz\,;[\![P']\!]_\pi\right)\right) \parallel \left(\overline{[\![Q]\!]_\pi} \otimes \mathit{И}z.\left(\overline{xz}\,;[\![Q']\!]_\pi\right)\right) \parallel [\![P]\!]_\pi \parallel [\![Q]\!]_\pi \parallel \Im z.\left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right) & \\
\longrightarrow \left(\left(\overline{[\![P]\!]_\pi} \parallel [\![P]\!]_\pi\right) \otimes \exists z\left(xz\,;[\![P']\!]_\pi\right)\right) \parallel \left(\left(\overline{[\![Q]\!]_\pi} \parallel [\![Q]\!]_\pi\right) \otimes \mathit{И}z\left(\overline{xz}\,;[\![Q']\!]_\pi\right)\right) \parallel \Im z.\left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right) & \\
\longrightarrow \exists z.\left(xz\,;[\![P']\!]_\pi\right) \parallel \mathit{И}z.\left(\overline{xz}\,;[\![Q']\!]_\pi\right) \parallel \Im z.\left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right) & \text{switch and Proposition 3} \\
\longrightarrow \exists z.\left(xz\,;[\![P']\!]_\pi\right) \parallel \mathit{И}z.\left(\left(\overline{xz}\,;[\![Q']\!]_\pi\right) \parallel \left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right)\right) & \text{close} \\
\longrightarrow \mathit{И}z.\left(\exists z.\left(xz\,;[\![P']\!]_\pi\right) \parallel \left(\overline{xz}\,;[\![Q']\!]_\pi\right) \parallel \left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right)\right) & \text{extrude new} \\
\longrightarrow \mathit{И}z.\left(\left(xz\,;[\![P']\!]_\pi\right) \parallel \left(\overline{xz}\,;[\![Q']\!]_\pi\right) \parallel \left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right)\right) & \text{select1} \\
\longrightarrow \mathit{И}z.\left((xz \parallel \overline{xz})\,;\left([\![P']\!]_\pi \parallel [\![Q']\!]_\pi \parallel \left(\overline{[\![P']\!]_\pi} \otimes \overline{[\![Q']\!]_\pi}\right)\right)\right) & \text{sequence} \\
\longrightarrow \mathit{И}z.\left((xz \parallel \overline{xz})\,;\left(\left([\![P']\!]_\pi \parallel \overline{[\![P']\!]_\pi}\right) \otimes \left([\![Q']\!]_\pi \parallel \overline{[\![Q']\!]_\pi}\right)\right)\right) & \text{switch} \\
\longrightarrow \mathit{И}z.\mathrm{I} \longrightarrow \mathrm{I} & \text{Proposition 3 and tidy new}
\end{array}
$$

Hence by Theorem 4, $\vdash [\![P \parallel Q]\!]_\pi \parallel \overline{[\![\nu z.(P' \parallel Q')]\!]_\pi}$, as required.

As the induction hypothesis, assume that $\vdash [\![P]\!]_\pi \parallel \Im z.\left(xz\,;\overline{[\![Q]\!]_\pi}\right)$ holds, where y is such that $x \neq y$ and $z \neq y$. Thereby the following proof can be constructed directly as required.

$$
\begin{array}{rcll}
[\![\nu y.P]\!]_\pi \parallel \Im z.\left(xz\,;\overline{[\![\nu y.Q]\!]_\pi}\right) = & \mathit{И}y.[\![P]\!]_\pi \parallel \Im z.\left(xz\,;\Im y.\overline{[\![Q]\!]_\pi}\right) & & \text{by definition} \\
\longrightarrow & \mathit{И}y.[\![P]\!]_\pi \parallel \Im z.\Im y.\left(xz\,;\overline{[\![Q]\!]_\pi}\right) & & \text{right wen} \\
\equiv & \mathit{И}y.[\![P]\!]_\pi \parallel \Im y.\Im z.\left(xz\,;\overline{[\![Q]\!]_\pi}\right) & & \text{equivariance} \\
\longrightarrow & \mathit{И}y.\left([\![P]\!]_\pi \parallel \Im z.\left(xz\,;\overline{[\![Q]\!]_\pi}\right)\right) \longrightarrow \mathit{И}y.\mathrm{I} \longrightarrow \mathrm{I} & & \text{induction and tidy}
\end{array}
$$

The proof concludes by inductively applying Theorem 4 to each transition forming a trace. □

The above proposition also holds for a processes-as-predicates embedding for the πI-calculus [32], where input of private names is such that $[\![x(y).P]\!]_{\pi I} = \Im y.(xy\,;[\![P]\!]_{\pi I})$ and output of private names is such that $[\![\overline{x}(y).P]\!]_{\pi I} = \mathit{И}y.(\overline{xy}\,;[\![P]\!]_{\pi I})$. The labelled transition

system for the πI-calculus, is such that $\overline{x}(y).P \xrightarrow{\overline{x}(y)} P$ and complete traces are such that, if $P \xrightarrow{x(y)} Q$ and Q has completed trace tr then P has completed trace $И y.(\overline{xy}\,;tr)$. We envision models of processes exploiting more of the expressive power of MAV1, such as the primitives employed for modelling session types using MAV [20].

For a basic process calculus with only parallel composition and prefix the converse to Proposition 6 is known to hold [7]. The converse direction for the π-calculus relies on techniques beyond cut elimination, hence will receive separate attention in a future paper.

Linear implication v.s. completed traces. We re-emphasise that the above completed trace semantics is a minimal justification for design decisions. Completed traces and linear implication are at the opposite ends of the linear-time/branching-time [39] and interleaving/-causal [33] spectra.

A characteristic distinction between linear-time and branching-time preorders is in how processes of the form $\alpha.(P+Q)$ and $\alpha.P+\alpha.Q$ are related. For completed traces they are equivalent but for linear implication, only the direction $\vdash [\![\alpha.(P+Q)]\!]_\pi \multimap [\![\alpha.P+\alpha.Q]\!]_\pi$ holds. Hence linear implication is at the branching-time end of the spectrum.

Simulation preorders are also at the branching-time end of the spectrum. However, many simulation preorders interleave events as characterised by *expansion* rules [26] which equate processes with identical interleavings. For linear implication, expansion holds in one direction only. For example, processes $\alpha.\alpha$ and $\alpha \parallel \alpha$ have the same interleavings, but linear implication holds only in the direction $\vdash [\![\alpha.\alpha]\!]_\pi \multimap [\![\alpha \parallel \alpha]\!]_\pi$. As a further example, processes $\alpha.(\alpha.\beta \parallel \beta)$ and $\alpha.\beta \parallel \alpha.\beta$ have identical interleavings but linear implication holds only in the direction established by the following proof.

$$
\begin{array}{rcll}
[\![\alpha.(\alpha.\beta \parallel \beta)]\!]_\pi \multimap [\![\alpha.\beta \parallel \alpha.\beta]\!]_\pi = & \big(\overline{\alpha}\,;\big(\big(\overline{\alpha}\,;\overline{\beta}\big)\otimes\overline{\beta}\big)\big) \parallel (\alpha\,;\beta) \parallel (\alpha\,;\beta) & \text{by definition} \\
\longrightarrow & \big((\overline{\alpha} \parallel \alpha)\,;\big(\big(\big(\overline{\alpha}\,;\overline{\beta}\big)\otimes\overline{\beta}\big) \parallel \beta\big)\big) \parallel (\alpha\,;\beta) & \text{sequence} \\
\longrightarrow & \big(\big(\overline{\alpha}\,;\overline{\beta}\big)\otimes\overline{\beta}\big) \parallel \beta \parallel (\alpha\,;\beta) & \text{interaction} \\
\longrightarrow & \big(\overline{\alpha}\,;\overline{\beta}\big)\otimes\big(\overline{\beta} \parallel \beta\big) \parallel (\alpha\,;\beta) & \text{switch} \\
\longrightarrow & \big(\overline{\alpha}\,;\overline{\beta}\big) \parallel (\alpha\,;\beta) & \text{interaction} \\
\longrightarrow & (\overline{\alpha} \parallel \alpha)\,;\big(\overline{\beta} \parallel \beta\big) & \text{sequence} \\
\longrightarrow & \mathrm{I} & \text{interaction}
\end{array}
$$

By not identifying interleavings, we argue that linear implication respects the causal order of events. Existing notions of simulation that respect the causal order of events, such as history preserving simulation [4, 40, 41], have not yet been extended to the setting with private names. Thereby MAV1 enables us to objectively explore the properties of fresh names in this part of the spectrum. For example, the implication $\vdash [\![\nu x.\,(P \parallel Q\{{}^{y}/_{z}\})]\!]_\pi \multimap [\![\nu x.(\overline{x}y.P \parallel x(z).Q)]\!]_\pi$ is provable. However, the converse is not provable. Intuitively, although no other process can communicate on channel x, the processes can be distinguished by a network partition interrupting communication the private channel.

Observe that no bisimulation can be complete with respect to logical equivalence. To see this observe that the embeddings $[\![\alpha.(\beta+\gamma)+\alpha.\beta]\!]_\pi$ and $[\![\alpha.(\beta+\gamma)]\!]_\pi$ are logically equivalent; whereas any bisimulation distinguishes these processes. Logical equivalence, as with *mutual simulation* [39], is checked using a preorder in each direction, by proving predicate $(P \multimap Q)\,\&\,(Q \multimap P)$.

Questions formally relating observational preorders and linear implication rely on cut elimination for MAV1. This leads us to the cut elimination result of this paper.

4 Logical Properties of the Pair of Nominal Quantifiers

This section offers deeper insight into the nature of the nominal quantifiers И and Э, and sketches the cut elimination proof.[1] Cut elimination (Theorem 4) is achieved by advancing methods developed in the propositional sub-system MAV [20]. A direct proof of *co-rule elimination* for universal quantifiers (Lemma 11) simplifies *splitting* (Lemma 16), and *context reduction* (Lemma 20) is adapted for existential quantifiers by working up-to substitutions. Lemmas check the interplay between nominal quantifiers and other connectives, as illustrated by the discussion in the following subsection.

4.1 Discussion on the Rules for Nominal Quantifiers

The rules for the nominal quantifiers *new* И and *wen* Э require some justification. The *close* and *tidy name* rules ensure the reflexivity of implication for nominal quantifiers. Using the *extrude new* rule (and Proposition 3) we can establish the following proof, and its dual statement $\vdash \Im x.P \multimap \exists x.P$.

$$\forall x.P \multimap Ux.P = \exists x.\overline{P} \parallel Ux.P \longrightarrow Ux.\big(\exists x.\overline{P} \parallel P\big) \longrightarrow Ux.\big(\overline{P} \parallel P\big) \longrightarrow Ux.\mathrm{I} \longrightarrow \mathrm{I}$$

Using the *fresh* rule we can establish the following implication between *new* and *wen*.

$$Ux.P \multimap \Im x.P = \Im x.\overline{P} \parallel \Im x.P \longrightarrow Ux.\overline{P} \parallel \Im x.P \longrightarrow Ux.\big(\overline{P} \parallel P\big) \longrightarrow Ux.\mathrm{I} \longrightarrow \mathrm{I}$$

This completes the chain $\vdash \forall x.P \multimap Ux.P$, $\vdash Ux.P \multimap \Im x.P$ and $\vdash \Im x.P \multimap \exists x.P$. These linear implications are strict unless $x \# P$, in which case, for $\eth \in \{\forall, \exists, U, \Im\}$, $\eth x.P$ is logically equivalent to P. For example, using the *fresh* and *extrude new* rules, the following holds given $x \# P$.

$$Ux.P \multimap P = \Im x.\overline{P} \parallel P \longrightarrow Ux.\overline{P} \parallel P \longrightarrow Ux.\big(\overline{P} \parallel P\big) \longrightarrow Ux.\mathrm{I} \longrightarrow \mathrm{I}$$

Alternatively, the *extrude new* and *fresh* rules could be replaced by extending the congruence with $Ux.P \equiv P \equiv \Im x.P$, where $x \# P$. However, this has the disadvantage that an arbitrary number of nominal quantifiers can be introduced during proof search thereby jeopardising analyticity [8].

The *medial new* rule is particular to handling nominal quantifiers in the presence of the self-dual non-commutative operator *seq*. To see why this rule cannot be excluded, consider the following.

$$(a \,;\, \Im x.(b \,;\, c)) \otimes (d \,;\, \Im x.(e \,;\, f)) \multimap (a \,;\, \exists x.b \,;\, \exists x.c) \otimes (d \,;\, \exists x.e \,;\, \exists x.f)$$
$$(a \,;\, \exists x.b \,;\, \exists x.c) \otimes (d \,;\, \exists x.e \,;\, \exists x.f) \multimap ((a \,;\, \exists x.b) \otimes (d \,;\, \exists x.e)) \,;\, (\exists x.c \otimes \exists x.f)$$

Without using the *medial new* rule, the above predicates are provable but the following predicate would not be provable; hence cut elimination cannot hold without the *medial new* rule.

$$(a \,;\, \Im x.(b \,;\, c)) \otimes (d \,;\, \Im x.(e \,;\, f)) \multimap ((a \,;\, \exists x.b) \otimes (d \,;\, \exists x.e)) \,;\, (\exists x.c \otimes \exists x.f)$$

[1] Details of proofs appear in a technical report at: `http://iit.iit.tuiasi.ro/TR/reports/fml1502.pdf`

In contrast, with the *medial new* rule the above predicate is provable, verified by the following proof.

$$
\begin{array}{ll}
& \big(\overline{a}\,;\mathit{И}x.\,\big(\overline{b}\,;\overline{c}\big)\big) \parallel \big(\overline{d}\,;\mathit{И}x.\,\big(\overline{e}\,;\overline{f}\big)\big) \parallel ((a\,;\exists x.b)\otimes(d\,;\exists x.e))\,;(\exists x.c\otimes\exists x.f)\\
\longrightarrow & \big(\overline{a}\,;\mathit{И}x.\overline{b}\,;\mathit{И}x.\overline{c}\big) \parallel \big(\overline{d}\,;\mathit{И}x.\overline{e}\,;\mathit{И}x.\overline{f}\big) \parallel ((a\,;\exists x.b)\otimes(d\,;\exists x.e))\,;(\exists x.c\otimes\exists x.f)\\
\longrightarrow & \big(\big(\overline{a}\,;\mathit{И}x.\overline{b}\big) \parallel \big(\overline{d}\,;\mathit{И}x.\overline{e}\big)\big)\,;\big(\mathit{И}x.\overline{c} \parallel \mathit{И}x.\overline{f}\big) \parallel ((a\,;\exists x.b)\otimes(d\,;\exists x.e))\,;(\exists x.c\otimes\exists x.f)\\
\longrightarrow & \big(\big(\overline{a}\,;\mathit{И}x.\overline{b}\big) \parallel \big(\overline{d}\,;\mathit{И}x.\overline{e}\big) \parallel ((a\,;\exists x.b)\otimes(d\,;\exists x.e))\big)\,;\big(\mathit{И}x.\overline{c} \parallel \mathit{И}x.\overline{f} \parallel (\exists x.c\otimes\exists x.f)\big)\\
\longrightarrow & \big(\big(\big(\overline{a}\,;\mathit{И}x.\overline{b}\big) \parallel (a\,;\exists x.b)\big)\otimes\big(\big(\overline{d}\,;\mathit{И}x.\overline{e}\big) \parallel (d\,;\exists x.e)\big)\big)\,;\big((\mathit{И}x.\overline{c} \parallel \exists x.c)\otimes\big(\mathit{И}x.\overline{f} \parallel \exists x.f\big)\big)\\
\longrightarrow & \big(\big((\overline{a} \parallel a)\,;\big(\mathit{И}x.\overline{b} \parallel \exists x.b\big)\big)\otimes\big(\big(\overline{d} \parallel d\big)\,;(\mathit{И}x.\overline{e} \parallel \exists x.e)\big)\big)\,;\big((\mathit{И}x.\overline{c} \parallel \exists x.c)\otimes\big(\mathit{И}x.\overline{f} \parallel \exists x.f\big)\big)\\
\longrightarrow & \big(\big((\overline{a} \parallel a)\,;\mathit{И}x.\big(\overline{b} \parallel \exists x.b\big)\big)\otimes\big(\big(\overline{d} \parallel d\big)\,;\mathit{И}x.(\overline{e} \parallel \exists x.e)\big)\big)\,;\big(\mathit{И}x.(\overline{c} \parallel \exists x.c)\otimes\mathit{И}x.\big(\overline{f} \parallel \exists x.f\big)\big)\\
\longrightarrow & \big(\big((\overline{a} \parallel a)\,;\mathit{И}x.\big(\overline{b} \parallel b\big)\big)\otimes\big(\big(\overline{d} \parallel d\big)\,;\mathit{И}x.(\overline{e} \parallel e)\big)\big)\,;\big(\mathit{И}x.(\overline{c} \parallel c)\otimes\mathit{И}x.\big(\overline{f} \parallel f\big)\big)\\
\longrightarrow & (\mathit{И}x.\mathrm{I}\otimes\mathit{И}x.\mathrm{I})\,;(\mathit{И}x.\mathrm{I}\otimes\mathit{И}x.\mathrm{I}) \longrightarrow \mathrm{I}
\end{array}
$$

Notice that the above proof uses only the *medial new*, *extrude new* and *tidy name* rules for nominals. These three rules are of the same form as the rules for universal quantifiers, hence the same argument holds for the necessity of the *medial1* rule.

Including the *medial new* rule forces the *medial wen* rule to be included. To see this observe that $(\mathit{И}x.a\,;\mathit{И}x.b)\otimes(\mathit{И}x.c\,;\mathit{И}x.d) \multimap \mathit{И}x.(a\,;b)\otimes\mathit{И}x.(c\,;d)$ and $\mathit{И}x.(a\,;b)\otimes\mathit{И}x.(c\,;d) \multimap \mathit{И}x.((a\,;b)\otimes(c\,;d))$ are provable. However, without the *medial wen* rule the following implication is not provable, which would contradict the main cut elimination result of this paper.

$$(\mathit{И}x.a\,;\mathit{И}x.b)\otimes(\mathit{И}x.c\,;\mathit{И}x.d) \multimap \mathit{И}x.((a\,;b)\otimes(c\,;d))$$

Fortunately, including the *medial wen* rule ensures that the above implication is provable. A similar argument justifies the inclusion of the *left wen* and *right wen* rules.

As with focussed proof search [3, 10], assigning a positive or negative polarity to operators explains certain distributivity properties. Consider $\parallel$, &, $\forall$ and И to be negative operators, and $\otimes$, $\oplus$, $\exists$ and $\Im$ to be positive operators, where *seq* is neuter. The negative quantifier И distributes over all positive operators. Considering positive operator *times* for example, $\vdash \mathit{И}x.\alpha\otimes\mathit{И}x.\beta \multimap \mathit{И}x.\,(\alpha\otimes\beta)$ holds but the converse implication does not hold. Furthermore, assuming x appears free in α and β, $\Im x.\alpha\otimes\Im x.\beta$ and $\Im x.\,(\alpha\otimes\beta)$ are unrelated by linear implication. Dually, for the negative operator *par* the only distributivity property that holds for nominal quantifiers is $\vdash \Im x.\,(\alpha \parallel \beta) \multimap \Im x.\alpha \parallel \Im x.\beta$. The *new wen* rule completes this picture of *new* distributing over positive operators and *wen* distributing over negative operators. From the perspective of embedding name-passing process calculi in logic, the above distributivity properties of *new* and *wen* suggest that processes should be encoded using negative operators И and $\parallel$ for private names and parallel composition (or perhaps dually, using positive operators $\Im$ and $\otimes$), so as to avoid private names distributing over parallel composition, which we have shown to be problematic in the introduction.

The control of distributivity exercised by *new* and *wen* contrast with the situation for universal and existential quantifiers, where $\exists$ commutes in one direction over all operators and $\forall$ commutes with all operators in the opposite direction. For a system with *equivariance* some of these distributivity properties for И over & and $\forall$ are explicit rules *all name*, *with name*, *left name*, *right name*. These rules allow И quantifiers to propagate to the front of certain contexts. In the sense of control of distributivity, *new* and *wen* behave more like multiplicatives than additives, but are unrelated to multiplicative quantifiers in the logic of bunched implications [27].

4.2 Preliminary Lemmas and Killing Contexts

We extend a trick employed for MAV [20] to MAV1 where *with* & and *all* ∀ receive a more direct treatment than other operators. The proof for *with* has a knock on effect for the nominal quantifiers requiring some vacuous *new* and *wen* quantifiers to be removed; while the proof for *all* requires closure of rules under substitution of terms for variables. This leads to the following five lemmas, established directly by functions over predicates.

▶ **Lemma 7** (Substitution). *If* $P \longrightarrow Q$ *holds then* $P\{^{v}/_{x}\} \longrightarrow Q\{^{v}/_{x}\}$ *holds.*

▶ **Lemma 8** (Vacuous new). *If* $\vdash \mathcal{C}\{\ И x.P\ \}$ *holds and* $x \# P$ *then* $\vdash \mathcal{C}\{\ P\ \}$ *holds.*

▶ **Lemma 9** (Vacuous wen). *If* $\vdash \mathcal{C}\{\ \Im x.P\ \}$ *holds and* $x \# P$ *then* $\vdash \mathcal{C}\{\ P\ \}$ *holds.*

▶ **Lemma 10** (Branching). *If* $\vdash \mathcal{C}\{\ P \,\&\, Q\ \}$ *holds then both* $\vdash \mathcal{C}\{\ P\ \}$ *and* $\vdash \mathcal{C}\{\ Q\ \}$ *hold.*

▶ **Lemma 11** (Universal). *If* $\vdash \mathcal{C}\{\ \forall x.P\ \}$ *holds then, for all terms* v, $\vdash \mathcal{C}\{\ P\{^{v}/_{x}\}\ \}$ *holds.*

We require a restricted form of context called a killing context (terminology is from [10]). A killing context is a context with one or more holes, defined as follows.

▶ **Definition 12.** A killing context is a context defined by the following grammar.

$$\mathcal{T}\{\ \} ::= \{\ \cdot\ \} \mid \mathcal{T}\{\ \} \,\&\, \mathcal{T}\{\ \} \mid \forall x.\mathcal{T}\{\ \} \mid И x.\mathcal{T}\{\ \}$$

In the above, $\{\ \cdot\ \}$ is a hole into which any predicate can be plugged. An n-ary killing context is a killing context in which n holes appear.

A killing context represents a context that cannot in general be removed until all other rules in a proof have been applied, hence the corresponding *tidy* rules are suspended until the end of a proof. A killing context has properties that are applied frequently in proofs, characterised by the following lemma.

▶ **Lemma 13.** *For any killing context* $\mathcal{T}\{\ \}$, $\vdash \mathcal{T}\{\ \mathrm{I}, \ldots, \mathrm{I}\ \}$ *holds; and, assuming the variables of* $\mathcal{T}\{\ \}$ *and the free variables of* P *are disjoint,* $P \parallel \mathcal{T}\{\ Q_1, Q_2, \ldots Q_n\ \} \longrightarrow \mathcal{T}\{\ P \parallel Q_1, P \parallel Q_2, \ldots P \parallel Q_n\ \}$.

For readability of large predicates involving an n-ary killing context, we introduce the notation $\mathcal{T}\{\ Q_i \colon 1 \leq i \leq n\ \}$ as shorthand for $\mathcal{T}\{\ Q_1, Q_2, \ldots, Q_n\ \}$; and $\mathcal{T}\{\ Q_i \colon i \in I\ \}$ for a family of predicates indexed by finite subset of natural numbers I. Killing contexts also satisfy the following property that is necessary for handling the *seq* operator, which interacts subtly with killing contexts.

▶ **Lemma 14.** *Assume that* I *is a finite subset of natural numbers,* P_i *and* Q_i *are predicates, for* $i \in I$*, and* $\mathcal{T}\{\ \}$ *is a killing context. There exist killing contexts* $\mathcal{T}^0\{\ \}$ *and* $\mathcal{T}^1\{\ \}$ *and sets of natural numbers* $J \subseteq I$ *and* $K \subseteq I$ *such that the following derivation holds.*

$$\mathcal{T}\{\ P_i \,;\, Q_i \colon i \in I\ \} \longrightarrow \mathcal{T}^0\{\ P_j \colon j \in J\ \} \,;\, \mathcal{T}^1\{\ Q_k \colon k \in K\ \}$$

The following lemma checks that *wen* quantifiers can propagate to the front of a killing context.

▶ **Lemma 15.** *Consider an* n*-ary killing context* $\mathcal{T}\{\ \}$ *and predicates such that* $x \# P_i$ *and either* $P_i = \Im x.Q_i$ *or* $P_i = Q_i$*, for* $1 \leq i \leq n$*. If for some* i *such that* $1 \leq i \leq n$*,* $P_i = \Im x.Q_i$*, then* $\mathcal{T}\{\ P_1, P_2, \ldots P_n\ \} \longrightarrow \Im x.\mathcal{T}\{\ Q_1, Q_2, \ldots, Q_n\ \}$.

4.3 The Splitting Technique for Simulating the Sequent Calculus

The technique called splitting [17, 18] generalises the application of rules in the sequent calculus. In the sequent calculus, any root connective in a sequent can be selected and some rule for that connective can be applied. In this setting, a sequent corresponds to a *shallow context* of the form $\{\ \cdot\ \} \parallel Q$. Splitting proves that a distinguished *principal predicate* P in a shallow context $\{\ P\ \} \parallel Q$ can always be rewritten to a form such that a rule for the root connective of the principal predicate may be applied. The cases for *times*, *seq*, *new* and *wen* are treated together in a Lemma 16. These operators give rise to *commutative cases*, where rules for these operators can permute with any principal predicate, swapping the order of rules in a proof. *Principal cases* are where the root connective of the principal predicate is directly involved in the bottommost rule of a proof. As with MAV [20], the *principal cases* for *seq* are challenging, demanding Lemma 14. The principal case induced by *medial new* demands Lemma 15. The cases where two nominal quantifiers commute are also interesting, particularly where the case arises due to *equivariance*.

▶ **Lemma 16** (Core Splitting). *The following statements hold.*

- *If* $\vdash (P \otimes Q) \parallel R$, *then there exist predicates* V_i *and* W_i *such that* $\vdash P \parallel V_i$ *and* $\vdash Q \parallel W_i$, *where* $1 \leq i \leq n$, *and* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $R \longrightarrow \mathcal{T}\{\ V_1 \parallel W_1, V_2 \parallel W_2, \ldots, V_n \parallel W_n\ \}$ *and if* x *appears in* $\mathcal{T}\{\ \}$ *then* $x \# (P \otimes Q)$.
- *If* $\vdash (P\,;Q) \parallel R$, *then there exist predicates* V_i *and* W_i *such that* $\vdash P \parallel V_i$ *and* $\vdash Q \parallel W_i$, *where* $1 \leq i \leq n$, *and* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $R \longrightarrow \mathcal{T}\{\ V_1\,;W_1, V_2\,;W_2, \ldots, V_n\,;W_n\ \}$ *and if* x *appears in* $\mathcal{T}\{\ \}$ *then* $x \# (P\,;Q)$.
- *If* $\vdash \text{И}x.P \parallel Q$, *then there exist predicates* V *and* W *where* $x \# V$ *and* $\vdash P \parallel W$ *and either* $V = W$ *or* $V = \ni x.W$, *such that derivation* $Q \longrightarrow V$ *holds.*
- *If* $\vdash \ni x.P \parallel Q$, *then there exist predicates* V *and* W *where* $x \# V$ *and* $\vdash P \parallel W$ *and either* $V = W$ *or* $V = \text{И}x.W$, *such that derivation* $Q \longrightarrow V$ *holds.*

Furthermore, for all $1 \leq i \leq n$, *in the first two cases the size of the proofs* [2] *of* $P \parallel V_i$ *and* $Q \parallel W_i$ *are bounded above by the size of the proofs of* $(P \otimes Q) \parallel R$ *and* $(P\,;Q) \parallel R$. *In the third and fourth cases, the size of the proof* $P \parallel W$ *is bounded above by the size of the proofs of* $\text{И}x.P \parallel Q$ *and* $\ni x.P \parallel Q$.

The final three splitting lemmas mainly involve checking commutative cases, which follow a pattern.

▶ **Lemma 17.** *If* $\vdash \exists x.P \parallel Q$, *then there exist predicates* V_i *and values* v_i *such that* $\vdash P\{^{v_i}/_x\} \parallel V_i$, *where* $1 \leq i \leq n$, *and* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $Q \longrightarrow \mathcal{T}\{\ V_1, V_2, \ldots, V_n\ \}$ *and if* y *appears in* $\mathcal{T}\{\ \}$ *then* $y \# (\exists x.P)$.

▶ **Lemma 18.** *The following statements hold, for any atom* α, *where if* x *appears in* $\mathcal{T}\{\ \}$ *then* $x \# \alpha$.

- *If* $\vdash \overline{\alpha} \parallel Q$, *then there exist* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $Q \longrightarrow \mathcal{T}\{\ \alpha, \alpha, \ldots, \alpha\ \}$.
- *If* $\vdash \alpha \parallel Q$, *then there exist* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $Q \longrightarrow \mathcal{T}\{\ \overline{\alpha}, \overline{\alpha}, \ldots, \overline{\alpha}\ \}$.

▶ **Lemma 19.** *If* $\vdash (P \oplus Q) \parallel R$, *then there exist predicates* W_i *such that either* $\vdash P \parallel W_i$ *or* $\vdash Q \parallel W_i$ *where* $1 \leq i \leq n$, *and* n*-ary killing context* $\mathcal{T}\{\ \}$ *such that* $R \longrightarrow \mathcal{T}\{\ W_1, W_2, \ldots, W_n\ \}$ *and if* x *appears in* $\mathcal{T}\{\ \}$ *then* $x \# (P \oplus Q)$.

[2] For the multiset measure of the size of a proof see `http://iit.iit.tuiasi.ro/TR/reports/fml1502.pdf`.

4.4 Context Reduction and the Admissibility of Co-rules

Splitting is always performed in a *shallow context*, i.e. a hole directly inside a parallel composition. Context reduction extends rules simulated by splitting to any context.

▶ **Lemma 20** (Context reduction). *If* $\vdash P\sigma \parallel R$ *yields that* $\vdash Q\sigma \parallel R$*, for any predicate* R *and substitution of terms for variables* σ*, then* $\vdash \mathcal{C}\{\ P\ \}$ *yields* $\vdash \mathcal{C}\{\ Q\ \}$*, for any context* $\mathcal{C}\{\ \}$*.*

The subtlety of context reduction is to accommodate *plus* and *some* by the following stronger induction invariant: If $\vdash \mathcal{C}\{\ T\ \}$, then there exist predicates U_i and substitutions σ_i such that $\vdash T\sigma_i \parallel U_i$, for $1 \leq i \leq n$; and n-ary killing context $\mathcal{T}\{\ \}$ such that for any predicate V there exist W_i such that either $W_i = V\sigma_i \parallel U_i$ or $W_i = \mathrm{I}$, for $1 \leq i \leq n$, and the following holds: $\mathcal{C}\{\ V\ \} \longrightarrow \mathcal{T}\{\ W_1, W_2, \ldots, W_n\ \}$.

For every rule there is a *co-rule*, where for rule $P \longrightarrow Q$, the co-rule is of the form $\overline{Q} \longrightarrow \overline{P}$. For example *close* has co-rule $\mathcal{C}\{\ \Im x.(P \otimes Q)\ \} \longrightarrow \mathcal{C}\{\ \Im x.P \otimes И x.Q\ \}$ and *extrude1* has co-rule if $x \# Q$ then $\mathcal{C}\{\ \exists x.(P \otimes Q)\ \} \longrightarrow \mathcal{C}\{\ \exists x.P \otimes Q\ \}$. The following eight lemmas establish that a co-rule is admissible in MAV1. In each case, the proof proceeds by applying splitting in a shallow context, forming a new proof, and finally applying Lemma 20.

▶ **Lemma 21** (Co-close). *If* $\vdash \mathcal{C}\{\ \Im x.P \otimes И x.Q\ \}$ *holds then* $\vdash \mathcal{C}\{\ \Im x.(P \otimes Q)\ \}$ *holds.*

▶ **Lemma 22** (Co-tidy name). *If* $\vdash \mathcal{C}\{\ \Im x.\mathrm{I}\ \}$ *holds then* $\vdash \mathcal{C}\{\ \mathrm{I}\ \}$ *holds.*

▶ **Lemma 23** (Co-extrude1). *If* $x \# Q$ *and* $\vdash \mathcal{C}\{\ \exists x.P \otimes Q\ \}$ *holds then* $\vdash \mathcal{C}\{\ \exists x.(P \otimes Q)\ \}$ *holds.*

▶ **Lemma 24** (Co-tidy1). *If* $\vdash \mathcal{C}\{\ \exists x.\mathrm{I}\ \}$ *holds then* $\vdash \mathcal{C}\{\ \mathrm{I}\ \}$ *holds.*

The above four lemmas are particular to MAV1. The proofs for the four lemmas below are similar to the corresponding cases in MAV [20].

▶ **Lemma 25** (Co-external). *If* $\vdash \mathcal{C}\{\ P \otimes (Q \oplus R)\ \}$ *holds then* $\vdash \mathcal{C}\{\ (P \otimes Q) \oplus (P \otimes R)\ \}$ *holds.*

▶ **Lemma 26** (Co-tidy). *If* $\vdash \mathcal{C}\{\ \mathrm{I} \oplus \mathrm{I}\ \}$ *holds, then* $\vdash \mathcal{C}\{\ \mathrm{I}\ \}$ *holds.*

▶ **Lemma 27** (Co-sequence). *If* $\vdash \mathcal{C}\{\ (P\,;Q) \otimes (R\,;S)\ \}$ *holds then* $\vdash \mathcal{C}\{\ (P \otimes R)\,;(Q \otimes S)\ \}$ *holds.*

▶ **Lemma 28** (Atomic co-interaction). *If* $\vdash \mathcal{C}\{\ \alpha \otimes \overline{\alpha}\ \}$ *holds then* $\vdash \mathcal{C}\{\ \mathrm{I}\ \}$ *holds.*

The Proof of Theorem 4. The main result of this paper follows by induction on the structure of P in a predicate of the form $\vdash \mathcal{C}\{\ P \otimes \overline{P}\ \}$, by applying the above eight co-rule elimination lemmas and also Lemma 10 in the cases for *with* and *plus*, and Lemma 11 in the cases for *all* and *some*.

Co-rules are interesting in their own right, since derivations extended with all co-rules coincide with provable linear implications. Suppose that SMAV1 is the system MAV1 extended with all co-rules. The following corollary is a consequence of Theorem 4.

▶ **Corollary 29.** $\vdash P \multimap Q$ *in MAV1 if and only if* $Q \longrightarrow P$ *in SMAV1.*

An advantage of defining linear implication using provability, is that MAV1 is *analytic* [8]; hence, with some care taken for existential quantifiers [5, 23], each predicate gives rise to finitely many derivations up-to congruence. Consequently, proof search is decidable.

▶ **Proposition 30.** *It is decidable whether a predicate in MAV1 is provable.*

5 Conclusion

This paper makes two significant contributions: a novel de Morgan dual pair of nominal quantifiers and the first direct cut elimination result for first-order quantifiers in the calculus of structures. As a consequence of cut-elimination (Theorem 4), we obtain the first consistent proof system that features both non-commutative operator *seq* and first-order quantifiers. The novelty of the nominal quantifiers *new* and *wen* is in how they distribute over positive and negative operators. This greater control of bookkeeping of names enables private names to be modelled in direct embeddings of processes as predicates in MAV1.

This paper continues a line of work on logical systems defined using the calculus of structures with applications to modelling processes [7, 12, 20]. Our approach is distinct from related work on nominal logic [2, 13, 38] where processes are terms, rather than predicates, and an operational semantics is given either as an inductive definition or a logical theory. The related approach is capable of encoding observational preorders and bisimulations, but has the drawback that implication cannot be directly used to compare processes. Our approach is also distinct from the proofs-as-processes Curry-Howard inspired approach to session types [9, 42]. Instead, we adopt a processes-as-predicates approach, setting up a discussion on the relationship between linear implication in MAV1 and observational preorders. Amongst the consequences of cut elimination (Theorem 4) is that linear implication defines a branching-time precongruence over processes that fully respects causality.

Acknowledgements. We are grateful to Catuscia Palamidessi and the CONCUR panel for their thorough analysis of a draft.

References

1 Luca Aceto and Matthew Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, 115(2):179–247, 1994. doi:10.1006/inco.1994.1096.

2 Andrei Alexandru and Gabriel Ciobanu. Nominal techniques for πI-calculus. *Romanian Journal of Information Science and Technology*, 16(4):261–286, 2013.

3 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992. doi:10.1093/logcom/2.3.297.

4 Marek Bednarczyk. Hereditary history preserving bisimulations. Technical report, Polish Academy of Sciences, Gdańsk, 1991.

5 Kai Brünnler. *Deep inference and symmetry in classical proofs*. PhD thesis, TU Dresden, 2003.

6 Kai Brünnler. Locality for classical logic. *Notre Dame J. Form. Log.*, 47(4):557–580, 2006.

7 Paola Bruscoli. A purely logical account of sequentiality in proof search. In *ICLP*, volume 2401 of *LNCS*, pages 302–316. Springer, 2002. doi:10.1007/3-540-45619-8_21.

8 Paola Bruscoli and Alessio Guglielmi. On the proof complexity of deep inference. *ACM Transactions on Computational Logic (TOCL)*, 10(2), 2009. doi:10.1145/1462179.1462186.

9 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.

10 Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Straßburger. The focused calculus of structures. In *EACSL*, volume 12, pages 159–173, 2011. doi:10.4230/LIPIcs.CSL.2011.159.

11 Gabriel Ciobanu and Ross Horne. Non-interleaving operational semantics for geographically replicated databases. In *SYNASC 2013*, pages 440–447, 2013. doi:10.1109/SYNASC.2013.64.

12 Gabriel Ciobanu and Ross Horne. Behavioural analysis of sessions using the calculus of structures. In *PSI 2015, 25-27 August, Kazan, Russia*, volume 9609 of *LNCS*, 2015.

13 Murdoch J. Gabbay. The π-calculus in FM. In *Thirty Five Years of Automating Mathematics*, pages 247–269. Springer, 2003. doi:10.1007/978-94-017-0253-9_10.

14 Andrew Gacek, Dale Miller, and Gopalan Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011. doi:10.1016/j.ic.2010.09.004.

15 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–112, 1987. doi:10.1016/0304-3975(87)90045-4.

16 Alessio Guglielmi. Re:encoding pi calculus in calculus of structures. post on public mailing list http://permalink.gmane.org/gmane.science.mathematics.frogs/161, 2004.

17 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1), 2007. doi:10.1145/1182613.1182614.

18 Alessio Guglielmi and Lutz Straßburger. A system of interaction and structure V: The exponentials and splitting. *Math. Struct. Comp. Sci.*, 21(03):563–584, 2011. doi:10.1017/S096012951100003X.

19 Kohei Honda et al. Scribbling interactions with a formal foundation. In *ICDCIT 2011*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011. doi:10.1007/978-3-642-19056-8_4.

20 Ross Horne. The consistency and complexity of multiplicative additive system virtual. *Sci. Ann. Comp. Sci.*, 25(2):245–316, 2015. URL: http://dx.doi.org/10.7561/SACS.2015.2.245.

21 Naoki Kobayashi and Akinori Yonezawa. ACL – a concurrent linear logic programming paradigm. In *ILPS'93*, pages 279–294. MIT Press, 1993.

22 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

23 Patrick Lincoln and Natarajan Shankar. Proof search in first-order linear logic and other cut-free sequent calculi. In *LICS'94*, pages 282–291. IEEE, 1994. doi:10.1109/LICS.1994.316061.

24 José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992. doi:10.1016/0304-3975(92)90182-F.

25 Dale Miller and Alwen Tiu. A proof theory for generic judgements. *ACM Transactions on Computational Logic (TOCL)*, 6(4):749–783, 2005. doi:10.1145/1094622.1094628.

26 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, 1992. doi:10.1016/0890-5401(92)90008-4.

27 Peter O'Hearn and David Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

28 Andrew Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2), 2003. doi:10.1016/S0890-5401(03)00138-X.

29 Vaughan Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986. doi:10.1007/BF01379149.

30 Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In *TLCA'97*, volume 1210 of *LNCS*, pages 300–318. Springer, 1997. doi:10.1007/3-540-62688-3_43.

31 Luca Roversi. A deep inference system with a self-dual binder which is complete for linear lambda calculus. *J. of Log. and Comp.*, 26(2):677–698, 2016. doi:10.1093/logcom/exu033.

32 Davide Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(1):235–274, 1996. doi:10.1016/0304-3975(96)00075-8.

33 Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: towards a classification. *Th. Comp. Sci.*, 170(1-2):297–348, 1996. doi:10.1016/S0304-3975(96)80710-9.

34 Lutz Straßburger. *Linear logic and noncommutativity in the calculus of structures.* PhD thesis, TU Dresden, 2003.

35 Lutz Straßburger. Some observations on the proof theory of second order propositional multiplicative linear logic. In *TLCA 2009*, volume 5608 of *LNCS*, pages 309–324. Springer, 2009. doi:10.1007/978-3-642-02273-9_23.

36 Lutz Straßburger and Alessio Guglielmi. A system of interaction and structure IV: the exponentials and decomposition. *TOCL*, 12(4):23, 2011. doi:10.1145/1970398.1970399.

37 Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2:4):1–24, 2006. doi:10.2168/LMCS-2(2:4)2006.

38 Alwen Tiu and Dale Miller. Proof search specifications of bisimulation and modal logics for the π-calculus. *TOCL*, 11(2):13, 2010. doi:10.1145/1656242.1656248.

39 Rob van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR '90*, volume 458 of *LNCS*, pages 278–297. Springer, 1990. doi:10.1007/BFb0039066.

40 Rob van Glabbeek. Structure preserving bisimilarity, supporting an operational Petri net semantics of CCSP. In *Correct System Design*, volume 9360 of *LNCS*, pages 99–130. Springer, 2015. doi:10.1007/978-3-319-23506-6_9.

41 Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4-5):229–327, 2001. doi:10.1007/s002360000041.

42 Philip Wadler. Propositions as sessions. *J. of Fun. Prog.*, 24(2-3):384–418, 2014. doi:10.1145/2364527.2364568.

Causality *vs.* Interleavings in Concurrent Game Semantics

Simon Castellan[1] and Pierre Clairambault[2]

1 **Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP**
2 **Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP**

Abstract

We investigate relationships between interleaving and causal notions of game semantics for concurrent programming languages, focusing on the existence of canonical compact causal representations of the interleaving game semantics of programs.

We perform our study on an affine variant of Idealized Parallel Algol (IPA), for which we present two games model: an interleaving model (an adaptation of Ghica and Murawski's fully abstract games model for IPA up to may-testing), and a causal model (a variant of Rideau and Winskel's games on event structures). Both models are sound and adequate for affine IPA. Then, we relate the two models. First we give a causality-forgetting operation mapping functorially the causal model to the interleaving one. We show that from an interleaving strategy we can reconstruct a causal strategy, from which it follows that the interleaving model is the observational quotient of the causal one. Then, we investigate several reconstructions of causal strategies from interleaving ones, showing finally that there are programs which are inherently causally ambiguous, with several distinct minimal causal representations.

1998 ACM Subject Classification F.3.2 Denotational Semantics

Keywords and phrases Game semantics, concurrency, causality, event structures

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.32

1 Introduction

Game semantics present a program as a representation of its behaviour under execution, against any execution environment. This interpretation is computed compositionally, following the methodology of denotational semantics. Game semantics and interactive semantics in general have been developed for a variety of programming language features. They are an established theoretical tool in the foundational study of logic and programming languages, with a growing body of research on applications to various topics, *e.g.* model-checking [1, 10], hardware [4] or software [13] compilation, for higher-order programs. These works exploit the ability of game semantics to provide compositionally a clean and elegant presentation of the operational behaviour of a program, which can then give an invariant for program transformations, or be exploited for analysis.

One subject where game semantics particularly shine is for reasoning about program equivalence. Indeed, game semantics models are often *fully abstract*: they characterise programs up to contextual equivalence, meaning that two programs behave in the same way in all contexts if and only if the corresponding strategies have the same plays. Concurrent languages are no exception: Ghica and Murawski's games model for IPA [5] is fully abstract *wrt.* may-testing. Although, in this language, contextual equivalence is undecidable even for second-order programs, decidability can be recovered for a restricted language [6]. But Ghica and Murawski's model represents concurrent programs with *interleavings*, so whether

one works in a decidable fragment or simply uses non automated tools, reasoning on the fully abstract model requires one to explore *all possible interleavings*. This is the so-called *state explosion problem* familiar in the verification of concurrent systems [7].

Partial order methods provide good tools to alleviate this problem. They provide more compact representations of concurrent programs, avoiding the enumeration of all interleavings. For IPA, recent advances in partial-order based game semantics [11, 3] allow us to restate Ghica and Murawski's model based on partial orders or *event structures*. But can we get back full abstraction this way? Since the interleaving model is fully abstract, the question is: can we give a clean, compact, presentation of the interleaving games model of IPA *via* partial orders? As it is, the interpretation of IPA in *e.g.* [3] is certainly not fully abstract since it retains intensional information (such as the point of non-deterministic branching) invisible up to may-testing. But can we rework it so it yields canonical partial-order representatives for strategies in the interleaving model? In this paper, we show that already in an *affine* setting, the answer is no.

Our contributions are the following. We describe an affine variant of IPA – it is mostly there to provide illustrations and an operational light. For this affine IPA, we give two new categories of games. The first is an affine version of Ghica and Murawski's model. The second draws inspiration from Rideau and Winskel's category of strategies as event structures, without the information on the point of non-deterministic branching, which is irrelevant up to may-testing. Via a collapse of the causal model into the interleaving one, we show that the latter is the observational quotient of the former. We describe several causal reconstructions from an interleaving strategy, aiming for minimality. Finally, we show that interleaving strategies have in general no canonical minimal causal representation.

On the game semantics front, our two models are arena-based, in the spirit of HO games [8]. They both operate on a notion of arenas enriched with conflict, which is required in an affine setting. Our interleaving model is *not* fully abstract for affine IPA. Indeed, we have omitted well-bracketing (as well as bad variables and semaphores) in an effort to make the presentation lighter. These aspects are orthogonal to the problem at hand, and our developments would apply just as well with those. Apart from well-bracketing, our interleaving model is fully compatible with Ghica and Murawski's – strategies in our sense can easily be read as strategies in their sense, as pointers can be uniquely recovered.

2 Affine IPA and its interleaving game semantics

In this section we introduce affine IPA, and the category **GM** of interleaving strategies.

2.1 Affine IPA

▸ **Definition 1.** The **types** of affine IPA are $A, B ::= \mathbb{B} \mid \mathbf{com} \mid A \multimap B \mid \mathbf{ref}_r \mid \mathbf{ref}_w$.

We have types for booleans, commands, and a linear function space. Finally we have two types $\mathbf{ref}_r$ and $\mathbf{ref}_w$ for read-only and write-only variables (this splitting of **ref** is necessary to make the variables non-trivial in an affine setting).

The **terms** of affine IPA are the following:

$$\begin{array}{rcl} M, N & ::= & x \mid M\,N \mid \lambda x.\,M \mid \mathtt{tt} \mid \mathbf{ff} \mid \mathbf{if}\,M\,N_1\,N_2 \mid \bot \\ & & \mid \mathbf{skip} \mid M; N \mid \mathbf{newref}\,v\,\mathbf{in}\,M \mid M := \mathtt{tt} \mid !M \mid M \parallel N \end{array}$$

References are considered initialized to **ff**. As they can only be read once, the only useful value to write is **tt**, hence the restricted assignment command. Typing rules are standard,

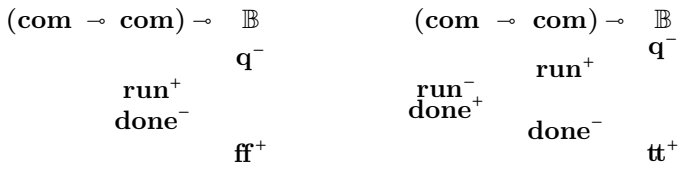


Figure 1 Maximal plays of the alternating game semantics of **strict**.

we only mention a few. Firstly, affine function application and boolean elimination.

$$\frac{\Gamma \vdash M : A \multimap B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash M\, N : B} \qquad \frac{\Gamma \vdash M : \mathbb{B} \qquad \Delta \vdash N_1 : A \qquad \Delta \vdash N_2 : A}{\Gamma, \Delta \vdash \mathbf{if}\, M\, N_1\, N_2 : A}$$

Crucially the first rule treats the context multiplicatively, making the language affine. Secondly, here are the rules for reference manipulation.

$$\frac{\Gamma, r : \mathbf{ref}_r, r : \mathbf{ref}_w \vdash M : \mathbb{B}}{\Gamma \vdash \mathbf{newref}\, r\, \mathbf{in}\, M : \mathbb{B}} \qquad \frac{\Gamma \vdash M : \mathbf{ref}_r}{\Gamma \vdash !M : \mathbb{B}} \qquad \frac{\Gamma \vdash M : \mathbf{ref}_w}{\Gamma \vdash M := \mathbf{tt} : \mathbf{com}}$$

Splitting between the read and write capabilities of the variable type is necessary for the variables to be used in a non-trivial way. For example, the following term is typable:

$$\mathbf{strict} = \lambda f^{\mathbf{com} \multimap \mathbf{com}}.\, \mathbf{newref}\, r\, \mathbf{in}\, (f\, (r := \mathbf{tt})); !r : (\mathbf{com} \multimap \mathbf{com}) \multimap \mathbb{B}$$

The language is equipped with the same operational semantics as in [5] – we skip the details. The operational semantics yields an *evaluation* relation: for $\vdash M : \mathbb{B}$, we write $M \Downarrow_{\text{may}} b$ to mean that M *may* evaluate to the boolean b, or just $M \Downarrow_{\text{may}}$ to mean that M *may* converge. From the combination of concurrency and state, affine IPA is a non-deterministic language.

2.2 Arenas

In game semantics, one interprets a program as a set of *interactions*, usually called *plays*, with its execution environment. For instance, some maximal plays of the interpretation $[\![\mathbf{strict}]\!]$ of the term $\mathbf{strict} : (\mathbf{com} \multimap \mathbf{com}) \multimap \mathbb{B}$ defined above are displayed in Figure 1. Those diagrams are read from top to bottom, and moves have polarity either Player (+, Program) or Opponent (−, Environment). In the first play of Figure 1 Opponent behaves like a constant, where in Figure 1 he is strict. Although the programs are stateful, plays do not carry state: instead, we only see how the state influences Player's behaviour.

To make this formal, we first extract from the type the computational events on which plays such as the above are formed. These are organized into *arenas*.

▸ **Definition 2.** An **event structure with polarities** is a tuple $(A, \leq_A, \sharp_A, \mathrm{pol}_A)$ where A is a set of **moves** or **events**, $\leq_A$ is a partial order on A such that for any $a \in A$, $[a] = \{a' \in A \mid a' \leq_A a\}$ is finite, $\sharp_A$ is an irreflexive symmetric **conflict** relation such that for all $a \sharp_A a'$, for all $a' \leq_A a'_0$, we also have $a \sharp_A a'_0$. Finally, $\mathrm{pol}_A : A \to \{-, +\}$ is a polarity function.

Apart from the fact that we only have binary conflict, this is the same notion of event structures with polarities as in [11]. A **configuration** of A, written $x \in \mathscr{C}(A)$, is a finite

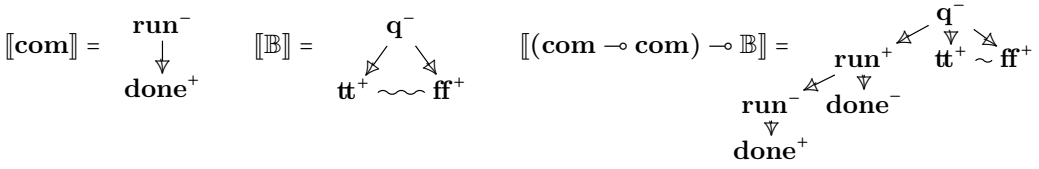
$x \subseteq A$ which is *down-closed* (if $a \in x$ and $a' \leq_A a$, then $a' \in x$ as well) and *consistent* (for all $a_1, a_2 \in x$, $\neg(a_1 \sharp_A a_2)$). For $a_1, a_2 \in A$, we say that a_1 **immediately causes** a_2, written $a_1 \rightarrowtriangle a_2$, when $a_1 <_A a_2$ and for all $a_1 \leq a \leq a_2$ we have either $a_1 = a$ or $a = a_2$. We also write $a_1 \sim a_2$ if a_1 and a_2 are in **immediate conflict**, meaning $a_1 \sharp_A a_2$ and for all $a_1' \leq_A a_1$, $a_2' \leq_A a_2$ (with *at least* one of them strict), we have $\neg(a_1' \sharp_A a_2')$. Finally, we write $\min(A)$ for the set of minimal events of A.

Arenas are certain event structures with polarities:

▸ **Definition 3.** An **arena** is an event structure with polarities such that $\leq_A$ is a *forest* (for all $a_1, a_2 \leq_A a$, either $a_1 \leq_A a_2$ or $a_2 \leq_A a_1$), is *alternating* (for all $a_1 \rightarrowtriangle a_2$, $\mathrm{pol}_A(a_1) \neq \mathrm{pol}_A(a_2)$), and *race-free* (if $a_1 \sim a_2$, then $\mathrm{pol}(a_1) = \mathrm{pol}(a_2)$).

Although our formulation is slightly different, our arenas are very close to the standard notion of [8]: the three differences is that we have no Question/Answer distinction, our arenas are not necessarily negative, and we have a conflict relation.

▸ **Example 4.** We display below the arenas for some types of IPA.



On $[\![\mathbf{com}]\!]$, Opponent may start running the command ($\mathbf{run}^-$), which may or may not terminate ($\mathbf{done}^+$). On $[\![\mathbb{B}]\!]$, Opponent may interrogate the boolean ($\mathbf{q}^-$), and Player may or may not answer. If he does, it will be with exactly *one* of the incompatible $\mathbf{tt}^+$ and $\mathbf{ff}^+$.

We will see later on how to systematically interpret types of IPA as arenas. For now on though, we give two simple constructions on arenas.

▸ **Definition 5.** Let A be an arena. Its **dual**, written $A^\perp$, has the same data as A but polarity reversed. If A and B are arenas, then their **parallel composition** $A \parallel B$, also written $A \otimes B$ for the **tensor**, has components:

- *Events/moves.* the disjoint union $\{1\} \times A \cup \{2\} \times B$,
- *Causality, conflict.* Inherited from A and B.

In this paper, we will define two categories **GM** and **PO** with arenas as objects.

2.3 Interleaving-based game semantics on arenas

Now, we define a compact closed category of games called **GM**, by reference to Ghica and Murawski's model of IPA [5]. Our category will be much simpler though, as it will be an affine version of theirs, without bracketing conditions. Firstly, we need to define *plays*.

▸ **Definition 6.** Let A be an arena. A **play** s on A, written $s \in \mathcal{P}_A$, is a total order $s = (|s|, \leq_s)$ of moves of A such that $|s| \in \mathscr{C}(A)$, and for any $a, b \in s$, if $a \leq_A b$ then $a \leq_s b$. We write $s \sqsubseteq t$ for the usual prefix ordering on plays.

In [5], strategies are closed under some *saturation conditions*: for instance, if $sa^+b^- \in \sigma$ and b does not actually depend on a in the game, then σ can always *delay* a until after b was played. In other words, we have $sba \in \sigma$ as well. In our affine variant, we will have a slightly different formulation of saturation. First we define an order on plays.

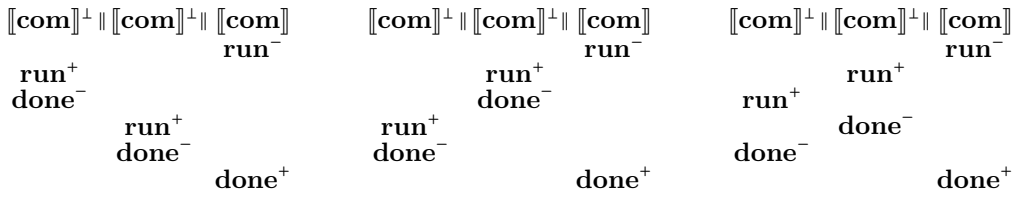


Figure 2 Some plays in $[\![\|]\!]_{\mathbf{GM}}$.

▸ **Definition 7.** Let $s, t \in \mathcal{P}_A$ for A an arena. Then we say that $s \preceq t$ iff $|s| \subseteq |t|$, and:
- If $a_1^+ \leq_s a_2^-$, then $a_1 \leq_t a_2$.
- For $a_2^+ \in |s|$, if $a_1{}^- \leq_t a_2^+$, then $a_1 \in |s|$ and $a_1 \leq_s a_2$.

Clearly, $\preceq$ is a partial order on $\mathcal{P}_A$. Intuitively, going upwards in $\preceq$ corresponds to strengthening causal information by pushing Opponent moves behind Player moves, hence implying that those Opponent moves were not true dependencies for the Player moves. The partial order $\preceq$ is generated by elementary permutations, as in the saturation conditions in [5], along with the prefix ordering. We now define:

▸ **Definition 8.** A **GM**-strategy on arena A, written $\sigma : A$, is a set $\sigma \subseteq \mathcal{P}_A$ which is:
- *Saturated:* if $s \in \sigma$ and $t \preceq s$, then $t \in \sigma$ as well,
- *Receptive:* if $s \in \sigma$ and $|s| \subset |s| \cup \{a^-\} \in \mathscr{C}(A)$, then $sa \in \sigma$ as well.

▸ **Example 9.** The **GM**-strategy $[\![\|]\!]_{\mathbf{GM}} : [\![\mathbf{com}]\!]^\perp \parr [\![\mathbf{com}]\!]^\perp \parr [\![\mathbf{com}]\!]$ comprises all plays on $[\![\mathbf{com}]\!]^\perp \parr [\![\mathbf{com}]\!]^\perp \parr [\![\mathbf{com}]\!]$ such that:
- If $\mathbf{run}^+$ appears on either occurrence of $[\![\mathbf{com}]\!]^\perp$, then $\mathbf{run}^-$ must appear before,
- If $\mathbf{done}^+$ appears, then both $\mathbf{done}^-$ must appear before.

Figure 2 displays several plays of $[\![\|]\!]_{\mathbf{GM}}$. In total, $[\![\|]\!]_{\mathbf{GM}}$ has six maximal plays.

As usual in play-based game semantics, operations on **GM**-strategies rely crucially on a notion of *restriction* of plays. Consider A an arena, $s \in \mathcal{P}_A$, and B some sub-component on A (we leave the notion of sub-component intentionally somewhat vague: for instance A is a subcomponent of $A \otimes B$, and $A_1 \parr B_1$ is a sub-component of $A_1 \otimes A_2 \parr B_1 \otimes B_2$). The **restriction** $s \upharpoonright B \in \mathcal{P}_B$ is the subsequence of s of moves in component B, in the same order.

Using that, we can now define the **copycat strategy** on A to be:

$$\mathbb{c}_A = \{s \in \mathcal{P}_{A^\perp \parr A} \mid \forall s' \sqsubseteq s,\ \forall a \in |s' \upharpoonright A^\perp|,\ \mathrm{pol}_{A^\perp}(a) = + \implies a \in |s' \upharpoonright A|,\ \forall a \in |s' \upharpoonright A|, \mathrm{pol}_A(a) = + \implies a \in |s' \upharpoonright A^\perp|\}$$

It is a **GM**-strategy. Using the usual *parallel composition plus hiding* mechanism, we can also define *composition*. Given $\sigma : A^\perp \parr B$ and $\tau : B^\perp \parr C$, first define their **interaction** $\tau \circledast \sigma = \{u \in \mathcal{P}_{A \parr B \parr C} \mid u \upharpoonright A \parr B \in \sigma \ \&\ u \upharpoonright B \parr C \in \tau\}$. The **composition** $\tau \odot \sigma : A^\perp \parr C$ is obtained through *hiding*, by $\tau \odot \sigma = \{u \upharpoonright (A \parr C) \mid u \in \tau \circledast \sigma\}$. Altogether:

▸ Proposition 1. There is a compact closed category **GM** with arenas as objects, and as morphisms from A to B, **GM**-strategies $\sigma : A^\perp \parr B$. We also write $\sigma : A \overset{\mathbf{GM}}{\to} B$.

Proof. The operation $\otimes$ on arenas is extended to **GM**-strategies by setting, for $\sigma_1 : A_1 \overset{\mathbf{GM}}{\to} B_1$ and $\sigma_2 : A_2 \overset{\mathbf{GM}}{\to} B_2$, $\sigma_1 \otimes \sigma_2 = \{s \in \mathcal{P}_{(A_1 \otimes A_2)^\perp \parr B_1 \otimes B_2} \mid s \upharpoonright A_1^\perp \parr B_1 \in \sigma_1 \ \&\ s \upharpoonright A_2^\perp \parr B_2 \in \sigma_2\}$. Rather than detailing explicitly the rest of the structure, we will inherit it from the forthcoming category **PO**. All laws will then follow from Proposition 4. ◂

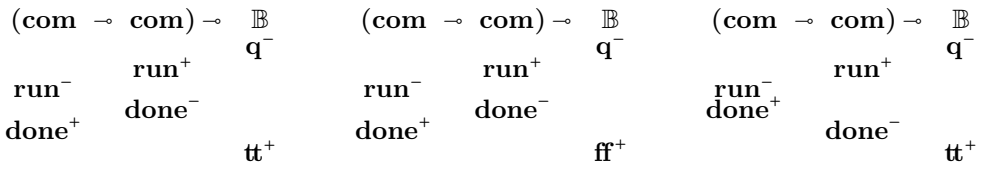


Figure 3 Some maximal plays of the non-alternating game semantics of **strict**.

For now we do not show how to interpret affine IPA in **GM** – for that one actually needs a symmetric monoidal closed subcategory of *negative* arenas, which seems difficult to define without appealing to **PO**. However, we illustrate this interpretation by revisiting Figure 1.

▸ **Example 10.** The **GM**-strategy corresponding to **strict** will contain, among others, the maximal plays described in Figure 3.

Although **strict** is a sequential program, the fact that in **GM**, Opponent may *not* be sequential (and, in this case, non well-bracketed either) allows us to observe new behaviours from **strict**. For instance, in the first two plays of Figure 3, Opponent concurrently *answers* and *asks for the argument* on **com** ⊸ **com**. This triggers a race between the subterms $r := \mathbf{tt}$ and $!r$ of **strict**. As a consequence, one can observe both **tt** and **ff** as final results of the computation. However, if Opponent was to answer only *after* $r := \mathbf{tt}$ was evaluated (as in the third play of Figure 3), the only possible final result would be **tt**.

There are, in total, ten maximal non-alternating plays in the **GM**-strategy for **strict**.

3 Causal game semantics for affine IPA

We give a *causal* variant of **GM**, where plays are partial orders. This yields a category **PO**, close to the category of *concurrent games* of Rideau and Winskel [11] – the main difference is that strategies in **PO** omit information about the point of non-deterministic branching.

3.1 Po-plays and po-strategies

First, we define the notion of partially ordered play.

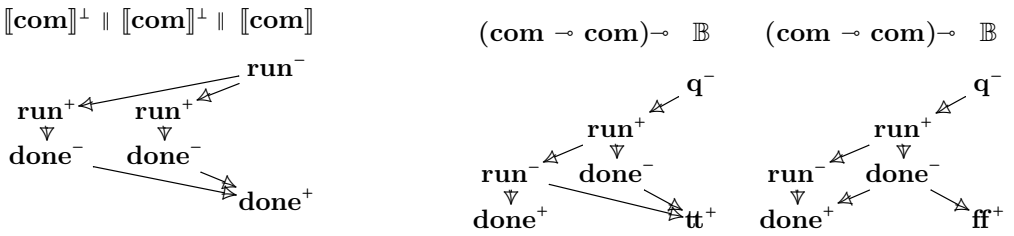
▸ **Definition 11.** A **partially ordered play (po-play)** on arena A is a partial order $\mathbf{q} = (|\mathbf{q}|, \leq_{\mathbf{q}})$ where $|\mathbf{q}| \in \mathscr{C}(A)$, and $\mathbf{q}$ satisfies the following properties:

- *Respects the game:* for $a_1, a_2 \in |\mathbf{q}|$, if $a_1 \leq_A a_2$ then $a_1 \leq_{\mathbf{q}} a_2$,
- *Is courteous:* if $a_1^+ \rightarrowtriangle_{\mathbf{q}} a_2$ then $a_1 \rightarrowtriangle_A a_2$, and if $a_1 \rightarrowtriangle_{\mathbf{q}} a_2^-$, then $a_1 \rightarrowtriangle_A a_2$.

We write $\mathcal{P}_A^{\copyright}$ for the set of po-plays on arena A.

Unlike usual (alternating or non-alternating) plays, po-plays are not chronologically ordered, but carry *causal* information about Player's choices. Hence, a po-play cannot express that an Opponent event happens *after* a given event, unless that dependency is already present in the arena. In fact, a po-play cannot force a dependency between two Player moves either: such a dependency may be broken by an asynchronous execution environment.

Although one po-play may carry information about many interleavings, representing a **GM**-strategy might take *several*. Indeed, a po-play is by itself only able to represent a process which is deterministic *up to the choice of the scheduler* (note that parallel composition is indeed deterministic up to the choice of the scheduler, it is only via its inter-



(a) A po-play for parallel composition.

(b) The two maximal po-plays of $[\![\mathbf{strict}]\!]_{\mathbf{PO}}$.

Figure 4 Some po-plays.

action with *e.g.* a shared memory that non-determinism arises). For instance, the **GM**-strategy $\mathbf{coin} : [\![\mathbb{B}]\!] = \{\epsilon, \mathbf{q}^-, \mathbf{q}^-\mathbf{tt}^+, \mathbf{q}^-\mathbf{ff}^+\}$ can only be represented via *two* maximal po-plays: $\mathbf{q}^- \rightarrowtriangle \mathbf{tt}^+$ and $\mathbf{q}^- \rightarrowtriangle \mathbf{ff}^+$. It features actual non-determinism, independent from the scheduler.

To express such non-determinism, Rideau and Winskel [11] formalize strategies as *event structures* rather than partial orders. Our causal notion of strategies builds on their work; but since the present paper is only interested in relating causal with interleaving game semantics (therefore with may-testing), we drop the explicit non-deterministic branching point and consider po-strategies to be certain *sets of partial orders*. For that we first define:

▸ **Definition 12.** Let $\mathbf{q}, \mathbf{q}'$ be two partial orders. We say that $\mathbf{q}$ is **rigidly included** in $\mathbf{q}'$, or that $\mathbf{q}$ is a **prefix** of $\mathbf{q}'$, written $\mathbf{q} \hookrightarrow \mathbf{q}'$, if we have the inclusion $|\mathbf{q}| \subseteq |\mathbf{q}'|$, for any $a_1, a_2 \in |\mathbf{q}|$ we have $a_1 \leq_{\mathbf{q}} a_2$ iff $a_1 \leq_{\mathbf{q}'} a_2$, and $\mathbf{q}$ is down-closed in $\mathbf{q}'$.

We are now in position to define **PO**-strategies.

▸ **Definition 13.** A **PO-strategy** on A, written $\sigma :: A$, is a non-empty prefix-closed $\sigma \subseteq \mathcal{P}^{\copyright}_A$, which is additionally **receptive**: for all $\mathbf{q} \in \sigma$, if $|\mathbf{q}| \in \mathscr{C}(A)$ extends to $|\mathbf{q}| \cup \{a^-\} \in \mathscr{C}(A)$, then there is $\mathbf{q} \hookrightarrow \mathbf{q}' \in \sigma$ such that $|\mathbf{q}'| = |\mathbf{q}| \cup \{a\}$.

It follows by courtesy that $\mathbf{q}'$ is necessarily unique: the immediate dependency of a in $\mathbf{q}'$ is forced by its immediate dependency in A.

Clearly, the set of prefixes of the po-play of Figure 4a gives a **PO**-strategy. For a non-trivial non-deterministic example, we give in Figure 4b the two maximal (up to prefix / rigid inclusion) po-plays of the **PO**-strategy corresponding to **strict**. This gives a quite compact representation of all of the ten maximal plays of the **GM**-strategy for **strict** of Example 10.

3.2 The compact closed category PO

To construct **PO** we start with the causal copycat, which is – configuration-wise – as in [11].

▸ **Definition 14.** Let A be an arena. We define a partial order $\leq_{\mathbb{CC}_A}$ on $A^\perp \parr A$:

$$\begin{aligned} \leq_{\mathbb{CC}^{\copyright}_A} \quad = \quad & (\{((1,a),(1,a')) \mid a \leq_A a'\} \cup \{((2,a),(2,a')) \mid a \leq_A a'\} \cup \\ & \{((1,a),(2,a)) \mid \mathrm{pol}_A(a) = +\} \cup \{((2,a),(1,a)) \mid \mathrm{pol}_A(a) = -\})^+ \end{aligned}$$

where $(-)^+$ denotes the transitive closure of a relation. Then, $\mathbb{cc}^{\copyright}_A :: A^\perp \parr A$ comprises all $x \parr y \in \mathscr{C}(A^\perp \parr A)$ down-closed for $\leq_{\mathbb{CC}^{\copyright}_A}$, with the induced partial order.

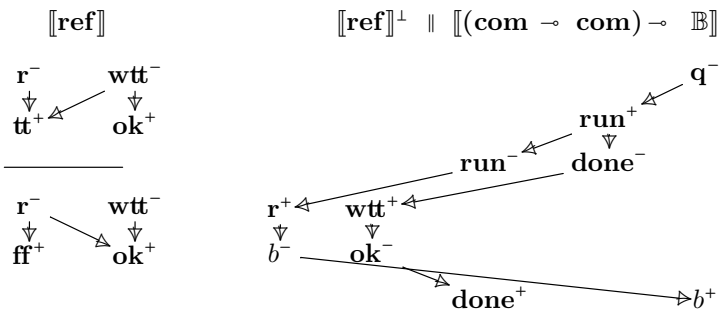


Figure 5 **cell** :: ⟦**ref**⟧ and $\llbracket \lambda f^{\mathbf{com} \multimap \mathbf{com}}.\, f\,(r := \mathbf{tt});\, !r \rrbracket :: \llbracket \mathbf{ref} \rrbracket^{\perp} \parr \llbracket (\mathbf{com} \multimap \mathbf{com}) \multimap \mathbb{B} \rrbracket$.

We will see in Proposition 4 that this is indeed a causal version of $\mathbb{c}_A : A^{\perp} \parr A$. Now, we define composition of **PO**-strategies. We first define composition of po-plays (via interaction plus hiding, essentially as in [11]), before lifting it component-wise to **PO**-strategies.

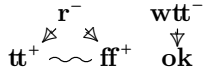
▸ **Definition 15.** Two dual po-plays $\mathbf{q} \in \mathcal{P}^{\copyright}_{A}$, $\mathbf{q}' \in \mathcal{P}^{\copyright}_{A^{\perp}}$ such that $|\mathbf{q}| = |\mathbf{q}'|$ are **causally compatible** if $(\leq_{\mathbf{q}} \cup \leq_{\mathbf{q}'})^*$ is a partial order, *i.e.* is acyclic. Then we write $\mathbf{q} \wedge \mathbf{q}' = (|\mathbf{q}|, \leq_{\mathbf{q} \wedge \mathbf{q}'})$ for the resulting partial order.

If $\mathbf{q}$ and $\mathbf{q}'$ are causally compatible po-plays on dual games as above, the events of $\mathbf{q} \wedge \mathbf{q}'$ have no well-defined polarity, so it is not a po-play. If $\mathbf{q} \in \mathcal{P}^{\copyright}_{A^{\perp} \parr B}$ and $\mathbf{q}' \in \mathcal{P}^{\copyright}_{B^{\perp} \parr C}$ are not dual but composable, we say that they are **causally compatible** if $|\mathbf{q}| = x_A \parr x_B$, $|\mathbf{q}'| = x_B \parr x_C$, plus $(\mathbf{q} \parr x_C)$ and $(x_A \parr \mathbf{q}')$ are causally compatible (where x_A, x_C inherit the order from A, C – in particular, x_A is regarded as a member of $\mathcal{P}^{\copyright}_{A}$, and x_C as a member of $\mathcal{P}^{\copyright}_{C^{\perp}}$), we define their open interaction $\mathbf{q}' \circledast \mathbf{q} = (\mathbf{q} \parr x_C) \wedge (x_A \parr \mathbf{q}')$.

In that case we define $\mathbf{q}' \odot \mathbf{q} \in \mathcal{P}^{\copyright}_{A^{\perp} \parr C}$ as the **projection** $\mathbf{q}' \circledast \mathbf{q} \downarrow A^{\perp} \parr C$, with events those of $\mathbf{q}' \circledast \mathbf{q}$ that are in A or C, and partial order as in $\leq_{\mathbf{q}' \circledast \mathbf{q}}$. This being a po-play is a variation on the stability by composition of courtesy in [11] (there called *innocence*).

▸ **Definition 16.** Let $\sigma :: A^{\perp} \parr B$ and $\tau :: B^{\perp} \parr C$ be **PO**-strategies. Their **composition** is $\tau \odot \sigma = \{\mathbf{q}' \odot \mathbf{q} \mid \mathbf{q}' \in \tau \;\&\; \mathbf{q} \in \sigma \text{ causally compatible}\}$. Then, $\tau \odot \sigma :: A^{\perp} \parr C$ is a **PO**-strategy.

The construction is a simplification of [11]: po-plays are certain concurrent strategies, and their composition is close to the composition of concurrent strategies with the simplification that events of po-plays are those of the games rather than only *labeled* by the game.

▸ **Example 17.** Consider $\llbracket \mathbf{ref}_r \rrbracket \otimes \llbracket \mathbf{ref}_w \rrbracket =$ 

, for the type of references.

By abuse of notation, we write ⟦**ref**⟧ for $\llbracket \mathbf{ref}_w \rrbracket \otimes \llbracket \mathbf{ref}_r \rrbracket$. The **PO**-strategy interpreting **strict** is the composition of the **PO**-strategy with maximal po-play at the right hand side of Figure 5 (interpreting $r : \mathbf{ref}_w, r : \mathbf{ref}_r \vdash \lambda f^{\mathbf{com} \multimap \mathbf{com}}.\, f\,(r := \mathbf{tt});\, !r$ following Section 3.3), and **cell** :: ⟦**ref**⟧ for the memory cell (with maximal po-plays at the left hand side of Figure 5). Performing composition as above produces the two maximal po-plays of Figure 4b.

▸ Proposition 2. There is a compact closed category **PO** with *arenas* as objects, and **PO**-strategies $\sigma :: A^{\perp} \parr B$ as morphisms from A to B, also written $\sigma : A \overset{\mathbf{PO}}{\to} B$.

Proof. The tensor $\mathbf{q}_1 \otimes \mathbf{q}_2$ of $\mathbf{q}_1 \in \mathcal{P}^{\copyright}_{A_1^\perp \parallel B_1}$ and $\mathbf{q}_2 \in \mathcal{P}^{\copyright}_{A_2^\perp \parallel B_2}$ is the obvious inherited partial order on $(A_1 \parallel A_2)^\perp \parallel (B_1 \parallel B_2)$. The tensor $\sigma_1 \otimes \sigma_2$ of **PO**-strategies $\sigma_1 :: A_1^\perp \parallel B_1$ and $\sigma_2 : A_2^\perp \parallel B_2$ is defined component-wise. Structural morphisms are copycat **PO**-strategies.

PO simplifies (omitting explicit non-deterministic branching information) the bicategory of concurrent games [11], whose compact closed structure is established with details in [2]. ◂

3.3 Interpretation of affine IPA

For completeness, we succinctly describe how one can define the interpretation of affine IPA in **PO**. In fact, affine IPA will not be interpreted directly in **PO**, which does not support weakening of variables as the empty arena 1, unit for the tensor, is not terminal (since **PO**-strategies can have minimal positive events, there are in general several **PO**-strategies on $A^\perp \parallel 1$ as soon as A has at least one minimal negative event). We have to restrict to a proper subcategory of **PO**, defined as follows.

▸ **Definition 18.** An event structure with polarities A is **negative** if $\mathrm{pol}(\min(A)) \subseteq \{-\}$.

The category $\mathbf{PO}^-$ is the subcategory of **PO** with objects **negative arenas**, and morphisms the **negative PO-strategies** whose po-plays are all negative.

The empty arena 1 is terminal in $\mathbf{PO}^-$: if A is negative then $A^\perp \parallel 1$ has no negative minimal event. Therefore a negative $\sigma :: A^\perp \parallel 1$ must be empty, as a potential minimal event would be in particular minimal in $A^\perp \parallel 1$. However, restricting to $\mathbf{PO}^-$ has a price: we lose the closure $A^\perp \parallel B$, which is in general not negative and hence not an object of $\mathbf{PO}^-$. Thus we build a negative version, where the minimal events of A depend on those of B.

▸ **Definition 19.** Let A, B be two negative arenas. The arena $A \multimap B$ has:

- *Events/polarity:* $(\|_{b \in \min(B)} A^\perp) \parallel B$.
- *Causality:* $(\|_{b \in \min(B)} A^\perp) \parallel B$, enriched with $((2, b), (1, (b, a)))$ for $a \in A$ and $b \in \min(B)$.
- *Conflict:* $(\|_{b \in \min(B)} A^\perp) \parallel B$, plus those inherited by $(1, (b_1, a)) \sim (1, (b_2, a))$ for $b_1 \neq b_2$.

If A, B are conflict-free and B has a unique minimal event, then $A \multimap B$ coincides with the usual arrow arena construction in Hyland-Ong games [8]. In general if B has a unique minimal event, then $A \multimap B$ does not introduce new conflicts or copies of A, and only differs from $A^\perp \parallel B$ by the fact that events of $A^\perp$ now depend on the minimal event of B – see Example 4 for such an arrow arena. However, if B has several minimal events, then multiple copies of A are created; fortunately we can use conflict to maintain linearity.

The arena $A \multimap B$ does not yet give a closure with respect to the tensor. The issue is that there are more **PO**-strategies in $A \multimap B$ than in $A^\perp \parallel B$. Indeed, consider a **PO**-strategy $\sigma :: \mathbb{B}^\perp \parallel (\mathbb{B} \otimes \mathbb{B})$, that plays $\mathbf{q}^+$ in the left hand side occurrence of $\mathbb{B}$ whenever Opponent plays $\mathbf{q}^-$ in *both* right hand side occurrences of $\mathbb{B}$. Then on $\mathbb{B} \multimap (\mathbb{B} \otimes \mathbb{B})$ there are *two* ways to replicate this, as they are two copies of the left hand side $\mathbb{B}$ in the arena. To get back a closed structure, we need to restrict the category further.

▸ **Definition 20.** A negative **PO**-strategy $\sigma :: A$ is **well-threaded** iff, for any $\mathbf{q} \in \sigma$, $\mathbf{q}$ has at most one minimal event. Copycat is well-threaded and well-threaded **PO**-strategies are stable under composition – they form a subcategory $\mathbf{PO}^-_{\mathrm{wt}}$ of $\mathbf{PO}^-$.

Up to renaming of events, negative well-threaded strategies on $(A \parallel B)^\perp \parallel C$ exactly coincide with those on $A^\perp \parallel B \multimap C$. Leveraging the compact closed structure of **PO**, it follows that $\mathbf{PO}^-_{\mathrm{wt}}$ is symmetric monoidal closed (where the monoidal unit 1 is terminal). As such, it supports the interpretation of the affine λ-calculus: any term $x_1 : A_1, \ldots, x_n : A_n \vdash$

$M : B$ is interpreted as a **PO**-strategy $[\![M]\!] : [\![A_1]\!] \otimes \ldots \otimes [\![A_n]\!] \overset{\mathbf{PO}^{-}_{\mathrm{wt}}}{\to} [\![B]\!]$. Along with the **PO**-strategy with unique po-play that of Figure 4a for parallel composition, the interpretation of the **newref** construct as sketched in Example 17, and the obvious **PO**-strategies for the other affine IPA combinators, we get an interpretation $[\![-]\!]$ of affine IPA into $\mathbf{PO}^{-}_{\mathrm{wt}}$, which is a subcategory of **PO**. Standard techniques entail:

▸ Proposition 3. The interpretation $[\![-]\!]$ is sound and adequate for affine IPA, *i.e.* for $\vdash M :$ **com**, we have $M \Downarrow_{\mathrm{may}}$ iff $[\![M]\!]$ contains a positive event.

4 From PO to GM and back

We finally enter the final section of this paper, and relate the two semantics.

4.1 Forgetting causality

We start with the easy part: that **PO** can be embedded into **GM**. As partial orders are more informative than plays, it is easy to move from the former to the latter.

▸ **Definition 21.** Let $\mathbf{q} \in \mathcal{P}^{\copyright}_{A}$. A **play** in $\mathbf{q}$ is $s \in \mathcal{P}_A$ such that $|s| \subseteq |\mathbf{q}|$, and such that for all a_2 in $|s|$, if $a_1 \leq_{\mathbf{q}} a_2$, then $a_1 \in |s|$ and $a_1 \leq_s a_2$. We write $\mathrm{Plays}(\mathbf{q})$ for the set of plays in $\mathbf{q}$.

From courtesy of $\mathbf{q}$ it follows that $\mathrm{Plays}(\mathbf{q})$ satisfies the saturation condition of Definition 8. For $\sigma :: A$ a **PO**-strategy, we have $\mathrm{Plays}(\sigma) = \bigcup\{\mathrm{Plays}(\mathbf{q}) \mid \mathbf{q} \in \sigma\}$ a **GM**-strategy, as receptivity follows from receptivity of σ. In fact, we have:

▸ Proposition 4. There is an identity-on-object functor $\mathrm{Plays} : \mathbf{PO} \to \mathbf{GM}$.

This is a direct verification. As in Section 2.2 we have by anticipation defined the compact closed structure of **GM** to be the image of that of **PO** through Plays, this functor preserves the compact closed structure by construction. Combined with the interpretation $[\![-]\!]$ of affine IPA in **PO**, this gives a sound and adequate interpretation $\mathrm{Plays} \circ [\![-]\!]$ of affine IPA in **GM**. Providing a direct sound interpretation to **GM** without **PO** would be awkward, as it is unclear how to define well-threaded **GM**-strategies with no access to causality.

As emphasized in the introduction, the interpretation $\mathrm{Plays} \circ [\![-]\!]$ is *not* fully abstract for affine IPA. However, let us emphasize again that we are not interested in full abstraction for affine IPA; rather this serves as a simpler setting in which to study the relationship between the fully abstract model for IPA [5] and its causal variant in *e.g.* [3].

4.2 Recovering causality

We now investigate how one can recover a **PO**-strategy from a **GM**-strategy.

4.2.1 A naive causal reconstruction

As a first step, we simply reverse the construction of Definition 21.

▸ **Definition 22.** A **causal resolution** $\sigma : A$ is any $\mathbf{q} \in \mathcal{P}^{\copyright}_{A}$ such that $\mathrm{Plays}(\mathbf{q}) \subseteq \sigma$.

Because some **GM**-strategies (such as $\mathbf{coin} : \mathbb{B}$) are inherently non-deterministic, it is hopeless to try to describe them with a unique maximal causal resolution. A first rough causal reconstruction for a **GM**-strategy consists simply in taking *all* causal resolutions.

▸ Proposition 5. Let $\sigma : A$ be a **GM**-strategy. Then, $\mathrm{Caus}(\sigma) = \{\mathbf{q} \in \mathcal{P}^{\copyright}_A \mid \mathrm{Plays}(\mathbf{q}) \subseteq \sigma\}$ is a **PO**-strategy such that $\mathrm{Plays}(\mathrm{Caus}(\sigma)) = \sigma$. Moreover, this yields a lax functor $\mathrm{Caus} : \mathbf{GM} \to \mathbf{PO}$, *i.e.* we have $\mathbb{c}^{\copyright}_A \subseteq \mathrm{Caus}(\mathbb{c}_A)$ and $\mathrm{Caus}(\tau) \odot \mathrm{Caus}(\sigma) \subseteq \mathrm{Caus}(\tau \odot \sigma)$ for all $\sigma : A^{\perp} \parr B$ and $\tau : B^{\perp} \parr C$ (but neither of the other inclusions hold).

Proof. Each causal resolution is courteous by definition; receptivity and closure under prefix are immediate. Each play $s \in \sigma$ appears in a causal resolution $\mathbf{q}_s$, whose plays are exactly those $t \preceq s$ obtained by saturation from s. Finally, lax functoriality is straightforward.

To see why $\mathrm{Caus}(-)$ is only lax functorial, take $A = \{a^-\}$, $C = \{c^-\}$ and $B = 1$. Take the **PO**-strategy $\sigma :: A^{\perp} \parr B$ to have as only non-empty po-play the singleton a^+, while $\tau :: B^{\perp} \parr C$ has only non-empty po-play the singleton c^-. Then the **GM**-strategy $\mathrm{Plays}(\tau) \odot \mathrm{Plays}(\sigma)$ admits $c^- \twoheadrightarrow a^+$ as a causal resolution, which is therefore a po-play of $\mathrm{Caus}(\mathrm{Plays}(\tau) \odot \mathrm{Plays}(\sigma))$. On the other hand, $\mathrm{Caus}(\mathrm{Plays}(\tau)) \odot \mathrm{Caus}(\mathrm{Plays}(\sigma)) = \tau \odot \sigma$ has only one maximal po-play, with causally independent c^- and a^+. ◂

In particular, each **GM**-strategy is *definable* as a **PO**-strategy. Along with Proposition 4, and the fact that (just as in [5]) two distinct **GM**-strategies can always be distinguished by a **GM**-strategy, this entails that **GM** is the **observational quotient** of **PO**, in the sense that for $\sigma_1, \sigma_2 :: A$, $\mathrm{Plays}(\sigma_1) = \mathrm{Plays}(\sigma_2)$ iff for all $\alpha :: A^{\perp} \parr \mathbf{com}$, $\alpha \odot \sigma_1 = \alpha \odot \sigma_2$.

There are in general many **PO**-strategies corresponding to one **GM**-strategy, as **GM**-strategies only remember the *observable* behaviour. Some **PO**-strategies are more succinct than others for a fixed **GM**-strategy; and the causal reconstruction $\mathrm{Caus}(-)$ is not very economical as it constructs the *biggest* such causal representation. For instance, the **PO**-strategy $\mathrm{Caus}(\llbracket \parr \rrbracket_{\mathbf{GM}})$ not only comprises the po-play of Figure 4a, but also the linear po-play of *sequential* command composition.

4.2.2 Extremal causal resolutions

As we have seen, the construction $\mathrm{Caus}(-)$ presented above does not yield a satisfactory causal representation of a **GM**-strategy because it is not *minimal*. Seeking a minimal canonical causal representation of a **GM**-strategy, we now investigate when certain causal resolutions are subsumed by others, and hence can be removed without changing $\mathrm{Plays}(\sigma)$.

For $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{P}^{\copyright}_A$ with $|\mathbf{q}_1| = |\mathbf{q}_2|$, considering $\mathbf{q}_1$ subsumed by $\mathbf{q}_2$ when $\mathrm{Plays}(\mathbf{q}_1) \subseteq \mathrm{Plays}(\mathbf{q}_2)$ is a bit too naive. Indeed, consider $\mathbf{cell} :: \llbracket \mathbf{ref} \rrbracket$ of Figure 5. We have:

$$\mathrm{Plays}\left(\begin{array}{ccc} \mathbf{r}^- & & \mathbf{wtt}^- \\ & \searrow & \downarrow \\ & & \mathbf{ok}^+ \end{array}\right) \subseteq \mathrm{Plays}\left(\begin{array}{ccc} \mathbf{r}^- & & \mathbf{wtt}^- \\ & & \downarrow \\ & & \mathbf{ok}^+ \end{array}\right)$$

However, moving from the former to the latter does not preserve the future: namely, whereas any play in the left hand side can only be extended with $\mathbf{ff}^+$, there are plays in the right hand side that can be extended with $\mathbf{tt}^+$ as well. So, the left hand side has to be kept.

To address this relaxation of causality while taking account of the future, for $|\mathbf{q}_1| = |\mathbf{q}_2|$ with $\mathrm{Plays}(\mathbf{q}_1) \subseteq \mathrm{Plays}(\mathbf{q}_2)$, we will say that $\mathbf{q}_2$ *relaxes* $\mathbf{q}_1$ if the inclusion of plays is automatically transferred to all possible rigid extensions of $\mathbf{q}_1$. More formally:

▸ **Definition 23.** We define a partial order $\leq$ called **relaxation** coinductively, by $\mathbf{q}_1 \leq \mathbf{q}_2$ iff $|\mathbf{q}_1| = |\mathbf{q}_2|$, $\mathrm{Plays}(\mathbf{q}_1) \subseteq \mathrm{Plays}(\mathbf{q}_2)$, and for all $\mathbf{q}_1 \hookrightarrow \mathbf{q}_1'$, there exists $\mathbf{q}_2 \hookrightarrow \mathbf{q}_2'$ such that $\mathbf{q}_1' \leq \mathbf{q}_2'$.

For $\sigma : A$ a **GM**-strategy and $\mathbf{q} \in \mathrm{Caus}(\sigma)$, we say that $\mathbf{q}$ is **extremal** in σ iff $\mathbf{q}$ is $\leq$-maximal. Let $\mathrm{Extr}(\sigma)$ be denote the set of extremal po-plays in σ.

▸ Proposition 6. For any $\sigma : A$, we have $\mathrm{Extr}(\sigma) :: A$ such that $\mathrm{Plays}(\mathrm{Extr}(\sigma)) = \sigma$.

The operation $\mathrm{Extr}(-)$ performs well on many examples: for instance, it recovers the proper **PO**-strategies for all the examples of **GM**-strategies in this paper until now. It also properly reverses $\mathrm{Plays}(-)$ for *deterministic* **PO**-strategies, with only one maximal po-play. In that case, it matches the previously known correspondence between Rideau and Winskel's *deterministic concurrent strategies* [12] and Melliès and Mimram's category of *receptive ingenuous strategies* [9].

In the general case however, $\mathrm{Extr}(-)$ is not even lax functorial. But more importantly, it turns out that $\mathrm{Extr}(\sigma)$ is still not necessarily a minimal causal representation of σ. We present an example outside of the interpretation of affine IPA as it is more succinct, but it is easy to find similar examples within the interpretation.

▸ **Example 24.** Let A be a non-negative arena, with two concurrent events $\ominus$ and $\oplus$. Consider the **GM**-strategy $\sigma : A_1 \parallel A_2$ with plays (annotations are for disambiguation):

$$\sigma = \mathrm{Plays}\left(\begin{array}{cc}\ominus_1 & \ominus_2\\ \downarrow & \downarrow\\ \oplus_1 & \oplus_2\end{array}\right) \cup \mathrm{Plays}\left(\begin{array}{cc}\ominus_1 & \ominus_2\\ & \searrow\\ \oplus_1 & \oplus_2\end{array}\right) \cup \mathrm{Plays}\left(\begin{array}{cc}\ominus_1 & \ominus_2\\ \swarrow & \\ \oplus_1 & \oplus_2\end{array}\right)$$

All three po-plays are extremal in σ. However, despite being extremal, the first po-play is **redundant**: it can be removed, yielding the same **GM**-strategy. Indeed, call the three po-plays above $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$; and take $s \in \mathrm{Plays}(\mathbf{q}_1)$. If $s \notin \mathrm{Plays}(\mathbf{q}_2)$, then $\oplus_2 \leq_s \ominus_1$ as this is the only constraint in $\mathbf{q}_2$. Likewise, $s \notin \mathrm{Plays}(\mathbf{q}_3)$ means that $\oplus_1 \leq_s \ominus_2$. But these constraints, put together with those of $\mathbf{q}_1$, yield a contradiction. Therefore $s \in \mathrm{Plays}(\mathbf{q}_2) \cup \mathrm{Plays}(\mathbf{q}_3)$. The two extremal po-plays $\mathbf{q}_2, \mathbf{q}_3$ yield a smaller representation of σ.

In the example above, $\{\mathbf{q}_2, \mathbf{q}_3\}$ is the *unique* minimal causal representation for σ. But can we always reach such a canonical representation by removing redundant extremals?

4.2.3 Causally ambiguous GM-strategies

Until this point, and including Example 24, all the examples of **GM**-strategies considered in this paper have a unique minimal causal representation, *i.e.* a unique set of extremal po-plays with minimal cardinality. They are all *causally unambiguous*:

▸ **Definition 25.** For A a finite arena, a **GM**-strategy $\sigma : A$ is **causally ambiguous** if there are (at least) two distinct sets of extremal po-plays of minimal cardinality $X = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ and $Y = \{\mathbf{q}'_1, \ldots, \mathbf{q}'_n\}$, such that $\sigma = \bigcup_{1 \leq i \leq n} \mathrm{Plays}(\mathbf{q}_i) = \bigcup_{1 \leq i \leq n} \mathrm{Plays}(\mathbf{q}'_i)$.

To conclude this paper, we show the following result.

▸ **Theorem 26.** *There is a term of affine IPA:*

$$\vdash M : ((\mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com}) \multimap \mathbf{com}) \multimap \mathbf{com}$$

such that $[\![M]\!]_{\mathbf{GM}}$ *is causally ambiguous.*

Proof. We first exhibit a causally ambiguous **GM**-strategy outside of the interpretation of affine IPA, and then sketch how the same phenomenon can be replicated via a term.

Figure 6 displays five po-plays $\mathbf{q}_1, \ldots, \mathbf{q}_5$, generating a **GM**-strategy $\sigma = \cup_{1 \leq i \leq 5} \mathrm{Plays}(\mathbf{q}_i)$ – the game A is the same as in Example 24. A rather tedious but direct verification ensures that they are all extremal: for that, it suffices to check that for each of these po-plays,

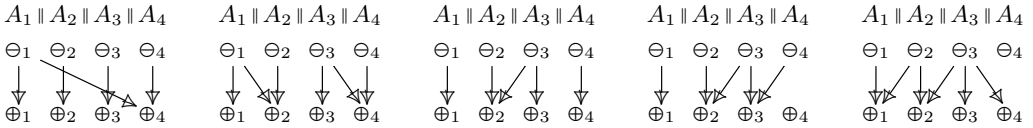


Figure 6 Extremal generators $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4$ and $\mathbf{q}_5$ of a causally ambiguous **GM**-strategy.

dropping any of the causal links unlocks a play not yet in σ. For instance, dropping the diagonal immediate causal link in $\mathbf{q}_1$ unlocks the play $\ominus_4 \oplus_4 \ominus_2 \oplus_2 \notin \sigma$.

Then, we note that $\mathbf{q}_2$ is redundant. Indeed, $\mathrm{Plays}(\mathbf{q}_2) \subseteq \mathrm{Plays}(\mathbf{q}_1) \cup \mathrm{Plays}(\mathbf{q}_3)$: as in Example 24, we cannot have at the same time $\oplus_4 \leq_s \ominus_1$ and $\oplus_2 \leq_s \ominus_3$ in $s \in \mathrm{Plays}(\mathbf{q}_2)$. Perhaps less obviously, $\mathbf{q}_3$ is redundant as well: we have $\mathrm{Plays}(\mathbf{q}_3) \subseteq \mathrm{Plays}(\mathbf{q}_2) \cup \mathrm{Plays}(\mathbf{q}_4) \cup \mathrm{Plays}(\mathbf{q}_5)$. Indeed, take $s \in \mathrm{Plays}(\mathbf{q}_3)$. If $s \notin \mathrm{Plays}(\mathbf{q}_4)$, then $\oplus_3 \leq_s \ominus_4$. If $s \notin \mathrm{Plays}(\mathbf{q}_5)$, then either $\oplus_1 \leq_s \ominus_2$ or $\oplus_4 \leq_s \ominus_3$, but the latter is incompatible as the constraints we already have on $\ominus_3, \oplus_3, \ominus_4, \oplus_4$ yield a cycle. Thus $\oplus_1 \leq_s \ominus_2$. But then if $s \notin \mathrm{Plays}(\mathbf{q}_2)$, then $\oplus_2 \leq_s \ominus_1$ or $\oplus_4 \leq_s \ominus_3$, but both possibilities yield a cycle; absurd.

None of $\mathbf{q}_1, \mathbf{q}_4, \mathbf{q}_5$ are redundant: only $\mathbf{q}_2$ and $\mathbf{q}_3$. Removing *both* $\mathbf{q}_2$ and $\mathbf{q}_3$ leads to the loss of the play $\ominus_3 \oplus_3 \ominus_4 \oplus_4 \ominus_1 \oplus_1$. There are two distinct minimal sets of extremals $\{\mathbf{q}_1, \mathbf{q}_3, \mathbf{q}_4, \mathbf{q}_5\}$ and $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_4, \mathbf{q}_5\}$, both generating σ – so σ is causally ambiguous.

We replicate this in affine IPA. First, we replace each A with $\mathbf{com}$. However, $\mathbf{q}_4$ and $\mathbf{q}_5$ do not have the causal link $\ominus_4 \rightarrowtriangle \oplus_4$; so we need *five* occurrences of $\mathbf{com}$, organised as $\mathbf{com}_1 \parallel \mathbf{com}_2 \parallel \mathbf{com}_3 \parallel \mathbf{com}_4 \parallel \mathbf{com}'_4$, where $\mathbf{run}'_4$, $\mathbf{done}_4$ play the role of $\ominus_4, \oplus_4$ and $\oplus'_4$ is ignored. This yields $\sigma' : \mathbf{com}_1 \parallel \mathbf{com}_2 \parallel \mathbf{com}_3 \parallel \mathbf{com}_4 \parallel \mathbf{com}'_4$ causally ambiguous. This is not a type of affine IPA (and σ' is not well-threaded), so instead we lift σ' to:

$$\sigma'' : [\![((\mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com} \multimap \mathbf{com}) \multimap \mathbf{com}) \multimap \mathbf{com}]\!]$$

Using variables, one can implement in affine IPA each of the po-plays corresponding in this type to the $\mathbf{q}_i$s above. It is also easy to define a non-deterministic choice operation in affine IPA, using which these are put together to define M such that $[\![M]\!]_{\mathbf{GM}} = \sigma''$. ◂

5 Conclusions

The phenomenon presented here is fairly robust, and causally ambiguous strategies would most likely emerge as well in other concurrent programming languages. Since interleaving games models are inherently related with observational equivalence as they exactly capture the observable behaviour of programs, it seems that unfortunately we cannot use the causal model presented here or those of *e.g.* [11, 3] to give canonical compact representations of concurrent programs up to contextual equivalence.

Causal structures are however still very relevant for other purposes (*e.g.* model-checking, error diagnostics, weak memory models, ...), and constructing them compositionally from programs remains an interesting challenge.

Acknowledgements. We are also grateful to Andrzej Murawski for interesting discussions on the topic.

References

1. Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, and C.-H. Luke Ong. Applying game semantics to compositional software modeling and verification. In Kurt Jensen and Andreas Podelski, editors, *TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 421–435. Springer, 2004.
2. Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. Concurrent games. *ArXiv*, 2015. URL: `http://arxiv.org/abs/1604.04390`.
3. Simon Castellan, Pierre Clairambault, and Glynn Winskel. The parallel intensionally fully abstract games model of PCF. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 232–243. IEEE, 2015.
4. Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 363–375. ACM, 2007.
5. Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Logic*, 151(2-3):89–114, 2008.
6. Dan R. Ghica, Andrzej S. Murawski, and C.-H. Luke Ong. Syntactic control of concurrency. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 683–694. Springer, 2004.
7. Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.
8. J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.
9. Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 395–411. Springer, 2007.
10. C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS) 2006, 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006.
11. Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS*, pages 409–418. IEEE Computer Society, 2011.
12. Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS*, volume 11, pages 409–418, 2011.
13. Ulrich Schöpp. On the relation of interaction semantics to continuations and defunctionalization. *Logical Methods in Computer Science*, 10(4), 2014.

Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types*

Marco Carbone[1], Sam Lindley[2], Fabrizio Montesi[3], Carsten Schürmann[4], and Philip Wadler[5]

1 **IT University of Copenhagen**
2 **University of Edinburgh**
3 **University of Southern Denmark**
4 **IT University of Copenhagen**
5 **University of Edinburgh**

Abstract

Wadler introduced Classical Processes (CP), a calculus based on a propositions-as-types correspondence between propositions of classical linear logic and session types. Carbone *et al.* introduced Multiparty Classical Processes, a calculus that generalises CP to multiparty session types, by replacing the duality of classical linear logic (relating two types) with a more general notion of coherence (relating an arbitrary number of types). This paper introduces variants of CP and MCP, plus a new intermediate calculus of Globally-governed Classical Processes (GCP). We show a tight relation between these three calculi, giving semantics-preserving translations from GCP to CP and from MCP to GCP. The translation from GCP to CP interprets a coherence proof as an arbiter process that mediates communications in a session, while MCP adds annotations that permit processes to communicate directly without centralised control.

1998 ACM Subject Classification F1.2 Modes of Computation: Parallelism and concurrency; F4.1 Mathematical logic: Proof theory

Keywords and phrases Multiparty Session Types, Linear Logic, Propositions as Types

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.33

1 Introduction

Session types, introduced by Honda, Vasconcelos, and Kubo [11, 20], are protocols that describe valid communication patterns in process calculi. A correspondence between process calculi and classical linear logic was found by Abramsky [1] and Bellin and Scott [3], and another between session types and intuitionistic linear logic by Caires and Pfenning [6, 7], in both of which channel types correspond to propositions of linear logic, processes to proofs, and communication to proof normalisation. Based on these, Wadler [22] introduced Classical Processes (CP), in which session types correspond to propositions of classical linear logic, processes to proofs, and communication to cut elimination. Key properties of session types such as deadlock freedom follow from key properties of linear logic such as cut elimination.

* Montesi was supported by the CRC project, grant no. DFF–4005-00304 from the Danish Council for Independent Research. Schürmann was partly supported by DemTech, grant no. 10-092309 from the Danish Council for Strategic Research. Lindley and Wadler were supported by the EPSRC grant ABCD (EP/K034413/1). This work was also supported by the EU COST Action IC1201 BETTY.

Last year, Carbone *et al.* [9] introduced Multiparty Classical Processes (MCP), which extends CP to the multiparty session types introduced by Honda, Yoshida, and Carbone [12]. In CP duality is defined between two propositions, whereas in MCP duality is replaced by coherence among multiple propositions. Coherence relates a global type (a description of a multiparty protocol) to many local types (the behaviours of each participant in a session).

MCP came at a cost as compared to CP. First, MCP required annotating logical connectives with roles, and it was unclear how such annotations related to classical linear logic. Second, MCP omitted the axiom and atomic and quantified propositions, losing support for parametric polymorphism. Third, MCP inverted the usual interpretation of connectives $\otimes$ and $\parr$, treating output as input and vice versa, which we no longer believe is a tenable position. This work presents an updated version of MCP that overcomes these shortcomings.

Section 2 introduces our variant of CP. We modify restriction to replace a single channel by two endpoints, yielding a logical reconstruction of the covariable formulation of session types due to Vasconcelos [20]. We partition types into input and output types (a slight variation of positive and negative polarities [13, 17]), which allows us to orient the axiom rule. And, in order to align with the subsequent development, we restrict the cut of axiom against a process to atomic variables, applying η-expansion to handle the remaining cases.

Section 3 introduces a calculus of Globally-governed Classical Processes (GCP), intermediate between CP and MCP. GCP uses global types to describe multiparty sessions, and coherence to relate the global type to many local types. GCP differs from MCP in not requiring annotations; types in GCP are the standard propositions of CP. We present a semantics-preserving translation from GCP into CP, where a global type is translated into an *arbiter*, an auxiliary process that coordinates communication. Caires and Perez [4] introduce a *medium* which is similar to our arbiter; we compare our work with theirs in Section 5. We show that under our translation GCP is simulated by CP. We also give a translation from CP to GCP and show that it too is a simulation.

Section 4 introduces our variant of MCP. MCP augments GCP with annotations which permit processes to communicate directly without centralised control. Whereas the original MCP refers to sessions and roles, our variant uses a simpler formulation based exclusively on endpoints. Our variant also restores the proper correspondence of $\otimes$ with output and $\parr$ with input, and supports parametric polymorphism. We present a semantics-preserving translation of MCP into GCP, which simply consists of erasing annotations. We show under our translation there is a bisimulation between MCP and GCP.

Section 5 discusses related and future work.

We illustrate our encodings using the classic 2-buyer protocol [12] as a running example. Two buyers, B_1 and B_2, attempt to buy a book together from a seller S. First, B_1 sends the title of the book that she wishes to purchase to S, who in turn sends a quote to both B_1 and B_2. Then, B_1 decides on how much she wishes to contribute, and informs B_2, who either pays the rest or cancels the transaction. A similar example appears in the original paper on MCP [9], but is degenerate in that it represents data using the unit type. There, it is not possible to replace the unit type with a non-degenerate type because of the inversion of $\otimes$ and $\parr$. Here, we present non-degenerate encodings in CP, GCP, and MCP.

2 Classical Processes (CP)

We describe our version of Classical Processes (CP), originally introduced by Wadler [22].

Types. We start by introducing propositions, which we interpret as session types. Let A, B, C, D range over propositions and X, Y range over atomic propositions.

A, B, C, D	::=	$A \otimes B$	*(send A, proceed as B)*	$A \parr B$	*(receive A, proceed as B)*
		$A \oplus B$	*(select A or B)*	$A \& B$	*(offer A or B)*
		0	*(unit for $\oplus$)*	$\top$	*(unit for $\&$)*
		1	*(unit for $\otimes$)*	$\bot$	*(unit for $\parr$)*
		$?A$	*(client request)*	$!A$	*(server accept)*
		X	*(atomic propositions)*	$X^\bot$	*(dual of atomic proposition)*
		$\exists X.A$	*(existential)*	$\forall X.A$	*(universal)*

We give a behavioural explanation to the types above. Proposition $A \otimes B$ is the type of a channel over which we send a fresh channel of type A and then continue as B. Dually, $A \parr B$ is the type of a channel over which we receive a channel of type A and then continue as B. Proposition $A \oplus B$ is the type of a channel over which we can select to proceed either with type A or with type B. Dually, $A \& B$ is the type of a channel offering a choice of proceeding with either type A or type B. Propositions 0, $\top$, 1 and $\bot$ are units for $\oplus$, $\&$, $\otimes$ and $\parr$, respectively. Proposition $?A$ is the type of a channel over which a client may request multiple invocations of a server. Dually, $!A$ is the type of a channel over which a server may accept multiple invocations from a client. Atomic propositions X and $X^\bot$, and universal propositions $\exists X.A$ and $\forall X.A$ model polymorphic channels [22].

Each type in the left-hand column is dual to the corresponding type in the right-hand column. We write $A^\bot$ for the dual of A. We refer to types on the left as *output* types, and types on the right as *input* types. Our output and input types correspond, respectively, to the standard notions in logic of positive and negative types [13]. The one exception are the exponentials: we classify ? as an output type and ! as an input type, while logicians classify ? as negative and ! as positive. Exponentials are already known to have a less strong correspondence with positive and negative types than other connectives. For instance, negative types have invertible rules while positive types do not, the exception being exponentials, where the rule for ! is invertible while the rules for weakening and contraction are not.

Processes. In CP, proofs correspond to proof terms, expressed in a π-calculus with sessions. Let x, y, and z range over channel endpoints. The syntax of processes is given by the proof terms (denoted in red) shown in Figure 1. In an output operation the sent object is always contained in square brackets $[\ldots]$, and, dually, in an input operation the received variable is always bound in round parentheses $(\ldots)$. As in the internal π-calculus [18], the object y in a send $x[y].(P \mid Q)$ and in a client request $?x[y].P$ is bound (this is not the case in selection and send type). A link process $x \to y^B$ forwards communications from x to y. A restriction $(\boldsymbol{\nu} x^A y)\,(P \mid Q)$ pairs two endpoints x and y into a session (as done by Vasconcelos [20]).

We give the type rules for CP in Figure 1. All rules are standard CLL rules. In the logic, link corresponds to axiom and restriction to cut. Their interpretation as proof terms follows that of Wadler [22], with rules Axiom and Cut adopting the new syntax.

Semantics. CLL is equipped with proof transformations that give semantics to CP processes. Figure 2 displays structural equivalence rules, η-expansions and β-reductions for processes and cuts. Structural equivalences permit swapping the names in a link and in a restriction, and reassociating two restrictions. The η-expansions for link do not appear in the original presentation of CP [22], but are standard and appear elsewhere [16]. We reformulate CP to use them, as they prove helpful in defining GCP in the next section. The expansion replaces

$$\frac{}{x \to y^{A} \vdash x : A^{\perp}, y : A}\ \textsc{Axiom} \qquad \frac{P \vdash \Gamma, x : A \quad Q \vdash \Delta, y : A^{\perp}}{(\nu x^{A} y)\,(P \mid Q) \vdash \Gamma, \Delta}\ \textsc{Cut}$$

$$\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}\ \otimes \qquad \frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}\ \parr$$

$$\frac{P \vdash \Gamma, x : A}{x[\mathsf{inl}].P \vdash \Gamma, x : A \oplus B}\ \oplus_1 \qquad \frac{P \vdash \Gamma, x : B}{x[\mathsf{inr}].P \vdash \Gamma, x : A \oplus B}\ \oplus_2 \qquad \frac{P \vdash \Gamma, x : A \quad Q \vdash \Gamma, x : B}{x.\mathsf{case}(P, Q) \vdash \Gamma, x : A \,\&\, B}\ \&$$

$$\frac{P \vdash \Gamma, y : A}{?x[y].P \vdash \Gamma, x : ?A}\ ? \qquad \frac{P \vdash\ ?\Gamma, y : A}{!x(y).P \vdash\ ?\Gamma, x : !A}\ ! \qquad \frac{P \vdash \Gamma}{P \vdash \Gamma, x : ?A}\ \textsc{Weaken}$$

$$\frac{P \vdash \Gamma, x : B[A/X]}{x[A].P \vdash \Gamma, x : \exists X.B}\ \exists \qquad \frac{P \vdash \Gamma, x : B \quad X \notin \mathsf{ftv}(\Gamma)}{x(X).P \vdash \Gamma, x : \forall X.B}\ \forall \qquad \frac{P \vdash \Gamma, y : ?A, z : ?A}{P\{x/y, x/z\} \vdash \Gamma, x : ?A}\ \textsc{Contract}$$

$$\frac{}{x[] \vdash x : 1}\ 1 \qquad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \bot}\ \bot \qquad \text{(no rule for 0)} \qquad \frac{}{x.\mathsf{case}() \vdash \Gamma, x : \top}\ \top$$

Figure 1 CP, Type Rules.

a link, step-by-step, by processes that perform the communications required by its type. The β-reductions of CP correspond to the cut elimination rules in CLL, and are standard.

We omit commuting conversions, which lift prefixes out of cuts on different endpoints; they are as in [22]. Structural equivalence and reductions apply inside a cut, but not inside a prefix. We use juxtaposition for the composition of relations, write R^{+} for the transitive closure and R^{*} for the transitive reflexive closure of relation R. We write $\Longrightarrow$ for $\equiv \longrightarrow \equiv$.

▶ **Theorem 1** (Subject reduction for CP). *If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$.*

▶ **Theorem 2** (Cut elimination for CP). *If $P \vdash \Gamma$ then there exists a Q such that $P \Longrightarrow^{*} Q$ and Q is not a cut.*

The proof of cut elimination for CP is standard [22]. All other theorems in this paper follow by straightforward induction.

▶ **Example 3.** We now describe the 2-buyer protocol in CP.

First, we proceed by providing a set of suitable types for the endpoints along which B_1, B_2, and S communicate. We assume that $\mathbf{cost}, \mathbf{name}, \mathbf{addr}$ are atomic propositions. In CP, B_1's endpoint has type $\mathbf{name} \otimes \mathbf{cost}^{\perp} \parr \mathbf{cost} \otimes 1$, B_2's endpoint has type $\mathbf{cost}^{\perp} \parr \mathbf{cost}^{\perp} \parr \big((\mathbf{addr} \otimes 1) \oplus 1\big)$, and S's endpoint has type $\mathbf{name}^{\perp} \parr \mathbf{cost} \otimes \mathbf{cost} \otimes \big((\mathbf{addr}^{\perp} \parr \bot) \& \bot\big)$. In terms of traditional multiparty sessions, these types amount to local types, but with the roles erased.

Second, we define an *arbiter* process over three endpoints, b_1, b_2, s, one each for communication with B_1, B_2, and S, respectively.

$$\begin{aligned} &b_1(b_1').s[s'].(b_1' \to s'^{\,\mathbf{name}} \mid s(s').b_1[b_1'].(s' \to b_1'^{\,\mathbf{cost}} \mid \\ &\quad s(s').b_2[b_2'].(s' \to b_2'^{\,\mathbf{cost}} \mid b_1(b_1').b_2[b_2'].(b_1' \to b_2'^{\,\mathbf{cost}} \mid \\ &\qquad b_2.\mathsf{case}(s[\mathsf{inl}].b_2(b_2').s[s'].(b_2' \to s'^{\,\mathbf{addr}} \mid b_1().b_2().s[]), s[\mathsf{inr}].b_1().b_2().s[])))) \end{aligned}$$

The arbiter process mediates between the parties, incorporating the missing role information that would normally appear in local types. In terms of traditional multiparty sessions, the arbiter corresponds to the dual of a global type.

Finally, an instantiation of the protocol must cut implementations of B_1, B_2, and S against the arbiter process.

Structural equivalence (Processes)

$$y \to x^{A^\perp} \equiv x \to y^A \qquad (\boldsymbol{\nu} y^{A^\perp} x)\,(Q \mid P) \equiv (\boldsymbol{\nu} x^A y)\,(P \mid Q)$$
$$(\boldsymbol{\nu} w^B z)\,(P \mid (\boldsymbol{\nu} x^A y)\,(Q \mid R)) \equiv (\boldsymbol{\nu} x^A y)\,((\boldsymbol{\nu} w^B z)\,(P \mid Q) \mid R)$$

η-expansions (Processes)

$$\begin{array}{rcl@{\qquad}rcl} x \to y^{A\otimes B} & \longrightarrow & x(u).y[v].(u \to v^A \mid x \to y^B) & x \to y^1 & \longrightarrow & x().y[] \\ x \to y^{A\oplus B} & \longrightarrow & x.\mathsf{case}(y[\mathsf{inl}].x \to y^A, y[\mathsf{inr}].x \to y^B) & x \to y^0 & \longrightarrow & x.\mathsf{case}() \\ x \to y^{?A} & \longrightarrow & !x(u).?y[v].u \to v^A & x \to y^{\exists X.A} & \longrightarrow & x(X).y[X].x \to y^A \end{array}$$

β-reductions (Processes)

$$\begin{array}{rcl} (\boldsymbol{\nu} x^X y)\,(w \to x^X \mid Q) & \longrightarrow & Q\{w/y\} \\ (\boldsymbol{\nu} x^{A\otimes B} y)\,(x[u].(P \mid Q) \mid y(v).R) & \longrightarrow & (\boldsymbol{\nu} u^A v)\,(P \mid (\boldsymbol{\nu} x^B y)\,(Q \mid R)) \\ (\boldsymbol{\nu} x^1 y)\,(x[] \mid y().P) & \longrightarrow & P \\ (\boldsymbol{\nu} x^{A\oplus B} y)\,(x[\mathsf{inl}].P \mid x.\mathsf{case}(Q,R)) & \longrightarrow & (\boldsymbol{\nu} x^A y)\,(P \mid Q) \\ (\boldsymbol{\nu} x^{A\oplus B} y)\,(x[\mathsf{inr}].P \mid x.\mathsf{case}(Q,R)) & \longrightarrow & (\boldsymbol{\nu} x^B y)\,(P \mid R) \\ \multicolumn{3}{c}{\text{(no rule for 0 with } \top\text{)}} \\ (\boldsymbol{\nu} x^{?A} y)\,(?x[u].Q \mid !y(v).P) & \longrightarrow & (\boldsymbol{\nu} u^A v)\,(P \mid Q) \\ (\boldsymbol{\nu} x^{?A} y)\,(P \mid !y(v).Q) & \longrightarrow & P \\ (\boldsymbol{\nu} x^{?A} y)\,(P\{x/x', x/x''\} \mid !y(v).Q) & \longrightarrow & (\boldsymbol{\nu} x'^{?A} y')\,(((\boldsymbol{\nu} x''^{?A} y'')\,(P \mid !y'(v).Q)) \mid !y''(v).Q) \\ (\boldsymbol{\nu} x^{\exists X.B} y)\,(x[A].P \mid y(X).Q) & \longrightarrow & (\boldsymbol{\nu} x^{B\{A/X\}} y)\,(P \mid Q\{A/X\}) \end{array}$$

Figure 2 CP, Structural Equivalence and Reduction Rules.

Directing link and restriction. We make a useful observation here that does not appear in [22]. A link or a restriction will always be between an input type and its dual output type. The swap rule $y \to x^{A^\perp} \equiv x \to y^A$ may always be applied to orient a link so that the input type is on the left and the output type on the right, and in this case the flow of information in the link will always be from left to right. Similarly, the swap rule may always be applied to orient a restriction so that the output type is on the left and the input type on the right, and in this case the flow of information in the restriction will always be from left to right. These descriptions are pleasingly similar, but note that input and output swap positions in the description of link and restriction!

May one invert output and input? The original presentation of MCP [9] states: "Our work inverts the interpretation of $\otimes$ as output and $\parr$ as input given in [3]. This makes our process terms in line with previous developments of multiparty session types, where communications go from one sender to many receivers [10]." Implicit is the claim that whether one assigns $\otimes$ to output and $\parr$ to input is a convention that may be inverted without harm. Here we argue that such a view is not tenable. As an illustration, consider the derivation in Figure 3; it types a process that inputs a single bit (or, more precisely, offers a choice between two units) and outputs two copies of that bit (or, more precisely, twice makes a selection between two units, both times echoing the choice made on input).

Would it make sense to modify our interpretation of the process terms (in red), so that inputs are considered as outputs and vice versa? Absolutely not! The interpretation of $\&$ as offering a choice and $\oplus$ as making a selection is uncontroversial (it is also accepted in [9]). Once the interpretations of $\&$ and $\oplus$ are fixed then in this example the only way to view $\parr$ is as input and $\otimes$ is as output. Nor is it possible to assign a sensible view if we swap the interpretations of $\&$ and $\oplus$, since then the process would input two bits which must be

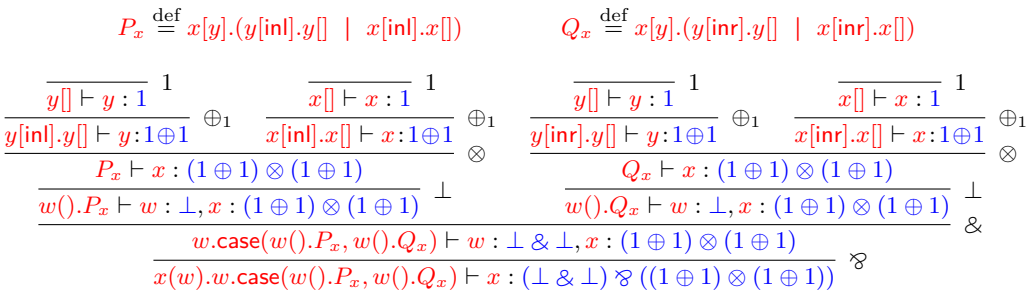


Figure 3 Example: duplicating a bit.

identical and output one bit which is equal to both inputs; this would eliminate the idea that each channel can have its value chosen independently, and we cannot see how to make sense of the process calculus that would result. More generally, it makes perfect sense when outputting one channel along another channel to assign the behaviour of the output channel and of remaining behaviour of the original channel to two separate processes (as happens when $\otimes$ is interpreted by output), and when inputting one channel along another channel to assign the behaviour of both channels to a single process (as happens when $\parr$ is interpreted by input). But it makes no sense to take the inverse interpretation, and when inputting one channel along another channel assign the behaviour of the input channel and the behaviour of the remainder of the original channel to two separate processes—then the behaviour of the input could have no effect on the behaviour of the remainder of the original channel, which contradicts the notion of how input is intended to behave. (We are grateful to Bob Atkey for this general argument about input and output.)

3 Globally-governed Classical Processes (GCP)

In CP, communications between two parties take place over a binary cut. In this section, we introduce GCP, by replacing the *binary* cut in CP with a *multiparty* cut (called coherence cut), where communications among multiple parties are governed by a *global type*.

Types. Coherence, introduced in [9], generalises the notion of duality found in classical linear logic. Two propositions A and B are dual if each output type in A is matched by an input type in B and vice versa. Duality ensures that two processes can be composed safely, by connecting two respective endpoints that have compatible interfaces (dual types). In GCP, we wish to also compose more than two processes, and therefore, we need a notion that generalises duality to compatibility among n processes. The notion of coherence serves this purpose and it is given as a proof system whose judgements have the form $G \vDash \Gamma$ where Γ is a set of compatible types and G, called the *global type*, is the corresponding proof term.

Global types give the flow of communications that sessions must follow. Their syntax is given by the proof terms (in red) in Figure 4. Let G, H range over global types, and write $\tilde{x}$ to abbreviate the sequence $(x_i)_i$. In $\tilde{x} \to y(G).H$, endpoints $\tilde{x}$ each send a message to y to create a new session of type G and then continue as H. In $\tilde{x} \to y$, endpoints $\tilde{x}$ each send a message to y to terminate the session. In $x \to \tilde{y}.\mathsf{case}(G, H)$, endpoint x sends a choice to endpoints $\tilde{y}$ on whether to proceed as G or H. In $x \to \tilde{y}.\mathsf{case}()$, x sends an empty choice to $\tilde{y}$. In $!x \to \tilde{y}(G)$, client x sends a request to servers $\tilde{y}$ to create a session of global type G. In

$$\frac{G \vDash (x_i : A_i)_i, y : C \quad H \vDash \Gamma, (x_i : B_i)_i, y : D}{\tilde{x} \to y(G).H \vDash \Gamma, (x_i : A_i \otimes B_i)_i, y : C \parr D}\ \otimes\parr \qquad \frac{}{\tilde{x} \to y \vDash (x_i : 1)_i, y : \bot}\ 1\bot$$

$$\frac{G \vDash \Gamma, x : A, (y_i : C_i)_i \quad H \vDash \Gamma, x : B, (y_i : D_i)_i}{x \to \tilde{y}.\mathsf{case}(G, H) \vDash \Gamma, x : A \oplus B, (y_i : C_i \,\&\, D_i)_i}\ \oplus\& \qquad \frac{}{x \to \tilde{y}.\mathsf{case}() \vDash \Gamma, x : 0, (y_i : \top)_i}\ 0\top$$

$$\frac{G \vDash x : A, (y_i : B_i)_i}{!x \to \tilde{y}(G) \vDash x : ?A, (y_i : !B_i)_i}\ ?! \qquad \frac{G \vDash \Gamma, x : A, (y_i : B_i)_i \quad X \notin \mathsf{ftv}(\Gamma)}{x \to \tilde{y}.(X)G \vDash \Gamma, x : \exists X.A, (y_i : \forall X.B_i)_i}\ \exists\forall$$

$$\frac{}{x^A \to y \vDash x : A, y : A^\perp}\ \textsc{Axiom} \qquad \frac{(P_i \vdash \Gamma_i, x_i : A_i)_i \quad G \vDash (x_i : A_i)_i}{(\boldsymbol{\nu} \tilde{x}^{\tilde{A}} : G)\,(\tilde{P}) \vdash \tilde{\Gamma}}\ \textsc{CCut}$$

Figure 4 GCP, Coherence Rules and Coherence Cut.

$x \to \tilde{y}.(X)G$, endpoint x sends a type to endpoints $\tilde{y}$ and the protocol proceeds as G. In $x^A \to y$, x is connected to y. Thus, a coherence cut in which the global type is an axiom behaves exactly like a binary cut.

Types for endpoints in GCP are identical to those of CP. Then, coherence, denoted by $\vDash$, is defined by the rules given in Figure 4. Rule $\otimes\parr$ says that if we have some endpoints of type $A_i \otimes B_i$ and an endpoint of type $C \parr D$ then a communication can happen (denoted as $\tilde{x} \to y$ in the global type) which will create a new session with endpoints of type $(A_i)_i$ and C, and the old session will continue as $\Gamma, (B_i)_i, D$. All other rules are similar.

The rules and the proof terms for GCP are identical to those of CP save that the standard binary cut Cut is replaced by the coherence cut CCut, given in Figure 4, with the proof term $(\boldsymbol{\nu} \tilde{x}^{\tilde{A}} : G)\,(\tilde{P})$. In CCut, the use of coherence becomes clear: the global type G governs all communication between the processes $\tilde{P}$. Each endpoint x_i in each P_i is bound with type A_i, and the coherence relation $G \vDash (x_i : A_i)_i$ ensures that such processes can safely communicate on such endpoints. The types $\tilde{A}$ adorning the endpoints $\tilde{x}$ are superfluous, since they are determined by G, but will come in handy when we write the translation from GCP to CP. It will follow from the translation, presented below, that if $G \vDash (x_i : A_i)_i$ holds then $\vdash (A_i^\perp)_i$ is derivable in classical linear logic. As in the original formulation of MCP [9], the restriction of coherence to two parties is exactly duality: $G \vDash x : A, y : B$ if and only if $A = B^\perp$. But, as we shall see at the end of this section, the connection between coherence and duality goes deeper than this.

Semantics. Figure 5 displays the interesting structural equivalences, η-expansions, and β-reductions of GCP. In addition we retain the structural equivalence for axiom and η-expansions of processes from Figure 2. We omit the straightforward commuting conversions, each one allowing a prefix to be lifted out of a coherence cut; see [9]. The β-rules are similar to CP, but engage multiple parties and all communication is coordinated by global types. Structural equivalence and reduction applies inside a coherence cut (including inside a global type), but not inside a prefix. Note that η-expansion on processes is necessary for cut-elimination to hold. For example, the process $(\boldsymbol{\nu} x^{A \otimes B} y^{A^\perp \parr C} z^D : x \to y(G).H)\,(w \to x^{A \otimes B} \mid y(y').P \mid Q)$ can only reduce if we first expand $w \to x^{A \otimes B}$ to $w(w').x[x'].(w' \to x'^A \mid w \to x^B)$.

▶ **Theorem 4** (Subject reduction for GCP). *If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$.*

▶ **Theorem 5** (Cut elimination for GCP). *If $P \vdash \Gamma$, then there exists Q such that $P \Longrightarrow^* Q$ and Q is not a cut.*

Structural equivalence (Global types and processes)

$$
\begin{aligned}
y^{A^\perp} \to x &\equiv x^A \to y \\
(\boldsymbol{\nu}\tilde{w}, y, x, \tilde{z} : G)\,(\tilde{P} \mid R \mid Q \mid \tilde{S}) &\equiv (\boldsymbol{\nu}\tilde{w}, x, y, \tilde{z} : G)\,(\tilde{P} \mid Q \mid R \mid \tilde{S}) \\
(\boldsymbol{\nu}z, \tilde{w} : H)\,((\boldsymbol{\nu}x, \tilde{y} : G)\,(P \mid \tilde{R}) \mid \tilde{Q}) &\equiv (\boldsymbol{\nu}x, \tilde{y} : G)\,((\boldsymbol{\nu}z, \tilde{w} : H)\,(P \mid \tilde{Q}) \mid \tilde{R})
\end{aligned}
$$

η-expansions (Global types)

$$
\begin{aligned}
x^{A\otimes B} \to y &\longrightarrow x \to y(x^A \to y).x^B \to y \\
x^{A\oplus B} \to y &\longrightarrow x \to y.\mathsf{case}(x^A \to y, x^B \to y) \\
x^{?A} \to y &\longrightarrow !x \to y(x^A \to y)
\end{aligned}
\qquad
\begin{aligned}
x^1 \to y &\longrightarrow x \to y \\
x^0 \to y &\longrightarrow x \to y.\mathsf{case}() \\
x^{\exists X.A} \to y &\longrightarrow x \to y.(X)x^A \to y
\end{aligned}
$$

β-reductions (Global types and processes)

$$
\begin{aligned}
(\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : \tilde{x} \to y(G).H)\,((x_i[x_i'].(P_i \mid Q_i))_i \mid y(y').R \mid \tilde{S}) &\longrightarrow \\
(\boldsymbol{\nu}\tilde{x}', y' : G\{\tilde{x}'/\tilde{x}, y'/y\})\,(\tilde{P} \mid (\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : H)\,(\tilde{Q} \mid R \mid \tilde{S})) \\
(\boldsymbol{\nu}\tilde{x}, y : \tilde{x} \to y)\,((x_i[])_i \mid y().P) &\longrightarrow P \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,(x[\mathsf{inl}].P \mid (y_i.\mathsf{case}(Q_i, R_i))_i \mid \tilde{S}) &\longrightarrow (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : G)\,(P \mid \tilde{Q} \mid \tilde{S}) \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,(x[\mathsf{inr}].P \mid (y_i.\mathsf{case}(Q_i, R_i))_i \mid \tilde{S}) &\longrightarrow (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : H)\,(P \mid \tilde{R} \mid \tilde{S}) \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(?x[x'].P \mid (!y_i(y_i').Q)_i) &\longrightarrow (\boldsymbol{\nu}x', \tilde{y}' : G\{x'/x, \tilde{y}'/\tilde{y}\})\,(P \mid \tilde{Q}) \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(P \mid (!y_i(y_i').Q)_i) &\longrightarrow P, \quad \text{if } x \notin \mathsf{fv}(P) \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(P\{x/w, x/z\} \mid (!y_i(y_i').Q)_i) &\longrightarrow \\
(\boldsymbol{\nu}w, \tilde{y} : !w \to \tilde{y}(G\{w/x\}))\,((\boldsymbol{\nu}z, \tilde{y} : !z \to \tilde{y}(G\{z/x\}))\,(P \mid (!y_i(y_i').Q)_i) \mid (!y_i(y_i').Q)_i) & \\
(\boldsymbol{\nu}x, \tilde{y} : x \to \tilde{y}.(X)G)\,(x[A].P \mid (x_i(X).Q_i)) &\longrightarrow (\boldsymbol{\nu}x, \tilde{y} : G\{A/X\})\,(P \mid (\tilde{Q})\{A/X\}) \\
(\boldsymbol{\nu}x, y : x^X \to y)\,(w \to x^X \mid Q) &\longrightarrow Q\{w/y\} \\
(\boldsymbol{\nu}x, y : y^X \to x)\,(x \to w^X \mid Q) &\longrightarrow Q\{w/y\}
\end{aligned}
$$

Figure 5 GCP, Structural Equivalence and Reduction Rules.

Cut elimination in GCP depends crucially on the η-expansions both in processes and in global types. One could prove cut elimination directly for GCP, but instead we will appeal to the standard cut elimination result for CP.

As with the original MCP [9], it is sound to swap independent actions in global types in GCP (and our new variant of MCP). We omit these rules, which are the expected ones [9].

▶ **Example 6.** We continue with the exposition of our running example. To represent the 2-buyer protocol in GCP, it is no longer necessary to construct an unwieldy arbiter process as in Example 3 in CP, but rather construct a global type, which is easily derived in GCP. Let G be the global type:

$$
\begin{aligned}
&B_1 \to S(B_1{}^{\mathbf{name}} \to S).\ S \to B_1(S^{\mathbf{cost}} \to B_1).\\
&\quad S \to B_2(S^{\mathbf{cost}} \to B_2).\ B_1 \to B_2(B_1{}^{\mathbf{cost}} \to B_2).\\
&\qquad B_2 \to S.\mathsf{case}(B_2 \to S(B_2{}^{\mathbf{addr}} \to S).(B_1, B_2) \to S,\ (B_1, B_2) \to S))
\end{aligned}
$$

Then, we can immediately prove the following coherence judgement:

$$
G \vDash \begin{array}{rcl} B_1 & : & \mathbf{name} \otimes \mathbf{cost}^\perp \parr \mathbf{cost} \otimes 1, \\ B_2 & : & \mathbf{cost}^\perp \parr \mathbf{cost}^\perp \parr ((\mathbf{addr} \otimes 1) \oplus 1), \\ S & : & \mathbf{name}^\perp \parr \mathbf{cost} \otimes \mathbf{cost}^\perp \otimes ((\mathbf{addr}^\perp \parr \perp) \,\&\, \perp) \end{array}
$$

Using this global type in a CCut, we can compose the three processes for B_1, B_2, and S.

Translations. Translations from GCP to CP ($[\![-]\!]$) and from CP to GCP ($(\!|-|\!)$) are given in Figure 6. The function $[\![-]\!]$ from GCP to CP is a homomorphism on all process forms except coherence cut. Coherence cut is translated into a series of binary cuts where the global type

Global cut as binary cut

$$[\![(\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,(\tilde{P})]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu}x_1{}^{A_1}y_1)\,([\![P_1]\!] \mid \cdots(\boldsymbol{\nu}x_n{}^{A_n}y_n)\,([\![P_n]\!] \mid [\![G]\!]\{\tilde{y}/\tilde{x}\})\cdots), \qquad \tilde{y} \text{ fresh}$$

Global types as processes

$$\begin{aligned}
[\![\tilde{x} \to y(G).H]\!] &\stackrel{\text{def}}{=} x_1(u_1).\cdots x_n(u_n).y[v].([\![G]\!]\{\tilde{u}/\tilde{x}, v/y\} \mid [\![H]\!]), && \tilde{u}, v \text{ fresh}\\
[\![\tilde{x} \to y]\!] &\stackrel{\text{def}}{=} x_1().\cdots x_n().y[]\\
[\![x \to \tilde{y}.\mathsf{case}(G,H)]\!] &\stackrel{\text{def}}{=} x.\mathsf{case}(y_1[\mathsf{inl}].\cdots y_n[\mathsf{inl}].[\![G]\!], y_1[\mathsf{inr}].\cdots y_n[\mathsf{inr}].[\![H]\!])\\
[\![x \to \tilde{y}.\mathsf{case}()]\!] &\stackrel{\text{def}}{=} x.\mathsf{case}()\\
[\![!x \to \tilde{y}(G)]\!] &\stackrel{\text{def}}{=} !x(u).?y_1[v_1].\cdots?y_n[v_n].[\![G]\!]\{u/x, \tilde{v}/\tilde{y}\}, && u, \tilde{v} \text{ fresh}\\
[\![x \to \tilde{y}.(X)G]\!] &\stackrel{\text{def}}{=} x(X).y_1[X].\cdots y_n[X].[\![G]\!]\\
[\![x^A \to y]\!] &\stackrel{\text{def}}{=} x \to y^A
\end{aligned}$$

Binary cut as global cut

$$(\!|(\boldsymbol{\nu}x^A y)\,(P \mid Q)|\!) = (\boldsymbol{\nu}x, y : y^A \to x)\,((\!|P|\!) \mid (\!|Q|\!))$$

Figure 6 Translations Between CP and GCP.

becomes an arbiter process that mediates all communication. The function $(\!|-|\!)$ maps CP processes to GCP. It is a homomorphism on all process forms except binary cut. Binary cut is translated into a coherence cut with two processes in which the global type is a link.

Why η-expansion? We now pause to explain how η-expansion simplifies the system. In the current formulation, the axiom cut rules apply only at atomic types. If we attempt to allow axiom cut at other types, then we need some way of reducing a coherence cut over an axiom process in which the global type is not itself an axiom. For instance, consider $(\boldsymbol{\nu}x_1{}^1, x_2{}^{\perp} : x_1 \to x_2)\,(P \mid x_2 \to w^1)$. Translating to CP, we obtain $(\boldsymbol{\nu}x_1{}^1y_1)\,(P \mid (\boldsymbol{\nu}x_2{}^{\perp}y_2)\,(x_2 \to w^1 \mid y_1().y_2[]))$ which reduces by the unrestricted axiom cut rule to $(\boldsymbol{\nu}x_1{}^1y_1)\,(P \mid y_1().w[])$. An obstacle to mapping this reduction back to GCP is that the substitution of w for y_2 occurs inside the arbiter process, that is, the image of a global type. In order to support this substitution we must generalise the CCut rule to allow free variables in global types. This in turn necessitates further reduction rules to prevent free variables in global types from blocking reduction. A more complex system including an unrestricted axiom cut rule is of practical interest as it admits implementations that take advantage of type erasure, but is beyond the scope of this paper.

▶ **Theorem 7** (Type preservation from GCP to CP).

1. *If $P \vdash \Gamma$ in GCP, then $[\![P]\!] \vdash \Gamma$ in CP.*
2. *If $G \vDash \Gamma$ in GCP, then $[\![G]\!] \vdash \Gamma^{\perp}$ in CP.*

A coherence judgement $G \vDash \Gamma$ translates to a judgement $[\![G]\!] \vdash \Gamma^{\perp}$, where $[\![G]\!]$ is an arbiter processes acting as an intermediary between the processes of a global session. It is typed in the dual of the environment in which we type the global type G. Write $\longrightarrow_{\eta}$ for η-expansion.

▶ **Theorem 8** (Simulation of GCP in CP).

1. *If $P \vdash \Gamma$ and $P \equiv Q$ in GCP, then $[\![P]\!] \equiv [\![Q]\!]$ in CP.*
2. *If $P \vdash \Gamma$ and $P \longrightarrow_{\eta} Q$ in GCP, then $[\![P]\!] \longrightarrow_{\eta} [\![Q]\!]$ in CP.*
3. *If $G \vDash \Gamma$ and $G \longrightarrow_{\eta} H$ in GCP, then $[\![G]\!] \longrightarrow_{\eta} [\![H]\!]$ in CP.*
4. *If $P \vdash \Gamma$ and $P \longrightarrow Q$ in GCP, then $[\![P]\!] \Longrightarrow^{+} [\![Q]\!]$ in CP.*

Each reduction in GCP is simulated by one or more reductions in CP. For instance, a cut involving a global type $\tilde{x} \to y(G).H$ performs a series of sends to the arbiter along $\tilde{x}$ followed by a receive from the arbiter along y. Theorem 8 shows that GCP is strongly normalising, by strong normalisation of CP (a standard result for classical linear logic).

▶ **Theorem 9** (Reflection of CP in GCP). *If $P \vdash \Gamma$ in GCP and $[\![P]\!] \longrightarrow Q'$ in CP, then there exists Q such that $P \Longrightarrow Q$ in GCP and $Q' \Longrightarrow^* [\![Q]\!]$ in CP.*

Theorem 9 shows that cut-elimination (Theorem 5), and hence deadlock-freedom, holds for GCP, by cut-elimination for CP. For if P does not reduce, then $[\![P]\!]$ must not reduce either, which means it is not a cut, which in turn means that P is not a cut.

▶ **Theorem 10** (Type preservation from CP to GCP). *If $P \vdash \Gamma$, then $(\!|P|\!) \vdash \Gamma$.*

▶ **Theorem 11** (Simulation of CP in GCP).
1. *If $P \vdash \Gamma$ and $P \equiv Q$ in CP, then $(\!|P|\!) \equiv (\!|Q|\!)$ in GCP.*
2. *If $P \vdash \Gamma$ and $P \longrightarrow Q$ in CP, then $(\!|P|\!) \longrightarrow^+ (\!|Q|\!)$ in GCP.*

Structural equivalence in CP is simulated by structural equivalence in GCP. Each reduction in CP is simulated by one or two reductions in GCP. Before performing the main reduction, it is sometimes necessary to η-expand a global link.

Coherence generalises duality. It is well known that axiom at any type is admissible given axiom at atomic types, and that the proof of admissibility corresponds to η-expansion [19, Chapter 6]. If we restrict axioms to atomic propositions, then every cut-free derivation of a judgement $\vdash A^{\perp}, A$ in CLL corresponds to the η-expansion of a link process $x \to y^A$. Note the close relation between η-expansion and the translation of global types. For instance, compare the η-expansion of $\parr\otimes$ in CP with the translation of the global type for $\parr\otimes$ in GCP. For CP, we have that $x \to y^{A \otimes B}$ expands to

$$x(u).y[v].(P \mid Q)$$

where $P = u \to v^A$ and $Q = x \to y^B$, while for GCP, we have that $\tilde{x} \to y(G).H$ translates to

$$x_1(u_1).\cdots x_n(u_n).y[v].(P \mid Q)$$

where $P = [\![G]\!]\{\tilde{u}/\tilde{x}, v/y\}$ and $Q = [\![H]\!]$. The relation is similarly close for each of the other logical connectives. Hence, duality corresponds to η-expansion and coherence corresponds to a straightforward generalisation of η-expansion.

4 Multiparty Classical Processes (MCP)

The semantics of GCP is governed, unlike in standard multiparty session types [12]. For example, process $x[w].(P \mid Q)$ in GCP does not say to which other endpoint the message should be routed; a communication can happen only under a restriction with a global type that pairs such output action with an input action, e.g., when in a context such as in $(\boldsymbol{\nu} xy\tilde{z} : x \to y(G).H)\,(x[w].(P \mid Q) \mid \tilde{R})$. Thus, a global type is a central point of control. In standard multiparty session types, this is avoided by annotating actions with the endpoint that they should interact with. In our example, the process becomes $x^y[w].(P \mid Q)$, meaning that this output can synchronise only with an input performed at endpoint y (which, dually, has to express that it intends to synchronise with an output from x). In this section, we define a variant of the calculus of Multiparty Classical Processes (MCP) [9], which follows the standard methodology of multiparty session types, simply by annotating types and processes of GCP with endpoints. Formally, MCP is defined as GCP with the following modifications.

Coherence rules

$$\frac{G \vDash (x_i{:}A_i)_i,\ y{:}C \quad H \vDash \Gamma,\ (x_i{:}B_i)_i,\ y{:}D}{\tilde{x} \to y(G).H \vDash \Gamma,\ (x_i{:}A_i \otimes^{y} B_i)_i,\ y{:}C \parr^{\tilde{x}} D}\ \otimes\parr \qquad \frac{}{\tilde{x} \to y \vDash (x_i{:}1^{y})_i, y{:}\bot^{\tilde{x}}}\ 1\bot$$

$$\frac{G_1 \vDash \Gamma, x{:}A, (y_i{:}C_i)_i \quad G_2 \vDash \Gamma, x{:}B, (y_i{:}D_i)_i}{x \to \tilde{y}.\mathsf{case}(G_1, G_2) \vDash \Gamma, x{:}A \oplus^{\tilde{y}} B, (y_i{:}C_i \& {}^{x} D_i)_i}\ \oplus\& \qquad \frac{G \vDash x{:}A, (y_i{:}B_i)_i}{!x \to \tilde{y}(G) \vDash x{:}?^{\tilde{y}}A, (y_i{:}!^{x}B_i)_i}\ !?$$

$$\frac{}{x \to \tilde{y}.\mathsf{case}() \vDash \Gamma, x{:}0^{\tilde{y}}, (y_i{:}\top^{x})_i}\ 0\top \qquad \frac{G \vDash \Gamma, x{:}A, (y_i{:}B_i)_i \quad X \notin \mathsf{ftv}(\Gamma)}{x \to \tilde{y}.(X)G \vDash \Gamma, x{:}\exists^{\tilde{y}}X.A, (y_i{:}\forall^{x}X.B_i)_i}\ \exists\forall$$

$$\frac{|A|^{\perp} = |B|}{x^{A} \to y^{B} \vdash x : A, y : B}\ \textsc{Axiom}$$

Typing rules

$$\frac{|A|^{\perp} = |B|}{x^{A} \to y^{B} \vdash x : A, y : B}\ \textsc{Axiom} \qquad \frac{(P_i \vdash \Gamma_i, x_i : A_i)_i \quad G \vDash (x_i : A_i)_i}{(\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,(\tilde{P}) \vdash \tilde{\Gamma}}\ \textsc{CCut}$$

$$\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, x : B}{x^{z}[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes^{z} B}\ \otimes \qquad \frac{P \vdash \Gamma, y : A, x : B}{x^{\tilde{z}}(y).P \vdash \Gamma, x : A \parr^{\tilde{z}} B}\ \parr$$

$$\frac{P \vdash \Gamma, x : A}{x^{\tilde{z}}[\mathsf{inl}].P \vdash \Gamma, u : A \oplus^{\tilde{z}} B}\ \oplus_1 \qquad \frac{P \vdash \Gamma, x : B}{x^{\tilde{z}}[\mathsf{inr}].P \vdash \Gamma, x : A \oplus^{\tilde{z}} B}\ \oplus_2 \qquad \frac{P \vdash \Gamma, x : A \quad Q \vdash \Gamma, x : B}{x^{z}.\mathsf{case}(P, Q) \vdash \Gamma, x : A \& {}^{z} B}\ \&$$

$$\frac{P \vdash\ ?\Gamma, y : A}{!x^{z}(y).P \vdash\ ?\Gamma, x : !^{z}A}\ ! \qquad \frac{P \vdash \Gamma, y : A}{?x^{\tilde{z}}[y].P \vdash \Gamma, x : ?^{\tilde{z}}A}\ ? \qquad \frac{P \vdash \Gamma}{P \vdash \Gamma, x : ?^{\tilde{z}}A}\ \textsc{Weaken}$$

$$\frac{P \vdash \Gamma, x : B[A/X]}{x^{\tilde{z}}[A].P \vdash \Gamma, x : \exists^{\tilde{z}}X.B}\ \exists \qquad \frac{P \vdash \Gamma, x : B \quad X \notin \mathsf{ftv}(\Gamma)}{x^{z}(X).P \vdash \Gamma, x : \forall^{z}X.B}\ \forall \qquad \frac{P \vdash \Gamma, y : ?^{\tilde{w}}A, z : ?^{\tilde{w}}A}{P[x/y, x/z] \vdash \Gamma, x : ?^{\tilde{w}}A}\ \textsc{Contract}$$

$$\frac{}{x^{z}[] \vdash x : 1^{z}}\ 1 \qquad \frac{P \vdash \Gamma}{x^{\tilde{z}}().P \vdash \Gamma, x : \bot^{\tilde{z}}}\ \bot \qquad \text{no rule for } 0 \qquad \frac{}{x^{z}.\mathsf{case}() \vdash \Gamma, x : \top^{z}}\ \top$$

Figure 7 MCP, Coherence Rules and Typing Rules.

Types. The coherence relation for MCP, given in Figure 7, is identical to that of GCP, except from the type connectives, which are now annotated with the names of the endpoints with which they are supposed to interact. By an abuse of notation, we now let A, B, C, D range over types with annotations. Rule Axiom is also slightly different then that of GCP. In its premise, we use the operation $|A|$, which removes all annotations from a given proposition A. For instance, $|B_1 \otimes^{x} B_2| = |B_1| \otimes |B_2|$. The syntax of global types is the same as that given for GCP, with the exception of the extra annotation in the term for the axiom.

Endpoint annotations restrict which derivations we can use to prove that some types are coherent. As an example, for some A_i, B_i, and C, consider the following annotated types:

$$x{:}A_1 \otimes^{y} B_1, \quad y{:}A_2 \parr^{x} B_2, \quad z{:}A_3 \parr^{w} B_3, \quad w : C$$

In MCP, it is necessary to eventually apply rule $\otimes\parr$ to $x{:}A_1 \otimes^{y} B_1$ and $y{:}A_2 \parr^{x} B_2$. However, in GCP, this would not be necessary: there may also be a coherence proof in which we apply rule $\otimes\parr$ to $x{:}A_1 \otimes^{y} B_1$ and $z{:}A_3 \parr^{w} B_3$ instead (after removing annotations).

The typing rules for MCP processes are given in Figure 7. As for coherence, the typing rules of MCP are those of GCP, but now with annotations. Importantly, the annotation of each communication action must be the same as that of the corresponding type construct. E.g., the send process is now written as $x^{z}[y].(P \mid Q)$, meaning that endpoint x is sending y to endpoint z, and the corresponding $\otimes$ connective in the type is annotated with the same z. Again, by an abuse of notation, we now let P, Q, R range over processes with annotations.

Semantics. The semantics of MCP is the same as that of GCP, extended with the expected endpoint annotations. The consequence of annotations is that MCP enjoys an ungoverned semantics; it is fully distributed, as in the original theories of multiparty session types [12] and MCP [9]. As an example, here is the η-expansion rule for $\parr$ and $\otimes$:

$$x^{A \parr^{\tilde{w}} B} \to y^{C \otimes^{z} D} \longrightarrow x^{\tilde{w}}(u).y^{z}[v].(u^{A} \to v^{C} \mid x^{B} \to y^{D})$$

Similarly, here is the β-reduction rule for $\otimes$ and $\parr$:

$$\begin{aligned}(\boldsymbol{\nu}\tilde{z}, \tilde{x}, y : \tilde{x} \to y(G).H)\,(\tilde{S} \mid (x_i{}^{y}[x_i'].(P_i \mid Q_i))_i \mid y^{\tilde{x}}(y').R) \longrightarrow \\ (\boldsymbol{\nu}\tilde{x}', y' : G\{\tilde{x}'/\tilde{x}, y'/y\})\,(\tilde{S} \mid \tilde{P} \mid (\boldsymbol{\nu}\tilde{z}, \tilde{x}, y : H)\,(\tilde{Q} \mid R))\end{aligned}$$

In the reduction above (and all other reductions of MCP), the global type of the session is unnecessary for the communicating processes to know which others processes are involved in the communication; that information is instead taken from the endpoint annotations of their respective actions. Type preservation is ensured by the annotations used to type the processes, since that guarantees that coherence continues to hold for the reductum.

▶ **Theorem 12** (Subject reduction for MCP). *If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$.*

▶ **Theorem 13** (Cut elimination for MCP). *If $P \vdash \Gamma$, then there exists Q such that $P \Longrightarrow^{*} Q$ and Q is not a cut.*

Cut elimination for MCP follows from that for GCP (by Theorems 15, 16, 17, and 18).

▶ **Example 14.** We revisit the 2-buyer protocol from Example 6 in MCP. The global type is exactly the same (except for the axioms, which have the extra annotation on the left). However, the types of each endpoint are now appropriately annotated.

$$\begin{aligned}&B_1 : \mathbf{name} \otimes^{S} \mathbf{cost}^{\perp} \parr^{S} \mathbf{cost} \otimes^{B_2} 1^{S},\\ &B_2 : \mathbf{cost}^{\perp} \parr^{S} \mathbf{cost}^{\perp} \parr^{B_1} ((\mathbf{addr} \otimes^{S} 1^{S}) \oplus^{S} 1^{S}),\\ &S : \mathbf{name}^{\perp} \parr^{B_1} (\mathbf{cost} \otimes^{B_1} (\mathbf{cost} \otimes^{B_2} ((\mathbf{addr}^{\perp} \parr^{B_2} \perp^{B_1,B_2}) \,\&^{B_2}\, \perp^{B_1,B_2})))\end{aligned}$$

The annotations on the connectives say that, e.g., B_1 first sends a name to an implementation of endpoint S, and then, she receives a quote from S, before sending her bid to B_2.

Translations. Proofs in MCP translate to proofs in GCP, by erasing annotations. We write $|P|$ for the erasure of annotations from P. We obtain the following type preservation results.

▶ **Theorem 15** (Type preservation from MCP to GCP).
1. *If $P \vdash \Gamma$ in MCP, then $|P| \vdash |\Gamma|$ in GCP.*
2. *If $G \vDash \Gamma$ in MCP, then $|G| \vDash |\Gamma|$ in GCP.*

▶ **Theorem 16** (Type preservation from GCP to MCP).
1. *If $P \vdash \Gamma$ in GCP, then there exist Q, Δ such that $|Q| = P$, $|\Delta| = \Gamma$, and $Q \vdash \Delta$ in MCP.*
2. *If $G \vDash \Gamma$ in GCP, then there exist H, Δ such that $|H| = G$, $|\Delta| = \Gamma$, and $H \vDash \Delta$ in MCP.*

We also obtain a lockstep bisimulation between MCP and GCP.

▶ **Theorem 17** (Simulation of MCP in GCP). *Assume $P \vdash \Gamma$ and $G \vDash \Gamma$ in MCP.*
1. *If $P \equiv Q$ in MCP, then $|P| \equiv |Q|$ in GCP.*
2. *If $P \longrightarrow Q$ in MCP, then $|P| \longrightarrow |Q|$ in GCP.*
3. *If $G \longrightarrow H$ in MCP, then $|G| \longrightarrow |H|$ in GCP.*

▶ **Theorem 18** (Reflection of GCP in MCP). *Assume* $P \vdash \Gamma$ *and* $G \vDash \Gamma$ *in MCP.*

1. *If* $|P| \equiv Q$ *in GCP, then there exists* R *such that* $|R| = Q$ *and* $P \equiv R$ *in MCP.*
2. *If* $|P| \longrightarrow Q'$ *in GCP, then there exists* Q *such that* $|Q| = Q'$ *and* $P \longrightarrow Q$ *in MCP.*
3. *If* $|G| \longrightarrow H'$ *in GCP, then there exists* H *such that* $|H| = H'$ *and* $G \longrightarrow H$ *in MCP.*

Combining these results with those for the translation from GCP to CP, we obtain an end-to-end translation of distributed multiparty sessions into arbitrated binary sessions.

5 Related and Future Work

Arbiters. Caires and Perez [4] show how to translate multiparty sessions by translating a global type to a process, which they call a *medium*. Their medium is similar to our arbiter, and their translation of a global type to a medium is similar to our translation of GCP to CP, which takes a global type to an arbiter. Their system, like ours, guarantees fidelity and deadlock freedom via the translation; and, like us, they extend their system to support polymorphism. However there are several differences. Our work is based on classical linear logic, whereas theirs is based on intuitionistic linear logic; we suspect that their approach could be adopted to classical logic, and ours to intuitionistic. More importantly, where we use coherence, they use the standard definition of projection [12]. Projection is closer to the original formulation of multiparty session types [12], but lacks the tighter connection to logic offered by coherence, in particular the way in which coherence is organised around pairs of logical connectives, generalising duality. Unlike us, they do not consider replication and nesting in global types. Their medium process imposes global governance, similar to our GCP, but they offer no decentralised system based on direct communication like our MCP.

Multiparty session types. Coherence logically reconstructs the notion of well-formedness found in multiparty session types [12, 10], in the context of synchronous communication [2]. Polymorphism for binary session types is considered in a proposition-as-types setting by Wadler [21] and Caires *et al.* [5]; we have generalised this notion to the multiparty case. Our new coherence proof system extends the one presented in the original MCP [9] with rules for polymorphism. Since coherence yields an algorithm for extracting a global type from a set of types [9], ours is also the first work dealing with the extraction of polymorphic global types.

Issues with axiom. In the original formulation of CP [22], axiom reduction applies at all types and all reductions are independent of types. In the current formulation, axiom reduction applies only at atomic types and other instances are handled by η-rules that depend upon types. As a result, the implementation of type instantiation may be problematic. Alternative implementations may seek to restore a version of axiom that applies at all types. We have explored one variant that does so, but the formalism is more complicated as it requires free variables in global types. We leave further exploration to future work.

Coherence. Coherence in GCP may be generalised. In $\otimes\parr$, the context Γ in the conclusion may be distributed to the two premises. Similarly, in !? a context !Γ may be added to the premise and conclusion of the rule, splitting channels between those that retain ! in the premise and those that lose it. While such splits are straightforward in GCP, it is unclear how to add them without centralised control in MCP. We also leave this to future work.

Choreography. Carbone *et al.* [8] search for a propositions-as-types correspondence for the calculus of compositional choreographies of Montesi and Yoshida [15]. This work inspired the notion of coherence in the original MCP [9] — typing choreographies requires a similar handling of multiple connectives. However, it was limited to binary sessions, whereas the original theory by Montesi and Yoshida supported multiparty sessions. This work may close the circle: our generalisation of CP to GCP seems applicable to choreographies, and would yield an expressive choreography language with parametric polymorphism.

Relationship with standard multiparty session types. MCP is directly connected to classical linear logic, while also bearing a close resemblance to traditional multiparty session types. However, there remain important differences between the two. First, we do not handle recursive behaviour [14]. Second, multiparty session types support broadcast from one sender to many receivers, while our types gather information from many senders to one receiver – a choice dictated by our desire to translate MCP to CP. We look forward to further studying the relation between MCP, GCP, CP, and other systems with multiparty session types.

References

1 Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994.
2 Andi Bejleri and Nobuko Yoshida. Synchronous multiparty session types. *ENTCS*, 241:3–33, 2009.
3 Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *TCS*, 135(1):11–65, 1994.
4 Luís Caires and Jorge Perez. Multiparty session types within a canonical binary theory, and beyond. In *FORTE*, 2016.
5 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *ESOP*, pages 330–349, 2013.
6 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.
7 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *MSCS*, 26(3):367–423, 2016.
8 Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. In *CONCUR*, pages 47–62, 2014.
9 Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. In *CONCUR*, pages 412–426, 2015.
10 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760:1–65, 2015.
11 Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, pages 22–138, 1998.
12 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *JACM*, 63(1):9, 2016. Also: POPL, 2008, pages 273–284.
13 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *TCS*, 410(46):4747–4768, 2009.
14 Sam Lindley and Garrett Morris. Talking bananas: structural recursion for session types. In *ICFP*. ACM, 2016. To appear.
15 Fabrizio Montesi and Nobuko Yoshida. Compositional choreographies. In *CONCUR*, pages 425–439, 2013.
16 Jennifer Paykin and Steve Zdancewic. Linear $\lambda\mu$ is CP (more or less). In *A List of Successes That Can Change The World*, pages 273–291, 2016.

17 Frank Pfenning and Dennis Griffith. Polarized substructural session types. In *Foundations of Software Science and Computation Structures*, pages 3–22. Springer, 2015.

18 Davide Sangiorgi. Pi-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(1&2):235–274, 1996.

19 A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory (2nd Ed.)*. Cambridge University Press, 2000.

20 Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

21 Philip Wadler. Propositions as sessions. In *ICFP*, pages 273–286, 2012.

22 Philip Wadler. Propositions as sessions. *JFP*, 24(2–3):384–418, 2014. Also: ICFP, pages 273–286, 2012.

Global Caching for the Alternation-free μ-Calculus

Daniel Hausmann[1], Lutz Schröder[2], and Christoph Egger[3]

1 **Friedrich-Alexander Universität Erlangen-Nürnberg, Germany**
2 **Friedrich-Alexander Universität Erlangen-Nürnberg, Germany**
3 **Friedrich-Alexander Universität Erlangen-Nürnberg, Germany**

Abstract

We present a sound, complete, and optimal single-pass tableau algorithm for the alternation-free μ-calculus. The algorithm supports global caching with intermediate propagation and runs in time $2^{\mathcal{O}(n)}$. In game-theoretic terms, our algorithm integrates the steps for constructing and solving the Büchi game arising from the input tableau into a single procedure; this is done on-the-fly, i.e. may terminate before the game has been fully constructed. This suggests a slogan to the effect that *global caching = game solving on-the-fly*. A prototypical implementation shows promising initial results.

1998 ACM Subject Classification F.4.1 Mathematical Logic - Temporal Logic

Keywords and phrases modal logic, fixpoint logic, satisfiability, global caching, coalgebraic logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.34

1 Introduction

The modal μ-calculus [25, 3] serves as an expressive temporal logic for the specification of sequential and concurrent systems containing many standard formalisms such as linear time temporal logic LTL [28, 33], CTL [7], and PDL [34]. Satisfiability checking in the modal μ-calculus is ExpTime-complete [31, 10]. There appears to be, to date, no readily implementable reasoning algorithm for the μ-calculus, and in fact (prior to [23]) even for its fragment CTL, that is simultaneously *optimal*, i.e. runs in ExpTime, and *single-pass*, i.e. avoids building an exponential-sized data structure in a first pass. Typical data structures used in worst-case-optimal algorithms are automata [10], games [13], and, for sublogics such as CTL, first-pass tableaux [9].

The term *global caching* describes a family of single-pass tableau algorithms [18, 21] that build graph-shaped tableaux bottom-up in so-called *expansion* steps, with no label ever generated twice, and attempt to terminate before the tableau is completely expanded by means of judicious intermediate *propagation* of satisfiability and/or unsatisfiability through partially expanded tableaux. Global caching offers wide room for heuristic optimization, regarding standard tableau optimizations as well as the order in which expansion and propagation steps are triggered, and has been shown to perform competitively in practice; see [21] for an evaluation of heuristics in global caching for the description logic $\mathcal{ALCI}$. One major challenge with global caching algorithms is typically to prove soundness and completeness, which becomes harder in the presence of fixpoint operators. A global caching algorithm for PDL has been described by Goré and Widmann [20]; finding an optimal global caching algorithm even for CTL has been named as an open problem as late as 2014 [15] (a non-optimal, doubly exponential algorithm is known [15]).

The contribution of the present work is an optimal global-caching algorithm for satisfiability in the alternation-free μ-calculus, extending our earlier work on the single-variable (*flat*) fragment of the μ-calculus [23]. The algorithm actually works at the level of generality of the

alternation-free fragment of the coalgebraic μ-calculus [6], and thus covers also logics beyond the realm of standard Kripke semantics such as alternating-time temporal logic ATL [1], neighbourhood-based logics such as the monotone μ-calculus that underlies Parikh's game logic [32], or probabilistic fixpoint logic. To aid readability, we phrase our results in terms of the relational μ-calculus, and discuss the coalgebraic generalization only at the end of Section 4. The model construction in the completeness proof yields models of size $2^{\mathcal{O}(n)}$.

We have implemented of our algorithm as an extension of the Coalgebraic Ontology Logic Reasoner COOL, a generic reasoner for coalgebraic modal logics [22]; given the current state of the implementation of instance logics in COOL, this means that we effectively support alternation-free fragments of relational, monotone, and alternating-time [1] μ-calculi, thus in particular covering CTL and ATL. We have evaluated the tool in comparison with existing reasoners on benchmark formulas for CTL [19] (which appears to be the only candidate logic for which well-developed benchmarks are currently available) and on random formulas for ATL and the alternation-free relational μ-calculus, with promising results; details are discussed in Section 5.

Related Work. The theoretical upper bound ExpTime has been established for the full coalgebraic μ-calculus [6] (and earlier for instances such as the alternating-time μ-calculus AMC [36]), using a multi-pass algorithm that combines games and automata in a similar way as for the standard relational case, in particular involving the Safra construction. *Global caching* has been employed successfully for a variety of description logics [18, 21], and lifted to the level of generality of coalgebraic logics with global assumptions [16] and nominals [17].

A tableaux-based non-optimal (NExpTime) decision procedure for the full μ-calculus has been proposed in [24]. Friedmann and Lange [13] describe an optimal tableau method for the full μ-calculus that, unlike most other methods including the one we present here, makes do without requiring guardedness. Like earlier algorithms for the full μ-calculus, the algorithm constructs and solves a parity game, and in principle allows for an on-the-fly implementation. The models constructed in the completeness proof are asymptotically larger than ours, but presumably the proof can be adapted for the alternation-free case by using determinization of co-Büchi automata [29] instead of Safra's determinization of Büchi automata [35] to yield models of size $2^{\mathcal{O}(n)}$, like ours. For non-relational instances of the coalgebraic μ-calculus, including the alternation-free fragment of the alternating-time μ-calculus AMC, the $2^{\mathcal{O}(n)}$ bound on model size appears to be new, with the best known bound for the alternation-free AMC being $2^{\mathcal{O}(n \log n)}$ [36].

In comparison to our own recent work [23], we move from the flat to the alternation-free fragment, which means essentially that fixpoints may now be defined by mutual recursion, and thus can express properties such as 'all paths reach states satisfying p and q, respectively, in strict alternation until they eventually reach a state satisfying r'. Technically, the main additional challenge is the more involved structure of eventualities and deferrals, which now need to be represented using cascaded sequences of unfoldings in the focusing approach; this affects mainly the soundness proof, which now needs to organize termination counters in a tree structure. While the alternation-free algorithm instantiates to the algorithm from [23] for flat input formulas, its completeness proof includes a new model construction which yields a bound of $3^n \in 2^{\mathcal{O}(n)}$ on model size, slightly improving upon the bound $n \cdot 4^n$ from [23]. We present the new algorithm in terms that are amenable to a game-theoretic perspective, emphasizing the correspondence between global caching and game-solving. In fact, it turns out that global caching algorithms effectively consist in an integration of the separate steps of typical game-based methods for the μ-calculus [13, 14, 31] into a single

on-the-fly procedure that talks only about partially expanded tableau graphs, implicitly combining on-the-fly determinization of co-Büchi automata with on-the-fly solving of the resulting Büchi games [11]. This motivates the mentioned slogan that

global caching is on-the-fly determinization and game solving.

In particular, the propagation steps in the global caching pattern can be seen as solving an incomplete Büchi game that is built directly by the expansion steps, avoiding explicit determinization of co-Büchi automata analogously to [29]. One benefit of an explicit global caching algorithm integrating the pipeline from tableaux to game solving is the implementation freedom afforded by the global caching pattern, in which suitable heuristics can be used to trigger expansion and propagation steps in any order that looks promising.

2 Preliminaries: The μ-Calculus

We briefly recall the definition of the (relational) μ-calculus. We fix a set P of *propositions*, a set A of *actions*, and a set $\mathfrak{V}$ of fixpoint variables. Formulas ϕ, ψ of the μ-calculus are then defined by the grammar

$$\psi, \phi ::= \bot \mid \top \mid p \mid \neg p \mid X \mid \psi \wedge \phi \mid \psi \vee \phi \mid \langle a \rangle \psi \mid [a]\psi \mid \mu X.\, \psi \mid \nu X.\, \psi$$

where $p \in P$, $a \in A$, and $X \in \mathfrak{V}$; we write $|\psi|$ for the size of a formula ψ. Throughout the paper, we use η to denote one of the fixpoint operators μ or ν. We refer to formulas of the form $\eta X.\, \psi$ as *fixpoint literals*, to formulas of the form $\langle a \rangle \psi$ or $[a]\psi$ as *modal literals*, and to p, $\neg p$ as *propositional literals*. The operators μ and ν *bind* their variables, inducing a standard notion of *free variables* in formulas. We denote the set of free variables of a formula ψ by $FV(\psi)$. A formula ψ is *closed* if $FV(\psi) = \emptyset$, and *open* otherwise. We write $\psi \leq \phi$ ($\psi < \phi$) to indicate that ψ is a (proper) subformula of ϕ. We say that ϕ *occurs free* in ψ if ϕ occurs as a subformula in ψ that is not in the scope of any fixpoint. Throughout, we *restrict to formulas that are guarded*, i.e. have at least one modal operator between any occurrence of a variable X and an enclosing binder ηX. (This is standard although possibly not without loss of generality [13].) Moreover we assume w.l.o.g. that input formulas are *clean*, i.e. all fixpoint variables are distinct, and *irredundant*, i.e. $X \in FV(\psi)$ for all subformulas $\eta X.\, \psi$.

Formulas are evaluated over *Kripke structures* $\mathcal{K} = (W, (R_a)_{a \in A}, \pi)$, consisting of a set W of *states*, a family $(R_a)_{a \in A}$ of relations $R_a \subseteq W \times W$, and a valuation $\pi : P \to \mathcal{P}(W)$ of the propositions. Given an *interpretation* $i : \mathfrak{V} \to \mathcal{P}(W)$ of the fixpoint variables, define $[\![\psi]\!]_i \subseteq W$ by the obvious clauses for Boolean operators and propositions, $[\![X]\!]_i = i(X)$, $[\![\langle a \rangle \psi]\!]_i = \{v \in W \mid \exists w \in R_a(v).w \in [\![\psi]\!]_i\}$, $[\![[a]\psi]\!]_i = \{v \in W \mid \forall w \in R_a(v).w \in [\![\psi]\!]_i\}$, $[\![\mu X.\, \psi]\!]_i = \mu [\![\psi]\!]_i^X$ and $[\![\nu X.\, \psi]\!]_i = \nu [\![\psi]\!]_i^X$, where $R_a(v) = \{w \in W \mid (v, w) \in R_a\}$, $[\![\psi]\!]_i^X(G) = [\![\psi]\!]_{i[X \mapsto G]}$, and μ, ν take least and greatest fixpoints of monotone functions, respectively. If ψ is closed, then $[\![\psi]\!]_i$ does not depend on i, so we just write $[\![\psi]\!]$. We write $x \models \psi$ for $x \in [\![\psi]\!]$. The *alternation-free fragment* of the μ-calculus is obtained by prohibiting formulas in which some subformula contains both a free ν-variable and a free μ-variable. E.g. $\mu X.\, \mu Y.\, (\Box X \wedge \Diamond Y \wedge \nu Z.\, \Diamond Z)$ is alternation-free but $\nu Z.\, \mu X.\, (\Box X \wedge \nu Y.\, (\Diamond Y \wedge \Diamond Z))$ is not. CTL is contained in the alternation-free fragment.

We have the standard *tableau rules* (each consisting of one *premise* and a possibly empty set of *conclusions*) which will be interpreted AND-OR style, i.e. to show satisfiability of a set of formulas Δ, it will be necessary to show that *every* rule application that matches Δ has

some conclusion that is satisfiable. Our algorithm will use these rules in the expansion step.

$$(\bot)\ \frac{\Gamma, \bot}{} \qquad (\lightning)\ \frac{\Gamma, p, \neg p}{}$$

$$(\wedge)\ \frac{\Gamma, \psi \wedge \phi}{\Gamma, \psi, \phi} \qquad (\vee)\ \frac{\Gamma, \psi \vee \phi}{\Gamma, \psi \quad \Gamma, \phi}$$

$$(\langle a \rangle)\ \frac{\Gamma, [a]\psi_1, \ldots, [a]\psi_n, \langle a \rangle \phi}{\psi_1, \ldots, \psi_n, \phi} \qquad (\eta)\ \frac{\Gamma, \eta X.\, \psi}{\Gamma, \psi[X \mapsto \eta X.\, \psi]}$$

(for $a \in A$, $n \in \mathbb{N}$, $p \in P$); we refer to the set of modal rules $(\langle a \rangle)$ by $\mathcal{R}_m$ and to the set of the remaining rules by $\mathcal{R}_p$ and usually write rules with premise Γ and conclusion $\Sigma = \Gamma_1, \ldots, \Gamma_n$ in sequential form, i.e. as (Γ / Σ).

▶ **Example 1.** As our running example, we pick a non-flat formula, i.e. one that requires two recursion variables. Consider the alternation-free formulas

$$\psi_1 = \mu X.\, ((p \wedge (r \vee \Box \psi_2)) \vee (\neg q \wedge \Box X)) \qquad \psi_2 = \mu Y.\, ((q \wedge (r \vee \Box X)) \vee (\neg p \wedge \Box Y))$$

(where $A = \{*\}$ and we write $\Box = [*]$, $\Diamond = \langle * \rangle$). The formulas ψ_1 and $\psi_2[X \mapsto \psi_1]$ state that all paths will visit p and q in strict alternation until r is eventually reached, starting with p and with q, respectively. Using the measure of *entanglement* [2], one can show that these properties cannot be expressed using only one variable.

3 The Global Caching Algorithm

We proceed to describe our global caching algorithm for the alternation-free μ-calculus. First off, we need some syntactic notions regarding decomposition of fixpoint literals.

▶ **Definition 2** (Deferrals)**.** Given fixpoint literals $\chi_i = \eta X_i.\, \psi_i$, $i = 1, \ldots, n$, we say that a substitution $\sigma = [X_1 \mapsto \chi_1]; \ldots; [X_n \mapsto \chi_n]$ *sequentially unfolds* χ_n if $\chi_i <_f \chi_{i+1}$ for all $1 \leq i < n$, where we write $\psi <_f \eta X.\, \phi$ if $\psi \leq \phi$ and ψ is open and occurs free in ϕ (i.e. σ unfolds a nested sequence of fixpoints in χ_n innermost-first). We say that a formula χ is *irreducible* if for every substitution $[X_1 \mapsto \chi_1]; \ldots; [X_n \mapsto \chi_n]$ that sequentially unfolds χ_n, we have that $\chi = \chi_1([X_2 \mapsto \chi_2]; \ldots; [X_n \mapsto \chi_n])$ implies $n = 1$ (i.e. $\chi = \chi_1$). An *eventuality* is an irreducible closed least fixpoint literal. A formula ψ *belongs* to an eventuality θ_n, or is a θ_n*-deferral*, if $\psi = \alpha\sigma$ for some substitution $\sigma = [X_1 \mapsto \theta_1]; \ldots; [X_n \mapsto \theta_n]$ that sequentially unfolds θ_n and some $\alpha <_f \theta_1$. We denote the set of θ_n-deferrals by $dfr(\theta_n)$.

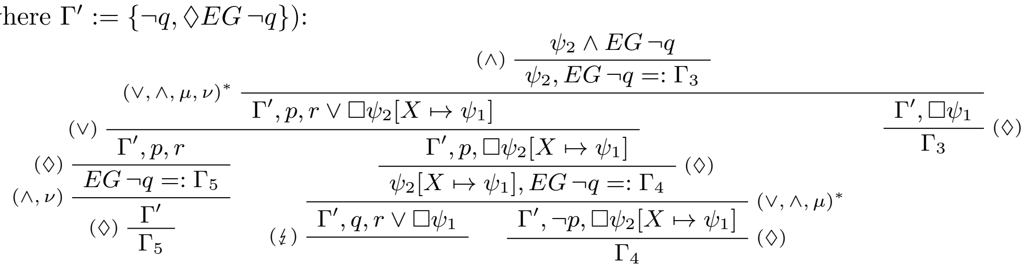
E.g. the substitution $\sigma = [Y \mapsto \mu Y.\, (\Box X \wedge \Diamond\Diamond Y)]; [X \mapsto \theta]$ sequentially unfolds the eventuality $\theta = \mu X.\, \mu Y.\, (\Box X \wedge \Diamond\Diamond Y)$, and $(\Diamond Y)\sigma = \Diamond \mu Y.\, (\Box\theta \wedge \Diamond\Diamond Y)$ is a θ-deferral. A fixpoint literal is irreducible if it is not an unfolding $\psi[X \mapsto \eta X.\, \psi]$ of a fixpoint literal $\eta X.\, \psi$; in particular, every clean irredundant fixpoint literal is irreducible.

▶ **Lemma 3.** *Each formula ψ belongs to at most one eventuality θ, and then $\theta \leq \psi$.*

▶ **Example 4.** Applying the tableau rules $\mathcal{R}_m$ and $\mathcal{R}_p$ to the formula $\psi_1 \wedge EG\, \neg r$, where ψ_1 is defined as in Example 1 and $EG\, \phi$ abbreviates $\nu X.\, (\phi \wedge \Diamond X)$, results in a cyclic graph, with relevant parts depicted as follows:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\psi_1 \wedge EG\, \neg r}{\psi_1, EG\, \neg r =: \Gamma_1}(\wedge)}{\Gamma, p, \Box\psi_2[X \mapsto \psi_1]}(\vee,\wedge,\nu,\mu)^*}{\psi_2[X \mapsto \psi_1], EG\, \neg r =: \Gamma_2}(\Diamond) \qquad \cfrac{\Gamma, \neg q, \Box\psi_1}{\Gamma_1}(\Diamond)}{\Gamma, q, \Box\psi_1 \quad \Gamma, \neg p, \Box\psi_2[X \mapsto \psi_1]}(\vee,\wedge,\nu,\mu)^* \qquad \cfrac{\Gamma, \neg q, \Box\psi_1}{\Gamma_1}(\Diamond)}{\Gamma_1 \qquad \Gamma_2}(\Diamond)$$

where $\Gamma = \{\neg r, \Diamond EG\,\neg r\}$. The graph contains three cycles, all of which contain but never *finish* a formula that belongs to ψ_1 (where a formula belonging to an eventuality ψ_1 is said to be *finished* if it evolves to a formula that does not belong to ψ_1): In the rightmost cycle, the deferral $\delta_1 := \psi_1$ evolves to the deferral $\delta_2 := \Box\psi_1$ which then evolves back to δ_1. For the cycle in the middle, δ_1 evolves to $\delta_3 := \Box\psi_2[X \mapsto \psi_1]$ which in turn evolves to $\delta_4 := \psi_2[X \mapsto \psi_1]$ before looping back to δ_3. In the leftmost cycle, δ_1 evolves via δ_3 and δ_4 to δ_2 before cycling back to δ_1. The satisfaction of ψ_1 is thus being postponed indefinitely, since $EG\,\neg r$ enforces the existence of a path on which r never holds. As a successful example, consider the graph that is obtained when attempting to show the satisfiability of $\psi_1 \wedge EG\,\neg q$, (where $\Gamma' := \{\neg q, \Diamond EG\,\neg q\}$):



The two loops through Γ_3 and Γ_4 are unsuccessful as they indefinitely postpone the satisfaction of the deferrals δ_2 and δ_3, respectively; also there is the unsuccessful clashing node $\Gamma', q, r \vee \Box\psi_1$, containing both q and $\neg q$. However, the loop through Γ_5 is successful since it contains no deferral that is never finished; as all branching in this example is disjunctive, the single successful loop suffices to show that the initial node is successful. Our algorithm implements this check for 'good' and 'bad' loops by *simultaneously* tracking all deferrals that occur through the proof graph, checking whether each deferral is eventually finished.

We fix an input formula ψ_0 and denote the Fischer-Ladner closure [26] of ψ_0 by $\mathbf{F}$; notice that $|\mathbf{F}| \leq |\psi_0|$. Let $\mathbf{N} = \mathcal{P}(\mathbf{F})$ be the set of all *nodes* and $\mathbf{S} \subseteq \mathbf{N}$ the set of all *state nodes*, i.e. nodes that contain only $\top$, non-clashing propositional literals (where p *clashes* with $\neg p$) and modal literals; so $|\mathbf{S}| \leq |\mathbf{N}| \leq 2^{|\psi_0|}$. Put

$$\mathbf{C} = \{(\Gamma, d) \in \mathbf{N} \times \mathcal{P}(\mathbf{F}) \mid d \subseteq \Gamma\}, \quad \text{and} \quad \mathbf{C}_G = \{(\Gamma, d) \in \mathbf{C} \mid \Gamma \in G\} \text{ for } G \subseteq \mathbf{N},$$

recalling that nodes are just sets of formulas; note $|\mathbf{C}| \leq 3^{|\psi_0|}$. Elements $v = (\Gamma, d) \in \mathbf{C}$ are called *focused nodes*, with *label* $l(v) = \Gamma$ and *focus* d. The idea of focusing single eventualities comes from work on LTL and CTL [27, 4]. In the alternation-free μ-calculus, eventualities may give rise to multiple deferrals so that one needs to focus *sets of deferrals* instead of single eventualities. Our algorithm incrementally builds a set of nodes but performs fixpoint computations on $\mathcal{P}(\mathbf{C})$, essentially computing winning regions of the corresponding Büchi game (with the target set of player 0 being the nodes with empty focus) on-the-fly.

▶ **Definition 5** (Conclusions). For a node $\Gamma \in \mathbf{N}$ and a set $\mathcal{S}$ of tableau rules, the set of *conclusions* of Γ under $\mathcal{S}$ is

$$Cn(\mathcal{S}, \Gamma) = \{\{\Gamma_1, \dots, \Gamma_n\} \in \mathcal{P}(\mathbf{N}) \mid (\Gamma/\Gamma_1 \dots \Gamma_n) \in \mathcal{S}\}.$$

We define $Cn(\Gamma)$ as $Cn(\mathcal{R}_m, \Gamma)$ if Γ is a state node and as $Cn(\mathcal{R}_p, \Gamma)$ otherwise. A set $N \subseteq \mathbf{N}$ of nodes is *fully expanded* if for each $\Gamma \in N$, $\bigcup Cn(\Gamma) \subseteq N$.

▶ **Definition 6** (Deferral tracking). Given a node $\Gamma = \psi_1, \dots, \psi_n, \phi$ and a state node $\Delta \in \mathbf{S}$ that contains $[a]\psi_1, \dots, [a]\psi_n, \langle a\rangle\phi$ as a subset, we say that Γ *inherits* ϕ *from* $(\langle a\rangle\phi, \Delta)$ and ψ_i from $([a]\psi_i, \Delta)$. For a non-state node $\Delta \in \mathbf{N}$, a node $\Gamma \in \mathbf{N}$ with $\phi \in \Gamma$, and $\psi \in \Delta$, Γ *inherits* ϕ *from* (ψ, Δ) if $\Gamma = \Gamma_i$ is conclusion of a non-modal rule $(\Gamma_0/\Gamma_1 \dots \Gamma_n)$ with

$\Gamma_0 = \Delta$ and either ψ has one of the forms ϕ, $\phi \vee \chi$, $\chi \vee \phi$, $\phi \wedge \chi$, $\chi \wedge \phi$, or $\psi = \eta X.\, \chi$ and $\phi = \chi[X \mapsto \psi]$. We put

$$\begin{aligned} \mathit{Inh}_m(\phi, \langle a\rangle\phi, \Delta) &= \{\Gamma \in \mathbf{N} \mid \Gamma \text{ inherits } \phi \text{ from } (\langle a\rangle\phi, \Delta)\} \\ \mathit{Inh}_m(\phi, [a]\phi, \Delta) &= \{\Gamma \in \mathbf{N} \mid \Gamma \text{ inherits } \phi \text{ from } ([a]\phi, \Delta)\} \\ \mathit{Inh}_p(\phi, \psi, \Delta) &= \{\Gamma \in \mathbf{N} \mid \Gamma \text{ inherits } \phi \text{ from } (\psi, \Delta)\}, \end{aligned}$$

where Δ is a state node in the first two clauses and a non-state node in the third clause. We write *evs* for the set of eventualities in $\mathbf{F}$. For a node $\Gamma \in \mathbf{N}$, the set of deferrals of Γ is

$$d(\Gamma) = \{\delta \in \Gamma \mid \exists \theta \in \mathit{evs}.\, \delta \in \mathit{dfr}(\theta)\}.$$

For a set $d \neq \emptyset$ of deferrals and nodes $\Gamma, \Delta \in \mathbf{N}$, we put

$$\begin{aligned} d_{\Delta \rightsquigarrow \Gamma} = \{\delta \in d(\Gamma) \mid\ & \exists \theta \in \mathit{evs}.\, \exists \langle a\rangle\delta \in d.\, \Gamma \in \mathit{Inh}_m(\delta, \langle a\rangle\delta, \Delta) \text{ and } \delta, \langle a\rangle\delta \in \mathit{dfr}(\theta) \text{ or} \\ & \exists [a]\delta \in d.\, \Gamma \in \mathit{Inh}_m(\delta, [a]\delta, \Delta) \text{ and } \delta, \langle a\rangle\delta \in \mathit{dfr}(\theta)\} \end{aligned}$$

if Δ is a state node, and

$$d_{\Delta \rightsquigarrow \Gamma} = \{\delta_1 \in d(\Gamma) \mid \exists \theta \in \mathit{evs}. \exists \delta_2 \in d.\, \Gamma \in \mathit{Inh}_p(\delta_1, \delta_2, \Delta) \text{ and } \delta_1, \delta_2 \in \mathit{dfr}(\theta)\}$$

if Δ is a non-state node. I.e. $d_{\Delta \rightsquigarrow \Gamma}$ is the set of deferrals that is obtained by *tracking* d from Δ to Γ, where Γ is the conclusion of a rule application to Δ. We put $\emptyset_{\Delta \rightsquigarrow \Gamma} = d(\Gamma)$, with the intuition that if the focus d is empty at (Δ, d), then we *refocus*, i.e. choose as new focus for the conclusion Γ the set $d(\Gamma)$ of *all* deferrals in Γ.

▶ **Example 7.** Revisiting the proof graphs from Example 4, we fix additional abbreviations $\Gamma_6 := \Gamma, \neg p, \Box\psi_2[X \mapsto \psi_1]$, $\Gamma_7 := \Gamma', p, r \vee \Box\psi_2[X \mapsto \psi_1]$ and $\Gamma_8 := \Gamma', p, r$. In the first graph, e.g. $d(\Gamma_6) = \{\delta_3\}$ and $d(\Gamma_2) = \{\delta_4\}$; in the second graph, e.g. $d(\Gamma_7) = \{r \vee \Box\psi_2[X \mapsto \psi_1]\}$ and $d(\Gamma_8) = \emptyset$. In the first graph, the node Γ_6 inherits the deferral δ_3 from δ_4 at Γ_2, i.e. $d(\Gamma_2)_{\Gamma_2 \rightsquigarrow \Gamma_6} = \{\delta_4\}_{\Gamma_2 \rightsquigarrow \Gamma_6} = \{\delta_3\}$ since $\Gamma_6 \in \mathit{Inh}_m(\psi_2[X \mapsto \psi_1], \Box\psi_2[X \mapsto \psi_1], \Gamma_2)$. Regarding the second graph, Γ_8 does not inherit any deferral from Γ_7, i.e. $d(\Gamma_7)_{\Gamma_8 \rightsquigarrow \Gamma_7} = \{r \vee \Box\psi_2[X \mapsto \psi_1]\}_{\Gamma_8 \rightsquigarrow \Gamma_7} = \emptyset$ since $\Gamma_8 \in \mathit{Inh}_p(r, r \vee \Box\psi_2[X \mapsto \psi_1], \Gamma_7)$ but $r \vee \Box\psi_2[X \mapsto \psi_1] \in \mathit{dfr}(\psi_1)$ while $r \notin \mathit{dfr}(\psi_1)$, i.e. $r \vee \Box\psi_2[X \mapsto \psi_1]$ belongs to ψ_1 but r does not. This corresponds to the intuition that Γ_8 represents a branch originating from Γ_7 that actually finishes the deferral $r \vee \Box\psi_2[X \mapsto \psi_1]$.

We next introduce the functionals underlying the fixpoint computations for propagation of satisfiability and unsatisfiability.

▶ **Definition 8.** Let $C \subseteq \mathbf{C}$ be a set of focused nodes. We define the functions $f : \mathcal{P}(C) \to \mathcal{P}(C)$ and $g : \mathcal{P}(C) \to \mathcal{P}(C)$ by

$$\begin{aligned} f(Y) &= \{(\Delta, d) \in C \mid \forall \Sigma \in \mathit{Cn}(\Delta).\, \exists \Gamma \in \Sigma.\, (\Gamma, d_{\Delta \rightsquigarrow \Gamma}) \in Y\} \\ g(Y) &= \{(\Delta, d) \in C \mid \exists \Sigma \in \mathit{Cn}(\Delta).\, \forall \Gamma \in \Sigma.\, (\Gamma, d_{\Delta \rightsquigarrow \Gamma}) \in Y\} \end{aligned}$$

for $Y \subseteq C$. We refer to C as the *base set* of f and g.

That is, a focused node (Δ, d) is in $f(Y)$ if each rule matching Δ has a conclusion Γ such that $(\Gamma, d') \in Y$, where the focus d' is the set of deferrals obtained by tracking d from Δ to Γ.

▶ **Definition 9** (Proof transitionals). For $X \subseteq C \subseteq \mathbf{C}$, we define the *proof transitionals* $\hat{f}_X : \mathcal{P}(C) \to \mathcal{P}(C)$, $\hat{g}_X : \mathcal{P}(C) \to \mathcal{P}(C)$ by

$$\hat{f}_X(Y) := (f(Y) \cap \overline{F}) \cup (f(X) \cap F) = f(Y) \cup (f(X) \cap F)$$
$$\hat{g}_X(Y) := (g(Y) \cup F) \cap (g(X) \cup \overline{F}) = g(X) \cup (g(Y) \cap \overline{F}),$$

for $Y \subseteq C$, where $F = \{(\Gamma, d) \in C \mid d = \emptyset\}$ and $\overline{F} = \{(\Gamma, d) \in C \mid d \neq \emptyset\}$ are the sets of focused nodes with empty and non-empty focus, respectively, and where C is the base set of f and g.

That is, $\hat{f}_X(Y)$ contains nodes with non-empty focus that have for each matching rule a successor node in Y as well as nodes with empty focus that have for each matching rule a successor node in X. The least fixpoint of $\hat{f}_X$ thus consists of those nodes that finish their focus – by eventually reaching nodes from F with empty focus – and loop to X afterwards.

▶ **Lemma 10.** *The proof transitionals are monotone w.r.t. set inclusion, i.e. if $X' \subseteq X$, $Y' \subseteq Y$, then $\hat{f}_{X'}(Y') \subseteq \hat{f}_X(Y)$ and $\hat{g}_{X'}(Y') \subseteq \hat{g}_X(Y)$.*

▶ **Definition 11** (Propagation). For $G \subseteq \mathbf{N}$, we define $E_G, A_G \subseteq \mathbf{C}_G$ as

$$E_G = \nu X.\mu Y.\ \hat{f}_X(Y) \quad \text{and} \quad A_G = \mu X.\nu Y.\ \hat{g}_X(Y),$$

where $\mathbf{C}_G$ is the base set of f and g.

Notice that in terms of games, the computation of E_G and A_G corresponds to solving an incomplete Büchi game. The set E_G contains nodes (Γ, d) for which player 0 has a strategy to enforce – for each infinite play starting at (Γ, d) – the Büchi condition that nodes in F, i.e. with empty focus, are visited infinitely often; similarly A_G is the winning region of player 1 in the corresponding game, i.e. contains the nodes for which player 1 has a strategy to enforce an infinite play that passes F only finitely often or a finite play that gets stuck in a winning position for player 1.

▶ **Example 12.** Returning to Example 4, we have $(\Gamma_1, d(\Gamma_1)) = (\Gamma_1, \{\psi_1\}) \in A_{G_1}$ and $(\Gamma_3, d(\Gamma_3)) = (\Gamma_3, \{\psi_1\}) \in E_{G_2}$ where G_1 and G_2 denote the set of all nodes of the first and the second proof graph, respectively; the global caching algorithm described later will therefore answer 'unsatisfiable' to Γ_1, and 'satisfiable' to Γ_3. To see $(\Gamma_1, \{\psi_1\}) \in A_{G_1}$ note that $A_{G_1} = \nu Y.\, \hat{g}_{A_{G_1}}(Y)$ by definition, so $A_{G_1} = (\hat{g}_{A_{G_1}})^n(\mathbf{C}_{G_1})$ for some n. For each focused node $(\Delta, d) \in \mathbf{C}_{G_1}$ there is a rule matching Δ all whose conclusions Γ satisfy $(\Gamma, d_{\Delta \rightsquigarrow \Gamma}) \in \mathbf{C}_{G_1}$, i.e. $g(\mathbf{C}_{G_1}) = \mathbf{C}_{G_1}$. Moreover, since all loops in G_1 indefinitely postpone some eventuality, no node with non-empty focus ever reaches one with empty focus, so $\hat{g}_\emptyset(\mathbf{C}_{G_1}) = \overline{F}$. Since $\hat{g}$ is monotone and $(\Gamma_1, \{\psi_1\}) \in \overline{F}$, we obtain by induction over n that $(\Gamma_1, \{\psi_1\}) \in (\hat{g}_{A_{G_1}})^n(\mathbf{C}_{G_1})$. To see $(\Gamma_3, d(\Gamma_3)) = (\Gamma_3, \{\psi_1\}) \in E_{G_2}$, note that that starting from Γ_3, the single deferral ψ_1 can be finished in finite time while staying in E_{G_2}. This holds because we can reach $(\Gamma_8, \emptyset)$ by branching to the left twice and $(\Gamma_8, \emptyset) \in E_{G_2}$, since the loop through Γ_5 does not contain any deferrals whose satisfaction is postponed indefinitely and hence is contained in E_{G_2}.

▶ **Lemma 13.** *If $G' \subseteq G$, then $E_{G'} \subseteq E_G$ and $A_{G'} \subseteq A_G$.*

▶ **Lemma 14.** *Let $G \subseteq \mathbf{N}$ be fully expanded. Then $E_G = \overline{A_G}$.*

Our algorithm constructs a partial tableau, maintaining sets $G, U \subseteq \mathbf{N}$ of *expanded* and *unexpanded* nodes, respectively. It computes $E_G, A_G \subseteq \mathbf{C}_G$ in the propagation steps; as these sets grow monotonically, they can be computed incrementally.

Algorithm (Global caching). Decide satisfiability of a closed formula ϕ_0.

1. (Initialization) Let $G := \emptyset$, $\Gamma_0 := \{\phi_0\}$, $U := \{\Gamma_0\}$.
2. (Expansion) Pick $t \in U$ and let $G := G \cup \{t\}$, $U := (U - \{t\}) \cup (\bigcup Cn(t) - G)$.
3. (Intermediate propagation) Optional: Compute E_G and/or A_G. If $(\Gamma_0, d(\Gamma_0)) \in E_G$, return 'Yes'. If $(\Gamma_0, d(\Gamma_0)) \in A_G$, return 'No'.
4. If $U \neq \emptyset$, continue with Step 2.
5. (Final propagation) Compute E_G. If $(\Gamma_0, d(\Gamma_0)) \in E_G$, return 'Yes', else 'No'.

Note that in Step 5, G is fully expanded. For purposes of the soundness proof, we note an immediate consequence of Lemmas 13 and 14:

▶ **Lemma 15.** *If some run of the algorithm without intermediate propagation steps is successful on input ϕ_0, then all runs on input ϕ_0 are successful.*

▶ Remark. For alternation-free fixpoint logics, the game-based approach (e.g. [14]) is to (1.) define a nondeterministic co-Büchi automaton of size $\mathcal{O}(n)$ that recognizes unsuccessful branches of the tableau. This automaton is then (2.) determinized to a deterministic co-Büchi automaton of size $2^{\mathcal{O}(n)}$ (avoiding the Safra construction using instead the method of [29]; here, alternation-freeness is crucial) and (3.) complemented to a deterministic Büchi automaton of the same size that recognizes successful branches of the tableau. A Büchi game is (4.) constructed as the product game of the carrier of the tableau and the carrier of the Büchi automaton. This game is of size $2^{\mathcal{O}(n)}$ and can be (5.) solved in time $2^{\mathcal{O}(n)}$. Our global caching algorithm integrates analogues of items (1.) to (5.) in one go: We directly construct the Büchi game (thus replacing (1.) through (4.) by a single definition) step-by-step during the computation of the sets E and A of (un)successful nodes as nested fixpoints of the proof transitionals; the propagation step corresponds to (5.). Our algorithm allows for intermediate propagation, corresponding to solving the Büchi game on-the-fly, i.e. before it has been fully constructed.

4 Soundness, Completeness and Complexity

Soundness. Let ϕ_0 be a satisfiable formula. By Lemma 15, it suffices to show that a run without intermediate propagation is successful.

▶ **Definition 16.** For a formula ψ, we define $\psi_X(\phi) = \psi[X \mapsto \phi]$, $\psi_X^0 = \bot$ and $\psi_X^{n+1} = \psi_X(\psi_X^n)$. We say that a Kripke structure $\mathcal{K}$ is *stabilizing* if for each state x in $\mathcal{K}$, each $\mu X.\, \psi$, and each fixpoint-free context $c(-)$ such that $x \models c(\mu X.\, \psi)$, there is $n \geq 0$ such that $x \models c(\psi_X^n)$.

We note that finite Kripke structures are stabilizing and import the finite model property (without requiring a bound on model size) for the μ-calculus from [26]; for the rest of the section, we thus fix w.l.o.g. a stabilizing Kripke structure $\mathcal{K} = (W, (R_a)_{a \in A}, \pi)$ satisfying the target formula ϕ_0 in some state.

▶ **Definition 17** (Unfolding tree)**.** Given a formula ψ, an *unfolding tree* t for ψ consists of the syntax tree of ψ together with a natural number as additional label for each node that represents a least fixpoint operator. We denote this number by $t(\kappa, \mu X.\, \phi)$ for an occurrence of a fixpoint literal $\mu X.\, \phi$ at position $\kappa \in \{0,1\}^*$ in ψ. We define the *unfolding* $\psi(t)$ of ψ according to an unfolding tree t for ψ by

$$X(t) = X \qquad (\phi_1 \wedge \phi_2)(t) = \phi_1(t_1) \wedge \phi_2(t_2) \quad (\mu X.\, \phi_1)(t) = (\phi_1(t_1))_X^{t(\epsilon, \mu X.\, \phi_1)},$$

where t_i is the i-th child of the root of t, and similar clauses for $\langle a \rangle$, $[a]$, $\vee$, and ν as for $\wedge$.

Given a formula ψ, we define the order $<_\psi$ on unfolding trees for ψ by lexically ordering the lists of labels obtained by pre-order traversal of the syntax tree of ψ.

▶ **Definition 18** (Unfolding). The *unfolding* of a formula ψ at a state x with $x \models \psi$ is defined as $unf(\psi, x) = \psi(t)$, where t is the least unfolding tree for ψ (w.r.t. $<_\psi$) such that $x \models \psi(t)$ (such a t exists by stabilization).

Note that in unfoldings, all least fixpoint literals $\mu X.\, \phi$ are replaced with finite iterates of ϕ.

▶ **Theorem 19** (Soundness). *The algorithm returns 'Yes' on input ϕ_0 if ϕ_0 is satisfiable.*

Proof. (Sketch) We show that any node (Γ, d) that is constructed by the algorithm and whose label is satisfied at some state x in $\mathcal{K}$ is successful, i.e. $(\Gamma, d) \in E_G$; the proof is by induction over the maximal modal depth of $unf(\delta, x)$ for $\delta \in d$. ◀

Completeness. Assume that the algorithm answers 'Yes' on input ϕ_0, having constructed the set $E := E_G$ of successful nodes. Put $D = \{(\Gamma, d) \in E \mid \Gamma \in \mathbf{S}\}$; note $|D| \leq |E| \leq 3^{|\phi_0|}$.

▶ **Definition 20** (Propositional entailment). For a finite set Ψ of formulas, we write $\bigwedge \Psi$ for the conjunction of the elements of Ψ. We say that Ψ *propositionally entails* a formula ϕ (written $\Psi \vdash_{PL} \phi$) if $\bigwedge \Psi \to \phi$ is a propositional tautology, where modal literals are treated as propositional atoms and fixpoint literals $\eta X.\phi$ are unfolded to $\phi(\eta X.\phi)$ (recall that fixpoint operators are guarded).

▶ **Definition 21.** We denote the set of formulas in a node Γ that do *not* belong to an eventuality θ by

$$N(\Gamma, \theta) = \{\phi \in \Gamma \mid \phi \notin dfr(\theta)\}.$$

A set d of deferrals is *sufficient* for $\delta \in dfr(\theta)$ at a node Γ, in symbols $d \vdash_\Gamma \delta$, if $d \cup N(\Gamma, \theta) \vdash_{PL} \delta$. We write $\vdash_\Gamma \delta$ to abbreviate $\emptyset \vdash_\Gamma \delta$.

▶ **Definition 22** (Timed-out tableau). Let $U \subseteq \mathbf{S} \times \mathbf{S}$ and let $L \subseteq U \times U$. We denote the set of L-successors of $v \in U$ by $L(v) = \{w \mid (v, w) \in L\}$. Let d be a set of deferrals. We put $to(\emptyset, n) = U$ for all n (to for *timeout*). For $d \neq \emptyset$, we put $to(d, 0) = \emptyset$ and define $to(d, m+1)$ to be the set of of focused nodes (Δ, d') such that writing $Cn(\Delta) = \{\Sigma_1, \dots, \Sigma_n\}$, we have $L(\Delta, d') = \{(\Gamma_1, d_1), \dots, (\Gamma_n, d_n)\}$ where for each i there exists $\Gamma \in \Sigma_i$ such that

- $\Gamma_i \vdash_{PL} \bigwedge \Gamma$ and $d_i \vdash_{\Gamma_i} d'_{\Delta \rightsquigarrow \Gamma}$, and
- $(\Gamma_i, d_i) \in to(d'', m)$ for some $d'' \subseteq d(\Gamma_i)$ with $d'' \vdash_{\Gamma_i} d_{\Delta \rightsquigarrow \Gamma}$.

If for each focused node $(\Gamma, d) \in U$ there is a number m such that $(\Gamma, d) \in to(d(\Gamma), m)$, then L is a *timed-out tableau* over U.

Roughly, $to(d, m)$ can be understood as the set of all focused nodes in U that finish all deferrals in d within m modal steps, i.e. with *time-out* m; this is similar to Kozen's *μ-counters* [25].

▶ **Lemma 23** (Tableau existence). *There exists a timed-out tableau over D.*

Proof sketch. Since $D \subseteq E_G$, we can define $L \subseteq D \times D$ in such a way that all paths in L visit F (the set of nodes with empty focus) infinitely often, so every deferral contained in some node in D will be focused by the unavoidable eventual refocusing; this new focus will in turn eventually be finished so that L is a timed-out tableau. ◀

For the rest of the section, we fix a timed-out tableau L over D and define a Kripke structure $\mathcal{K} = (D, (R_a)_{a \in A}, \pi)$ by taking $R_a(v)$ to be the set of focused nodes in $L(v)$ whose label is the conclusion of an $(\langle a \rangle)$-rule that matches $l(v)$ and by putting $\pi(p) = \{v \in D \mid p \in l(v)\}$.

▶ **Definition 24** (Pseudo-extension). The *pseudo-extension* $\widehat{[\![\phi]\!]}$ of ϕ in D is

$$\widehat{[\![\phi]\!]} = \{v \in D \mid l(v) \vdash_{PL} \phi\}.$$

▶ **Lemma 25** (Truth). *In the Kripke structure $\mathcal{K}$, $\widehat{[\![\psi]\!]} \subseteq [\![\psi]\!]$ for all $\psi \in \mathbf{F}$.*

Proof sketch. Induction on ψ, with an additional induction on time-outs in the case for least fixpoint literals, exploiting alternation-freeness. ◀

▶ **Corollary 26** (Completeness). *If a run of the algorithm with input ϕ_0 returns 'Yes', then ϕ_0 is satisfiable.*

Proof sketch. Combine the existence lemma and the truth lemma to obtain a model over D. Since $(\{\phi_0\}, d(\{\phi_0\})) \in E$ and $\widehat{[\![\phi_0]\!]} \subseteq [\![\phi_0]\!]$, there is a focused node in D that satisfies ϕ_0. ◀

As a by-product, our model construction yields

▶ **Corollary 27.** *Every satisfiable alternation-free fixpoint formula ϕ_0 has a model of size at most $3^{|\phi_0|}$.*

Thus we recover the bound of $2^{\mathcal{O}(n)}$ for the alternation-free relational μ-calculus, which can be obtained, e.g., by carefully adapting results from [13] to the alternation-free case; for the alternation-free fragment of the alternating-time μ-calculus, covered by the coalgebraic generalization discussed next, the best previous bound appears to be $n^{\mathcal{O}(n)} = 2^{\mathcal{O}(n \log n)}$ [36].

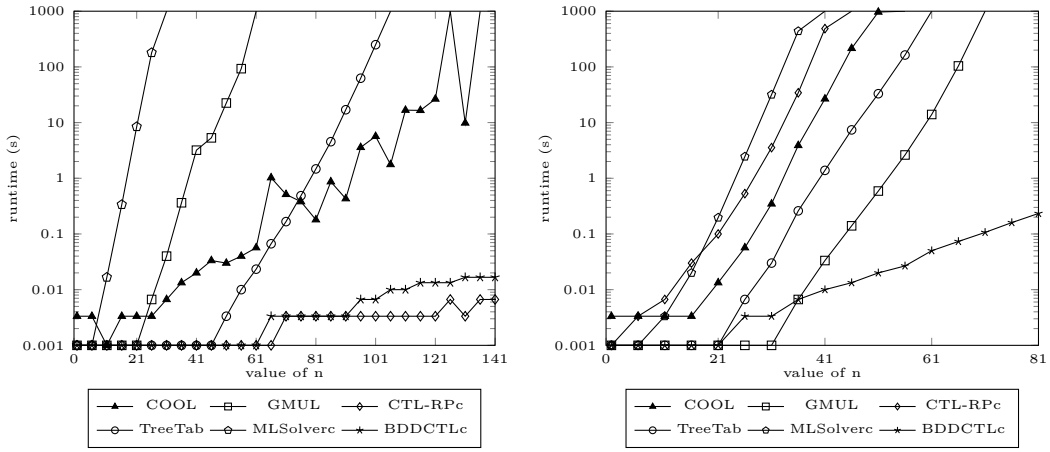
Complexity. Our algorithm has optimal complexity (given that the problem is known to be ExpTime-hard):

▶ **Theorem 28.** *The global caching algorithm decides the satisfiability problem of the alternation-free μ-calculus in* ExpTime*, more precisely in time $2^{\mathcal{O}(n)}$.*

The Alternation-Free Coalgebraic μ-Calculus. Coalgebraic logic [6] serves as a unifying framework for modal logics beyond standard relational semantics, subsuming systems with, e.g., probabilistic, weighted, game-oriented, or preference-based behaviour under the concept of coalgebras for a set functor F. All our results lift to the level of generality of the (alternation-free) coalgebraic μ-calculus [5]; details are in a technical report at `https://www8.cs.fau.de/hausmann/afgc.pdf`. In consequence, our results apply also to the alternation-free fragments of the alternating-time μ-calculus [1], probabilistic fixpoint logics, and the monotone μ-calculus (the ambient fixpoint logic of Parikh's game logic [32]), as all these can be cast as instances of the coalgebraic μ-calculus.

5 Implementation and Benchmarking

The global caching algorithm has been implemented as an extension of the *Coalgebraic Ontology Logic Reasoner* (COOL) [22], a generic reasoner for coalgebraic modal logics, available at `https://www8.cs.fau.de/research:software:cool`. COOL achieves its genericity by instantiating an abstract core reasoner that works for all coalgebraic logics to concrete instances of logics; our global caching algorithm extends this core. Instance logics implemented in COOL currently include relational, monotone, and alternating-time logics, as well as any logics that arise as fusions thereof. In particular, this makes COOL, to our knowledge, the only implemented reasoner for the alternation-free fragment of the alternating-time



(a) Montali, $n = 1$ (satisfiable). **(b)** Montali, $n = 1$ (unsatisfiable).

Figure 1 Runtimes for the Montali-formulas.

Table 1 Runtimes (in s) for the Alternating Bit Protocol formulas

Type of formula	COOL	TreeTab	GMUL	MLSolverc	BDDCTLc	CTL-RPc
(i)	<0.01	<0.01	<0.01	0.02	<0.01	0.02
(ii)	0.12	–	0.02	0.95	<0.01	0.15
(iii)	0.12	–	0.02	0.87	<0.01	0.16

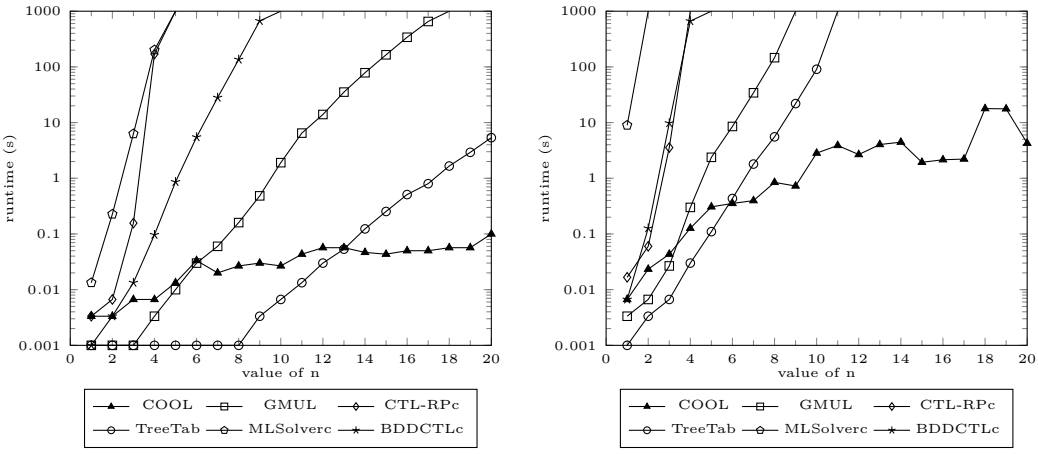
μ-calculus (a tableau calculus for the sublogic ATL is prototypically implemented in the TATL reasoner [8]) and the star-nesting free fragment of Parikh's game logic.

Although our tool supports the full alternation-free μ-calculus, we concentrate on CTL for experiments, as this appears to be the only candidate logic for which substantial sets of benchmark formulas are available [19]. CTL reasoners can be broadly classified as being either *top-down*, i.e. building graphs or tableaux by recursion over the formula, or *bottom-up*; the two groups perform very differently [19]. We compare our implementation with the top-down solvers TreeTab [15], GMUL [19], MLSolver [12] and the bottom-up solvers CTL-RP [37] and BDDCTL [19]. Out of the top-down solvers, only TreeTab is single-pass like COOL; however, TreeTab has suboptimal (doubly exponential) worst-case runtime. MLSolver supports the full μ-calculus. For MLSolver, CTL-RP and BDDCTL, formulas have first been *compacted* [19]. All tests have been executed on a system with Intel Core i7 3.60GHz CPU with 16GB RAM, and a stack limit of 512MB.

On the benchmark formulas of [19], COOL essentially performs similarly as the other top-down tools, and closer to the better tools when substantial differences show up. As an example, the runtimes of COOL, TreeTab, GMUL, MLSolver, CTL-RP, and BDDCTL on the Montali-formulas [30, 19] are shown in Figure 1. To single out one more example, Table 1 shows the runtimes for the alternating bit protocol benchmark from [19]; COOL performs closer to GMUL than to MLSolverc on these formulas.

This part of the evaluation may be summed up as saying that COOL performs well despite being, at the moment, essentially unoptimized: the only heuristics currently implemented is a simple-minded dependency of the frequency of intermediate propagation on the number of unexpanded nodes.

In addition, we design two series of unsatisfiable benchmark formulas that have an exponen-



(a) $early(n, 4, 2)$ (unsatisfiable). **(b)** $early_{gc}(n, 4, 2)$ (unsatisfiable).

Figure 2 Formulas with exponential search space and sub-exponential refutations.

tially large search space but allow for detection of unsatisfiability at an early stage. Recall that in CTL we can express the statement 'in the next step, the n-bit counter x represented by the variables $x_1, \ldots, x_n$ will be incremented' (with wraparound) as a formula $c(x, n)$ of polynomial size in n. We define unsatisfiable formulas $early(n, j, k)$ that specify an n-bit counter p with n bits and additionally branch after 2^j steps (i.e. when p_j holds) to start a counter r with k bits which in turn forever postpones the eventuality $EF\ p$:

$$\begin{aligned} early(n, j, k) = {} & start_p \wedge init(p, n) \wedge init(r, k) \wedge AG\ ((r \to c(r, k)) \wedge (p \to c(p, n))) \wedge \\ & AG\ ((\textstyle\bigwedge_{0 \le i \le j} p_i \to EX(start_r \wedge EF\ p)) \wedge \neg(p \wedge r) \wedge (r \to AX\ r)) \\ init(x, m) = {} & AG\ ((start_x \to (x \wedge \textstyle\bigwedge_{0 \le i < m} \neg x_i)) \wedge (x \to EX\ x)). \end{aligned}$$

Note here that *init* uses x as a string argument; $start_x$ is an atom indicating the start of counter x, and the atom x itself indicates that the counter x is running. The second series of unsatisfiable formulas $early_{gc}(n, j, k)$ is obtained by extending the formulas $early(n, j, k)$ with the additional requirement that a further counter q with n bits is started infinitely often, but at most at every second step:

$$\begin{aligned} early_{gc}(n, j, k) = {} & early(n, j, k) \wedge b \wedge init(q, n) \wedge AG\ (\neg(p \wedge q) \wedge \neg(q \wedge r) \wedge (q \to c(q, n))) \\ & \wedge\ AG\ (AF\ b \wedge (b \to (EX\ p \wedge EX\ start_q \wedge AX\ \neg b))) \end{aligned}$$

Figure 2 shows the respective runtimes for these formulas. In all cases, COOL finishes before the tableau is fully expanded, while GMUL and MLSolver will necessarily complete their first pass before being able to decide the formulas, and hence exhibit exponential behaviour; TreeTab seems not to benefit substantially from its capability to close tableaux early. For the $early_{gc}$ formulas, the ability to cache previously seen nodes appears to provide COOL with additional advantages. The $early_{gc}$ series can be converted into satisfiable formulas by replacing AX with EX, with similar results.

Due to the apparent lack of benchmarking formulas for the alternation-free μ-calculus and ATL, we compare runtimes on random formulas for these logics. For the alternation-free μ-calculus, formulas were built from 250 random operators (where disjunction and conjunction are twice as likely as the other operators). The experiment was conducted with formulas over three and over ten propositional atoms, respectively. MLSolver ran out of memory on

21% on the formulas over three atoms and on 16% of the formulas over ten atoms. COOL answered all queries without exceeding memory restrictions, and in under one second for all queries but one. Altogether, COOL was faster than MLSolver for more than 98% of the random alternation-free formulas, with the median of the ratios of the runtimes being 0.0431 in favour of COOL for formulas over three atoms and 0.0833 for formulas over ten atoms (recall however that MLSolver supports the full μ-calculus). For alternating-time temporal logic ATL, we compared the runtimes of TATL and COOL on random formulas consisting of 50 random operators; COOL answered faster than TATL on all of the formulas, with the median of the ratios of runtimes being 0.000668 in favour of COOL.

6 Conclusion

We have presented a tableau-based global caching algorithm of optimal (ExpTime) complexity for satisfiability in the alternation-free coalgebraic μ-calculus; the algorithm instantiates to the alternation-free fragments of e.g. the relational μ-calculus, the alternating-time μ-calculus (AMC) and the serial monotone μ-calculus. Essentially, it simultaneously generates and solves a deterministic Büchi game on-the-fly in a direct construction, in particular skipping the determinization of co-Büchi automata; the correctness proof, however, is stand-alone. We have generalized the $2^{\mathcal{O}(n)}$ bound on model size for alternation-free fixpoint formulas from the relational case to the coalgebraic level of generality, in particular to the AMC.

We have implemented the algorithm as part of the generic solver COOL; the implementation shows promising performance for CTL, ATL and the alternation-free relational μ-calculus. An extension of our global caching algorithm to the full μ-calculus would have to integrate Safra-style determinization of Büchi automata [35] and solving of the resulting parity game, both on-the-fly.

References

1. Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002.
2. Dietmar Berwanger, Erich Grädel, and Giacomo Lenzi. The variable hierarchy of the μ-calculus is strict. *Theory Comput. Sys.*, 40:437–466, 2007.
3. Julian Bradfield and Colin Stirling. Modal μ-calculi. In *Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.
4. Kai Brünnler and Martin Lange. Cut-free sequent systems for temporal logic. *J. Log. Algebr. Prog.*, 76:216–225, 2008.
5. Corina Cîrstea, Clemens Kupke, and Dirk Pattinson. EXPTIME tableaux for the coalgebraic μ-calculus. *Log. Meth. Comput. Sci.*, 7, 2011.
6. Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comput. J.*, 54:31–41, 2011.
7. Edmund Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.
8. Amélie David. TATL: Implementation of ATL tableau-based decision procedure. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2013*, volume 8123 of *LNCS*, pages 97–103. Springer, 2013.
9. E. Allen Emerson and Joseph Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Sys. Sci.*, 30:1–24, 1985.
10. E. Allen Emerson and Charanjit Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, September 1999.

11 Oliver Friedmann and Martin Lange. Local strategy improvement for parity game solving. In *Games, Automata, Logic, and Formal Verification, GANDALF 2010*, volume 25 of *EPTCS*, pages 118–131. Open Publishing Association, 2010.

12 Oliver Friedmann and Martin Lange. A solver for modal fixpoint logics. In *Methods for Modalities, M4M-6 2009*, volume 262 of *ENTCS*, pages 99–111, 2010.

13 Oliver Friedmann and Martin Lange. Deciding the unguarded modal μ-calculus. *J. Appl. Non-Classical Log.*, 23:353–371, 2013.

14 Oliver Friedmann, Markus Latte, and Martin Lange. Satisfiability games for branching-time logics. *Log. Methods Comput. Sci.*, 9, 2013.

15 Rajeev Goré. And-Or tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL. In *Automated Reasoning, IJCAR 2014*, volume 8562 of *LNCS*, pages 26–45. Springer, 2014.

16 Rajeev Goré, Clemens Kupke, and Dirk Pattinson. Optimal tableau algorithms for coalgebraic logics. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, volume 6015 of *LNCS*, pages 114–128. Springer, 2010.

17 Rajeev Goré, Clemens Kupke, Dirk Pattinson, and Lutz Schröder. Global caching for coalgebraic description logics. In *Automated Reasoning, IJCAR 2010*, volume 6173 of *LNCS*, pages 46–60. Springer, 2010.

18 Rajeev Goré and Linh Anh Nguyen. Exptime tableaux for ALC using sound global caching. *J. Autom. Reasoning*, 50:355–381, 2013.

19 Rajeev Goré, Jimmy Thomson, and Florian Widmann. An experimental comparison of theorem provers for CTL. In *Temporal Representation and Reasoning, TIME 2011*, pages 49–56. IEEE, 2011.

20 Rajeev Goré and Florian Widmann. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In *Automated Deduction, CADE 2009*, volume 5663 of *LNCS*, pages 437–452. Springer, 2009.

21 Rajeev Goré and Florian Widmann. Sound global state caching for $\mathcal{ALC}$ with inverse roles. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2009*, volume 5607 of *LNCS*, pages 205–219. Springer, 2009.

22 Daniel Gorín, Dirk Pattinson, Lutz Schröder, Florian Widmann, and Thorsten Wißmann. COOL – a generic reasoner for coalgebraic hybrid logics (system description). In *Automated Reasoning, IJCAR 2014*, volume 8562 of *LNCS*, pages 396–402. Springer, 2014.

23 Daniel Hausmann and Lutz Schröder. Global caching for the flat coalgebraic μ-calculus. In *Temporal Representation and Reasoning, TIME 2015*, pages 121–143. IEEE, 2015.

24 Natthapong Jungteerapanich. A tableau system for the modal μ-calculus. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2009*, volume 5607 of *LNCS*, pages 220–234. Springer, 2009.

25 Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

26 Dexter Kozen. A finite model theorem for the propositional μ-calculus. *Stud. Log.*, 47:233–241, 1988.

27 Martin Lange and Colin Stirling. Focus games for satisfiability and completeness of temporal logic. In *Logic in Computer Science, LICS 2001*, pages 357–365. IEEE Computer Society, 2001.

28 Zohar Manna and Amir Pnueli. The modal logic of programs. In *Automata, Languages and Programming, ICALP 1979*, volume 71 of *LNCS*, pages 385–409. Springer, 1979.

29 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32:321–330, 1984.

30 Marco Montali, Paolo Torroni, Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, and Paola Mello. Verification from declarative specifications using logic programming. In *Logic Programming, ICLP 2008*, volume 5366 of *LNCS*, pages 440–454. Springer, 2008.

31 Damian Niwinski and Igor Walukiewicz. Games for the μ-calculus. *Theor. Comput. Sci.*, 163:99–116, 1996.

32 Rohit Parikh. The logic of games and its applications. *Ann. Discr. Math.*, 24:111–140, 1985.

33 Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, FOCS 1977*, pages 46–57. IEEE Computer Society, 1977.

34 Vaughan Pratt. Semantical considerations on Floyd-Hoare logic. In *Foundations of Computer Science, FOCS 1976*, pages 109–121. IEEE Computer Society, 1976.

35 Shmuel Safra. On the complexity of omega-automata. In *Foundations of Computer Science, FOCS 1988*, pages 319–327. IEEE Computer Society, 1988.

36 Sven Schewe. *Synthesis of distributed systems*. PhD thesis, Universität des Saarlands, 2008.

37 Lan Zhang, Ullrich Hustadt, and Clare Dixon. A resolution calculus for the branching-time temporal logic CTL. *ACM Trans. Comput. Log.*, 15, 2014.

Up-To Techniques for Generalized Bisimulation Metrics

Konstantinos Chatzikokolakis[1], Catuscia Palamidessi[2], and Valeria Vignudelli[3]

1 **CNRS and LIX, Ecole Polytechnique**
2 **Inria and LIX, Ecole Polytechnique**
3 **University of Bologna and Inria**

Abstract

Bisimulation metrics allow us to compute distances between the behaviors of probabilistic systems. In this paper we present enhancements of the proof method based on bisimulation metrics, by extending the theory of up-to techniques to (pre)metrics on discrete probabilistic concurrent processes.

Up-to techniques have proved to be a powerful proof method for showing that two systems are bisimilar, since they make it possible to build (and thereby check) smaller relations in bisimulation proofs. We define soundness conditions for up-to techniques on metrics, and study compatibility properties that allow us to safely compose up-to techniques with each other. As an example, we derive the soundness of the up-to-bisimilarity-metric-and-context technique.

The study is carried out for a generalized version of the bisimulation metrics, in which the Kantorovich lifting is parametrized with respect to a distance function. The standard bisimulation metrics, as well as metrics aimed at capturing multiplicative properties such as differential privacy, are specific instances of this general definition.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases bisimulation, metrics, up-to techniques, Kantorovich, differential privacy

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.35

1 Introduction

Bisimulation has played a fundamental role in the analysis and verification of traditional concurrent systems. In recent times, however, there is a growing tendency to consider probabilistic frameworks, partly to capture the random nature of interactions in distributed systems, partly to model and reason about protocols which make use of randomized mechanisms, such as those used in security and privacy. In this context, equivalences are not suitable, because they are not robust w.r.t. small variation of the transition probabilities, and they are usually replaced by (pseudo-)metrics: unlike an equivalence relation, a metric can vary smoothly as a function of the probabilities, and it can be used to measure the similarity of two systems in a more informative way than an equivalence relation.

Bisimulation metrics are particularly successful, especially in the area of concurrency, They can be defined by generalizing to metrics the bisimilarity "progress" relation; using a terminology introduced by Sangiorgi [12], we say that a relation between processes $\mathcal{R}$ *progresses to* $\mathcal{S}$ if for every pair of processes in $\mathcal{R}$, every transition from one process is matched by a transition from the other, and the derivative processes are related by $\mathcal{S}$. A bisimulation can then be defined as a relation that progresses to itself. Using the same terminology for probabilistic transitions, a metric d on states progresses to a metric l on distributions over

states if, for all processes at d-distance ε, every transition from one process is matched by a transition from the other and the resulting distributions are at l-distance at most ε. Then d is a bisimulation metric if it progresses to its own *lifting* $K(d)$ on distributions.

Among the bisimulation metrics, those based on the Kantorovich lifting are the most popular. Originally proposed in the seminal works of Desharnais et al. [5, 6, 7] and of van Breugel and Worrel [13, 14], the traditional Kantorovich lifting has been extended in [3] so as to capture privacy properties such as *differential privacy* [8]. Part of their success is due to the *Kantorovich-Rubinstein duality*, which allows us to compute the lifting efficiently using linear programming algorithms [1, 13, 15, 16].

Analogously to the bisimilarity relation $\sim$, which is defined as the union of all bisimulations, the bisimilarity metric bm is defined coinductively as the smallest bisimulation metric. This means that we can extend the bisimulation proof method to metrics: given two processes P and Q, to prove $P \sim Q$ it is sufficient to find a bisimulation $\mathcal{R}$ such that $P \mathcal{R} Q$. Similarly, to show that $bm(P,Q) \leq \varepsilon$, it is sufficient to find a bisimulation metric d such that $d(P,Q) \leq \varepsilon$. The main difficulty in the bisimulation method is that the cost of naively checking that $\mathcal{R}$ is a bisimulation can be proportional to its size. Indeed, we need to prove that *for all pairs* of processes in $\mathcal{R}$, the derivatives of the matching transitions are still related by $\mathcal{R}$. Now, the size of bisimulations typically depends on the complexity of the underlying transition system, and if the transition system is unbounded, bisimulations are, in general, infinite sets. This difficulty translates immediately to the metric level: to prove that d is a bisimulation metric we need to prove that *for all pairs* of processes at d-distance ε, the distributions resulting from the matching transitions have $K(d)$-distance at most ε.

One well known and general approach, originally due to Milner [9], for reducing the sizes of bisimulations, is to represent them *up to* a different relation that identifies redundant pairs of process expressions. For instance, he showed that, when we consider the relation between the derivative processes, we can reason *modulo* bisimilarity. In other words, to prove $P \sim Q$ it is sufficient to find a relation $\mathcal{R}$ that relates P and Q, and that progresses to $\sim \mathcal{R} \sim$. In other words, if P' and Q' are the derivative processes, we do not need to show $P' \mathcal{R} Q'$, we only need to find a pair or processes P'' and Q'' such that $P' \sim P''$, $P'' \mathcal{R} Q''$, and $Q'' \sim Q'$. Such an $\mathcal{R}$ is called *bisimulation up to bisimilarity*. This technique was successively generalized by Sangiorgi [12], who introduced the notion of *bisimulation up to* $\mathcal{F}$, where $\mathcal{F}$ is a function from relations to relations. The idea is that $\mathcal{F}(\mathcal{R})$ contains the pairs of derivatives. The method is sound if, whenever $\mathcal{R}$ *progresses to* $\mathcal{F}(\mathcal{R})$, then $\mathcal{R} \subseteq \sim$. The paper also defines *respectfulness* for up-to techniques, later generalized as compatibility [11], which guarantees that it is sound to compose them with each other. The up-to techniques can be so effective that they may reduce the size of the relation to be checked from infinite to finite, and even, in some cases, to a singleton.

In this paper we aim at generalizing the up-to bisimulation method to the Kantorovich bisimulation metrics (in the extended version of [3]), thus enhancing the corresponding proof technique. The aim is to obtain a proof method that allows us to prove that $bm(P,Q) \leq \varepsilon$ by finding a metric d such that $d(P,Q) \leq \varepsilon$, and such that the set of pairs of processes for which we have to check the progress relation is relatively small. In other words, a metric d which gives maximal distance (and therefore the progress relation is verified trivially) between all processes except a small set. As an example, consider the following processes (from a probabilistic version of CCS):

$$A = a.([\tfrac{1}{2}]A \mid b \oplus [\tfrac{1}{2}]c) \qquad A' = a.([\tfrac{1}{2}]A' \mid b \oplus [\tfrac{1}{4}]c \oplus [\tfrac{1}{4}]d)$$

After performing an a-action, process A has one half probability of going back to itself, with the additional possibility of performing an action b in parallel, and one half probability of

performing action c. Process A' behaves similarly to A, but with probability one fourth it performs action d instead of c. In order to prove that $bm(A, A') \leq \frac{1}{2}$, we should define a metric assigning distance one half not only to the pair (A, A'), but also to all pairs of the form $A \,|\, b^n$ and $A \,|\, b^n$, where b^n is the parallel composition of n instances of b, representing the pairs to be inspected after the action a is performed for the n-th time. Each of these pairs should then be proved to satisfy the bisimulation metric clauses. Using up-to techniques, we can prove that $bm(A, A') \leq \frac{1}{2}$ just by considering a (pre)metric assigning one half distance to (A, A'), and maximal distance to all other non-identical states. When A performs a, then A' replies with the same action and the (probabilistic) up-to-context technique guarantees that it is sound to directly use the distance on (A, A') in place of the distance on $(A \,|\, b, A' \,|\, b)$.

Plan of the paper. Section 2 recalls some preliminary notions. Section 3 introduces some operators on premetrics and discusses some relevant properties of them. Section 4 presents the extension to metrics of the up-to techniques. Section 5 shows some examples of these techniques applied to probabilistic CCS and to the verification of differential privacy. Finally, Section 6 concludes. Some proofs were omitted for space reasons, they can be found in the report version of this paper [4].

2 Preliminaries

2.1 Premetrics and metrics

An (extended) *premetric* on a set X is a very relaxed form of metric, namely a function $m : X^2 \to [0, +\infty]$ satisfying only reflexivity ($m(x, x) = 0$). An (extended, pseudo) *metric* d on X is a premetric also satisfying symmetry ($d(x, y) = d(y, x)$) and the triangle inequality ($d(x, z) \leq d(x, y) + d(y, z)$). For simplicity we drop "extended" and "pseudo" but they are always implied; we denote by $\mathcal{M}(X), \mathcal{M}_d(X)$ the set of premetrics and metrics on X respectively. The *kernel* $\ker(m)$ of m is an equivalence relation on X relating elements at distance 0, i.e. $(x, y) \in \ker(m)$ iff $m(x, y) = 0$.

Premetrics $\mathcal{M}(X)$ bounded by some maximal distance $\top \in [0, \infty]$ form a complete lattice under element-wise ordering ($m \leq m'$ iff $m(x, y) \leq m'(x, y)$ for all x, y), with suprema and infima given by $(\bigvee A)(x, y) = \sup_{m \in A} m(x, y)$ and $(\bigwedge A)(x, y) = \inf_{m \in A} m(x, y)$. Note that the lattice depends on the choice of $\top$ – the value (possibly $+\infty$) assigned by the top premetric $\top_{\mathcal{M}(X)}$ to all distinct elements – which we generally leave implicit.

Metrics $\mathcal{M}_d(X)$ bounded by $\top$ also form a complete lattice under $\leq$, with the same supremum operator. On the other hand, the infimum operator, denoted by $\bigwedge_d$, is different since the inf of metrics is not necessarily a metric. Still, infima exist and can be obtained by $\bigwedge_d A = \bigvee(\downarrow_d A)$, where $\downarrow_d A = \{d \in \mathcal{M}_d(X) \mid \forall d' \in A : d \leq d'\}$.

2.2 Probabilistic automata, bisimilarity and metrics

Let S be a countable set of *states*.[1] We denote by $\mathcal{P}(S)$ the set of all (discrete) probability measures Δ, Θ over S; the Dirac measure on s by $\delta(s)$. A *Probabilistic automaton* (henceforth PA) $\mathcal{A}$ is a tuple (S, A, D) where A is a countable set of action *labels*, and $D \subseteq S \times A \times \mathcal{P}(S)$ is a *transition relation*. We write $s \xrightarrow{\alpha} \Delta$ for $(s, \alpha, \Delta) \in D$, and define a family of functions $\to_\alpha : S \to 2^{\mathcal{P}(S)}$ as $\to_\alpha (s) = \{\Delta \mid s \xrightarrow{\alpha} \Delta\}$.

[1] A countable state space is assumed for simplicity; however, the proofs of several results do not rely on this assumption, and we expect those that do to be extendible to the continuous case.

Let $R \subseteq S \times S$ be an equivalence relation on S; its lifting $\mathcal{L}(R)$ is an equivalence relation on $\mathcal{P}(S)$, defined as $(\Delta, \Theta) \in \mathcal{L}(R)$ iff Δ, Θ assign the same probability to all equivalence classes of R. *Probabilistic bisimilarity* $\sim$ can be defined as the largest equivalence relation R on S such that $(s,t) \in R$ and $s \xrightarrow{\alpha} \Delta$ imply $t \xrightarrow{\alpha} \Theta$ with $(\Delta, \Theta) \in \mathcal{L}(R)$.

Bisimilarity is a strong notion that often fails in probabilistic systems due to some "small" mismatch of probabilities. Hence, it is natural to define a metric that tells us "how much" different two states are, and such that its kernel coincides with $\sim$. Let $K : \mathcal{M}_d(S) \to \mathcal{M}_d(\mathcal{P}(S))$ be a lifting operator mapping metrics on S to metrics on distributions over S. A well known such operator is the *Kantorovich* lifting, but it is not unique: in fact, the Kantorovich itself can be generalized to a family of liftings, parametrized by an underlying distance (c.f. Section 3.2).

A metric $d \in \mathcal{M}_d(S)$ is a *bisimulation metric* if $d(s,t) < \top$ and $s \xrightarrow{\alpha} \Delta$ imply $t \xrightarrow{\alpha} \Theta$ with $K(d)(\Delta, \Theta) \leq d(s,t)$.[2] The *bisimilarity metric* bm can be defined as the $\bigwedge_d$ of all bisimulation metrics. Note that the lattice order of metrics has inverse meaning than the one of relations: a smaller metric corresponds to a larger relation.

It should be emphasized that, although $\sim$ is a uniquely defined relation, bm depends first on the choice of $\top$ and second, on the choice of the K operator. If $K, \mathcal{L}$ commute with ker, i.e. $\ker(K(d)) = \mathcal{L}(\ker(d))$ for all $d \in \mathcal{M}_d(S)$, it can be shown that $\sim = \ker(bm)$ [3]. In other words, we can have different metrics, all characterizing bisimilarity at their kernel, but which do not coincide on the distance they assign to non-bisimilar states.

Note that, although $\sim$ was defined as the union of all *equivalence* relations satisfying the bisimulation property, the "equivalence" requirement is only for convenience, so that the lifting $\mathcal{L}(R)$ has a simple form; we could obtain the same $\sim$ as the union of all *arbitrary* relations R satisfying the same property. The same is true for bm: although in the literature it is typically defined as the $\bigwedge_d$ of bisimulation *metrics*, we show in Section 4.1 that it can be constructed as the $\bigwedge$ of bisimulation *premetrics*. The advantage of using premetrics (resp. arbitrary relations) is that one has to construct a simpler bisimulation premetric m (resp. bisimulation relation R) not necessarily satisfying the triangle inequality (resp. transitivity), in order to bound the bisimilarity distance between two states.

3 Premetrics: operations and their properties

In this section we discuss various operations on premetrics and their properties. These will provide the technical building blocks for developing the up-to techniques in Section 4.

3.1 Lipschitz property and reverse maps

Lipschitz is a fundamental strong notion of continuity that plays a central role in all constructions of this work. A function $f : A \to B$ is *Lipschitz* (or nonexpansive) wrt the metrics m_A, m_B, written m_A, m_B-Lip, iff

$$m_B(f(a), f(a')) \leq m_A(a, a') \qquad \forall a, a' \in A$$

Tightly connected to this property is the *reverse map* on premetrics $f^{\leftarrow} : \mathcal{M}(B) \to \mathcal{M}(A)$ induced by $f : A \to B$, defined as $f^{\leftarrow}(m_B)(a, a') = m_B(f(a), f(a'))$.

▶ Proposition 1. The following hold:

[2] Note that if $d(s,t) = \top$ (i.e. s, t are maximally "non-bisimilar") then $t \xrightarrow{a} \Theta$ is not required at all.

1. f is m_A, m_B-Lip iff $m_A \geq f^{\leftarrow}(m_B)$.
2. $f^{\leftarrow}$ is monotone.
3. $f^{\leftarrow}$ preserves metrics: $m_B \in \mathcal{M}_d(B)$ implies $f^{\leftarrow}(m_B) \in \mathcal{M}_d(A)$.
4. $f^{\leftarrow}$ preserves $\bigwedge, \bigvee$, that is: $f^{\leftarrow}(\bigwedge M) = \bigwedge f^{\leftarrow}(M)$ and $f^{\leftarrow}(\bigvee M) = \bigvee f^{\leftarrow}(M)$.

Note that, from the first property above, we have that $m_A = f^{\leftarrow}(m_B)$ is the smallest premetric such that f is m_A, m_B-Lip.

3.2 Generalized Kantorovich lifting

To construct metrics for probabilistic systems, as described in Section 2, one needs to lift (pre)metrics on the state space S to (pre)metrics on $\mathcal{P}(S)$. One well known such lifting is the Kantorovich metric, defined either via Lipschitz functions, or dually as a transportation problem. In [3] a generalization of this construction is given by extending the range of Lipschitz functions from $(\mathbb{R}, |\cdot|)$ to a generic metric space (V, d_V), where $V \subseteq \mathbb{R}$ is a convex subset of the reals and $d_V \in \mathcal{M}_d(V)$.

A function $f : S \to V$ can be lifted to a function $\hat{f} : \mathcal{P}(S) \to V$ by taking expectations: $\hat{f}(\Delta) = \int_S f d\Delta$. The requirement that V is convex ensures that $\hat{f}(\Delta) \in V$. Then, given a premetric $m \in \mathcal{M}(S)$, we can define a lifted metric $K(m) \in \mathcal{M}(\mathcal{P}(S))$ as:

$$K(m)(\Delta, \Theta) = \sup\{d_V(\hat{f}(\Delta), \hat{f}(\Theta)) \mid f \text{ is } m, d_V\text{-Lip}\}$$

The lifting K depends on the choice of (V, d_V) that we generally leave implicit: many results are given for any member of the family, while some state specific conditions on d_V. Note the difference between m, the premetric being lifted, and d_V, a parameter of the construction. Using the construction of Section 2, each member of the family gives rise to a different bisimilarity metric bm, and under mild assumptions it can be shown that all of them characterize bisimilarity at their kernel [3].[3]

Of particular interest is the classical Kantorovich $K_{\oplus}$, corresponding to $(V, d_V) = (\mathbb{R}, |\cdot|)$, and the *multiplicative* variant $K_{\otimes}$, corresponding to $(V, d_V) = ((0, +\infty), d_{\otimes})$ where $d_{\otimes}(a, b) = |\ln a - \ln b|$. The corresponding bisimilarity metric obtained from the classical Kantorovich has been extensively studied; an important property of it is that $bm(s, t)$ is a bound on the total variation distance between the trace distributions originated from states s, t (a quantitative analogue of the fact that bisimilarity implies trace equivalence). The multiplicative Kantorovich provides the same bound, but for the multiplicative total variation distance, a metric of central importance to the area of differential privacy. Hence, the multiplicative variant provides a means for verifying privacy for concurrent systems.

Somewhat unexpectedly, it turns out that $K(m)$ is a proper metric, even if m itself is only a premetric: the metric properties of $K(m)$ come from those of d_V.

▶ Proposition 2. The following hold:

1. K is monotone.
2. $K(m) \in \mathcal{M}_d(S)$ (a proper metric) for all premetrics $m \in \mathcal{M}(S)$.

Another interesting property of K concerns its relationship with $f^{\leftarrow}$. Given $f : A \to B$, let $f_* : \mathcal{P}(A) \to \mathcal{P}(B)$ denote the function mapping Δ to its *pushforward measure*, given by

$$f_*(\Delta)(Z) = \Delta(f^{-1}(Z)) \quad \text{for all measurable} \quad Z \subseteq B$$

[3] Note that these "mild assumptions" are orthogonal to the results of this paper. If they are not satisfied, $\ker(bm)$ might be strictly included in $\sim$, without violating any of our results.

Then, we can map metrics in $\mathcal{M}(B)$ to those in $\mathcal{M}(\mathcal{P}(A))$ by either applying $f^{\leftarrow}$ followed by K, or applying K followed by ${f_*}^{\leftarrow}$. The two options are related by the following result:

▶ Proposition 3. Let $f : A \to B$ and $m_B \in \mathcal{M}(B)$. Then $(K \circ f^{\leftarrow})(m_B) \geq ({f_*}^{\leftarrow} \circ K)(m_B)$.

Due to the above result, K can be shown to preserve the Lip property (c.f. Section 3.4), which in turn is crucial for establishing the soundness of the up-to context techniques.

Dual form on premetrics. The classical Kantorovich lifting can be dually expressed as a transportation problem. The primal and dual formulations are well-known to coincide on metrics; however, this is no longer the case when we work on premetrics. To see this, notice that in the transportation problem, the distance $K^d(m)(\delta(s), \delta(t))$ (where K^d denotes the dual Kantorovich) between two point distributions is exactly $m(s,t)$, in other words $\delta^{\leftarrow} \circ K^d = id_{\mathcal{M}(S)}$. On the other hand, $K(m)$ is always a metric, and it can be shown that $\delta^{\leftarrow} \circ K$ gives the metric closure operator.

Note that the dual forms of both the classical and the multiplicative Kantorovich are particularly useful since, in contrast to the primal form, they provide direct algorithms for computing the distance between finite distributions. Since the two forms no longer coincide, we should ensure that both of them are sound when used in the up-to techniques. For a general Kantorovich lifting K, let K^d be a monotone lifting that coincides with K on metrics. It can be shown that $K^d(m) \leq K(m)$ for all premetrics m, which in turn means that replacing K with K^d in the up-to techniques of Section 4 is sound.

3.3 Metric closure and chaining

A metric can be thought of as a generalization of an equivalence relation, since it satisfies reflexivity, symmetry and transitivity (in the form of the triangle inequality). Similarly to the equivalence closure, it is natural to define the *metric* closure $m^{\triangledown}$ of m: intuitively, the goal is to decrease m just enough to enforce the metric properties. Since $\mathcal{M}_d$ is a complete lattice, $m^{\triangledown}$ can be naturally defined as the greatest metric below m:

$$m^{\triangledown} = \bigvee(\mathcal{M}_d \cap {\downarrow} m)$$

It can be shown that $m \mapsto m^{\triangledown}$ is a closure operator whose fixpoints are exactly $\mathcal{M}_d(S)$.

Let $M^{\triangledown}$ denote the set $\{m^{\triangledown} \mid m \in M\}$. We can show that metric closure commutes with the infima of the two lattices.

▶ Proposition 4. Let $M \subseteq \mathcal{M}$. Then $\bigwedge_d(M^{\triangledown}) = (\bigwedge M)^{\triangledown}$.

This, in turn, means that the metric infimum $\bigwedge_d$ can be obtained by the premetric infimum followed by metric closure, that is: $\bigwedge_d D = (\bigwedge D)^{\triangledown}$ for $D \subseteq \mathcal{M}_d(S)$. Based on this, we extend the $\bigwedge_d$ operator to premetrics, defined as $\bigwedge_d M = (\bigwedge M)^{\triangledown}$.

Finally, we can define the *chaining* $m_1 \curlywedge m_2$ of two premetrics as:

$$(m_1 \curlywedge m_2)(s_1, s_2) = \inf_{t \in S}(m_1(s_1, t) + m_2(t, s_2))$$

Chaining combines two premetrics by passing through some midway point, and will be used as a primitive block for constructing up-to techniques in Section 4.

▶ Proposition 5. The following hold:

1. $\curlywedge$ is associative and monotone on both arguments
2. $m_1 \wedge_d m_2 \leq m_1 \curlywedge m_2 \leq m_1 \wedge m_2$
3. $K(m_1 \curlywedge m_2) \leq K(m_1) \curlywedge K(m_2)$

3.4 Operations that preserve Lipschitz

The Lipschitz property plays a central role in all constructions of this work, since both the Kantorovich lifting and the notion of progression depend on it. The following operations preserving this property will play a crucial role in the up-to techniques developed in Section 4.

▶ **Theorem 1.** *Let $f : A \to B$ and assume it is m_A, m_B-Lip. Moreover, let $M_A = \{m_A^i\}_{i\in I}$ and $M_B = \{m_B^i\}_{i\in I}$ such that f is m_A^i, m_B^i-Lip for all $i \in I$. The following hold:*

1. *Inc/dec-reasing the source/target metric: f is m'_A, m'_B-Lip $\quad \forall m'_A \geq m_A, m'_B \leq m_B$*
2. *Infima and suprema: f is $\bigvee M_A, \bigvee M_B$-Lip and $\bigwedge M_A, \bigwedge M_B$-Lip*
3. *Metric closure: f is ${m_A}^\nabla, {m_B}^\nabla$-Lip*
4. *Kantorovich lifting: f_* is $K(m_A), K(m_B)$-Lip*

Note that the property (3) above implies that $K(m) = K(m^\nabla)$ since the sup in the definition of K for both sides ranges over the same set of functions.

3.5 Convex and quasiconvex premetrics

If X is a convex set then X^2 can be also viewed as a convex set of vectors (x, y), where $\sum_i \lambda_i(x_i, y_i) = (\sum_i \lambda_i x_i, \sum_i \lambda_i y_i)$ for all λ_i's such that $\sum_i \lambda_i = 1$. This allows us to talk about the convexity of a premetric jointly on both arguments. We say that $m \in \mathcal{M}(X)$ is:

- *convex* iff $m(\sum_i \lambda_i(x_i, y_i)) \leq \sum_i \lambda_i m(x_i, y_i)$
- *quasiconvex* iff $m(\sum_i \lambda_i(x_i, y_i)) \leq \max_i m(x_i, y_i)$

Note that there exist several distinct abstract notions of convexity for general metric spaces, here (quasi)convexity is used in the usual sense of (quasi)convex functions.

The set $\mathcal{P}(S)$ is convex and so is V used in the construction of the Kantorovich lifting. It can be shown that if d_V is convex (resp. quasiconvex) then $K(m)$ is also convex (resp. quasiconvex) for all $m \in \mathcal{M}(S)$. As a consequence, the classical Kantorovich $K_\oplus(m)$ is convex (since $|\cdot|$ is convex), while the multiplicative variant $K_\otimes(m)$ is quasiconvex (since $d_\otimes$ is quasiconvex).

4 Up-to techniques

In this section, we extend to the metric case the theory of up-to techniques presented in [12]. All the constructions assume some fixed underlying PA, which could be produced by a process calculus like the probabilistic CCS of Section 5. In what follows, we use l to denote premetrics on $\mathcal{P}(S)$.

4.1 Progressions

For a relation $\mathcal{R}$ on states of a non-probabilistic automaton, bisimulation can be defined in terms of progressions. A relation $\mathcal{R}$ progresses to $\mathcal{R}'$, denoted by $\mathcal{R} \rightarrowtail \mathcal{R}'$, if whenever $s\,\mathcal{R}\,t$ and $s \xrightarrow{\alpha} s'$ then $t \xrightarrow{\alpha} t'$ and $s'\,\mathcal{R}'\,t'$, and vice versa. A bisimulation can be thereby defined as a relation that progresses to itself, i.e. $\mathcal{R} \rightarrowtail \mathcal{R}$.

An important difference in the probabilistic case is that progressions have different source and target domains. A premetric m on S (the source premetric) progresses to a premetric l on $\mathcal{P}(S)$ (the target premetric).

▶ **Definition 2.** Given $m \in \mathcal{M}(S), l \in \mathcal{M}(\mathcal{P}(S))$ we say that m progresses to l, written $m \rightarrowtail l$, iff $m(s, t) < \top$ implies that:

- whenever $s \xrightarrow{\alpha} \Delta$ then $t \xrightarrow{\alpha} \Theta$ with $l(\Delta, \Theta) \leq m(s, t)$

- whenever $t \xrightarrow{\alpha} \Theta$ then $s \xrightarrow{\alpha} \Delta$ with $l(\Delta, \Theta) \leq m(s,t)$

Using the Hausdorff metric, progression can be written as a Lipschitz property:[4]

$$m \rightarrowtail l \quad \text{iff} \quad \forall \alpha : \rightarrow_\alpha \text{ is } m, H(l)\text{-Lip}$$

From the results about operations preserving Lipschitz, and the fact that Hausdorff is monotone, we obtain the following useful properties of the progress relation:

- $m \rightarrowtail l$ implies $m' \rightarrowtail l'$ for all $m' \geq m, l' \leq l$.
- Let $d \in \mathcal{M}_d(\mathcal{P}(S))$. Then $m \rightarrowtail d$ implies $m^\triangledown \rightarrowtail d$.
- Let $m = \bigwedge_i m_i$ and $l = \bigwedge l_i$ such that for all i: $m_i \rightarrowtail l_i$. Then $m \rightarrowtail l$.

From the definition of bisimulation (pre)metrics (Section 2), we have that $m \in \mathcal{M}(S)$ is a bisimulation (pre)metric iff $m \rightarrowtail K(m)$. The bisimilarity metric is traditionally defined as the $\bigwedge_d$ of all bisimulation metrics. Since metric closure preserves the Lip property, it also preserves the bisimulation property, which means that we can equivalently obtain bm as the $\bigwedge$ of all bisimulation premetrics.

▶ **Theorem 3.** *m is a bisimulation premetric iff $m^\triangledown$ is a bisimulation metric. Hence:* $bm = \bigwedge_d \{d \in \mathcal{M}_d(S) \mid d \rightarrowtail K(d)\} = \bigwedge \{m \in \mathcal{M}(m) \mid m \rightarrowtail K(m)\}$

Proof. Assuming that m is a bisimulation premetric, we have that $\rightarrow_\alpha$ is $m, H(K(m))$-Lip for all α. Since $H(K(m))$ is a metric, from Theorem 1 we get that $\rightarrow_\alpha$ is $m^\triangledown, H(K(m))$-Lip and since $K(m^\triangledown) = K(m)$ we get that $\rightarrow_\alpha$ is $m^\triangledown, H(K(m^\triangledown))$-Lip which implies that $m^\triangledown$ is a bisimulation metric. ◀

4.2 $\mathcal{F}$ functions, soundness, respectfulness

We can define an up-to technique using a function $\mathcal{F}$ on $\mathcal{M}(\mathcal{P}(S))$. Ideally, for a premetric m on states, we want to allow the distance $\mathcal{F}(K(m))(\Delta, \Theta)$ to be used instead of $K(m)(\Delta, \Theta)$ in a bisimulation proof, since a bound to $\mathcal{F}(K(m))$ could be easier to compute. Therefore, we consider progressions of the form $m \rightarrowtail \mathcal{F}(K(m))$, where $\mathcal{F} : \mathcal{M}(\mathcal{P}(S)) \to \mathcal{M}(\mathcal{P}(S))$.

▶ **Definition 4.** A function $\mathcal{F} : \mathcal{M}(\mathcal{P}(S)) \to \mathcal{M}(\mathcal{P}(S))$ is sound if $m \rightarrowtail \mathcal{F}(K(m))$ implies $bm \leq m$.

Hence, if $\mathcal{F}$ is a sound function then a bisimulation premetric up-to $\mathcal{F}$ allows us to derive upper-bounds to the distance between two states. At the same time, using $\mathcal{F}$ in the target metric allows us to simplify the proof that the states actually satisfy these bounds.

Respectful functions. Given a function $\mathcal{F} : \mathcal{M}(\mathcal{P}(S)) \to \mathcal{M}(\mathcal{P}(S))$, one can prove that it is a sound up-to technique by means of a direct proof. However, it is known that the composition of sound functions on relations is not necessarily a sound function, and the standard counterexamples apply to the metric setting as well. In the non-probabilistic case, this has led to the definition of "respectfulness": an up-to function $\mathcal{F}$ on relations is respectful if whenever $\mathcal{R} \rightarrowtail \mathcal{R}'$ and $\mathcal{R} \subseteq \mathcal{R}'$, then $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{R}')$ and $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{R}')$. Respectfulness implies soundness and at the same time is closed under composition [12].
On metrics, the definition of respectfulness must take care of the fact that the source and target metrics have different domains, and that the function $\mathcal{F}$ is defined on the domain $\mathcal{P}(S)$ of the target metric. Hence, a "corresponding" function $\mathcal{G} : \mathcal{M}(S) \to \mathcal{M}(S)$ on the

[4] We could also define progression as a Lipschitz property of a *single* function $\rightarrow(s) = \{(\alpha, \Delta) \mid s \xrightarrow{\alpha} \Delta\}$.

source metric has to be defined. Instead of constructing a specific such $\mathcal{G}$, we only assume its existence and that it "plays well" with $\mathcal{F}$ and K, meaning that $(K \circ \mathcal{G})(m) \leq (\mathcal{F} \circ K)(m)$. A concrete $\mathcal{G}$ is then chosen in the respectfulness proof of each up-to technique $\mathcal{F}$.

▶ **Definition 5.** A function $\mathcal{F} : \mathcal{M}(\mathcal{P}(S)) \to \mathcal{M}(\mathcal{P}(S))$ is *respectful* iff it is monotone and there exists $\mathcal{G} : \mathcal{M}(S) \to \mathcal{M}(S)$ such that for all $m, m' \in \mathcal{M}(S)$:

- $(K \circ \mathcal{G})(m) \leq (\mathcal{F} \circ K)(m)$
- $m \rightarrowtail K(m')$ and $m \geq m'$ imply $\mathcal{G}(m) \rightarrowtail K(\mathcal{G}(m'))$ and $\mathcal{G}(m) \geq \mathcal{G}(m')$

▶ **Theorem 6.** *Any respectful function is sound.*

Proof. Let $\mathcal{F}$ be respectful and let $\mathcal{G}$ be its corresponding source map from the definition of respectfulness. Assume that $m \rightarrowtail \mathcal{F}(K(m))$. Analogously to the proof in [12], we define a sequence of metrics $m_n, n \geq 0$ as: $m_0 = m$ and $m_{n+1} = \mathcal{G}(m_n) \wedge m_n$. By construction, $m_n \geq m_{n+1}$ for all $n \geq 0$. We now show that $m_n \rightarrowtail K(m_{n+1})$ for all $n \geq 0$ For the base case $n = 0$, from the respectfulness of $\mathcal{F}$ and the monotonicity of K we have that $\mathcal{F}(K(m)) \geq K(\mathcal{G}(m)) \geq K(\mathcal{G}(m) \wedge m)$. Hence $m \rightarrowtail \mathcal{F}(K(m))$ implies $m_0 = m \rightarrowtail K(\mathcal{G}(m) \wedge m) = K(m_1)$. For the inductive step, we want to show that $m_{n+1} \rightarrowtail K(m_{n+2})$, that is, $\mathcal{G}(m_n) \wedge m_n \rightarrowtail K(\mathcal{G}(m_{n+1}) \wedge m_{n+1})$. We have that:

$$
\begin{array}{lrl}
 & m_n \rightarrowtail K(m_{n+1}) & \text{induction hypothesis} \\
\Rightarrow & \mathcal{G}(m_n) \rightarrowtail K(\mathcal{G}(m_{n+1})) & \text{respectfulness, } m_n \geq m_{n+1} \\
\Rightarrow & \mathcal{G}(m_n) \wedge m_n \rightarrowtail K(\mathcal{G}(m_{n+1})) \wedge K(m_{n+1}) & \wedge \text{ preserves } \rightarrowtail \\
\Rightarrow & \mathcal{G}(m_n) \wedge m_n \rightarrowtail K(\mathcal{G}(m_{n+1}) \wedge m_{n+1}) & K(a \wedge b) \leq K(a) \wedge K(b)
\end{array}
$$

Since progressions are closed under infima, $\bigwedge_{n\geq 0} m_n \rightarrowtail K(\bigwedge_{n\geq 0} m_n)$. Hence, $\bigwedge_{n\geq 0} m_n$ is a bisimulation metric, and $m \geq \bigwedge_{n\geq 0} m_n$, which concludes the proof. ◀

4.2.1 Composing up-to techniques

The advantage of the respectfulness condition is that it makes it possible to derive the soundness of a composed up-to function just by proving the respectfulness of its components. We present here three operations that preserve respectfulness: function composition, function chaining, and taking the infimum of a set of functions (these operations respectively correspond to composition, chaining and union in the relational case).

▶ **Theorem 7.** *The composition of respectful functions is respectful.*

The theorem is proved by showing that, given two respectful functions $\mathcal{F}_1, \mathcal{F}_2$ and their corresponding source maps $\mathcal{G}_1, \mathcal{G}_2$ from the definition of respectfulness, $\mathcal{F} = \mathcal{F}_1 \circ \mathcal{F}_2$ and $\mathcal{G} = \mathcal{G}_1 \circ \mathcal{G}_2$ satisfy the requirements of respectfulness.

The chaining of up-to functions is defined using the $\curlywedge$ operator from Section 4.2.1. Define the chaining of two functions $\mathcal{F}_1, \mathcal{F}_2$ as $(\mathcal{F}_1 \curlywedge \mathcal{F}_2)(m) = \mathcal{F}_1(m) \curlywedge \mathcal{F}_2(m)$. Using the properties of $\curlywedge$ proved in Proposition 5, we derive the following result.

▶ **Theorem 8.** *The chaining of respectful functions is respectful.*

Analogously to chaining, define the infimum of a countable set of functions $\bigwedge\{\mathcal{F}_i\}$ as $\bigwedge\{\mathcal{F}_i\}(m) = \bigwedge\{\mathcal{F}_i(m)\}$. Given a countable set $\{\mathcal{F}_i\}$ of respectful functions with corresponding source maps $\{\mathcal{G}_i\}$, we prove that the function $\bigwedge\{\mathcal{F}_i\}$ is respectful by using the source map $\bigwedge\{\mathcal{G}_i\}$.

▶ **Theorem 9.** *The infimum of a set of respectful functions is respectful.*

4.2.2 Up-to bisimilarity metric and up-to (quasi)convexity

The respectfulness (and soundness) of up-to techniques such as up-to-bisimilarity-metric can now be recovered by applying the operations presented in Section 4.2.1 to basic respectful functions.

▶ **Theorem 10.** *The identity $\mathcal{F}_{id}(l) = l$ and the constant-to-bm $\mathcal{F}_{bm}(l) = K(bm)$ functions are respectful.*

The result directly follows from the definition: for the first we take $\mathcal{G}_{id}(m) = m$, for the second $\mathcal{G}_{bm}(m) = bm$. The up-to-bisimilarity-metric function can be now simply constructed as $\mathcal{F}_{bm} \curlywedge \mathcal{F}_{id} \curlywedge \mathcal{F}_{bm}$, and it is respectful as the chaining of respectful functions is (Theorem 8). By Theorem 9, we can also derive the respectfulness of the up-to-triangle-inequality function (corresponding to the up-to-transitive-closure technique on relations), defined as $\bigwedge\{\curlywedge^n \mathcal{F}_{id}\}_{n\geq 1}$, where $\curlywedge^n \mathcal{F}_{id}$ is the chaining of $\mathcal{F}_{id}$ with itself n-times.
Another useful proof technique consists in the possibility of splitting probability distributions into components with common factors, and then only consider the (possibly weighted) distances between the components. Define the up-to-quasiconvexity and the up-to-convexity functions as follows:

- $\mathcal{F}_{qcv}(l)(\Delta, \Theta) = \inf\{\max_i l(\Delta_i, \Theta_i) | \, \Delta = \sum_i p_i \Delta_i \text{ and } \Theta = \sum_i p_i \Theta_i\}$
- $\mathcal{F}_{cv}(l)(\Delta, \Theta) = \inf\{\sum_i p_i l(\Delta_i, \Theta_i) | \, \Delta = \sum_i p_i \Delta_i \text{ and } \Theta = \sum_i p_i \Theta_i\}$

The respectfulness of the above up-to techniques depends on the (quasi)convexity of the Kantorovich operator. The following result is derived using the identity $\mathcal{G}_{id}$ as a source map.

▶ **Theorem 11.** *If K is quasiconvex (resp. convex) then $\mathcal{F}_{qcv}$ (resp. $\mathcal{F}_{cv}$) is respectful.*

4.3 Faithful contexts

With up-to context techniques, common contexts in the probability distributions reached in the bisimulation game are allowed to be safely removed. Given a set of states S, a context is a function $C : S \to S$. As usual, we write $C[s]$ to denote the image of s under C. We look at states in S as defined by a language whose terms are syntactically finite expressions, which justifies the following assumption: for any class $\mathcal{C}$ of contexts, there is only a finite number of states s' such that $s = C[s']$ for some $C \in \mathcal{C}$.

▶ **Definition 12.** Given a class of contexts $\mathcal{C}$, a premetric m is closed under $\mathcal{C}$ iff C is m, m-Lip for all $C \in \mathcal{C}$. The closure of m under $\mathcal{C}$, denoted by $\mathcal{C}(m)$, is defined as the greatest premetric below m that is closed under $\mathcal{C}$:

$$\mathcal{C}(m) = \bigvee\{m' \leq m \mid m' \text{ is closed under } \mathcal{C}\}$$

Let $\mathcal{C}_* = \{C_* \mid C \in \mathcal{C}\}$. The up-to faithful context function $\mathcal{F}_{\mathcal{C}}$ is defined as: $\mathcal{F}_{\mathcal{C}}(l) = \mathcal{C}_*(l)$.

Since the Lipschitz property is preserved by $\bigvee$ (Thm 1), it is easy to show that $\mathcal{C}(m)$ itself is closed under $\mathcal{C}$, that is, $\mathcal{C}(m)(C[s], C[t]) \leq \mathcal{C}(m)(s,t) \leq m(s,t)$ for all $C \in \mathcal{C}$. Moreover, it follows from Thm 1 that K preserves the closure under $\mathcal{C}$. Hence, $K(\mathcal{C}(m))$ is always closed under $\mathcal{C}_*$: for all $C \in \mathcal{C}$, $K(\mathcal{C}(m))(C_*[\Delta], C_*[\Theta]) \leq K(\mathcal{C}(m))(\Delta, \Theta) \leq K(m)(\Delta, \Theta)$.
The function $\mathcal{C}(m)$ (respectively: $\mathcal{C}_*(l)$) can be alternatively characterized by considering the infimum value of m when a common context is removed from two terms (respectively: from two distributions). The context closure $(s,t)^{\mathcal{C}}$ of the pair (s,t) is the set of all pairs of terms of the form $(C[s], C[t])$, for $C \in \mathcal{C}$. The context closure $(\Delta, \Theta)^{\mathcal{C}_*}$ is extended to probability distributions using the set of contexts $C_* \in \mathcal{C}_*$.

▶ **Theorem 13.** *The functions $\mathcal{C}$ and $\mathcal{C}_*$ can be alternatively characterized as follows:*

1. $\mathcal{C}(m)(s,t) = \inf\{m(s',t') \mid (s,t) \in (s',t')^{\mathcal{C}}\}$
2. $\mathcal{C}_*(l)(\Delta,\Theta) = \inf\{l(\Delta',\Theta') \mid (\Delta,\Theta) \in (\Delta',\Theta')^{\mathcal{C}_*}\}$

In what follows, we often write $C[\Delta]$ to denote $C_*[\Delta]$.

Instead of directly proving soundness (or respectfulness) for up-to context functions $\mathcal{F}_{\mathcal{C}}$ where $\mathcal{C}$ are contexts of a specific language, we follow [12] and define the class of faithful contexts. Faithfulness only depends on general properties of the semantics of the contexts, and the up-to-faithful-context function is respectful whenever used with a quasiconvex Kantorovich operator (Theorem 15). In Section 5, the contexts of a probabilistic extension of CCS are proved to satisfy the condition of faithfulness.

▶ **Definition 14.** A context class $\mathcal{C}$ is faithful if whenever $C \in \mathcal{C}$, all transitions of $C[s]$ are of the form $C[s] \xrightarrow{\alpha} \sum_i p_i C_i[\Delta]$, where $C_i \in \mathcal{C}$ and either

1. $\Delta = \delta(s)$ and $\forall t$: $C[t] \xrightarrow{\alpha} \sum_i p_i C_i[\delta(t)]$, or
2. $s \xrightarrow{\alpha'} \Delta$ and $\forall t$: if $t \xrightarrow{\alpha'} \Theta$ then $C[t] \xrightarrow{\alpha} \sum_i p_i C_i[\Theta]$.

We can now prove the respectfulness of $\mathcal{F}_{\mathcal{C}}$, assuming that the Kantorovich operator is quasiconvex. The reason for this extra condition is that faithfulness allows contexts to be probabilistic, meaning that when a transition is performed, the common context can be split into a weighted sum of contexts. Quasiconvexity then allows us to establish a bound to the distances between weighted sums of distributions with a common contexts (e.g., $\sum_i p_i C_i[\Delta']$ and $\sum_i p_i C_i[\Theta']$) based on the bounds of the components, which now are of the desired form ($C_i[\Delta']$ and $C_i[\Theta']$).

▶ **Theorem 15.** *If K is quasiconvex then $\mathcal{F}_{\mathcal{C}}$ is respectful.*

Proof. The monotonicity of $\mathcal{F}_{\mathcal{C}}$ comes directly from the definition of $\mathcal{C}(m)$. Let $\mathcal{G}(m) = \mathcal{C}(m)$, we prove that $\mathcal{G}$ is the source map required by the definition of respectfulness:

1. we prove $K(\mathcal{G}(m)) \leq \mathcal{F}_{\mathcal{C}}(K(m))$. From $\mathcal{G}(m) \leq m$ we derive $K(\mathcal{G}(m)) \leq K(m)$, and since $\mathcal{G}(m)$ is closed under $\mathcal{C}$ and K preserves closedness, then $K(\mathcal{G}(m))$ is closed under $\mathcal{C}_*$. Finally, $\mathcal{F}_{\mathcal{C}}(K(m))$ is the greatest premetric below $K(m)$ that is is closed under $\mathcal{C}_*$, from which the result follows;
2. suppose $m \rightarrowtail K(m')$ and $m \geq m'$. Then $\mathcal{G}(m) \geq \mathcal{G}(m')$ comes from the monotonicity of $\mathcal{C}(m)$, and it remains to prove that $\mathcal{G}(m) \rightarrowtail K(\mathcal{G}(m'))$. We first show that
   - ⋆ for any faithful context C, $C[s] \xrightarrow{\alpha} \Delta$ implies that, for all t, if $m(s,t) < \top$ then $C[t] \xrightarrow{\alpha} \Theta$ with $K(\mathcal{G}(m'))(\Delta,\Theta) \leq m(s,t)$

   by considering the two cases of the definition of respectfulness and using quasiconvexity to derive the result. Since a term has only a finite number of subterms, by Theorem 13 we have $\mathcal{G}(m)(s,t) = m(s',t')$ for some s',t' and C faithful such that $s = C[s']$ and $t = C[t']$. Hence, by property ⋆ we have that $\mathcal{G}(m) \rightarrowtail K(\mathcal{G}(m'))$.

◀

5 Up-to techniques for probabilistic CCS

The conditions of faithfulness are quite general and can be instantiated by several varieties of probabilistic languages. We consider here CCS with a probabilistic choice operator and prove that its unary contexts (i.e., terms with a single hole, occurring only once) are faithful. The terms of pCCS are defined by the following grammar:

$$P, Q ::= \mathbf{0} \mid \alpha.\oplus_i [p_i]P_i \mid P + Q \mid P \mid Q \mid (\nu a)P \mid A$$

$$\frac{}{a.\oplus_i [p_i]P_i \xrightarrow{\alpha} \sum_i p_i\delta(P_i)} \qquad \frac{P \xrightarrow{\alpha} \Delta}{P+Q \xrightarrow{\alpha} \Delta} \qquad \frac{P \xrightarrow{\alpha} \Delta}{P \,|\, Q \xrightarrow{\alpha} \Delta \,|\, \delta(Q)}$$

$$\frac{P \xrightarrow{\alpha} \Delta \qquad Q \xrightarrow{\bar{\alpha}} \Theta}{P \,|\, Q \xrightarrow{\tau} \Delta \,|\, \Theta} \qquad \frac{P \xrightarrow{\alpha} \Delta \qquad \alpha \neq a, \bar{a}}{(\nu a)P \xrightarrow{\alpha} \Delta} \qquad \frac{P \xrightarrow{\alpha} \Delta \qquad A = P}{A \xrightarrow{\alpha} \Delta}$$

Figure 1 Structured Operational Semantics for pCCS.

where $\alpha ::= a, \bar{a}, \tau$ is an action label, for some underlying set of labels such that $a \in Act$ iff $\bar{a} \in Act$, and $\bar{\bar{\alpha}} = \alpha$ for $\alpha \in Act$, where $\tau \notin Act$. The semantics is given by the rules in Figure 1, where the parallel composition of distributions Δ, Θ on pCCS terms is defined by $\Delta \,|\, \Theta(P) = \Delta(P_1) \cdot \Theta(P_2)$ if $P = P_1 \,|\, P_2$, and 0 otherwise. The symmetric rules for the nondeterministic choice and parallel composition are omitted. We assume that every constant A of the language is defined by an equation $A = P$ for some pCCS process P where A may occur guarded. When the distribution following an action label is a point distribution, the $\oplus_i$ is omitted.

▶ **Theorem 16.** *The (unary) contexts of* pCCS *are faithful.*

Theorem 16 is proved by induction on the structure of the contexts. Since the up-to context technique is respectful for faithful contexts (Theorem 15), it follows from Theorem 16 that the up-to context function $\mathcal{F}_{\mathcal{C}}$ where $\mathcal{C}$ is the set of pCCS contexts is respectful.

▶ **Example 17.** Let A and A' be the pCCS constants defined in the introduction. We prove that their distance in the bisimilarity metric $bm_{\oplus}$, based on the standard Kantorovich lifting $K_{\oplus}$ and with $\top = 1$, is bounded by $\frac{1}{2}$. Define the premetric m on pCCS terms as follows: $m(A, A') = \frac{1}{2}$ and, for all P, Q different from A, A', m is the discrete metric, i.e., $m(P, Q) = 0$ if $P = Q$ and $m(P, Q) = 1$ otherwise. We prove that m is a bisimulation premetric up-to $(\mathcal{F}_{cv} \circ \mathcal{F}_{\mathcal{C}}) \curlywedge \mathcal{F}_{id}$, i.e., the chaining of the up-to-convexity-and-context function with the up-to-identity function.
Suppose that A moves (the case when A' moves is symmetrical). If $A \xrightarrow{a} \Delta = \frac{1}{2}\cdot\delta(A) + \frac{1}{2}\cdot\delta(c)$, then $A' \xrightarrow{a} \Delta' = \frac{1}{2}\cdot\delta(A') + \frac{1}{4}\cdot\delta(c) + \frac{1}{4}\cdot\delta(d)$. Define $\Delta'' = \frac{1}{2}\cdot\delta(A') + \frac{1}{2}\cdot\delta(c)$. Then:

$$\begin{aligned}((\mathcal{F}_{cv} \circ \mathcal{F}_{\mathcal{C}}) \curlywedge \mathcal{F}_{id})(K_{\oplus}(m))(\Delta, \Delta') &\leq (\mathcal{F}_{cv} \circ \mathcal{F}_{\mathcal{C}})(K_{\oplus}(m))(\Delta, \Delta'') + (K_{\oplus}(m))(\Delta'', \Delta') \\ &\leq \tfrac{1}{2} \cdot (K_{\oplus}(m))(\delta(A), \delta(A')) + (K_{\oplus}(m))(\Delta'', \Delta') \\ &\leq \tfrac{1}{4} + \tfrac{1}{4}\end{aligned}$$

Note that the same premetric and the same proof can be applied when an arbitrary pCCS process P is substituted to b in the definition of the constants A, A'.

Finally, we give an example to illustrate how the generalized Kantorovich lifting captures differential privacy, and how the techniques developed in this paper can help to verify this property. Following [3], we model differential privacy in pCCS as a bound e^{ε} on the ratio between the probability that a process P produce a set of traces ψ, and the probability that an "adjacent" process P' produce the same set ψ, for any ψ. In [3] it is shown that in order to establish this property it is sufficient to show that $bm_{\otimes}(P, P') \leq \varepsilon$, where $bm_{\otimes}$ is defined based on the multiplicative Kantorovich $K_{\otimes}$ and $\top = +\infty$.

In the example, we consider a database D containing medical information relative to (at most) n patients. We assume that we are interested in obtaining statistical information

about a certain disease, and that for this purpose we are allowed to ask queries like "how many patients are affected by the disease". Queries of this kind are called *counting queries* and it is well known that they can be sanitized, i.e. made ε-differentially private, by adding *geometric noise* to the real answer, namely a noise distribution $p_y(z) = c_z e^{|z-y|\varepsilon}$, where y is the real answer, z is the reported answer (ranging between 0 and n), and c_z is a normalization constant that depends only on z. Another database D' is *adjacent* to D if it differs from D for only one record (i.e., one patient). Clearly, the (sanitized) answers to the above query in two adjacent databases will differ by at most 1, and it is easy to see that the ratio between $p_{y+1}(z)$ and $p_y(z)$ is at most e^{ε}, which proves that ε-differential privacy is satisfied by the geometrical-noise method.

▶ **Example 18.** Consider the adjacent databases D, D' where y and $y+1$ patients are affected by the disease, respectively. We model D and D' in pCCS as

$$D = q.\oplus_{z=0}^{n} [p_y(z)]\bar{v}_z.D \qquad D' = q.\oplus_{z=0}^{n} [p_{y+1}(z)]\bar{v}_z.D'$$

where the prefix q represents the acceptance of a query request, and the action $\bar{v}_z$ represents the delivery of the reported answer. Consider now a process Q that queries the database. This can be defined as $Q = \bar{q}.+_{z=0}^{n} v_z.\bar{w}_z$, where $+_{z=0}^{n} P_z$ denotes the nondeterministic choice $P_0 + P_2 + ... + P_n$. It is possible to prove that the processes $D \mid Q$ and $D' \mid Q$ satisfy ε-differential privacy, by proving that $bm_{\otimes}(D \mid Q, D' \mid Q) \leq \varepsilon$.

What we want to prove now is that the level of differential privacy decreases linearly with the number of queries (this is a well-known fact, the interest here is to show it using up-to techniques). Namely that if we define the processes P and P' as the parallel composition of i instances of Q and D and D' respectively, then $K_{\otimes}(P, P') \leq i\varepsilon$ We prove this for the case $i = 2$. Define the premetric m as $m(D \mid Q \mid Q, D' \mid Q \mid Q) = 2\varepsilon$, and as the discrete metric on all other pairs. The interesting case is when D (symmetrically: D') synchronizes with one of the queries. Suppose that $D \mid Q \mid Q \xrightarrow{\tau} \Delta$, with $\Delta = \sum_{z=0}^{n} p_y(z) \cdot \delta(\bar{v}_z.D \mid (+_{z=0}^{n} v_z.\bar{w}_z) \mid Q)$. Then $D' \mid Q \mid Q \xrightarrow{\tau} \Delta'$, with $\Delta' = \sum_{z=0}^{n} p_{y+1}(z) \cdot \delta(\bar{v}_z.D' \mid (+_{z=0}^{n} v_z.\bar{w}_z) \mid Q)$. We derive the result by exploiting the soundness of the composition of up-to-quasiconvexity, up-to-context and up-to-bm functions, chained with up-to-identity. Let $\Delta'' = \sum_{z=0}^{n} p_y(z) \cdot \delta(\bar{v}_z.D' \mid (+_{z=0}^{n} v_z.\bar{w}_z) \mid Q)$. We have:

$$\begin{aligned}
&((\mathcal{F}_{qcv} \circ \mathcal{F}_{\mathcal{C}} \circ \mathcal{F}_{bm}) \curlywedge \mathcal{F}_{id})(K_{\otimes}(m))(\Delta, \Delta') \\
&\leq (\mathcal{F}_{qcv} \circ \mathcal{F}_{\mathcal{C}} \circ \mathcal{F}_{bm})(K_{\otimes}(m))(\Delta, \Delta'') + (K_{\otimes}(m))(\Delta'', \Delta') \\
&\leq (K_{\otimes}(bm))(\delta(D \mid Q), \delta(D' \mid Q)) + (K_{\otimes}(m))(\Delta'', \Delta') \\
&\leq \varepsilon + \varepsilon
\end{aligned}$$

6 Conclusion and future work

In this paper we studied techniques to increase the efficiency of the bisimulation proof method in the case of the (extended) Kantorovich metric. To this purpose, we have explored properties of the Kantorovich lifting, and we have generalized to the case of metrics the bisimulation up to $\mathcal{F}$ method by Sangiorgi. This allows us to reduce the size of the set of pairs for which we have to show the progress relation.

The theory of compatibility [11] for up-to techniques generalizes the respectfulness conditions on relations in a lattice-theoretic setting, where general properties of the progress relation and of the up-to functions (seen as functionals on the same lattice) can be proved and later instantiated to capture bisimulation relations on automata. A more recent approach

[10] consists in directly focusing on the greatest compatible (or respectful) function. In this paper we considered probabilistic systems and metrics, where the domain and the target of the progress relation are not in the same lattice anymore, and the up-to functions are defined on the target domain. The generalization of the techniques presented in this paper to a lattice-theoretic setting provides an interesting line of research.
In [2], up-to techniques are developed in an abstract fibrational setting, from which one could be able to obtain techniques for metrics. Studying whether the techniques of this paper can be obtained in this way is left as future work.

References

1 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In *Proc. of TACAS*, volume 7795 of *LNCS*, pages 1–15. Springer, 2013.

2 Filippo Bonchi, Daniela Petrişan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In *Proc. of CSL-LICS*, pages 20:1–20:9. ACM, 2014.

3 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In *Proc. of CONCUR*, volume 8704 of *LNCS*, pages 32–46. Springer, 2014.

4 Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Valeria Vignudelli. Up-to techniques for generalized bisimulation metrics. Technical report, INRIA, 2016. URL: `https://hal.inria.fr/hal-01335234`.

5 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled markov systems. In *Proc. of CONCUR*, volume 1664 of *LNCS*, pages 258–273. Springer, 1999.

6 Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. of LICS*, pages 413–422. IEEE, 2002.

7 Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comp. Sci.*, 318(3):323–354, 2004.

8 Cynthia Dwork. Differential privacy. In *Proc. of ICALP*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

9 R. Milner. *Communication and Concurrency*. Series in Comp. Sci. Prentice Hall, 1989.

10 Damien Pous. Coinduction all the way up. To appear in *Proc. of LICS*, 2016.

11 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

12 Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

13 Franck van Breugel and James Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proc. of CONCUR*, volume 2154 of *LNCS*, pages 336–350. Springer, 2001.

14 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In *Proc. of ICALP*, volume 2076 of *LNCS*, pages 421–432. Springer, 2001.

15 Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comp. Sci.*, 360(1–3):373–385, 2006.

16 Franck van Breugel and James Worrell. The complexity of computing a bisimilarity pseudometric on probabilistic automata. In *Horizons of the Mind*, volume 8464 of *LNCS*, pages 191–213. Springer, 2014.

Modal Decomposition on Nondeterministic Probabilistic Processes

Valentina Castiglioni[1], Daniel Gebler[2], and Simone Tini[3]

1 **University of Insubria, Como, Italy**
`v.castiglioni2@uninsubria.it`
2 **VU University Amsterdam, Amsterdam, The Netherlands**
`e.d.gebler@vu.nl`
3 **University of Insubria, Como, Italy**
`simone.tini@uninsubria.it`

Abstract

We propose a SOS-based method for decomposing modal formulae for nondeterministic probabilistic processes. The purpose is to reduce the satisfaction problem of a formula for a process to verifying whether its subprocesses satisfy certain formulae obtained from its decomposition. By our decomposition, we obtain (pre)congruence formats for probabilistic bisimilarity, ready similarity and similarity.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases SOS, nondeterministic probabilistic process algebras, logical characterization, decomposition of modal formulae

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.36

1 Introduction

In this paper we provide a SOS [23] driven method for the decomposition of modal formulae for nondeterministic probabilistic transition systems (PTSs) [6, 24], which are a model in which nondeterminism and probability coexist. In essence, our target is to reduce the satisfaction problem of a modal formula for a process to the satisfaction of suitable formulae for its subprocesses, where these formulae are derived from the SOS transition rules.

In the non probabilistic setting, such a problem has been tackled in [2, 12–14, 22], by exploiting *ruloids* [3], which are SOS transition rules that are derived from the SOS specification and define the behavior of open processes in terms of the behavior of their variables. In [2,12,14] the decomposition of modal formulae is used to systematically derive expressive congruence formats for several behavioral equivalences and preorders from their modal characterizations. In [15] such an approach is applied to the reactive probabilistic model [21], which does not admit internal nondeterminism and is therefore less general than PTSs.

In the PTS model, processes perform actions and evolve to probability distributions over processes, i.e. an a-labeled transition is of the form $t \xrightarrow{a} \pi$, with t a process and π a distribution holding all information on the probabilistic behavior arising from this transition. All modal logics developed for the PTS model are equipped with modalities allowing for the specification of the quantitative properties of processes. In essence, this means that some modal formulae are (possibly indirectly) evaluated on distributions. In order to decompose this kind of formulae, we introduce a SOS machinery, called *distribution specification*, allowing us to infer transitions of the form $\pi \xrightarrow{q} t$ whenever the distribution π assigns probability

q to process t. Then we derive the *distribution ruloids*, which allow us to define the behavior of open distributions in terms of the behavior of their distribution variables. These distribution ruloids can support the decomposition of formulae in any modal logic for PTSs.

We present the decomposition of formulae from the two-sorted boolean-valued modal logic $\mathcal{L}$ of [7]. This is an expressive logic, which characterizes probabilistic bisimilarity [7] and bisimilarity metric [4]. We apply our decomposition method also to two subclasses of formulae in $\mathcal{L}$, denoted by $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{+}$, which we prove to characterize resp. probabilistic ready similarity and similarity. Finally, to show the robustness of our approach we apply it to derive the congruence theorem for probabilistic bisimilarity wrt. the PGSOS format [5] and the precongruence theorem for probabilistic ready similarity and similarity wrt. the PGSOS format and the positive PGSOS format, respectively. Summarizing:

1. We present new logical characterizations of probabilistic ready similarity and similarity obtained by means of two sublogics of $\mathcal{L}$, resp. $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{+}$.
2. We define a SOS machinery for the specification of the probabilistic behavior of processes, which can support the decomposition of any modal logic for PTSs.
3. We develop a method of decomposing formulae in $\mathcal{L}$ and in its sublogics $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{+}$.
4. We derive (pre)congruence formats for probabilistic bisimilarity, ready similarity and similarity by exploiting our decomposition method on the logics characterizing them.

2 Probabilistic Transition Systems

The PTS model. A *signature* Σ is a countable set of *operators*. We let $\mathfrak{n}$ range over the rank of the operators. We assume a countable set of (state) *variables* $\mathcal{V}_s$ disjoint from Σ. The set $\mathsf{T}(\Sigma, V)$ of *terms* over Σ and $V \subseteq \mathcal{V}_s$ is defined as usual. By $\mathcal{T}(\Sigma)$ we denote the set of the *closed terms* $\mathsf{T}(\Sigma, \emptyset)$. By $\mathbb{T}(\Sigma)$ we denote the set of the *open terms* $\mathsf{T}(\Sigma, \mathcal{V}_s)$.

Nondeterministic probabilistic transition systems (PTSs) [6,24] extend LTSs by allowing for probabilistic choices in the transitions. The state space is the set of the closed terms $\mathcal{T}(\Sigma)$. The *transitions* are of the form $t \xrightarrow{a} \pi$, with t a term in $\mathcal{T}(\Sigma)$, a an action label and π a probability distribution over $\mathcal{T}(\Sigma)$, i.e. a mapping $\pi\colon \mathcal{T}(\Sigma) \to [0,1]$ with $\sum_{t \in \mathcal{T}(\Sigma)} \pi(t) = 1$. By $\Delta(\mathcal{T}(\Sigma))$ we denote the set of all probability distributions over $\mathcal{T}(\Sigma)$.

▶ **Definition 1** (PTS, [6, 24])**.** A PTS is a triple $(\mathcal{T}(\Sigma), \mathcal{A}, \to)$, where:
(i) Σ is a signature,
(ii) $\mathcal{A}$ is a countable set of *actions*, and
(iii) $\to \subseteq \mathcal{T}(\Sigma) \times \mathcal{A} \times \Delta(\mathcal{T}(\Sigma))$ is a *transition relation*.

We say that a PTS $P = (\mathcal{T}(\Sigma), \mathcal{A}, \to)$ is *image finite* if each closed term in $\mathcal{T}(\Sigma)$ has finitely many outgoing a-labeled transitions for each $a \in \mathcal{A}$.

For $\pi \in \Delta(\mathcal{T}(\Sigma))$, $\mathsf{supp}(\pi) = \{t \in \mathcal{T}(\Sigma) \mid \pi(t) > 0\}$ is the *support* of π. For $t \in \mathcal{T}(\Sigma)$, δ_t is the *Dirac distribution* s.t. $\delta_t(t) = 1$ and $\delta_t(s) = 0$ for $s \neq t$. For $f \in \Sigma$ and $\pi_i \in \Delta(\mathcal{T}(\Sigma))$, $f(\pi_1, \ldots, \pi_{\mathfrak{n}})$ is the distribution defined by $f(\pi_1, \ldots, \pi_{\mathfrak{n}})(f(t_1, \ldots, t_{\mathfrak{n}})) = \prod_{i=1}^{\mathfrak{n}} \pi_i(t_i)$. The convex combination $\sum_{i \in I} p_i \pi_i$ of a family of distributions $\{\pi_i\}_{i \in I} \subseteq \Delta(\mathcal{T}(\Sigma))$ with $p_i \in (0,1]$ and $\sum_{i \in I} p_i = 1$ is defined by $(\sum_{i \in I} p_i \pi_i)(t) = \sum_{i \in I} (p_i \pi_i(t))$ for all $t \in \mathcal{T}(\Sigma)$.

Bisimulation. A (probabilistic) bisimulation is an equivalence relation over $\mathcal{T}(\Sigma)$ equating two terms if they can mimic each other's transitions and evolve to distributions related by the same bisimulation. To formalize this, we need to lift relations over terms to distributions.

▶ **Definition 2** (Relation lifting, [8]). The *lifting* of a relation $\mathcal{R} \subseteq \mathcal{T}(\Sigma) \times \mathcal{T}(\Sigma)$ is the relation $\mathcal{R}^\dagger \subseteq \Delta(\mathcal{T}(\Sigma)) \times \Delta(\mathcal{T}(\Sigma))$ with $\pi \,\mathcal{R}^\dagger\, \pi'$ whenever there is a countable set of indexes I s.t.:

(i) $\pi = \sum_{i\in I} p_i \delta_{s_i}$,
(ii) $\pi' = \sum_{i\in I} p_i \delta_{t_i}$, and
(iii) $s_i \;\mathcal{R}\; t_i$ for all $i \in I$.

▶ **Definition 3** (Probabilistic (bi)simulations, [21, 24]). Assume a PTS $(\mathcal{T}(\Sigma), \mathcal{A}, \to)$.

1. A binary relation $\mathcal{R} \subseteq \mathcal{T}(\Sigma) \times \mathcal{T}(\Sigma)$ is a *simulation* if, whenever $s \,\mathcal{R}\, t$, if $s \xrightarrow{a} \pi_s$ then there is a transition $t \xrightarrow{a} \pi_t$ s.t. $\pi_s \,\mathcal{R}^\dagger \pi_t$.
2. A simulation $\mathcal{R}$ is a *ready simulation* if, whenever $s \,\mathcal{R}\, t$, if $s \xrightarrow{a}\!\!\!\!\!\!/\;$ then $t \xrightarrow{a}\!\!\!\!\!\!/\;$.
3. A *bisimulation* is a symmetric simulation.

The union of all simulations (resp.: ready simulations, bisimulations) is the greatest simulation (resp.: ready simulation, bisimulation), denoted $\sqsubseteq$ (resp.: $\sqsubseteq_{\mathrm{r}}$, $\sim$), called *similarity* (resp.: *ready similarity*, *bisimilarity*), and is a preorder (resp.: preorder, equivalence).

Logical characterization. As a logic expressing behavioral properties over terms, we consider the modal logic $\mathcal{L}$ of [7], which extends HML [19] with a probabilistic choice modality.

▶ **Definition 4** (Modal logic $\mathcal{L}$, [7]). The classes of *state formulae* $\mathcal{L}^{\mathrm{s}}$ and *distribution formulae* $\mathcal{L}^{\mathrm{d}}$ over $\mathcal{A}$ are defined by the following BNF-like grammar:

$$\mathcal{L}^{\mathrm{s}}\colon\quad \varphi ::= \quad \top \mid \neg\varphi \mid \textstyle\bigwedge_{j\in J} \varphi_j \mid \langle a\rangle\psi \qquad\qquad \mathcal{L}^{\mathrm{d}}\colon\quad \psi ::= \quad \bigoplus_{i\in I} r_i\varphi_i$$

where:

(i) φ ranges over $\mathcal{L}^{\mathrm{s}}$,
(ii) ψ ranges over $\mathcal{L}^{\mathrm{d}}$,
(iii) $a \in \mathcal{A}$,
(iv) I, J are at most countable sets of indexes with $I, J \neq \emptyset$, and
(v) $r_i \in (0, 1]$ for each $i \in I$ and $\sum_{i\in I} r_i = 1$.

We shall write $\langle a\rangle\varphi$ for $\langle a\rangle \bigoplus_{i\in I} r_i\varphi_i$ with $I = \{i\}$, $r_i = 1$ and $\varphi_i = \varphi$.

▶ **Definition 5** (Satisfaction relation, [7]). The *satisfaction relation* $\models \subseteq (\mathcal{T}(\Sigma) \times \mathcal{L}^{\mathrm{s}}) \cup (\Delta(\mathcal{T}(\Sigma)) \times \mathcal{L}^{\mathrm{d}})$ is defined by structural induction on formulae by

- $t \models \top$ always;
- $t \models \neg\varphi$ iff $t \models \varphi$ does not hold;
- $t \models \bigwedge_{j\in J} \varphi_j$ iff $t \models \varphi_j$ for all $j \in J$;
- $t \models \langle a\rangle\psi$ iff $t \xrightarrow{a} \pi$ for a distribution $\pi \in \Delta(\mathcal{T}(\Sigma))$ with $\pi \models \psi$;
- $\pi \models \bigoplus_{i\in I} r_i\varphi_i$ iff $\pi = \sum_{i\in I} r_i\pi_i$ for distributions π_i with $t \models \varphi_i$ for all $t \in \mathsf{supp}(\pi_i)$.

Dealing with $\mathcal{L}$ is motivated by its characterization of bisimilarity, proved in [7] (see Thm. 6 below), bisimilarity metric, proved in [4], and similarity and ready similarity, proved here (see Thm. 8 below).

▶ **Theorem 6** ([7]). *Assume an image finite PTS $(\mathcal{T}(\Sigma), \mathcal{A}, \to)$ and terms $s, t \in \mathcal{T}(\Sigma)$. Then, $s \sim t$ if and only if they satisfy the same formulae in $\mathcal{L}^{\mathrm{s}}$.*

The characterization of ready similarity and similarity requires two subclasses of $\mathcal{L}$.

▶ **Definition 7.** The classes of *ready formulae* $\mathcal{L}_{\mathrm{r}}$ and *positive formulae* $\mathcal{L}_{+}$ are defined as

$$\begin{array}{lll} \mathcal{L}_{\mathrm{r}}^{\mathrm{s}}: & \varphi ::= & \top \mid \bar{a} \mid \bigwedge_{j\in J} \varphi_j \mid \langle a\rangle\psi \\ \mathcal{L}_{+}^{\mathrm{s}}: & \varphi ::= & \top \mid \bigwedge_{j\in J} \varphi_j \mid \langle a\rangle\psi \end{array} \qquad \begin{array}{lll} \mathcal{L}_{\mathrm{r}}^{\mathrm{d}}: & \psi ::= & \bigoplus_{i\in I} r_i\varphi_i \\ \mathcal{L}_{+}^{\mathrm{d}}: & \psi ::= & \bigoplus_{i\in I} r_i\varphi_i \end{array}$$

where $\bar{a}$ stays for $\neg\langle a\rangle\top$.

The classes $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{+}$ are strict sublogics of the one proposed in [9] for the characterization of failure similarity and forward similarity [24]. In particular, the logic used in [9] allows for arbitrary formulae to occur after the diamond modality. We can show that our sublogics are powerful enough for the characterization of ready similarity and similarity.

▶ **Theorem 8.** *Assume an image finite PTS* $(\mathcal{T}(\Sigma), \mathcal{A}, \to)$ *and terms* $s, t \in \mathcal{T}(\Sigma)$*. Then:*

1. $s \sqsubseteq_{\mathrm{r}} t$ *iff for any formula* $\varphi \in \mathcal{L}_{\mathrm{r}}^{\mathrm{s}}$*,* $s \models \varphi$ *implies* $t \models \varphi$*.*
2. $s \sqsubseteq t$ *iff for any formula* $\varphi \in \mathcal{L}_{+}^{\mathrm{s}}$*,* $s \models \varphi$ *implies* $t \models \varphi$*.*

Probabilistic transition system specifications. PTSs are usually defined by means of SOS rules, which are syntax-driven inference rules allowing us to infer the behavior of terms inductively wrt. their structure. Here we consider rules in the probabilistic GSOS format [5] (examples in Ex. 10), which allow for specifying most of probabilistic process algebras [16,18].

In these rules we need syntactic expressions that denote probability distributions. We assume a countable set of *distribution variables* $\mathcal{V}_d$. We denote by $\mathcal{V}$ the set of state and distribution variables $\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_d$. We let $\mu, \nu, \ldots$ range over $\mathcal{V}_d$ and ζ range over $\mathcal{V}$. The set of *distribution terms* over Σ, $V_s \subseteq \mathcal{V}_s$ and $V_d \subseteq \mathcal{V}_d$, notation $\mathsf{DT}(\Sigma, V_s, V_d)$, is the least set satisfying:

(i) $\{\delta_t \mid t \in \mathsf{T}(\Sigma, V_s)\} \subseteq \mathsf{DT}(\Sigma, V_s, V_d)$,
(ii) $V_d \subseteq \mathsf{DT}(\Sigma, V_s, V_d)$,
(iii) $f(\Theta_1, \ldots, \Theta_n) \in \mathsf{DT}(\Sigma, V_s, V_d)$ whenever $f \in \Sigma$ and $\Theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$, and
(iv) $\sum_{i\in I} p_i\Theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$ whenever $\Theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$ and $p_i \in (0, 1]$ with $\sum_{i\in I} p_i = 1$.

We write $\mathbb{DT}(\Sigma)$ for $\mathsf{DT}(\Sigma, \mathcal{V}_s, \mathcal{V}_d)$, i.e. the set of all *open distribution terms*, and $\mathcal{DT}(\Sigma)$ for $\mathsf{DT}(\Sigma, \emptyset, \emptyset)$, i.e. the set of all *closed distribution terms*. Distribution terms have the following meaning. An *instantiable Dirac distribution* δ_t instantiates to $\delta_{t'}$ if t instantiates to t'. A *distribution variable* $\mu \in \mathcal{V}_d$ is a variable that takes values from $\Delta(\mathcal{T}(\Sigma))$. Case (3) lifts the structural inductive construction of terms to distribution terms. Case (4) allows us to construct convex combinations of distributions. By $\mathrm{var}(t)$ (resp. $\mathrm{var}(\Theta)$) we denote the set of the variables occurring in t (resp. Θ).

A *positive (resp. negative) literal* is an expression of the form $t \xrightarrow{a} \Theta$ (resp. $t \not\xrightarrow{a}$) with $t \in \mathbb{T}(\Sigma)$, $a \in \mathcal{A}$ and $\Theta \in \mathbb{DT}(\Sigma)$. The literals $t \xrightarrow{a} \Theta$ and $t \not\xrightarrow{a}$ are said to *deny* each other.

▶ **Definition 9** (PGSOS rules, [5])**.** A *PGSOS rule* r has the form:

$$\frac{\{x_i \xrightarrow{a_{i,m}} \mu_{i,m} \mid i \in I, m \in M_i\} \qquad \{x_i \xrightarrow{a_{i,n}} \not\rightarrow \mid i \in I, n \in N_i\}}{f(x_1, \ldots, x_{\mathfrak{n}}) \xrightarrow{a} \Theta}$$

with $f \in \Sigma$, $I = \{1, \ldots, \mathfrak{n}\}$, M_i, N_i finite indexes sets, $a_{i,m}, a_{i,n}, a \in \mathcal{A}$ actions, $x_i \in \mathcal{V}_s, \mu_{i,m} \in \mathcal{V}_d$ variables and $\Theta \in \mathbb{DT}(\Sigma)$ a distribution term. Furthermore, all $\mu_{i,m}$ for $i \in I$ and $m \in M_i$ are distinct, all $x_1, \ldots, x_{\mathfrak{n}}$ are distinct, and $\mathrm{var}(\Theta) \subseteq \{\mu_{i,m} \mid i \in I, m \in M_i\} \cup \{x_1, \ldots, x_{\mathfrak{n}}\}$.

We say that $P = (\Sigma, \mathcal{A}, R)$, with Σ a signature, $\mathcal{A}$ a countable set of actions and R a finite set of PGSOS rules, is a *PGSOS probabilistic transition system specification (PGSOS-PTSS).*

▶ **Example 10.** The operators of synchronous parallel composition $|$ and probabilistic alternative composition $+_p$, with $p \in (0,1]$, are specified by the following PGSOS rules:

$$\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu}{x|y \xrightarrow{a} \mu|\nu} \qquad \frac{x \xrightarrow{a} \mu \quad y \not\xrightarrow{a}}{x +_p y \xrightarrow{a} \mu} \qquad \frac{x \not\xrightarrow{a} \quad y \xrightarrow{a} \nu}{x +_p y \xrightarrow{a} \nu} \qquad \frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu}{x +_p y \xrightarrow{a} p\mu + (1-p)\nu}\,.$$

For a PGSOS rule r, the positive (resp. negative) literals above the line are the *positive premises*, notation $\mathrm{pprem}(r)$ (resp. *negative premises*, notation $\mathrm{nprem}(r)$). The literal $f(x_1,\dots,x_{\mathtt{n}}) \xrightarrow{a} \Theta$ is called the *conclusion*, notation $\mathrm{conc}(r)$, the term $f(x_1,\dots,x_{\mathtt{n}})$ is called the *source* and the distribution term Θ is called the *target*.

A PGSOS rule r is said to be *positive* if $\mathrm{nprem}(r) = \emptyset$. Then we say that a PGSOS-PTSS $P = (\Sigma, \mathcal{A}, R)$ is *positive* if all the PGSOS rules in R are positive.

A PTS is derived from a PTSS through the notions of substitution and proof.

A *substitution* is a mapping $\sigma\colon \mathcal{V} \to \mathbb{T}(\Sigma) \cup \mathbb{DT}(\Sigma)$ s.t. $\sigma(x) \in \mathbb{T}(\Sigma)$ if $x \in \mathcal{V}_s$ and $\sigma(\mu) \in \mathbb{DT}(\Sigma)$ if $\mu \in \mathcal{V}_d$. It extends to terms, literals and rules by element-wise application. A substitution is *closed* if it maps variables to closed terms. A closed substitution instance of a literal (resp. PGSOS rule) is called a *closed literal* (resp. *closed PGSOS rule)*.

▶ **Definition 11** (Proof)**.** A *proof* from a PTSS P of a closed literal α is a well-founded, upwardly branching tree, with nodes labeled by closed literals, s.t. the root is labeled α and, if β is the label of a node $\mathfrak{q}$ and $\mathcal{K}$ is the set of labels of the nodes directly above $\mathfrak{q}$, then:

- either β is positive and $\mathcal{K}/\beta$ is a closed substitution instance of a rule in R,
- or β is negative and for each closed substitution instance of a rule in R whose conclusion denies β, a literal in $\mathcal{K}$ denies one of its premises.

A literal α is *provable* from P, notation $P \vdash \alpha$, if there exists a proof from P of α.

The set of literals provable from a PGSOS-PTSS P is unique and contains literals that do not deny each other [3]. The *model induced by* P is the PTS $(\mathcal{T}(\Sigma), \mathcal{A}, \to)$ whose transition relation $\to$ contains exactly the closed positive literals provable from P.

3 Distribution specifications

The decomposition of state formulae in Sec. 4 is based on a collection of rules extracted from the PTSS, called ruloids. To have a similar method for distribution formulae, we develop a SOS-like machinery allowing us to infer the expression $\Theta \xrightarrow{q} t$ whenever a closed distribution term Θ assigns probability weight q to a closed term t. Such a machinery can be exploited also to decompose formulae of any logic, and can be easily generalized to cover the case of sub-distributions that are used in models different from PTSs.

A *distribution literal* is of the form $\Theta \xrightarrow{q} t$, with $\Theta \in \mathbb{DT}(\Sigma)$, $q \in (0,1]$ and $t \in \mathbb{T}(\Sigma)$. A set of distribution literals $\{\Theta \xrightarrow{q_i} t_i \mid i \in I\}$ is a *distribution over terms* if $\sum_{i\in I} q_i = 1$ and all t_i are distinct. This expresses that Θ is the distribution over $\mathbb{T}(\Sigma)$ giving weight q_i to t_i.

To infer distributions over terms $\{\Theta \xrightarrow{q_i} t_i \mid i \in I\}$ inductively wrt. the structure of Θ, we introduce the Σ-*distribution rules*. Let $\delta_{\mathcal{V}_s} := \{\delta_x \mid x \in \mathcal{V}_s\}$ denote the set of all instantiable Dirac distributions with a variable as term, and $\vartheta, \vartheta_i, \dots$ denote distribution terms in $\mathbb{DT}(\Sigma)$ ranging over $\mathcal{V}_d \cup \delta_{\mathcal{V}_s}$. Then, for arbitrary sets $S_1,\dots,S_n$, we denote by $\bigtimes_{i=1}^{n} S_i$ the set of tuples $k = [s_1,\dots,s_n]$ with $s_i \in S_i$. The i-th element of k is denoted $k(i)$.

▶ **Definition 12** (Σ-distribution rules)**.** Assume a signature Σ. The set R_Σ of the Σ-*distribution rules* consists of the least set containing the following inference rules:

1. $\{\delta_x \xrightarrow{1} x\}$ for any state variable $x \in \mathcal{V}_s$;

2. $$\frac{\bigcup_{i=1,\dots,\mathfrak{n}} \left\{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i, \sum_{j\in J_i} q_{i,j} = 1\right\}}{\left\{f(\vartheta_1,\dots,\vartheta_\mathfrak{n}) \xrightarrow{q_k} f(x_{1,k(1)},\dots,x_{\mathfrak{n},k(\mathfrak{n})}) \mid q_k = \prod_{i=1,\dots,\mathfrak{n}} q_{i,k(i)},\, k \in \bigtimes_{i=1,\dots,\mathfrak{n}} J_i\right\}}$$
where $f \in \Sigma$, the distribution terms $\vartheta_i \in \mathcal{V}_d \cup \delta_{\mathcal{V}_s}$ are all distinct, and for each $i = 1,\dots,\mathfrak{n}$ the state variables $x_{i,j}$'s with $j \in J_i$ are pairwise distinct;

3. $$\frac{\bigcup_{i\in I} \left\{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i, \sum_{j\in J_i} q_{i,j} = 1\right\}}{\left\{\sum_{i\in I} p_i\vartheta_i \xrightarrow{q_x} x \mid q_x = \sum_{i\in I, j\in J_i \text{ s.t. } x_{i,j}=x} p_i \cdot q_{i,j} \text{ and } x \in \{x_{i,j} \mid j \in J_i, i \in I\}\right\}}$$
where I is an at most countable set of indexes, the distribution terms $\vartheta_i \in \mathcal{V}_d \cup \delta_{\mathcal{V}_s}$ are all distinct, and for each $i \in I$ the state variables $x_{i,j}$'s with $j \in J_i$ are pairwise distinct.

Then, the Σ-*distribution specification* (Σ-*DS*) is the pair $D_\Sigma = (\Sigma, R_\Sigma)$.

For each Σ-distribution rule r_D, all sets above the line are called *premises*, notation $\mathrm{prem}(r_\mathsf{D})$, and the set below the line is called *conclusion*, notation $\mathrm{conc}(r_\mathsf{D})$. It is not hard to see that all premises and the conclusion are distributions over terms.

▶ **Example 13.** An example of Σ-distribution rule with source $\mu|\nu$ is the following:

$$\frac{\{\mu \xrightarrow{1/4} x_1, \quad \mu \xrightarrow{3/4} x_2\} \quad \{\nu \xrightarrow{1/3} y_1, \quad \nu \xrightarrow{2/3} y_2\}}{\{\mu|\nu \xrightarrow{1/12} x_1|y_1, \quad \mu|\nu \xrightarrow{1/6} x_1|y_2, \quad \mu|\nu \xrightarrow{1/4} x_2|y_1, \quad \mu|\nu \xrightarrow{1/2} x_2|y_2\}}\,.$$

The following notion of reduction wrt. a substitution allows us to extend the notion of substitution to distributions over terms and, then, to Σ-distribution rules.

▶ **Definition 14** (Reduction wrt. a substitution)**.** Assume a substitution σ and a distribution over terms $L = \{\Theta \xrightarrow{q_i} t_i \mid i \in I\}$. We say that σ *reduces* L to the set of distribution literals $L' = \{\sigma(\Theta) \xrightarrow{q_j} t_j \mid j \in J\}$, or that L' is the *reduction wrt.* σ of L, notation $\sigma(L) = L'$, if:

- for each index $j \in J$ there is at least one index $i \in I$ with $\sigma(t_i) = t_j$;
- the terms $\{t_j \mid j \in J\}$ are pairwise distinct;
- for each index $j \in J$, we have $q_j = \sum_{\{i\in I \mid \sigma(t_i)=t_j\}} q_i$.

▶ **Proposition 15.** *For a substitution σ and a distribution over terms L, the set of distribution literals $\sigma(L)$ is a distribution over terms.*

▶ **Definition 16** (Reduced instance of a Σ-distribution rule)**.** The *reduced instance* of a Σ-distribution rule r_D wrt. a substitution σ is the inference rule $\sigma(r_\mathsf{D})$ s.t.:

1. If r_D is as in Def. 12.1, then $\sigma(r_\mathsf{D}) = \{\delta_{\sigma(x)} \xrightarrow{1} \sigma(x)\}$.
2. If r_D is as in Def. 12.2, then
$$\sigma(r_\mathsf{D}) = \frac{\bigcup_{i=1,\dots,\mathfrak{n}} \{\sigma(\vartheta_i) \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}}{\left\{f(\sigma(\vartheta_1),\dots,\sigma(\vartheta_\mathfrak{n})) \xrightarrow{q_\kappa} f(t_{1,\kappa(1)},\dots,t_{\mathfrak{n},\kappa(\mathfrak{n})}) \mid q_\kappa = \prod_{i=1,..,\mathfrak{n}} q_{i,\kappa(i)},\, \kappa \in \bigtimes_{i=1,..,\mathfrak{n}} H_i\right\}}$$
where $\{\sigma(\vartheta_i) \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\} = \sigma(\{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i\})$.
3. If r_D is as in Def. 12.3, then
$$\sigma(r_\mathsf{D}) = \frac{\bigcup_{i\in I} \{\sigma(\vartheta_i) \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}}{\left\{\sum_{i\in I} p_i\sigma(\vartheta_i) \xrightarrow{q_t} t \mid q_t = \sum_{i\in I, h\in H_i \text{ s.t. } t_{i,h}=t} p_i \cdot q_{i,h},\, t \in \{t_{i,h} \mid h \in H_i, i \in I\}\right\}}$$

where $\{\sigma(\vartheta_i) \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\} = \sigma(\{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i\})$.

Notice that Prop. 15 ensures that the premises of $\sigma(r_{\mathsf{D}})$ are distributions over terms. Moreover, it is not hard to see that also the conclusion of $\sigma(r_{\mathsf{D}})$ is a distribution over terms.

▶ **Definition 17** (Proof from the Σ-DS)**.** A *proof* from the Σ-DS D_Σ of a closed distribution over terms L is a well-founded, upwardly branching tree, whose nodes are labeled by closed distributions over terms, s.t. the root is labeled L, and, if β is the label of a node $\mathfrak{q}$ and $\mathcal{K}$ is the set of labels of the nodes directly above $\mathfrak{q}$, then $\mathcal{K}/\beta$ is a closed reduced instance of a Σ-distribution rule in R_Σ.

A closed distribution over terms L is *provable* from D_Σ, notation $D_\Sigma \vdash L$, if there exists a proof from D_Σ for L.

Since Σ-distribution rules have only positive premises, the set of the distribution over terms provable from the Σ-DS is unique. The following result confirms that all probability distributions over $\mathcal{T}(\Sigma)$ can be inferred through the Σ-DS.

▶ **Proposition 18.** *Assume a signature* Σ*. Let* $\pi \in \mathcal{DT}(\Sigma)$ *be a closed distribution term and* $\{t_m\}_{m\in M} \subseteq \mathcal{T}(\Sigma)$ *a set of pairwise distinct closed terms. Then*

$$D_\Sigma \vdash \{\pi \xrightarrow{q_m} t_m \mid m \in M\} \Leftrightarrow \textit{for all } m \in M \textit{ it holds } \pi(t_m) = q_m, \textit{ and } \sum_{m\in M} q_m = 1.$$

4 The decomposition method

In this section we present our method for decomposing formulae in $\mathcal{L}$, $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{+}$. Our aim is to reduce the satisfaction problem of a formula for a (distribution) term to the satisfaction problem of derived formulae for its subterms. In Sec. 4.1 we define *ruloids* and *distribution ruloids*, namely derived (distribution) rules allowing us to infer the behavior of any (distribution) term from the behavior of its variables. Both classes of ruloids are *sound and specifically witnessing* [3], i.e. a closed literal α (resp. a distribution over terms L) is provable from a PGSOS-PTSS (resp. the Σ-DS) iff α (resp. L) is an instance of the conclusion of a ruloid (resp. distribution ruloid) (Thm. 21 and Thm. 24). Then, in Sec. 4.2 we exploit the two classes of ruloids for the decomposition. The decomposition of state formulae follows [2, 12–15] and consists in assigning to each term $t \in \mathbb{T}(\Sigma)$ and formula $\varphi \in \mathcal{L}^{\mathrm{s}}$, a set of functions $\xi\colon \mathcal{V}_s \to \mathcal{L}^{\mathrm{s}}$, called *decomposition mappings*, assigning to each variable x in t a proper formula in $\mathcal{L}^{\mathrm{s}}$ s.t. for any closed substitution σ it holds that $\sigma(t) \models \varphi$ iff $\sigma(x) \models \xi(x)$ for each $x \in \mathrm{var}(t)$ (Thm. 28). Each mapping ξ is defined on a ruloid having t as source. The decomposition of distribution formulae consists in assigning to each distribution term $\Theta \in \mathbb{DT}(\Sigma)$ and distribution formula $\psi \in \mathcal{L}^{\mathrm{d}}$ a set of decomposition mappings $\eta\colon \mathcal{V} \to \mathcal{L}^{\mathrm{d}} \cup \mathcal{L}^{\mathrm{s}}$ s.t. for any closed substitution σ we get that $\sigma(\Theta) \models \psi$ iff $\sigma(\zeta) \models \eta(\zeta)$ for each $\zeta \in \mathrm{var}(\Theta)$ (Thm. 28).

4.1 Ruloids

Ruloids are defined by an inductive composition of PGSOS rules. All PGSOS rules are ruloids. Then, from a rule r and substitution σ, a ruloid ρ with conclusion $\sigma(\mathrm{conc}(r))$ is built as follows:

1. for each positive premise α in $\sigma(r)$, we take any ruloid having α as conclusion and we put its premises among the premises of ρ;

2. for each negative premise α in $\sigma(r)$ and for each ruloid ρ' having any literal denying α as conclusion, we select any premise β of ρ', we take any literal β' denying β, and we put β' among the premises of ρ.

For a PGSOS-PTSS $P = (\Sigma, \mathcal{A}, R)$, let $\mathrm{Lit}(P)$ denote the set of literals that can be built with terms in $\mathbb{T}(\Sigma) \cup \mathbb{DT}(\Sigma)$ and actions in $\mathcal{A}$.

▶ **Definition 19** (Ruloids). Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS. The set of *P-ruloids* $\Re^P$ is the smallest set s.t.:

- $\dfrac{x \xrightarrow{a} \mu}{x \xrightarrow{a} \mu}$ is a P-ruloid for all $x \in \mathcal{V}_s$, $a \in \mathcal{A}$ and $\mu \in \mathcal{V}_d$;
- $\dfrac{\bigcup_{i \in I} \left(\bigcup_{m \in M_i} \mathcal{H}_{i,m} \cup \bigcup_{n \in N_i} \mathcal{H}_{i,n} \right)}{f(t_1, \ldots, t_{\mathfrak{n}}) \xrightarrow{a} \Theta}$ is a P-ruloid if there is a PGSOS rule $r \in R$

$$\frac{\{x_i \xrightarrow{a_{i,m}} \mu_{i,m} \mid i \in I, m \in M_i\} \qquad \{x_i \xrightarrow{a_{i,n}} \not\rightarrow \mid i \in I, n \in N_i\}}{f(x_1, \ldots, x_{\mathfrak{n}}) \xrightarrow{a} \Theta'}$$

together with a substitution σ, with $\sigma(x_i) = t_i$ for $i = 1, \ldots, \mathfrak{n}$ and $\sigma(\Theta') = \Theta$, s.t.:

- for every positive premise $x_i \xrightarrow{a_{i,m}} \mu_{i,m}$ of r
  - either $\sigma(x_i)$ is a variable and $\mathcal{H}_{i,m} = \{\sigma(x_i) \xrightarrow{a_{i,m}} \sigma(\mu_{i,m})\}$,
  - or there is a P-ruloid $\rho_{i,m} = \mathcal{H}_{i,m} / \sigma(x_i) \xrightarrow{a_{i,m}} \sigma(\mu_{i,m})$;
- for every negative premise $x_i \xrightarrow{a_{i,n}} \not\rightarrow$ of r
  - either $\sigma(x_i)$ is a variable and $\mathcal{H}_{i,n} = \{\sigma(x_i) \xrightarrow{a_{i,n}} \not\rightarrow\}$,
  - or $\mathcal{H}_{i,n} = \mathrm{opp}(\mathrm{pick}(\Re^P_{(a_{i,n})}))$, where:
    - **(i)** define $\Re^P_{(a_{i,n})} \in \mathcal{P}(\mathcal{P}(\mathrm{Lit}(P)))$ as the set containing the sets of the premises of all P-ruloids with conclusion $\sigma(x_i) \xrightarrow{a_{i,n}} \theta$, formally

$$\Re^P_{(a_{i,n})} = \{\mathrm{prem}(\rho) \mid \rho \in \Re^P \text{ and } \mathrm{conc}(\rho) = \sigma(x_i) \xrightarrow{a_{i,n}} \theta \text{ for some } \theta \in \mathbb{DT}(\Sigma)\},$$

    - **(ii)** define any mapping $\mathrm{pick} \colon \mathcal{P}(\mathcal{P}(\mathrm{Lit}(P))) \to \mathcal{P}(\mathrm{Lit}(P))$ s.t. for any set of literals L_k with $k \in K$, $\mathrm{pick}(\{L_k \mid k \in K\}) = \{l_k \mid k \in K \wedge l_k \in L_k\}$,
    - **(iii)** define any mapping $\mathrm{opp} \colon \mathcal{P}(\mathrm{Lit}(P)) \to \mathcal{P}(\mathrm{Lit}(P))$ satisfying $\mathrm{opp}(L) = \{\mathrm{opp}(l) \mid l \in L\}$ for all set of literals L, where $\mathrm{opp}(t' \xrightarrow{a} \theta) = t' \xrightarrow{a} \not\rightarrow$, and $\mathrm{opp}(t' \xrightarrow{a} \not\rightarrow) = t' \xrightarrow{a} \theta$ for some fresh distribution term θ;
- right hand side variables $\mathrm{rhs}(\rho_{i,m})$ are all pairwise disjoint.

▶ **Example 20.** From rules in Ex. 10, we can build the following ruloids for term $x +_p (y|z)$:

$$\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \not\rightarrow}{x +_p (y|z) \xrightarrow{a} \mu} \quad \frac{x \xrightarrow{a} \mu \quad z \xrightarrow{a} \not\rightarrow}{x +_p (y|z) \xrightarrow{a} \mu} \quad \frac{x \xrightarrow{a} \not\rightarrow \quad y \xrightarrow{a} \nu \quad z \xrightarrow{a} \upsilon}{x +_p (y|z) \xrightarrow{a} \nu|\upsilon} \quad \frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu \quad z \xrightarrow{a} \upsilon}{x +_p (y|z) \xrightarrow{a} p\mu + (1-p)(\nu|\upsilon)}.$$

We describe the construction of the first ruloid: $\dfrac{\dfrac{x \xrightarrow{a} \mu}{x \xrightarrow{a} \mu} \qquad \dfrac{y \xrightarrow{a} \not\rightarrow}{y|z \xrightarrow{a} \not\rightarrow}}{x +_p (y|z) \xrightarrow{a} \mu}$.

It is not hard to see that if the PTSS is positive then also the derived ruloids are positive.

▶ **Theorem 21** (Ruloid theorem). *Assume a PGSOS-PTSS P and a closed substitution σ. Then $P \vdash \sigma(t) \xrightarrow{a} \Theta'$ for $t \in \mathbb{T}(\Sigma)$ and $\Theta' \in \mathcal{DT}(\Sigma)$ if and only if there are a P-ruloid $\dfrac{\mathcal{H}}{t \xrightarrow{a} \Theta}$ and a closed substitution σ' with $P \vdash \sigma'(\mathcal{H})$, $\sigma'(t) = \sigma(t)$ and $\sigma'(\Theta) = \Theta'$.*

As the Σ-DS is positive, the definition of Σ-*distribution ruloids* results technically simpler.

▶ **Definition 22** (Distribution ruloids). Let $D_\Sigma = (\Sigma, R_\Sigma)$ be the Σ-DS. The set of Σ-*distribution ruloids* $\Re^\Sigma$ is the smallest set s.t.:

- $\dfrac{\{\delta_x \xrightarrow{1} x\}}{\{\delta_x \xrightarrow{1} x\}}$ is a Σ-distribution ruloid for any $x \in \mathcal{V}_s$;
- $\dfrac{\{\mu \xrightarrow{q_i} x_i \mid \sum_{i\in I} q_i = 1\}}{\{\mu \xrightarrow{q_i} x_i \mid i \in I\}}$ is a Σ-distribution ruloid for any $\mu \in \mathcal{V}_d$;
- $\dfrac{\bigcup_{i=1,\dots,\mathfrak{n}} \mathcal{H}_i}{\left\{f(\Theta_1,\dots,\Theta_\mathfrak{n}) \xrightarrow{Q_m} f(t_{1,m},\dots,t_{\mathfrak{n},m}) \mid m \in M\right\}}$ is a Σ-distribution ruloid if there is a Σ-distribution rule $r_\mathsf{D} \in R_\Sigma$ of the form

$$\frac{\bigcup_{i=1,\dots,\mathfrak{n}} \{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i, \sum_{j\in J_i} q_{i,j} = 1\}}{\left\{f(\vartheta_1,\dots,\vartheta_\mathfrak{n}) \xrightarrow{q_k} f(x_{1,k(1)},\dots,x_{\mathfrak{n},k(\mathfrak{n})}) \mid q_k = \prod_{i=1,\dots,\mathfrak{n}} q_{i,k(i)},\ k \in \bigtimes_{i=1,\dots,\mathfrak{n}} J_i\right\}}$$

together with a substitution σ, with $\sigma(\vartheta_i) = \Theta_i$ for $i = 1,\dots,n$, s.t.:

- $\sigma(r_\mathsf{D}) = \dfrac{\bigcup_{i=1,\dots,\mathfrak{n}} \{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i, \sum_{h\in H_i} q_{i,h} = 1\}}{\left\{f(\Theta_1,\dots,\Theta_\mathfrak{n}) \xrightarrow{q_\kappa} f(t_{1,\kappa(1)},..,t_{\mathfrak{n},\kappa(\mathfrak{n})}) \mid q_\kappa = \prod_{i=1,..,\mathfrak{n}} q_{i,\kappa(i)},\ \kappa \in \bigtimes_{i=1,..,\mathfrak{n}} H_i\right\}}$,
- there is a bijection $\mathfrak{f}\colon \bigtimes_{i=1}^{\mathfrak{n}} H_i \to M$ such that $t_{i,\kappa(i)} = t_{i,\mathfrak{f}(\kappa)}$ and $q_\kappa = Q_{\mathfrak{f}(\kappa)}$,
- for every Θ_i with $i = 1,\dots,\mathfrak{n}$ we have that:
  - either Θ_i is a variable or a Dirac distribution and $\mathcal{H}_i = \{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}$,
  - or there is a Σ-distribution ruloid $\rho_i^\mathsf{D} = \mathcal{H}_i / \{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}$;
- $\dfrac{\bigcup_{i\in I} \mathcal{H}_i}{\left\{\sum_{i\in I} p_i\Theta_i \xrightarrow{Q_m} t_m \mid m \in M\right\}}$ is a Σ-distribution ruloid if there is a Σ-distribution rule $r_\mathsf{D} \in R_\Sigma$ of the form

$$\frac{\bigcup_{i\in I}\{\vartheta_i \xrightarrow{q_{i,j}} x_{i,j} \mid j \in J_i, \sum_{j\in J_i} q_{i,j} = 1\}}{\left\{\sum_{i\in I} p_i\vartheta_i \xrightarrow{q_x} x \mid q_x = \sum_{i\in I, j\in J_i \text{ s.t. } x_{i,j}=x} p_i \cdot q_{i,j},\ x \in \{x_{i,j} \mid j \in J_i, i \in I\}\right\}}$$

together with a substitution σ, with $\sigma(\vartheta_i) = \Theta_i$ for $i \in I$, s.t.:

- $\sigma(r_\mathsf{D}) = \dfrac{\bigcup_{i\in I}\{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i, \sum_{h\in H_i} q_{i,h} = 1\}}{\left\{\sum_{i\in I} p_i\Theta_i \xrightarrow{q_u} u \mid q_u = \sum_{i\in I, h\in H_i \text{ s.t. } t_{i,h}=u} p_i \cdot q_{i,h},\ u \in \{t_{i,h} \mid h \in H_i, i \in I\}\right\}}$
- there is a bijection $\mathfrak{f}\colon \{t_{i,h} \mid h \in H_i, i \in I\} \to M$ s.t. $u = t_{\mathfrak{f}(u)}$ and $q_u = Q_{\mathfrak{f}(u)}$,
- for every Θ_i with $i \in I$ we have that:
  - either Θ_i is a variable or a Dirac distribution and $\mathcal{H}_i = \{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}$,
  - or there is a Σ-distribution ruloid $\rho_i^\mathsf{D} = \mathcal{H}_i / \{\Theta_i \xrightarrow{q_{i,h}} t_{i,h} \mid h \in H_i\}$.

▶ **Example 23.** Consider the distribution term $\frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon)$, which is an instance of the target of the fourth ruloid in Ex. 20. Then, we can build the following Σ-distribution ruloid:

$$\dfrac{\dfrac{\{\mu \xrightarrow{1/4} x_1 \quad \mu \xrightarrow{3/4} x_2\}}{\{\mu \xrightarrow{1/4} x_1 \quad \mu \xrightarrow{3/4} x_2\}} \qquad \dfrac{\{\nu \xrightarrow{1/3} y_1, \quad \nu \xrightarrow{2/3} y_2\} \quad \{\upsilon \xrightarrow{1} w\}}{\{\nu|\upsilon \xrightarrow{1/3} y_1|w \quad \nu|\upsilon \xrightarrow{2/3} y_2|w\}}}{\left\{\frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon) \xrightarrow{\frac{1}{10}} x_1, \frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon) \xrightarrow{\frac{3}{10}} x_2, \frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon) \xrightarrow{\frac{1}{5}} y_1|w, \frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon) \xrightarrow{\frac{2}{5}} y_2|w\right\}}$$

▶ **Theorem 24** (Distribution ruloid theorem)**.** *Assume the Σ-DS D_Σ and a closed substitution σ. Then $D_\Sigma \vdash \{\sigma(\Theta) \xrightarrow{q_m} t_m \mid m \in M\}$ for $\Theta \in \mathbb{DT}(\Sigma)$ and $t_m \in \mathcal{T}(\Sigma)$ pairwise distinct if and only if there are a Σ-distribution ruloid $\dfrac{\mathcal{H}}{\{\Theta \xrightarrow{q_m} u_m \mid m \in M\}}$ and a closed substitution σ' with $D_\Sigma \vdash \sigma'(\mathcal{H})$, $\sigma'(\Theta) = \sigma(\Theta)$ and $\sigma'(u_m) = t_m$ for each $m \in M$.*

Although the construction of our ruloids resembles that in [15], the two classes are quite different. [15] bases on the rule format of [20] instead of the PGSOS format of [5], deals with reactive systems, which are less expressive than PTSs since they do not admit internal nondeterminism, and considers transitions of the form $t \xrightarrow{a,p} t'$, denoting that t evolves by a to t' with probability p. Informally, our ruloids generalize those in [15] in the same way PTSs generalize reactive systems. In fact, to deal with $t \xrightarrow{a,p} t'$, ruloids in [15] are defined by keeping track of rules and ruloids used in their construction, in order to obtain a partitioning over ruloids ensuring that the probabilities of all a-labeled transitions from a term t sum up to either 0 or 1. Here we do not need this technicality, since, given a term t, all ruloids in one partition for t of [15] are captured by one of our ruloids and one Σ-distribution ruloid. Our ruloid captures all the requirements that the subterms of t must satisfy to derive the transition to the desired distribution over terms. The proper probability weights are then assigned by the Σ-distribution ruloid.

4.2 Decomposition of modal formulae

First we need to introduce the notion of *matching* for a distribution term, seen as a probability distribution over terms, and a distribution formula, which can be viewed as a probability distribution over state formulae [4, 7].

▶ **Definition 25** (Matching)**.** Assume $\Theta \in \mathbb{DT}(\Sigma)$, a Σ-distribution ruloid $\mathcal{H}/\{\Theta \xrightarrow{q_m} t_m \mid m \in M\}$ and a distribution formula $\psi = \bigoplus_{i\in I} r_i\varphi_i \in \mathcal{L}^{\mathrm{d}}$. Then a *matching* for Θ and ψ is a distribution over the product space $\mathfrak{w} \in \Delta(\mathbb{T}(\Sigma) \times \mathcal{L}^{\mathrm{s}})$ having Θ and ψ as left and right marginals, that is $\sum_{i\in I} \mathfrak{w}(t_m, \varphi_i) = q_m$ for all $m \in M$ and $\sum_{m\in M} \mathfrak{w}(t_m, \varphi_i) = r_i$ for all $i \in I$. We denote by $\mathfrak{W}(\Theta, \psi)$ the set of all matchings for Θ and ψ.

▶ **Definition 26** (Decomposition of $\mathcal{L}$)**.** Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS and let D_Σ be the Σ-DS. We define the mapping $\cdot^{-1}\colon \mathbb{T}(\Sigma) \to (\mathcal{L}^{\mathrm{s}} \to \mathcal{P}(\mathcal{V}_s \to \mathcal{L}^{\mathrm{s}}))$ as the function that for each $t \in \mathbb{T}(\Sigma)$ and $\varphi \in \mathcal{L}^{\mathrm{s}}$ returns the set $t^{-1}(\varphi) \in \mathcal{P}(\mathcal{V}_s \to \mathcal{L}^{\mathrm{s}})$ of *decomposition mappings* $\xi\colon \mathcal{V}_s \to \mathcal{L}^{\mathrm{s}}$ generated as follows. Let t denote an univariate term. Then:

1. $\xi \in t^{-1}(\top)$ iff $\xi(x) = \top$ for all $x \in \mathcal{V}_s$;
2. $\xi \in t^{-1}(\neg\varphi)$ iff there is a function $\mathfrak{f}\colon t^{-1}(\varphi) \to \mathrm{var}(t)$ s.t.

$$\xi(x) = \bigwedge_{\xi' \in \mathfrak{f}^{-1}(x)} \neg\xi'(x), \text{ if } x \in \mathrm{var}(t), \text{ and } \xi(x) = \top, \text{ otherwise;}$$

3. $\xi \in t^{-1}(\bigwedge_{j\in J} \varphi_j)$ iff there exist decomposition mappings $\xi_j \in t^{-1}(\varphi_j)$, for $j \in J$, s.t.

$$\xi(x) = \bigwedge_{j\in J} \xi_j(x) \text{ for all } x \in \mathcal{V}_s;$$

4. $\xi \in t^{-1}(\langle a \rangle \psi)$ iff there are a P-ruloid $\frac{\mathcal{H}}{t \xrightarrow{a} \Theta}$ and a decomposition mapping $\eta \in \Theta^{-1}(\psi)$ s.t.

$$\xi(x) = \bigwedge_{x \xrightarrow{b} \mu \in \mathcal{H}} \langle b \rangle \eta(\mu) \wedge \bigwedge_{x \xrightarrow{c} \not\in \mathcal{H}} \neg \langle c \rangle \top \wedge \eta(x), \text{ if } x \in \mathrm{var}(t), \text{ and } \xi(x) = \top, \text{ otherwise;}$$

5. $\xi \in (\sigma(t))^{-1}(\varphi)$ for a non injective substitution $\sigma \colon \mathrm{var}(t) \to \mathcal{V}_s$ iff there is a decomposition mapping $\xi' \in t^{-1}(\varphi)$ s.t.

$$\xi(x) = \bigwedge_{y \in \sigma^{-1}(x)} \xi'(y) \quad \text{for all } x \in \mathcal{V}_s.$$

Then we define the mapping $\cdot^{-1} \colon \mathbb{DT}(\Sigma) \to (\mathcal{L}^{\mathrm{d}} \to \mathcal{P}(\mathcal{V} \to \mathcal{L}))$ as the function that for each $\Theta \in \mathbb{DT}(\Sigma)$ and $\psi \in \mathcal{L}^{\mathrm{d}}$ returns the set $\Theta^{-1}(\psi) \in \mathcal{P}(\mathcal{V} \to \mathcal{L})$ of *decomposition mappings* $\eta \colon \mathcal{V} \to \mathcal{L}$ generated as follows. Let Θ denote an univariate distribution term. Then:

6. $\eta \in \Theta^{-1}(\bigoplus_{i \in I} r_i \varphi_i)$ iff there are a Σ-distribution ruloid $\frac{\mathcal{H}}{\{\Theta \xrightarrow{q_m} t_m \mid m \in M\}}$ and a matching $\mathfrak{w} \in \mathfrak{W}(\Theta, \bigoplus_{i \in I} r_i \varphi_i)$ s.t. for all $m \in M$ and $i \in I$ there is a decomposition mapping $\xi_{m,i}$ with $\xi_{m,i} \in t_m^{-1}(\varphi_i)$, if $\mathfrak{w}(t_m, \varphi_i) > 0$, and $\xi_{m,i} \in t_m^{-1}(\top)$, otherwise, s.t.:
   a. for $\mu \in \mathcal{V}_d$ we have $\eta(\mu) = \begin{cases} \bigoplus_{\{\mu \xrightarrow{q_j} x_j \mid \sum_{j \in J} q_j = 1\} \in \mathcal{H}} q_j \bigwedge_{\substack{i \in I \\ m \in M}} \xi_{m,i}(x_j) & \text{if } \mu \in \mathrm{var}(\Theta) \\ 1\top & \text{otherwise} \end{cases}$
   b. for $x \in \mathcal{V}_s$ we have $\eta(x) = \begin{cases} \bigwedge_{i \in I, m \in M} \xi_{m,i}(x) & \text{if } x \in \mathrm{var}(\Theta) \\ \top & \text{otherwise.} \end{cases}$
7. $\eta \in (\sigma(\Theta))^{-1}(\psi)$ for a non injective substitution $\sigma \colon \mathrm{var}(\Theta) \to \mathcal{V}$ iff there is a decomposition mapping $\eta' \in \Theta^{-1}(\psi)$ s.t. for $\zeta \in \mathrm{var}(\sigma(\Theta))$ it holds $\eta'(z) = \eta'(z')$ for all $z, z' \in \sigma^{-1}(\zeta)$ and

$$\eta(\zeta) = \eta'(\tilde{z}) \text{ if } \zeta \in \mathrm{var}(\sigma(\Theta)) \text{ and } \tilde{z} \in \sigma^{-1}(\zeta), \text{ and } \eta(\zeta) = \top \text{ if } \zeta \notin \mathrm{var}(\sigma(\Theta)).$$

We discuss only the decomposition of $\psi = \bigoplus_{i \in I} r_i \varphi_i \in \mathcal{L}^{\mathrm{d}}$. Let σ be a closed substitution and consider $\Theta \in \mathbb{DT}(\Sigma)$. We have $\sigma(\Theta) \models \psi$ iff $\sigma(\Theta) = \sum_{i \in I} r_i \pi_i$ with $t \models \varphi_i$ for all $t \in \mathsf{supp}(\pi_i)$. So, we need to identify which properties each $\sigma(\zeta)$ with $\zeta \in \mathrm{var}(\Theta)$ must satisfy to guarantee that $\sigma(\Theta)$ is such a distribution $\sum_{i \in I} r_i \pi_i$. Assume $\mathsf{supp}(\sigma(\Theta)) = \{t_m \mid m \in M\}$ and $\sigma(\Theta)(t_m) = q_m$. By Prop. 18, this is equivalent to have $D_\Sigma \vdash \{\sigma(\Theta) \xrightarrow{q_m} t_m \mid m \in M\}$. From Thm. 24, $D_\Sigma \vdash \{\sigma(\Theta) \xrightarrow{q_m} t_m \mid m \in M\}$ iff there are a Σ-distribution ruloid $\mathcal{H}/\{\Theta \xrightarrow{q_m} u_m \mid m \in M\}$ and a closed substitution σ' with $\sigma'(\Theta) = \sigma(\Theta)$, $\sigma'(u_m) = t_m$ and $\mathsf{D}_\Sigma \vdash \sigma'(\mathcal{H})$. Since the weights q_m are univocally determined by the distributions over terms in $\mathcal{H}$, we can define, for each $\mu \in \mathrm{var}(\Theta) \cap \mathcal{V}_d$, $\eta(\mu)$ using as weights the q_j in $\{\mu \xrightarrow{q_j} x_j \mid \sum_{j \in J} q_j = 1\} \in \mathcal{H}$. Finally, to ensure that if $\sigma'(u_m) \in \mathsf{supp}(\pi_i)$, then $\sigma'(u_m) \models \varphi_i$, we define $\mathfrak{w}(u_m, \varphi_i)$ positive if $\sigma'(u_m) \in \mathsf{supp}(\pi_i)$ so that we can assign the proper decomposed formula $\xi_{m,i}(x)$ to each $x \in \mathrm{var}(u_m)$. Since each $\sigma'(u_m)$ may occur in the support of more than one π_i, we impose that each $x \in \mathrm{var}(u_m)$ satisfies the conjunction of all the decomposed formulae $\xi_{m,i}(x)$.

▶ **Example 27.** We exemplify two mappings in $t^{-1}(\varphi)$ for $\varphi = \langle a \rangle \psi$, with $\psi = \frac{1}{2}\langle a \rangle \top \oplus \frac{1}{2} \neg \langle a \rangle \top$, and $t = x +_{2/5} (y|z)$, which is the term in Ex. 20 with $p = 2/5$. Let ρ be the last

Table 1 Derived decomposition mappings.

$x_1^{-1}(\langle a\rangle\top) = \{\xi_1\}$	$\xi_1(x_1) = \langle a\rangle\top$, $\xi_1(x) = \top$ for all other variables
$x_2^{-1}(\neg\langle a\rangle\top) = \{\xi_2\}$	$\xi_2(x_2) = \neg\langle a\rangle\top$, $\xi_2(x) = \top$ for all other variables
$(y_1\|w)^{-1}(\neg\langle a\rangle\top) = \{\xi_3, \xi_4\}$	$\xi_3(y_1) = \neg\langle a\rangle\top \quad \xi_3(w) = \top$, $\xi_3(x) = \top$ for all other variables $\xi_4(y_1) = \top \quad \xi_4(w) = \neg\langle a\rangle\top$, $\xi_4(x) = \top$ for all other variables
$(y_2\|w)^{-1}(\langle a\rangle\top) = \{\xi_5\}$	$\xi_5(y_2) = \langle a\rangle\top \quad \xi_5(w) = \langle a\rangle\top$, $\xi_5(x) = \top$ for all other variables

ruloid for t in Ex. 20, $\Theta = \frac{2}{5}\mu + \frac{3}{5}(\nu|\upsilon)$ denote its target, and ρ^D be the Σ-distribution ruloid for Θ in Ex. 23. By Def. 26.4, the decomposition mappings $\xi \in t^{-1}(\varphi)$ built over ρ are s.t.:

$$\xi(x) = \langle a\rangle\eta(\mu) \qquad \xi(y) = \langle a\rangle\eta(\nu) \qquad \xi(z) = \langle a\rangle\eta(\upsilon) \tag{1}$$

where $\eta \in \Theta^{-1}(\psi)$. Consider the matching $\mathfrak{w} \in \mathfrak{W}(\Theta, \psi)$ for Θ and ψ defined through ρ^D by

$$\mathfrak{w}(x_1, \langle a\rangle\top) = 1/10 \quad \mathfrak{w}(x_2, \neg\langle a\rangle\top) = 3/10 \quad \mathfrak{w}(y_1|w, \neg\langle a\rangle\top) = 1/5 \quad \mathfrak{w}(y_2|w, \langle a\rangle\top) = 2/5.$$

For the terms and the formulae to which $\mathfrak{w}$ gives a positive weight, we obtain the decomposition mappings in Tab. 1, where ξ_3 and ξ_4 derive from Def. 26.2.

Next, we construct the decomposition mappings for the variable ν in Θ wrt. ρ^{D} and $\mathfrak{w}$. By Def. 26.6a we consider the weights of the premises of ρ^D having ν as left-hand side, namely $\mathcal{H}_\nu = \{\nu \xrightarrow{1/3} y_1, \quad \nu \xrightarrow{2/3} y_2\}$, and use them as weights of the $\bigoplus$ operator. Then for the variables y_1, y_2 in the right side of $\mathcal{H}_\nu$, we consider the conjunction of the formulae assigned to y_1, y_2 by one mapping from each set in the first column of Tab. 1. The choice of ξ_3 or ξ_4 generates two different mappings in $\Theta^{-1}(\psi)$: by ξ_3 we obtain the mapping $\eta_1 \in \Theta^{-1}(\psi)$ with $\eta_1(\nu) = 1/3\neg\langle a\rangle\top \oplus 2/3\langle a\rangle\top$ and by ξ_4 we obtain the mapping $\eta_2 \in \Theta^{-1}(\psi)$ with $\eta_2(\nu) = 1/3\top \oplus 2/3\langle a\rangle\top$. By applying the same reasoning to μ and υ we obtain

$$\eta_1(\mu) = 1/4\langle a\rangle\top \oplus 3/4\neg\langle a\rangle\top \quad \eta_1(\nu) = 1/3\neg\langle a\rangle\top \oplus 2/3\langle a\rangle\top \quad \eta_1(\upsilon) = 1(\top \wedge \langle a\rangle\top)$$

$$\eta_2(\mu) = 1/4\langle a\rangle\top \oplus 3/4\neg\langle a\rangle\top \quad \eta_2(\nu) = 1/3\top \oplus 2/3\langle a\rangle\top \quad \eta_2(\upsilon) = 1(\neg\langle a\rangle\top \wedge \langle a\rangle\top)$$

where we have omitted multiple occurrences of the $\top$ formulae in conjunctions. Finally, we obtain two mappings in $t^{-1}(\varphi)$ by substituting η with either η_1 or η_2 in Eq. (1).

The following result confirms that our decomposition method is correct.

▶ **Theorem 28** (Decomposition theorem). *Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS and let D_Σ be the Σ-DS. For any $t \in \mathbb{T}(\Sigma)$, closed substitution σ and $\varphi \in \mathcal{L}^{\mathrm{s}}$ we have*

$$\sigma(t) \models \varphi \Leftrightarrow \exists \xi \in t^{-1}(\varphi) \text{ s.t. for all } x \in \mathrm{var}(t) \text{ it holds } \sigma(x) \models \xi(x)$$

and for any $\Theta \in \mathbb{DT}(\Sigma)$, closed substitution σ and $\psi \in \mathcal{L}^{\mathrm{d}}$ we have

$$\sigma(\Theta) \models \psi \Leftrightarrow \exists \eta \in \Theta^{-1}(\psi) \text{ s.t. for all } \zeta \in \mathrm{var}(\Theta) \text{ it holds } \sigma(\zeta) \models \eta(\zeta).$$

The decompositions of formulae in $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_+$ can be derived from the one for $\mathcal{L}$.

▶ **Definition 29** (Decomposition of $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_+$). Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS and D_Σ be the Σ-DS. The mappings $\cdot^{-1}\colon \mathbb{T}(\Sigma) \to (\mathcal{L}_{\mathrm{r}}^{\mathrm{s}} \to \mathcal{P}(\mathcal{V}_s \to \mathcal{L}_{\mathrm{r}}^{\mathrm{s}}))$ and $\cdot^{-1}\colon \mathbb{DT}(\Sigma) \to (\mathcal{L}_{\mathrm{r}}^{\mathrm{d}} \to \mathcal{P}(\mathcal{V} \to \mathcal{L}_{\mathrm{r}}))$ are obtained as in Def. 26 by rewriting Def. 26.2 and Def. 26.4, resp., by

2. $\xi \in t^{-1}(\bar{a})$ iff there is a function $\mathfrak{f}\colon t^{-1}(\langle a\rangle\top) \to \mathrm{var}(t)$ s.t.

$$\xi(x) = \bigwedge_{\xi' \in \mathfrak{f}^{-1}(x)} \neg\xi'(x), \text{ if } x \in \mathrm{var}(t), \text{ and } \xi(x) = \top, \text{ otherwise;}$$

4. $\xi \in t^{-1}(\langle a\rangle\psi)$ iff there are a ruloid $\frac{\mathcal{H}}{t \xrightarrow{a} \Theta}$ and a decomposition mapping $\eta \in \Theta^{-1}(\psi)$ s.t.

$$\xi(x) = \bigwedge_{x \xrightarrow{b} \mu \in \mathcal{H}} \langle b\rangle\eta(\mu) \;\wedge\; \bigwedge_{x \xrightarrow{c} \not\in \mathcal{H}} \bar{c} \;\wedge\; \eta(x), \text{ if } x \in \mathrm{var}(t), \text{ and } \xi(x) = \top, \text{ otherwise.}$$

If P is positive, the mappings $\cdot^{-1}\colon \mathbb{T}(\Sigma) \to (\mathcal{L}_+^{\mathrm{s}} \to \mathcal{P}(\mathcal{V}_s \to \mathcal{L}_+^{\mathrm{s}}))$ and $\cdot^{-1}\colon \mathbb{DT}(\Sigma) \to (\mathcal{L}_+^{\mathrm{d}} \to \mathcal{P}(\mathcal{V} \to \mathcal{L}_+))$ are obtained as in Def. 26 by removing Def. 26.2 and by rewriting Def. 26.4 by

4. $\xi \in t^{-1}(\langle a\rangle\psi)$ iff there are a positive P-ruloid $\frac{\mathcal{H}}{t \xrightarrow{a} \Theta}$ and a decomposition mapping $\eta \in \Theta^{-1}(\psi)$ s.t.

$$\xi(x) = \bigwedge_{x \xrightarrow{b} \mu \in \mathcal{H}} \langle b\rangle\eta(\mu) \;\wedge\; \eta(x), \text{ if } x \in \mathrm{var}(t), \text{ and } \xi(x) = \top, \text{ otherwise.}$$

Notice that by decomposing formulae in $\mathcal{L}_{\mathrm{r}}$ (resp. $\mathcal{L}_+$) we get formulae in $\mathcal{L}_{\mathrm{r}}$ (resp. $\mathcal{L}_+$).

▶ **Theorem 30** (Decomposition theorem II). *Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS and D_Σ be the Σ-DS. Assume the decomposition mappings as in Definition 29. Then:*

- *The results in Theorem 28 hold for $\varphi \in \mathcal{L}_{\mathrm{r}}^{\mathrm{s}}$ and $\psi \in \mathcal{L}_{\mathrm{r}}^{\mathrm{d}}$.*
- *Moreover, if P is positive, then the results in Theorem 28 hold for $\varphi \in \mathcal{L}_+^{\mathrm{s}}$ and $\psi \in \mathcal{L}_+^{\mathrm{d}}$.*

4.3 Probabilistic bisimilarity as a congruence

To support the compositional reasoning, the congruence (resp. precongruence) property is required for any behavioral equivalence (resp. preorder) $\mathcal{R}$. It consists in verifying whether $f(t_1, \ldots, t_{\mathtt{n}}) \;\mathcal{R}\; f(t'_1, \ldots, t'_{\mathtt{n}})$ whenever $t_i \,\mathcal{R}\, t'_i$ for $i = 1, \ldots, \mathtt{n}$. In [5] it is proved that probabilistic bisimilarity is a congruence for all operators defined by a PGSOS-PTSS. We can restate this result as a direct consequence of the characterization result of [7] (Thm. 6) combined with our first decomposition result in Thm. 28. Then, by our characterization results in Thm. 8 and our decomposition results in Thm. 30 we can derive precongruence formats for both ready similarity and similarity.

▶ **Theorem 31.** *Let $P = (\Sigma, \mathcal{A}, R)$ be a PGSOS-PTSS. Then:*

1. *Probabilistic bisimilarity is a congruence for all operators defined by P;*
2. *Probabilistic ready similarity is a precongruence for all operators defined by P;*
3. *If P is positive, probabilistic similarity is a precongruence for all operators defined by P.*

5 Conclusions

We proposed distribution ruloids as a powerful tool supporting the decomposition of modalities over the PTS model. This allowed us to define modular proof systems for modal properties of probabilistic systems (Thms. 28, 30), from which we also derived congruence formats (Thm. 31). Our approach can be easily adapted to models with subdistributions.

We will continue this line of research as follows. We will apply our decomposition method to derive congruence formats for testing and trace equivalences. Next, we will use our decomposition method to systematically derive formats for bisimilarity metric [10,25], weak

metric semantics [11] and metric variants of branching bisimulation equivalence [1]. These metric semantics provide notions of *distance* over processes, and the formats will guarantee that a small variance in the behavior of the subprocesses leads to a bounded small variance in the behavior of the composed processes (*uniform continuity*, [16–18]). Then, we will study decomposition methods for real-valued modal formulae.

References

1 S. Andova and T.A.C. Willemse. Branching bisimulation for probabilistic systems: characteristics and decidability. *Theoret. Comput. Sci.*, 356(3):325–355, 2006.

2 B. Bloom, W. J. Fokkink, and R. J. van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Trans. Comput. Log.*, 5(1):26–78, 2004.

3 B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.

4 V. Castiglioni, D. Gebler, and S. Tini. Logical characterization of bisimulation metrics. In *Proc. QAPL 2016*, Electronic Proceedings in Theoretical Computer Science, 2016.

5 P. R. D'Argenio, D. Gebler, and M. D. Lee. Axiomatizing bisimulation equivalences and metrics from probabilistic SOS rules. In *Proc. FoSSaCS 2014*, volume 8412 of *Lecture Notes in Computer Science*, pages 289–303. Springer, 2014.

6 P. R. D'Argenio and M. D. Lee. Probabilistic transition system specification: Congruence and full abstraction of bisimulation. In *Proc. FoSSaCS 2012*, volume 7213 of *Lecture Notes in Computer Science*, pages 452–466. Springer, 2012.

7 Y. Deng and W. Du. Logical, metric, and algorithmic characterisations of probabilistic bisimulation. *CoRR*, abs/1103.4577, 2011.

8 Y. Deng and R. J. van Glabbeek. Characterising probabilistic processes logically - (extended abstract). In *Proc. LPAR-17*, volume 6397 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2010.

9 Y. Deng, R. J. van Glabbeek, M. Hennessy, and C. Morgan. Characterising testing preorders for finite probabilistic processes. *Logical Methods in Computer Science*, 4(4), 2008.

10 J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled markov processes. *Theoret. Comput. Sci.*, 318(3):323–354, 2004.

11 J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. LICS 2002*, pages 413–422. IEEE Computer Society, 2002.

12 W. J. Fokkink and R. J. van Glabbeek. Divide and congruence II: from decomposition of modal formulas to preservation of delay and weak bisimilarity. *CoRR*, abs/1604.07530, 2016.

13 W. J. Fokkink, R. J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic by structural operational semantics. *Theoret. Comput. Sci.*, 354(3):421–440, 2006.

14 W. J. Fokkink, R. J. van Glabbeek, and P. de Wind. Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity. *Inf. Comput.*, 214:59–85, 2012.

15 D. Gebler and W. J. Fokkink. Compositionality of probabilistic Hennessy-Milner logic through structural operational semantics. In *Proc. CONCUR 2012*, volume 7454 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2012.

16 D. Gebler, K. G. Larsen, and S. Tini. Compositional metric reasoning with Probabilistic Process Calculi. In *Proc. FoSSaCS'15*, volume 9034 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2015.

17 D. Gebler, K. G. Larsen, and S. Tini. Compositional metric reasoning with Probabilistic Process Calculi. *Logical Methods in Computer Science*, 2016.

18 D. Gebler and S. Tini. SOS specifications of probabilistic systems by uniformly continuous operators. In *Proc. CONCUR 2015*, volume 42 of *LIPIcs*, pages 155–168. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

19 M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32:137–161, 1985.

20 R. Lanotte and S. Tini. Probabilistic bisimulation as a congruence. *ACM Trans. Comput. Log.*, 10:1–48, 2009.

21 K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

22 K. G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *J. Log. Comput.*, 1(6):761–795, 1991.

23 G. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

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

25 F. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proc. CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

Diagnosis in Infinite-State Probabilistic Systems*

Nathalie Bertrand[1], Serge Haddad[2], and Engel Lefaucheux[3]

1 **Inria, France** nathalie.bertrand@inria.fr
2 **LSV, ENS Cachan & CNRS & Inria, France** serge.haddad@ens-cachan.fr
3 **Inria, France and**
LSV, ENS Cachan & CNRS & Inria, France engel.lefaucheux@ens-cachan.fr

Abstract

In a recent work, we introduced four variants of diagnosability (FA, IA, FF, IF) in (finite) probabilistic systems (pLTS) depending whether one considers (1) finite or infinite runs and (2) faulty or all runs. We studied their relationship and established that the corresponding decision problems are PSPACE-complete. A key ingredient of the decision procedures was a characterisation of diagnosability by the fact that a random run almost surely lies in an open set whose specification only depends on the qualitative behaviour of the pLTS. Here we investigate similar issues for infinite pLTS. We first show that this characterisation still holds for FF-diagnosability but with a G_δ set instead of an open set and also for IF- and IA-diagnosability when pLTS are finitely branching. We also prove that surprisingly FA-diagnosability cannot be characterised in this way even in the finitely branching case. Then we apply our characterisations for a partially observable probabilistic extension of visibly pushdown automata (POpVPA), yielding EXPSPACE procedures for solving diagnosability problems. In addition, we establish some computational lower bounds and show that slight extensions of POpVPA lead to undecidability.

1998 ACM Subject Classification D.2.5 Testing and Debugging; F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Diagnosis – Infinite-state systems – Partial observation – Markov chains

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.37

1 Introduction

Diagnosis. Monitoring (hardware and/or software) systems prone to faults involves several critical tasks: controlling the system to prevent faults as much as possible, deducing the cause of the faults, etc. Most of these tasks assume that an observer has the capability to assess the *status* of the current run based on the outputs of the system: providing information about the possible occurrence of faults. Such an observer is called a *diagnoser* and its associated task is called *diagnosis*. This framework leads to interesting decision and synthesis problems: "Does there exist a diagnoser?" and in the positive case "How to build such a diagnoser?", "Which kind of diagnoser is sufficient?", etc. The decision problem, on which we focus here, is called *diagnosability* [15].

Diagnosis of discrete event systems. In order to formally reason about diagnosability, the systems were first modelled by finite labelled transition systems (LTS). Then the specification of a diagnoser is defined by two requirements: *correctness*, meaning that the information provided by the diagnoser is accurate, and *reactivity*, ensuring that a fault will eventually

* S. Haddad has been supported by ERC project EQualIS (FP7-308087).

be detected. Within the framework of finite LTS, the decision problem was shown to be solvable in PTIME [10] and it is in fact NLOGSPACE-complete.

Diagnosis of probabilistic systems. A natural way of modelling partially observable systems consists in introducing probabilities (*e.g.* when the design is not fully known or the effects of the interaction with the environment is not predictible). Thus the notion of diagnosability was later extended to Markov chains with labels on transitions, also called probabilistic labelled transition systems (pLTS) [16]. In this context, the reactivity requirement now asks that faults will be almost surely eventually detected. Regarding correctness, two specifications have been proposed: either one sticks to the original definition and requires that the provided information is accurate, defining *A-diagnosability*; or one weakens the correctness by admitting errors in the provided information that should, however, have an arbitrary small probability defining *AA-diagnosability*. From a computational viewpoint, we recently proved that A-diagnosability is PSPACE-complete [3] and that AA-diagnosability can be solved in PTIME [4].

In case a system is not diagnosable, one may be able to control it, by forbidding some controllable actions, so that is becomes diagnosable. This property of *active diagnosability* has been studied for discrete-event systems [14, 9], and for probabilistic systems [2]. Interestingly, the diagnosability notion in the latter work slightly differs from the original one in [16]. Building on this variation, in [3] semantical issues have been investigated and four relevant notions of diagnosability (FA, IA, FF, IF) have been defined depending on (1) whether one considers finite or infinite runs and (2) faulty or all runs. In finite pLTS, it was shown that all these notions can be characterized by the fact that a random run almost surely lies in an open set, whose specification only depends on the qualitative behaviour of the pLTS.

Diagnosis of infinite-state systems. Diagnosability in infinite-state systems has been studied, on the one hand for restricted Petri nets [6], for which an accurate diagnoser can be designed, and on the other hand for visibly pushdown automata (VPA) [12], for which diagnosability can be decided via the determinisation procedure of [1]. However to the best of our knowledge diagnosis of probabilistic infinite-state systems has not yet been studied.

Contributions. The characterisations of diagnosability established in [3] strongly relied on the finiteness of the models. Our first aim is thus to establish characterisations in the infinite-state case. FF-diagnosability (the original notion of diagnosability) states that almost surely a faulty run will be detected in finite time. We establish that FF-diagnosability can be characterised by the fact that a random run almost surely lies in a G_δ set, only depending on the qualitative behaviour of the system. This characterisation also applies to IF-diagnosability for finitely-branching systems, since then the two notions coincide. An *ambiguous* infinite correct (resp. faulty) run is a run indistinguishable from a faulty (resp. correct) run. IA-diagnosability states that almost surely a run is unambiguous. The set of ambiguous runs is an analytic set (so a priori not known to be a Borel set). However in the finitely-branching case, we establish that the set of unambiguous runs is a G_δ set, yielding a characterisation of IA-diagnosability. FA-diagnosability states that the probability that a finite run is unambiguous goes to 1 when its length goes to infinity. Surprisingly, despite the fact that IA-diagnosability and FA-diagnosability are very close, we prove that FA-diagnosability cannot be characterised by the fact that a random run almost surely lies in a G_δ set. Furthermore we strenghten this result by another inexpressivess result also related to FA-diagnosability.

Partially observable probabilistic visibly pushdown automata (POpVPA) are models generating infinite-state probabilistic systems. We show how to exploit the above characterisations to design a decision procedure for diagnosability in POpVPA. More precisely we show that we can "encode" our characterisations in an enlarged probabilistic VPA and then exploit the decision procedures of [8] leading to an EXPSPACE algorithm. Since our characterisations are not regular, this requires some tricky machinery. Finally we complete this work by exhibiting an EXPTIME lower-bound and showing that slight extensions of POpVPA lead to undecidability of the diagnosability problem.

Organisation. In Section 2, we introduce probabilistic infinite-state systems, equip them with partial observation and faults, and define diagnosability notions. In Section 3, we establish characterisations of the diagnosability notions and inexpressiveness results. We exploit the characterisations to design decision procedures for POpVPA in Section 4, also proving hardness and undecidability results. We conclude and give some perspectives in Section 5. More details and all the proofs can be found in the associate research report [5].

2 Diagnosis specifications for infinite-state probabilistic systems

2.1 Probabilistic labelled transition systems

Probabilistic labelled transition systems (pLTS) are labelled transition systems equipped with probability distributions on transitions outgoing from a state.

▸ **Definition 1.** A pLTS is a tuple $\mathcal{M} = \langle Q, q_0, \Sigma, T, \mathbf{P}\rangle$ where:
- Q is a finite or countable set of states with $q_0 \in Q$ the initial state;
- Σ is a finite set of events;
- $T \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- $\mathbf{P} : T \to \mathbb{Q}_{>0}$ is the transition probability fulfilling: $\forall q \in Q,\ \sum_{(q,a,q')\in T} \mathbf{P}[q,a,q'] = 1$.

Given a pLTS $\mathcal{M}$, the transition relation of the *underlying LTS* $\mathcal{L}$ is defined by $q \xrightarrow{a} q'$ for $(q,a,q') \in T$; this transition is then said to be *enabled* in q. In order to emphasise the relation between the pLTS and the LTS, we sometimes write $\mathcal{M} = (\mathcal{L}, \mathbf{P})$. Note that since we assume the state space to be at most countable, a pLTS is by definition at most countably branching: from every state q, there are at most countably many transitions enabled in q.

▸ **Example 2.** The pLTS of Figure 1 represents a server that accepts jobs (event `in`) until it randomly decides to serve the jobs (event `serve`). When a job is done the result is delivered (event `out`). When all jobs are done, the server waits for a new batch of jobs. However randomly, the server may trigger a fault (event **f**) and then abort all remaining jobs (event `abort`). Afterwards, the server is reset (event `reset`). In the figure, the label of a transition (q,a,q') is depicted as $\mathbf{P}[q,a,q'] \cdot a$.

Let us now introduce some important notions and notations that will be used throughout the paper. A *run* ρ of a pLTS $\mathcal{M}$ is a (finite or infinite) sequence $\rho = q_0 a_0 q_1 \dots$ such that for all i, $q_i \in Q$, $a_i \in \Sigma$ and when q_{i+1} is defined, $q_i \xrightarrow{a_i} q_{i+1}$. The notion of run can be generalised, starting from an arbitrary state q. We write Ω for the set of all infinite runs of $\mathcal{M}$ starting from q_0, assuming the pLTS is clear from context. When it is finite, ρ ends in a state q and its *length*, denoted $|\rho|$, is the number of events occurring in it. Given a finite run $\rho = q_0 a_0 q_1 \dots q_n$ and a (finite or infinite) run $\rho' = q_n a_n q_{n+1} \dots$, the concatenation of ρ and ρ', written $\rho\rho'$, is the run $q_0 a_0 q_1 \dots q_n a_n q_{n+1} \dots$; the run ρ is then a *prefix* of $\rho\rho'$,

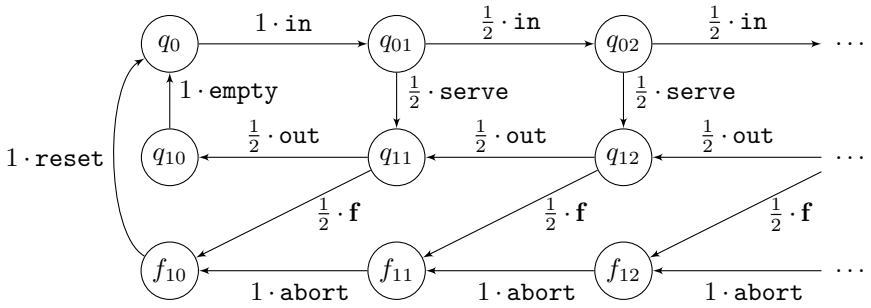


Figure 1 An infinite-state pLTS.

which we denote $\rho \preceq \rho\rho'$. The *cylinder* defined by a finite run ρ is the set of all infinite runs that extend ρ: $C(\rho) = \{\rho' \in \Omega \mid \rho \preceq \rho'\}$. Cylinders form a basis of open sets for the standard topology on the set of runs (which can be viewed as an infinite tree). One equips a pLTS with a probability measure on Ω with σ-algebra being $\mathcal{B}$, the set of Borel sets, and which is uniquely defined by Caratheodory's extension theorem from the probabilities of the cylinders:

$$\mathbb{P}(C(q_0 a_0 q_1 \dots q_n)) = \mathbf{P}[q_0, a_1, q_1] \cdots \mathbf{P}[q_{n-1}, a_{n-1}, q_n] \ .$$

We will sometimes omit the C and write $\mathbb{P}(\rho)$ for $\mathbb{P}(C(\rho))$. It is well-known that once the measure is fixed, one can enlarge the set of of measurable sets by considering the smallest σ-algebra containing $\mathcal{B}$ and the "null" sets: $\{A \mid \exists B \in \mathcal{B}\ A \subseteq B \wedge \mathbb{P}(B) = 0\}$ and then extend the original measure to a (complete) measure on this enlarged σ-algebra. We consider this measure in the sequel.

The sequence associated with $\rho = q a_0 q_1 \dots$ is the word $\sigma_\rho = a_0 a_1 \dots$, and we write either $q \xrightarrow{\rho}{}^*$ or $q \xrightarrow{\sigma_\rho}{}^*$ (resp. $q \xrightarrow{\rho}{}^* q'$ or $q \xrightarrow{\sigma_\rho}{}^* q'$) for an infinite (resp. finite) run ρ. A state q is *reachable* (from q_0) if there exists a run such that $q_0 \xrightarrow{\rho}{}^* q$, which we alternatively write $q_0 \to^* q$. The (infinite) language of pLTS $\mathcal{M}$ consists of all infinite words that label runs of $\mathcal{M}$ and is formally defined as $\mathsf{L}^\omega(\mathcal{M}) = \{\, \sigma \in \Sigma^\omega \mid q_0 \xrightarrow{\sigma}{}^* \,\}$.

2.2 Partial observation and faults

The observation of a pLTS is given by a mask function. This function projects every event to its observation. This observation is partial as an event can have no observation or shares its observation with another event, but it is deterministic.

▸ **Definition 3.** A *partially observable pLTS* (POpLTS) is a tuple $\mathcal{N} = \langle \mathcal{M}, \Sigma_o, \mathcal{P} \rangle$ consisting of a pLTS $\mathcal{M}$ equipped with a mapping $\mathcal{P} : \Sigma \to \Sigma_o \cup \{\varepsilon\}$ where Σ_o is the set of observations.

Note that our setting generalises most existing frameworks of fault diagnosis by considering a mask function $\mathcal{P}$ onto a possibly different alphabet rather than a partition of the event alphabet into observable and unobservable events. An event $a \in \Sigma$ is said *unobservable* if $\mathcal{P}(a) = \varepsilon$, otherwise, it is *observable* and we distinguish a being *fully observable* if $\mathcal{P}(a) \neq \varepsilon$ and $\mathcal{P}^{-1}(\{\mathcal{P}(a)\}) = \{a\}$ or *partially observable* if $\mathcal{P}(a) \neq \varepsilon$ and $|\mathcal{P}^{-1}(\{\mathcal{P}(a)\})| > 1$. The set of unobservable events is denoted Σ_u.

Let $\sigma \in \Sigma^*$ be a finite word; its length is denoted $|\sigma|$. The mapping $\mathcal{P}$ is extended to finite words inductively: $\mathcal{P}(\varepsilon) = \varepsilon$ and $\mathcal{P}(\sigma a) = \mathcal{P}(\sigma)\mathcal{P}(a)$. We say that $\mathcal{P}(\sigma)$ is the *mask* of σ. Write $|\sigma|_o$ for $|\mathcal{P}(\sigma)|$. When σ is an infinite word, its mask is the limit of the masks of its finite prefixes. This mask function is applicable to runs via their associated sequence; it can

be either finite or infinite. As usual the mask function is extended to languages. With respect to $\mathcal{P}$, a POpLTS $\mathcal{N}$ is *convergent* if there is no infinite sequence of unobservable events from any reachable state: $\mathsf{L}^\omega(\mathcal{M}) \cap \Sigma^*\Sigma_u^\omega = \varnothing$. When $\mathcal{N}$ is convergent, for every $\sigma \in \mathsf{L}^\omega(\mathcal{M})$, $\mathcal{P}(\sigma) \in \Sigma_o^\omega$. In the rest of the paper we assume that POpLTS are convergent. $\mathcal{P}$ can also be be viewed as a mapping from runs to Σ_o^ω by defining $\mathcal{P}(q_0a_0q_1a_1\ldots) = \mathcal{P}(a_0a_1\ldots)$. Remark that this mapping is continuous. We will refer to a *sequence* for a finite or infinite word over Σ, and an *observed sequence* for a finite or infinite sequence over Σ_o. Clearly, the application of the mask function onto Σ_o of a sequence yields an observed sequence.

The *observable length* of a run ρ denoted $|\rho|_o \in \mathbb{N} \cup \{\infty\}$, is the number of observable events that occur in it: $|\rho|_o = |\sigma_\rho|_o$. A *signalling* run is a finite run whose last event is observable. Signalling runs are precisely the relevant runs w.r.t. partial observation issues since each observable event provides additional information about the execution to an external observer. Given states q, q' and an observed sequence $\sigma \in \Sigma_o^+$, we write $q \overset{\sigma}{\Longrightarrow} q'$ if there is a signalling run from q to q' with observed sequence σ.

In the sequel starting from the initial state q_0, SR denotes the set of signalling runs, and SR_n the set of signalling runs of observable length n. Since we assume that the POpLTS are convergent, for all $n > 0$, SR_n is equipped with a probability distribution defined by assigning measure $\mathbb{P}(\rho)$ to each $\rho \in \mathsf{SR}_n$. Given ρ a finite or infinite run, and $n \leq |\rho|_o$, $\rho_{\downarrow n}$ denotes the signalling subrun of ρ of observable length n. For convenience, we consider the empty run q_0 to be the single signalling run, of null length.

2.3 Fault diagnosis for POpLTS

To model the problem of fault diagnosis in POpLTS, we assume the event alphabet Σ contains a special event $\mathbf{f} \in \Sigma$ called the *fault*. A run ρ is then said to be *faulty* if its associated sequence of events contains a fault, *i.e.* $\sigma_\rho \in \Sigma^*\mathbf{f}\Sigma^\omega$; otherwise it is *correct*. The set of faulty (resp. correct) runs is denoted F (resp. C). For $n \in \mathbb{N}$, we write F_n for the set of infinite runs ρ such that $\rho_{\downarrow n}$ is faulty and C_n for the set of infinite runs ρ such that $\rho_{\downarrow n}$ is correct. By definition, for all n, $\Omega = \mathsf{F}_n \uplus \mathsf{C}_n$, moreover, $\mathsf{F} = \bigcup_{n\in\mathbb{N}} \mathsf{F}_n$ and $\mathsf{C} = \bigcap_{n\in\mathbb{N}} \mathsf{C}_n$.

In order to reason about faults we partition sequences of observations into three subsets: an observed sequence $\sigma \in \Sigma_o^\omega$ is *surely correct* if $\mathcal{P}^{-1}(\sigma) \cap \mathsf{L}^\omega(\mathcal{M}) \subseteq (\Sigma \setminus \mathbf{f})^\omega$; it is *surely faulty* if $\mathcal{P}^{-1}(\sigma) \cap \mathsf{L}^\omega(\mathcal{M}) \subseteq \Sigma^*\mathbf{f}\Sigma^\omega$; otherwise, it is *ambiguous*. For finite sequences, we need to rely on signalling runs: a finite observed sequence $\sigma \in \Sigma_o^*$ is *surely faulty* (resp. *surely correct*) if for every signalling run ρ with $\mathcal{P}(\sigma_\rho) = \sigma$, ρ is faulty (resp. correct); otherwise it is ambiguous. A (finite signalling or infinite) run ρ is *surely faulty* (resp. *surely correct*, *ambiguous*) if $\mathcal{P}(\rho)$ is surely faulty (resp. surely correct, ambiguous).

In order to specify various requirements for diagnosability we need to refine the notion of ambiguity. Let $\mathcal{N}$ be a POpLTS and $n \in \mathbb{N}$ with $n \geq 1$. Then:

- FAmb_∞ (resp. CAmb_∞) is the set of infinite faulty (resp. correct) ambiguous runs of $\mathcal{N}$;
- FAmb_n (resp. CAmb_n) is the set of infinite runs of $\mathcal{N}$ whose signalling subrun of observable length n is faulty (resp. correct) and ambiguous;

At this point it is interesting to look at the status of the different subsets of runs we have introduced with respect to the Borel hierarchy. The complementary sets F_n and C_n are unions of cylinders; so they are open (and by complementation) closed sets. The set of faulty (resp. correct) runs F (resp. C) is an open (resp. closed) set as a union (resp. intersection) of open (resp. closed) sets. The sets FAmb_n and CAmb_n are unions of cylinders; so they are open. The sets FAmb_∞ and CAmb_∞ may be defined as follows. Consider $(\Sigma_o^2)^\omega$ and Ω^2 both equipped with the product topology. $\mathsf{SameObs} = \{(\rho, \rho') \mid \mathcal{P}(\rho) = \mathcal{P}(\rho')\}$ is the inverse image by a continuous mapping of the closed set $\{(\sigma, \sigma) \mid \sigma \in \Sigma_o^\omega\}$. Therefore $\mathsf{SameObs}$ is closed.

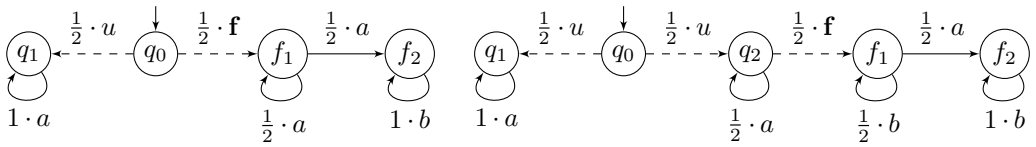


Figure 2 Left: a POpLTS that is IF-diagnosable but not IA-diagnosable. Right: a POpLTS that is IA-diagnosable but not FA-diagnosable.

Thus $\mathsf{C}\times\mathsf{F}\cap\mathsf{SameObs}$ is a Borel set. The first and second projections are exactly CAmb_∞ and FAmb_∞ which establishes that these sets are analytic sets (*i.e.* continuous images of Borel sets). The set of analytic sets is a strict superset of Borel sets but every analytic set is still measurable w.r.t. the complete measure [13, 2H8 p.83].

In the context of finite POpLTS, we introduced four possible specifications of diagnosability [3]. There are two discriminating criteria: whether the non ambiguity requirement holds for faulty runs only or for all runs, and whether ambiguity is defined at the infinite run level or for longer and longer finite signalling subruns.

▸ **Definition 4.** Let $\mathcal{N}$ be a POpLTS. Then:

- $\mathcal{N}$ is IF-*diagnosable* if $\mathbb{P}(\mathsf{FAmb}_\infty)=0$.
- $\mathcal{N}$ is IA-*diagnosable* if $\mathbb{P}(\mathsf{FAmb}_\infty\uplus\mathsf{CAmb}_\infty)=0$.
- $\mathcal{N}$ is FF-*diagnosable* if $\limsup_{n\to\infty}\mathbb{P}(\mathsf{FAmb}_n)=0$.
- $\mathcal{N}$ is FA-*diagnosable* if $\limsup_{n\to\infty}\mathbb{P}(\mathsf{FAmb}_n\uplus\mathsf{CAmb}_n)=0$.

We recall in the next theorem all the implications that hold between these definitions. Missing implications do not hold, already for finite-state POpLTS.

▸ **Theorem 5** ([3]). *Let $\mathcal{N}$ be a POpLTS. Then*

- $\mathcal{N}$ FA-*diagnosable* $\Rightarrow$ $\mathcal{N}$ IA-*diagnosable and* FF-*diagnosable;*
- $\mathcal{N}$ IA-*diagnosable or* FF-*diagnosable* $\Rightarrow$ $\mathcal{N}$ IF-*diagnosable;*
- *If $\mathcal{N}$ is finitely branching, then $\mathcal{N}$ is* IF-*diagnosable iff $\mathcal{N}$ is* FF-*diagnosable.*

In order to illustrate the different kinds of diagnosability, we describe below some discriminating examples, already presented in [3].

Consider the POpLTS $\mathcal{N}$ on the left of Figure 2 where $\{u,\mathbf{f}\}$ is the set of unobservable events (represented by dashed arrows) and $\mathcal{P}$ is the identity over the other events. A faulty run will almost surely produce a b-event that cannot be mimicked by the single correct run. Thus this POpLTS is IF-diagnosable. The unique correct run $\rho=q_0uq_1aq_1\dots$ has probability $\frac{1}{2}$ and its corresponding observed sequence a^ω is ambiguous. Thus the POpLTS is not IA-diagnosable. This simple example shows that, already for finite-state POpLTS, IF-diagnosability does not imply IA-diagnosability.

Similarly, let us look at the POpLTS on the right of Figure 2 where $\{u,\mathbf{f}\}$ is the set of unobservable events and $\mathcal{P}$ is the identity over the other events. Any infinite faulty run will contain a b-event, and cannot be mimicked by a correct run, therefore $\mathsf{FAmb}_\infty=\varnothing$. The two infinite correct runs have a^ω as observed sequence, and cannot be mimicked by a faulty run, thus $\mathsf{CAmb}_\infty=\varnothing$. As a consequence, this POpLTS is IA-diagnosable. Consider now the infinite correct run $\rho=q_0uq_1aq_1\dots$. It has probability $\frac{1}{2}$, and all its finite signalling subruns are ambiguous since their observed sequence is a^n, for some $n\in\mathbb{N}$. Thus for all $n\geq 1$, $\mathbb{P}(\mathsf{CAmb}_n)\geq\frac{1}{2}$, so that this POpLTS is not FA-diagnosable.

3 Characterisation of diagnosability

The aim of this section is to establish "simple" characterisations of the diagnosability notions for a POpLTS $\mathcal{N} = ((\mathcal{L}, \mathbf{P}), \Sigma_o, \mathcal{P})$ and more precisely to study whether one can express it as a Borel set $B \in \mathcal{B}$ only depending on the underlying LTS $\mathcal{L}$ and the mask function $\mathcal{P}$, such that almost surely a random run belongs to B if and only if $\mathcal{N}$ is diagnosable. Furthermore if possible, one looks for a set B belonging to a low level of the Borel hierarchy. Observe that for all notions, this requires some machinery since the finite runs-based notions FF and FA are expressed by a family of Borel sets and the infinite runs-based notions IF and IA are expressed by a set which is not *a priori* a Borel set.

Pursuing this goal, we introduce a language pathL for specifying Borel sets of runs. It is based on *path formulae*. A path formula α is a predicate over finite prefixes of runs. The (pseudo-)syntax of a formula of pathL is:

$$\phi ::= \alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \Diamond\phi$$

where α is a path formula. In the sequel we use the standard shortcut $\Box\phi \equiv \neg \Diamond \neg\phi$.

A formula is evaluated at some position k of a run $\rho = q_0 a_0 q_1 \ldots$. The prefix $\rho[0,k]$ of ρ is defined by $\rho[0,k] = q_0 a_0 q_1 \ldots q_k$. The semantics of pathL is inductively defined by:

- $\rho, k \vDash \alpha$ if and only if $\alpha(\rho[0,k])$;
- $\rho, k \vDash \neg\phi$ if and only if $\rho, k \nvDash \phi$;
- $\rho, k \vDash \phi_1 \wedge \phi_2$ if and only if $\rho, k \vDash \phi_1$ and $\rho, k \vDash \phi_2$;
- $\rho, k \vDash \Diamond\phi$ if and only if there exists $k' \geq k$ such that $\rho, k' \vDash \phi$.

Finally $\rho \vDash \phi$ if and only if $\rho, 0 \vDash \phi$. Due to the presence of path formulae (with no restriction) this language subsumes LTL and more generally any ω-regular specification language. In order to reason about the probabilistic behaviour of a POpLTS, we introduce qualitative probabilistic formulae $\mathbb{P}^{\bowtie p}(\phi)$ with $\bowtie \in \{<, >, =\}$, $p \in \{0, 1\}$ and $\phi \in$ pathL. The semantics is obvious: $\mathcal{N} \vDash \mathbb{P}^{\bowtie p}(\phi)$ if and only if $\mathbb{P}_{\mathcal{N}}(\{\rho \in \Omega \mid \rho \vDash \phi\}) \bowtie p$. Since pathL is closed by complementation the probabilistic formulae can be restricted to $\mathbb{P}^{=0}(\phi)$ and $\mathbb{P}^{>0}(\phi)$.

Let us give some examples of path formulae. Given a finite run $\rho = q_0 a_0 q_1 \ldots q_k$, let $\mathfrak{f}$ be defined by $\mathfrak{f}(\rho) =$ true if $a_i = \mathbf{f}$ for some index i. This path formula characterises the faulty finite runs. Let $\mathfrak{U}$ be defined by $\mathfrak{U}(\rho) =$ true if there exists a correct signalling run ρ' with $\mathcal{P}(\rho) = \mathcal{P}(\rho')$. Using the path formulae $\mathfrak{f}$ and $\mathfrak{U}$, we exhibit a formula of pathL that characterises FF-diagnosability.

▸ **Proposition 6.** *Let $\mathcal{N}$ be a POpLTS. Then $\mathcal{N}$ is* FF*-diagnosable iff $\mathcal{N} \vDash \mathbb{P}^{=0}(\Diamond \Box (\mathfrak{f} \wedge \mathfrak{U}))$.*

Due to Theorem 5, in finitely-branching POpLTS the above characterisation also holds for IF-diagnosability. We also need the finitely-branching assumption in order to characterise IA-diagnosability. To this goal, let us introduce a more intricate path formula. For $\sigma \in \Sigma_o^*$, we define firstf(σ) by firstf$(\sigma) = \min\{k \mid \exists \rho$ signalling run $\mathcal{P}(\rho) = \sigma \wedge \rho_{\downarrow k}$ is faulty$\}$ with the convention that $\min(\varnothing) = \infty$. Then the path formula $\mathfrak{W}$ is defined by: $\mathfrak{W}(\varepsilon) =$ false and $\mathfrak{W}(q_0 a_0 \ldots q_{n+1}) =$ true if firstf$(\mathcal{P}(q_0 a_0 \ldots q_{n+1})) =$ firstf$(\mathcal{P}(q_0 a_0 \ldots q_n)) < \infty$.

▸ **Proposition 7.** *Let $\mathcal{N}$ be a finitely branching POpLTS. Then $\mathcal{N}$ is* IA*-diagnosable iff $\mathcal{N} \vDash \mathbb{P}^{=0}(\Diamond \Box (\mathfrak{U} \wedge \mathfrak{W}))$.*

The POpLTS of Figure 3 illustrates the need for the finitely-branching assumption in Proposition 7. The set of unobservable events is $\{u, \mathbf{f}\}$ and $\mathcal{P}$ is the identity over the other events. Observation b occurs in every infinite correct run, while the observed sequence of the single infinite faulty run is a^ω. This POpLTS is thus IA-diagnosable. However, it does not satisfy $\mathbb{P}^{=0}(\Diamond \Box (\mathfrak{U} \wedge \mathfrak{W}))$ since the unique infinite faulty run has probability $\frac{1}{2}$ and satisfies at

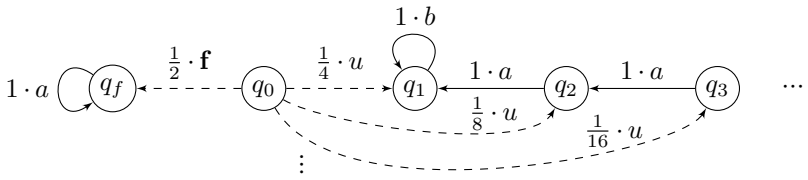


Figure 3 An infinitely-branching IA-diagnosable POpLTS.

the same time $\Box\mathfrak{W}$, by unicity, and $\Box\mathfrak{U}$. Indeed for every $n \in \mathbb{N}$, there is a correct signalling run with observed sequence a^n.

Observe that the sets of runs specified by the characterisations of FF-diagnosability ($\Diamond\Box(\mathfrak{f} \wedge \mathfrak{U})$) and IA-diagnosability ($\Diamond\Box(\mathfrak{U} \wedge \mathfrak{W})$) are F_σ sets, *i.e.* countable unions of closed sets. Surprisingly, we show that such a characterisation is impossible for FA-diagnosability: there is no F_σ set E such that a POpLTS $\mathcal{N}$ is FA-diagnosable if and only if $\mathcal{N} \vDash \mathbb{P}^{=0}(E)$.

▸ **Proposition 8.** *There exists a finitely-branching LTS $\mathcal{L}$ and a mask function $\mathcal{P}$ such that for every F_σ set E of runs, there exists a POpLTS $\mathcal{N} = ((\mathcal{L}, \mathbf{P}), \Sigma_o, \mathcal{P})$ such that:*

- *either $\mathcal{N}$ is* FA*-diagnosable and $\mathbb{P}_{\mathcal{N}}(E) > 0$;*
- *or $\mathcal{N}$ is not* FA*-diagnosable and $\mathbb{P}_{\mathcal{N}}(E) = 0$.*

We conjecture that the impossibility also holds for arbitrary Borel sets. The next proposition shows that a positive probability characterization cannot exist whatever the Borel set.

▸ **Proposition 9.** *There exists a finitely-branching LTS $\mathcal{L}$ and a mask function $\mathcal{P}$ such that for every Borel set E of runs, there exists a POpLTS $\mathcal{N} = ((\mathcal{L}, \mathbf{P}), \Sigma_o, \mathcal{P})$ such that:*

- *either $\mathcal{N}$ is* FA*-diagnosable and $\mathbb{P}_{\mathcal{N}}(E) = 0$;*
- *or $\mathcal{N}$ is not* FA*-diagnosable and $\mathbb{P}_{\mathcal{N}}(E) > 0$.*

4 Diagnosis for probabilistic pushdown automata

We now turn to a concrete model for infinite-state POpLTS, namely the ones generated by probabilistic pushdown automata, and more specifically by probabilistic visibly pushdown automata. Our goal is to use the characterisations from the previous section to decide the diagnosability of POpLTS generated by partially observable probabilistic visibly pushdown automata (POpVPA). To do so, we face the difficulty that the Borel sets that characterise IF-, IA- and IF-diagnosability are not *a priori* regular, even in the finite branching case. Yet, for POpVPA, we circumvent this problem, and manage to specify these sets by pLTL formulae on a determinisation of the model, tagged with the needed atomic propositions. The decidability of the qualitative model checking for recursive probabilistic systems [8] then yields the decidability of the above three diagnosability notions for POpVPA.

4.1 Probabilistic visibly pushdown automata

Among probabilistic infinite-state systems the ones generated by probabilistic pushdown automata [11, 8] support relevant decision procedures. Already in the non-probabilistic case, the subclass of visibly pushdown automata (VPA) [1] is more tractable than the general model. In VPA, the type of events determines whether the operation on the stack is a push, a pop, or possibly changes the top stack symbol, so that the languages defined by VPA enjoy most of the desirable properties regular languages have.

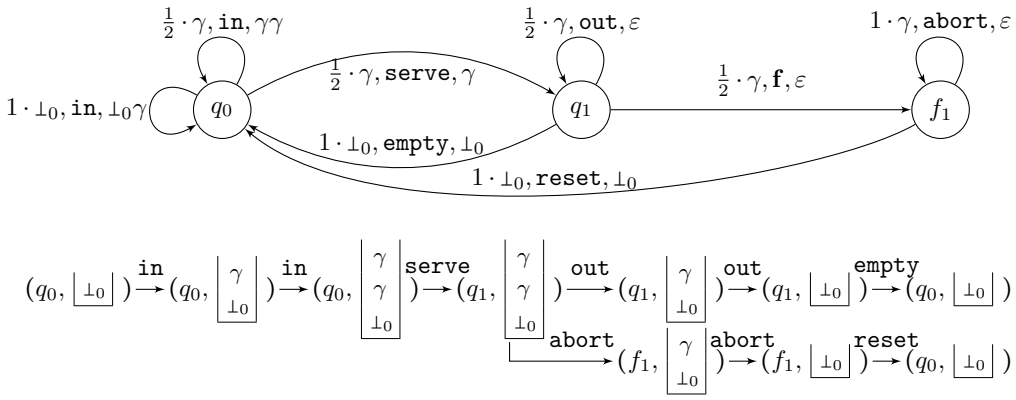


Figure 4 A pVPA generating the pLTS from Figure 1 with two finite runs.

▸ **Definition 10.** A *probabilistic visibly pushdown automaton* (pVPA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \mathbf{P})$ where:

- Q is a finite set of control states with q_0 the initial state;
- Σ is a finite alphabet of events, partitionned into local, push and pop events $\Sigma = \Sigma_\natural \uplus \Sigma_\sharp \uplus \Sigma_\flat$.
- Γ is a finite alphabet of stack symbols including a set of bottom stack symbols $\Gamma_\perp$ with initial symbol $\perp_0 \in \Gamma_\perp$;
- $\delta \subseteq Q \times \Gamma \times \Sigma \times Q \times \Gamma^*$ is the set of transitions such that for every $(q, \gamma, a, q', w) \in \delta$, $|w| \leq 2$, $\gamma \in \Gamma_\perp$ implies $w \in \Gamma_\perp(\Gamma \setminus \Gamma_\perp)^*$ and $\gamma \notin \Gamma_\perp$ implies $w \in (\Gamma \setminus \Gamma_\perp)^*$;
- $\mathbf{P}$ is the transition probability function fulfilling for every $q \in Q$ and $\gamma \in \Gamma$: $\sum_{(q,\gamma,a,q',w)\in\delta} \mathbf{P}[(q, \gamma, a, q', w)] = 1$.

A transition $t = (q, \gamma, a, q', w) \in \delta$ is said to be a *local* (resp. *push*, *pop*) transition if $|w| = 1$ (resp. $|w| = 2$, $|w| = 0$). We require that for every transition $t = (q, \gamma, a, q', w) \in \delta$, t is a local (resp. push, pop) transition iff a is a local (resp. push, pop) event.

The semantics of a pVPA is an infinite-state pLTS whose states are pairs (q, z) consisting of a control state and a stack contents.

▸ **Definition 11.** A pVPA $\mathcal{V} = (Q, \Sigma, \Gamma, \delta, \mathbf{P})$ defines a pLTS $\mathcal{M}_\mathcal{V} = (Q_\mathcal{V}, (q_0, \perp_0), \Sigma, T_\mathcal{V}, \mathbf{P}_\mathcal{V})$ where:

- $Q_\mathcal{V} = \{(q, z) \mid q \in Q \wedge z \in \Gamma_\perp(\Gamma \setminus \Gamma_\perp)^*\}$;
- $T_\mathcal{V} = \{((q, z\gamma), a, (q', zw)) \mid z\gamma \in \Gamma_\perp(\Gamma \setminus \Gamma_\perp)^* \wedge (q, \gamma, a, q', w) \in \delta\}$;
- For every $((q, z\gamma), a, (q', zw)) \in T_\mathcal{V}$, $\mathbf{P}_\mathcal{V}[((q, z\gamma), a, (q', zw))] = \mathbf{P}[(q, \gamma, a, q', w)]$.

▸ **Example 12.** Figure 4 gives an example of a pVPA. The event alphabet is composed of local events $\{\texttt{serve}, \texttt{empty}, \texttt{reset}\}$, a push event $\texttt{in}$ and pop events $\{\texttt{out}, \mathbf{f}, \texttt{abort}\}$. A transition $t = (q, \gamma, a, q', w)$ is represented by an edge from state q to state q' and labelled by $\mathbf{P}[t] \cdot \gamma, a, w$. The semantics of this pVPA is precisely the pLTS from Figure 1. Indeed, the stack alphabet consists of two letters $\Gamma = \{\gamma, \perp_0\}$ where the set of bottom stack symbol is $\Gamma_\perp = \{\perp_0\}$. Thus one can encode the stack using a counter that gives the number of γ in the stack. For instance, in the pLTS from Figure 1 the configuration $(q_1, \perp_0\gamma^n)$ of the pVPA corresponds to the state q_{1n}.

To define partially observable pVPA, we equip a pVPA with a mask function and require that only local events may be unobservable, and that pushes and pops can still be

distinguished. This restriction is crucial since it ensures that the observed sequence of a signalling run of a POpVPA still provides the information about the height of the stack.

▸ **Definition 13.** A *partially observable pVPA* (POpVPA) is a tuple $\langle \mathcal{V}, \Sigma_o, \mathcal{P} \rangle$ consisting of a pVPA $\mathcal{V}$ equipped with a mapping $\mathcal{P} : \Sigma \to \Sigma_o \cup \{\varepsilon\}$ such that:

- $\Sigma_o = \Sigma_{o,\natural} \uplus \Sigma_{o,\sharp} \uplus \Sigma_{o,\flat}$ is the set of observations;
- $\mathcal{P}(\Sigma_\natural) \subseteq \Sigma_{o,\natural} \cup \{\varepsilon\}$, $\mathcal{P}(\Sigma_\sharp) \subseteq \Sigma_{o,\sharp}$ and $\mathcal{P}(\Sigma_\flat) \subseteq \Sigma_{o,\flat}$.

In the sequel, we may identify a POpVPA with the POpLTS it generates. In particular, the various concepts of diagnosability are lifted from POpLTS to POpVPA.

4.2 Diagnosability for POpVPA

To obtain an algorithm for the diagnosability of POpVPA, we follow the finite-state case approach [3]. First, we determinise POpVPA $\mathcal{V}$ into $\mathcal{A}(\mathcal{V})$, with the diagnosis objective in mind, building on the deterministic automaton recognising unambiguous sequences from [9]. We therefore introduce tags that reflect the category of runs (faulty or correct) given an observed sequence with a distinction between "old" and "young" faulty runs. It then suffices to check whether the characterisations hold on the synchronised product $\widehat{\mathcal{V}} \times \mathcal{A}(\mathcal{V})$ where $\widehat{\mathcal{V}}$ enlarges $\mathcal{V}$ by keeping track of a fault occurrence. To reduce to a decidable model checking question, we specify the Borel sets from Section 3 by LTL formulae.

4.2.1 Diagnosis-oriented determinisation

The determinisation of $\mathcal{V}$ (where probabilities are irrelevant for this transformation) into $\mathcal{A}(\mathcal{V})$ exploits some ideas of the original determinisation by Alur and Madhusudan [1], yet, it is customised to diagnosis. In particular, it uses tags that were first defined to construct a deterministic Büchi automaton recognising the unambiguous sequences of a finite LTS [9]. The complete definition of the estimate VPA $\mathcal{A}(\mathcal{V})$ associated with a POpVPA $\mathcal{V}$ is technical and detailed in [5]. We emphasise here some aspects of the construction and illustrate them on an example. Figure 5 represents the deterministic VPA associated with our example POpVPA. For readability, we use shortcuts on the transitions in this figure, namely symbols a_0^{X}, a_1^{X}, etc. denote stack symbols of $\mathcal{A}(\mathcal{V})$.

Figure 6 displays two finite runs of the deterministic VPA $\mathcal{A}(\mathcal{V})$ from Figure 5 sharing most transitions to the exception of the last one.

States and stack symbols. The VPA $\mathcal{A}(\mathcal{V})$ tracks all runs with the same observation in parallel memorising their status w.r.t. faults. More precisely to the current set of runs corresponds the symbol on the top of the stack which is a set of tuples where each tuple is written as a fraction $\frac{\gamma,\mathsf{X},q}{\gamma^-,\mathsf{X}^-,q^-}$. Let us describe the meaning of this tuple:

- q is the current state of the run and γ is the symbol on the top of its stack;
- $\mathsf{X} \in \mathsf{Tg} = \{\mathsf{U}, \mathsf{V}, \mathsf{W}\}$ is the status of the run: U for a correct run, V for a young faulty run and W for an old faulty run;
- The denominator $(\gamma^-, \mathsf{X}^-, q^-)$, is related to the configuration just after the last push event of the run: γ^- is the stack symbol under the top symbol, while X^- is the status of the run reaching this configuration and q^- the state of this configuration.

A priori, a single state run would be enough. However the simulation of a pop event in the original VPA is performed in two steps requiring some additional states that we explain later.

Illustration. The initial configuration of the VPA $\mathcal{A}(\mathcal{V})$ of Figure 5 $(\mathsf{run}, \left|\{\frac{\perp_0,\mathsf{U},q_0}{\perp_0,\mathsf{U},q_0}\}\right|)$ corresponds to the empty run represented by a singleton. The denominator of bottom stack symbols is by convention $(\perp_0, \mathsf{U}, q_0)$ and is irrelevant for specifying the transitions of $\mathcal{A}(\mathcal{V})$.

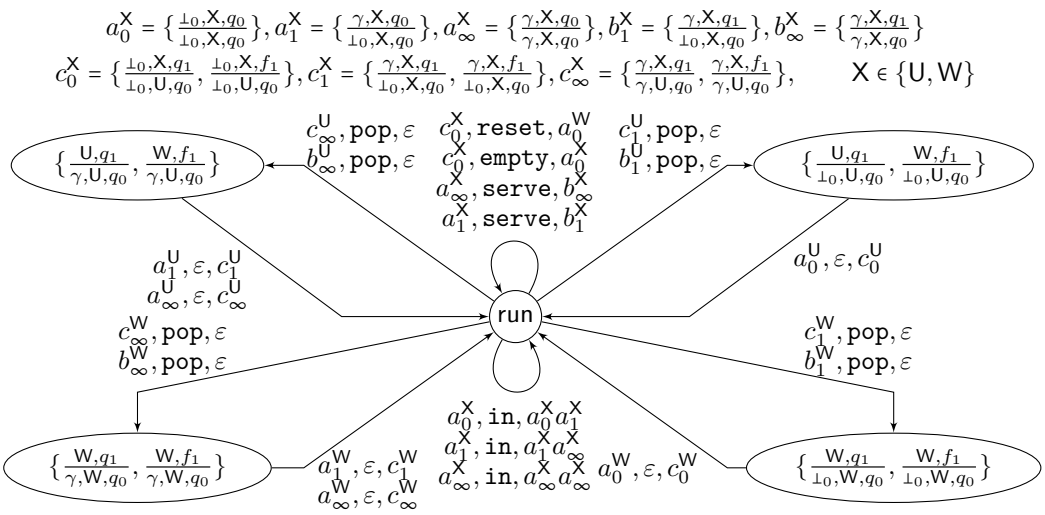


Figure 5 The VPA $\mathcal{A}(\mathcal{V})$ associated with the POpVPA $\mathcal{V}$ of Figure 4.

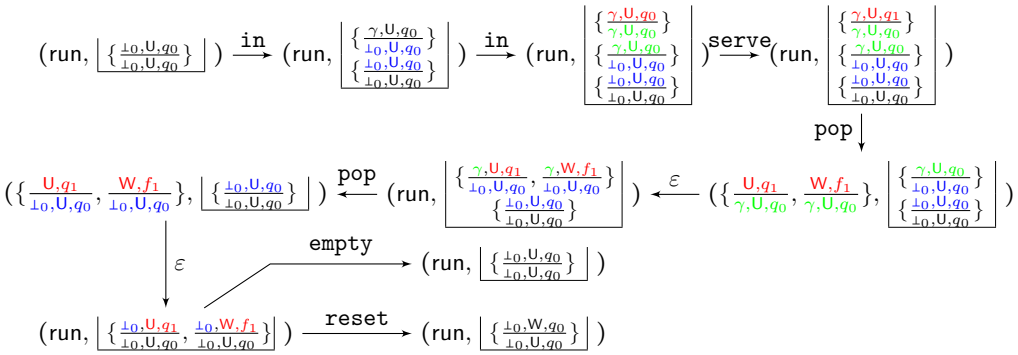


Figure 6 Two runs of the VPA from Figure 5.

Tag updates. Let us explain how the tag X of an item $\frac{\gamma,\mathsf{X},q}{\gamma^-,\mathsf{X}^-,q^-}$ of the current stack symbol is determined. If this item corresponds to a correct run then $\mathsf{X} = \mathsf{U}$. When, in a current state, after a transition of $\mathcal{A}(\mathcal{V})$ a (tracked) correct run becomes faulty in the next state, there are two cases. Either there was no tag W in (the numerators of items of) the top stack symbol of the current state then the run is tagged by W. Otherwise it is tagged by V meaning that it is a young faulty run. The tag V (young) becomes W (old) when, in the previous state, there was no tag W in the top stack symbol. A tag W is unchanged along the run.

Push transitions. Given an observed push event $o \in \Sigma_{o,\sharp}$, from the control state run with top stack symbol bel, there is a looping push transition $(\mathsf{run}, bel, o, \mathsf{run}, bel'bel'')$ in $\mathcal{A}(\mathcal{V})$ that encodes the possible signalling runs with observation o in $\mathcal{V}$. More precisely for every transition sequence $(q,\alpha) \overset{o}{\Longrightarrow} (r,\beta^-\beta)$ in $\mathcal{V}$ (*i.e.* a sequence of unobservable local events ending by an event e with $\mathcal{P}(e) = o$) and $\frac{\alpha,\mathsf{X},q}{\alpha^-,\mathsf{X}^-,q^-} \in bel$ one inserts $\frac{\beta^-,\mathsf{Y},r}{\alpha^-,\mathsf{X}^-,q^-}$ in bel' and $\frac{\beta,\mathsf{Y},r}{\beta^-,\mathsf{Y},r}$ in bel''. The value of Y follows the rules of tag updates.

Illustration. In Figure 5 several transitions correspond to the transition $(q_0, \bot_0, in, q_0, \bot_0\gamma)$ of $\mathcal{V}$, including $(\mathsf{run}, \{\frac{\bot_0,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\}, \mathtt{in}, \mathsf{run}, \{\frac{\bot_0,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\}\{\frac{\gamma,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\})$ and several transitions correspond to the transition $(q_0, \gamma, \mathtt{in}, q_0, \gamma\gamma)$ of $\mathcal{V}$, including $(\mathsf{run}, \{\frac{\gamma,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\}, \mathtt{in}, \mathsf{run}, \{\frac{\gamma,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\}\{\frac{\gamma,\mathsf{U},q_0}{\gamma,\mathsf{U},q_0}\})$. Here, the specification of the tag updates is straightforward since it does not involve faulty runs. The runs represented in Figure 6 use these two transitions from the initial state.

Local transitions. Given an observed local event $o \in \Sigma_{o,\natural}$, from the control state run with top stack symbol bel, there is a looping local transitions $(\mathsf{run}, bel, o, \mathsf{run}, bel')$ in $\mathcal{A}(\mathcal{V})$ that encodes the possible signalling runs with observation o in $\mathcal{V}$. More precisely for every transition sequence $(q, \alpha) \overset{o}{\Longrightarrow} (r, \beta)$ in $\mathcal{V}$ (*i.e.* a sequence of unobservable local events ended by an event e with $\mathcal{P}(e) = o$) and $\frac{\alpha,\mathsf{X},q}{\alpha^-,\mathsf{X}^-,q^-} \in bel$ one inserts $\frac{\beta,\mathsf{Y},r}{\alpha^-,\mathsf{X}^-,q^-}$ in bel'. The value of Y follows the rules of tag updates.
Illustration. In the VPA $\mathcal{A}(\mathcal{V})$ of Figure 5 there are several transitions corresponding to transition $(q_0, \gamma, \mathtt{serve}, q_1, \gamma)$ of $\mathcal{V}$ including $(\mathsf{run}, \{\frac{\gamma,\mathsf{U},q_0}{\gamma,\mathsf{U},q_0}\}, \mathtt{serve}, \mathsf{run}, \{\frac{\gamma,\mathsf{U},q_1}{\gamma,\mathsf{U},q_0}\})$. The runs represented in Figure 6 use this transition.
Pop transitions. Given an observed local event $o \in \Sigma_{o,\natural}$, from the control state run with top stack symbol bel, the "pop operation" is performed by a sequence of two transitions: a pop transition labelled by o that keeps in the next state all the information needed by the next (local) transition labelled by ε to move back to state run with a consistent stack symbol. Given an intermediate stack symbol, there is exactly one possible such transition. Thus despite these transitions, $\mathcal{A}(\mathcal{V})$ is still deterministic. The first transition $(\mathsf{run}, bel, o, \ell, \varepsilon)$ in $\mathcal{A}(\mathcal{V})$ is specified as follows. The next state ℓ is a set of items of the following shape $\frac{\mathsf{X},q}{\alpha^-,\mathsf{X}^-,q^-}$. More precisely for every transition sequence $(q, \alpha) \overset{o}{\Longrightarrow} (r, \varepsilon)$ in $\mathcal{V}$ (i.e. a sequence of unobservable local events ended by an event e with $\mathcal{P}(e) = o$) and $\frac{\alpha,\mathsf{X},q}{\alpha^-,\mathsf{X}^-,q^-} \in bel$ one inserts $\frac{\mathsf{Y},r}{\alpha^-,\mathsf{X}^-,q^-}$ in ℓ. The value of Y follows the rules of tag updates. A transition $(\ell, bel, \varepsilon, \mathsf{run}, bel')$ is specified as follows. For every $\frac{\mathsf{X}',q'}{\gamma,\mathsf{X},q}$ in ℓ and $\frac{\gamma,\mathsf{X},q}{\gamma^-,\mathsf{X}^-,q^-}$ in bel (*i.e.* the denominator of the first fraction and the numerator of the second fraction match), one inserts $\frac{\gamma,\mathsf{X}',q'}{\gamma^-,\mathsf{X}^-,q^-}$ in bel'.

Illustration. Let us describe how the $\mathtt{pop}$ event is performed by two transitions in the runs of the VPA of Figure 6 from the state reached after event $\mathtt{serve}$. From q_1 with γ as top of the stack there are two transitions whose observation is $\mathtt{pop}$: $(q_1, \gamma, \mathtt{out}, q_1, \varepsilon)$ and $(q_1, \gamma, \mathbf{f}, f_1, \varepsilon)$. Thus starting from run with top stack symbol $\{\frac{\gamma,\mathsf{U},q_1}{\gamma,\mathsf{U},q_0}\}$, one reaches state $\ell = \{\frac{\mathsf{U},q_1}{\gamma,\mathsf{U},q_0}, \frac{\mathsf{W},f_1}{\gamma,\mathsf{U},q_0}\}$. The faulty run is tagged with W as there was no tag W in the former top stack symbol. In the next configuration, the top stack symbol is $\{\frac{\gamma,\mathsf{U},q_0}{\bot_0,\mathsf{U},q_0}\}$. So the transition labelled by ε moves back to state run with updated top stack symbol $\{\frac{\gamma,\mathsf{U},q_1}{\bot_0,\mathsf{U},q_0}, \frac{\gamma,\mathsf{W},f_1}{\bot_0,\mathsf{U},q_0}\}$.

4.2.2 Product VPA

To recover the probabilistic behaviour of $\mathcal{V}$, we need to construct a synchronised product of $\mathcal{V}$ and the deterministic VPA $\mathcal{A}(\mathcal{V})$. In order to track the presence of a fault in a run of this product, we first enrich $\mathcal{V}$ to track occurrences of $\mathbf{f}$. We thus define the POpVPA $\widehat{\mathcal{V}}$ whose set of states $\widehat{Q}$ is a duplication of Q in correct states Q_c and faulty states Q_f. Given a transition of $\mathcal{V}$ starting from q leading to q', there is in $\widehat{\mathcal{V}}$ a transition starting from q_f leading to q'_f and a transition starting from q_c leading either to q'_c if the event is not $\mathbf{f}$ or to q'_f otherwise. We then construct $\mathcal{V}_{\mathcal{A}(\mathcal{V})} = \widehat{\mathcal{V}} \times \mathcal{A}(\mathcal{V})$ the product automaton of $\widehat{\mathcal{V}}$ and $\mathcal{A}(\mathcal{V})$ synchronised on the alphabet of observed events Σ_o. The transitions of $\widehat{\mathcal{V}}$ labelled by unobservable events do not change the second component of the state and the transitions of $\mathcal{A}(\mathcal{V})$ labelled by

ε do not change the first component of the state. Due to the determinism of $\mathcal{A}(\mathcal{V})$, $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ has the same probabilistic behaviour as the one of $\mathcal{V}$ except that it memorises additional information along the run. More precisely, let ρ be a run of $\mathcal{V}$, then $\bar{\rho}$, a run of $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$, is obtained from ρ by following the same transitions and adding the single $\ominus$ transition firable after any pop transition. One immediately gets $\mathbb{P}_{\mathcal{V}_{\mathcal{A}(\mathcal{V})}}(\bar{\rho}) = \mathbb{P}_{\mathcal{V}}(\rho)$.

Let us explain how to transform the paths formulae $\mathfrak{f}$, $\mathfrak{U}$ and $\mathfrak{W}$ into atomic propositions on the pairs $((q, \mathsf{run})(\gamma, bel))$ consisting of a control state of $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ together with a top stack contents. For path formula $\mathfrak{f}$, we define the corresponding atomic proposition ν_f by $\nu_f((q, \mathsf{run})(\gamma, bel)) = \mathsf{true}$ if and only if $q \in Q_f$. Let $bel \subseteq (\Gamma \times \mathsf{Tg} \times Q)^2$, we say that X *occurs* in bel if there exists $\frac{\gamma,\mathsf{X},q}{\gamma^-,\mathsf{X}^-,q^-} \in bel$. We define atomic propositions ν_u and ν_w by: $\nu_u((q, \mathsf{run})(\gamma, bel)) = \mathsf{true}$ if and only if U occurs in bel; and $\nu_w((q, \mathsf{run})(\gamma, bel)) = \mathsf{true}$ if and only if W occurs in bel.

Given a run ρ of $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$, we write $\mathsf{last}(\rho)$ for the pair formed of the control state and top stack symbol in $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ after ρ. The atomic propositions ν_f and ν_u perfectly reflect the paths formula $\mathfrak{f}$ and $\mathfrak{U}$, and ν_w is eventually forever true if and only if $\mathfrak{W}$ is.

▸ **Proposition 14.** *Let ρ be an infinite run of $\mathcal{V}$. Then:*

- *For all $k \in \mathbb{N}$, $\mathfrak{f}(\rho_{\downarrow k}) \Leftrightarrow \nu_f(\mathsf{last}(\bar{\rho}_{\downarrow k}))$ and $\mathfrak{U}(\rho_{\downarrow k}) \Leftrightarrow \nu_u(\mathsf{last}(\bar{\rho}_{\downarrow k}))$;*
- $\rho \vDash \Diamond \Box\, \mathfrak{W} \Leftrightarrow \exists K \forall k \geq K.\ \nu_w(\mathsf{last}(\bar{\rho}_{\downarrow k})) = \mathsf{true}$.

4.2.3 Complexity of diagnosability for POpVPA

Thanks to the relationships between the paths formulae and the atomic propositions, and using the characterisations from Section 3, we manage to reduce the FF-, IF- and IA-diagnosis to the model checking of a pLTL formula on the product VPA $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$. Model checking qualitative pLTL for probabilistic pushdown automata is achievable in polynomial space in the size of the model [8]. In our case, $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ is exponential in the size of $\mathcal{V}$. We thus obtain the decidability and a complexity upper-bound for the diagnosability problems for POpVPA.

▸ **Theorem 15.** FF*-diagnosability,* IF*-diagnosability and* IA*-diagnosability are decidable in* EXPSPACE *for POpVPA.*

Proof. For $\mathcal{V}$ a POpVPA, $\mathcal{V}$ and $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ have the same probabilistic behaviour. Therefore, using the relation between path formulae and atomic propositions from Proposition 14, we reformulate Propositions 6 and 7 into pLTL characterisations of diagnosability:

- $\mathcal{V}$ is FF-diagnosable iff $\mathcal{V}_{\mathcal{A}(\mathcal{V})} \vDash \mathbb{P}^{=0}(\Diamond \Box\, (\nu_f \wedge \nu_u))$;
- $\mathcal{V}$ is IA-diagnosable iff $\mathcal{V}_{\mathcal{A}(\mathcal{V})} \vDash \mathbb{P}^{=0}(\Diamond \Box\, (\nu_u \wedge \nu_w))$.

Moreover, since the POpLTS generated by POpPDA are finitely-branching, IF-diagnosability coincides with FF-diagnosability [3] (see also Theorem 5). The two above qualitative pLTL formulae can be checked on general probabilistic pushdown automata (beyond visibly pushdown ones) thanks to [8]. More precisely, one can transform $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ into a recursive Markov chain (the transformation is linear) [7]. Then, the model checking of qualitative pLTL on recursive Markov chains is doable in PSPACE in the size of the Recursive Markov Chain and EXPTIME in the size of the formulae [8]. In our case, the product VPA $\mathcal{V}_{\mathcal{A}(\mathcal{V})}$ is exponential in the size of $\mathcal{V}$ and the size of the formulae is constant. This yields an EXPSPACE algorithm for checking diagnosability of POpVPA. ◂

Reducing the universality problem for VPA, which is known to be EXPTIME-complete [1], we obtain the EXPTIME-hardness of all diagnosability variants for POpVPA.

▸ **Theorem 16.** *Diagnosability is* EXPTIME*-hard for POpVPA.*

The restriction to visibly pushdown automata is motivated by the unfeasibility of diagnosis for general probabilistic pushdown automata.

▸ **Theorem 17.** *Diagnosability is undecidable for POpPDA.*

The undecidability can be obtained by adapting the proof for diagnosis of *non-probabilistic* pushdown automata [12]. However, in order to show how robust the result is, we rather reduce from the Post Correspondence Problem and prove the undecidability of diagnosability for restricted classes of partially observable probabilistic pushdown automata. In particular, undecidability already holds for two (incomparable) subclasses of POpPDA [5] with restriction on what is observable and on the number of phases of any run, where a phase is a portion of run in which the stack either never decreases or never increases.

5 Conclusion

We studied the diagnosability problem for infinite-state probabilistic systems, both from a semantical perspective, and from an algorithmic one when considering probabilistic visibly pushdown automata. A natural research aim is to reduce the complexity gap for the diagnosability of POpVPA (currently EXPTIME-hard and in EXPSPACE). We could also investigate the diagnosability problem for other probabilistic extensions infinite state systems, such as lossy channel systems or VASS. Another research direction would be to consider the fault diagnosis problem for continuous-time probabilistic models, starting with CTMC.

References

1 R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. STOC'04*, pages 202–211. ACM, 2004.

2 N. Bertrand, É. Fabre, S. Haar, S. Haddad, and L. Hélouët. Active diagnosis for probabilistic systems. In *Proc. FoSSaCS'14*, volume 8412 of *LNCS*, pages 29–42. Springer, 2014.

3 N. Bertrand, S. Haddad, and E. Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. In *Proc. FSTTCS'14*, volume 29 of *LIPIcs*, pages 417–429. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

4 N. Bertrand, S. Haddad, and E. Lefaucheux. Accurate approximate diagnosability of stochastic systems. In *Proc. LATA'16*, volume 9618 of *LNCS*, pages 549–561. Springer, 2016.

5 N. Bertrand, S. Haddad, and E. Lefaucheux. Diagnosis in infinite-state probabilistic systems (long version). Technical report, HAL Inria, 2016. `https://hal.inria.fr/hal-01334218`.

6 M. P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of discrete-event systems using labeled Petri nets. *IEEE Trans. Automation Science and Engineering*, 11(1):144–153, 2014.

7 K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1), 2009.

8 K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Trans. Computational Logic*, 13(2):12, 2012.

9 S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. In *Proc. FSTTCS'13*, volume 24 of *LIPIcs*, pages 527–539. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

10 S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Trans. Automatic Control*, 46(8):1318–1321, 2001.

11 A. Kučera, J. Esparza, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

12 C. Morvan and S. Pinchinat. Diagnosability of pushdown systems. In *Proc. HVC'09*, volume 6405 of *LNCS*, pages 21–33. Springer, 2009.

13 Y. N. Moschovakis. *Descriptive Set Theory*. Mathematical Surveys and Monographs. AMS, 2009.

14 M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. Automatic Control*, 43(7):908–929, 1998.

15 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Automatic Control*, 40(9):1555–1575, 1995.

16 D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Trans. Automatic Control*, 50(4):476–492, 2005.