

Lars Birkedal (Ed.)

ARCoSS

LNCS 7213

Foundations of Software Science and Computational Structures

15th International Conference, FOSSACS 2012
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2012
Tallinn, Estonia, March/April 2012, Proceedings



 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison, UK

Josef Kittler, UK

Alfred Kobsa, USA

John C. Mitchell, USA

Oscar Nierstrasz, Switzerland

Bernhard Steffen, Germany

Demetri Terzopoulos, USA

Gerhard Weikum, Germany

Takeo Kanade, USA

Jon M. Kleinberg, USA

Friedemann Mattern, Switzerland

Moni Naor, Israel

C. Pandu Rangan, India

Madhu Sudan, USA

Doug Tygar, USA

Advanced Research in Computing and Software Science

Subline of Lectures Notes in Computer Science

Subline Series Editors

Giorgio Ausiello, *University of Rome 'La Sapienza', Italy*

Vladimiro Sassone, *University of Southampton, UK*

Subline Advisory Board

Susanne Albers, *University of Freiburg, Germany*

Benjamin C. Pierce, *University of Pennsylvania, USA*

Bernhard Steffen, *University of Dortmund, Germany*

Madhu Sudan, *Microsoft Research, Cambridge, MA, USA*

Deng Xiaotie, *City University of Hong Kong*

Jeannette M. Wing, *Carnegie Mellon University, Pittsburgh, PA, USA*

Lars Birkedal (Ed.)

Foundations of Software Science and Computational Structures

15th International Conference, FOSSACS 2012
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2012
Tallinn, Estonia, March 24 – April 1, 2012
Proceedings

Volume Editor

Lars Birkedal
IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen, Denmark
E-mail: birkedal@itu.dk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-28728-2 e-ISBN 978-3-642-28729-9
DOI 10.1007/978-3-642-28729-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012932640

CR Subject Classification (1998): F.3, F.1, F.4, D.3, D.2, I.2.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2012

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

ETAPS 2012 is the fifteenth instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference that was established in 1998 by combining a number of existing and new conferences. This year it comprised six sister conferences (CC, ESOP, FASE, FOSSACS, POST, TACAS), 21 satellite workshops (ACCAT, AIPA, BX, BYTECODE, CMCS, DICE, FESCA, FICS, FIT, GRAPHITE, GT-VMT, HAS, IWIGP, LDTA, LINEARITY, MBT, MSFP, PLACES, QAPL, VSSE and WRLA), and eight invited lectures (excluding those specific to the satellite events).

The six main conferences received this year 606 submissions (including 21 tool demonstration papers), 159 of which were accepted (6 tool demos), giving an overall acceptance rate just above 26%. Congratulations therefore to all the authors who made it to the final programme! I hope that most of the other authors will still have found a way to participate in this exciting event, and that you will all continue to submit to ETAPS and contribute to making it the best conference on software science and engineering.

The events that comprise ETAPS address various aspects of the system development process, including specification, design, implementation, analysis, security and improvement. The languages, methodologies and tools that support these activities are all well within its scope. Different blends of theory and practice are represented, with an inclination towards theory with a practical motivation on the one hand and soundly based practice on the other. Many of the issues involved in software design apply to systems in general, including hardware systems, and the emphasis on software is not intended to be exclusive.

ETAPS is a confederation in which each event retains its own identity, with a separate Programme Committee and proceedings. Its format is open-ended, allowing it to grow and evolve as time goes by. Contributed talks and system demonstrations are in synchronised parallel sessions, with invited lectures in plenary sessions. Two of the invited lectures are reserved for ‘unifying’ talks on topics of interest to the whole range of ETAPS attendees. The aim of cramming all this activity into a single one-week meeting is to create a strong magnet for academic and industrial researchers working on topics within its scope, giving them the opportunity to learn about research in related areas, and thereby to foster new and existing links between work in areas that were formerly addressed in separate meetings.

This year, ETAPS welcomes a new main conference, *Principles of Security and Trust*, as a candidate to become a permanent member conference of ETAPS. POST is the first addition to our main programme since 1998, when the original five conferences met in Lisbon for the first ETAPS event. It combines the practically important subject matter of security and trust with strong technical connections to traditional ETAPS areas.

A step towards the consolidation of ETAPS and its institutional activities has been undertaken by the Steering Committee with the establishment of *ETAPS e.V.*, a non-profit association under German law. ETAPS e.V. was founded on April 1st, 2011 in Saarbrücken, and we are currently in the process of defining its structure, scope and strategy.

ETAPS 2012 was organised by the *Institute of Cybernetics at Tallinn University of Technology*, in cooperation with

- ▷ European Association for Theoretical Computer Science (EATCS)
- ▷ European Association for Programming Languages and Systems (EAPLS)
- ▷ European Association of Software Science and Technology (EASST)

and with support from the following sponsors, which we gratefully thank:

INSTITUTE OF CYBERNETICS AT TUT; TALLINN UNIVERSITY OF TECHNOLOGY (TUT); ESTONIAN CENTRE OF EXCELLENCE IN COMPUTER SCIENCE (EXCS) FUNDED BY THE EUROPEAN REGIONAL DEVELOPMENT FUND (ERDF); ESTONIAN CONVENTION BUREAU; and MICROSOFT RESEARCH.

The organising team comprised:

General Chair: *Tarmo Uustalu*

Satellite Events: *Keiko Nakata*

Organising Committee: *James Chapman, Juhan Ernits, Tiina Laasma, Monika Perkmann* and their colleagues in the *Logic and Semantics* group and *administration* of the *Institute of Cybernetics*

The ETAPS portal at <http://www.etaps.org> is maintained by *RWTH Aachen University*.

Overall planning for ETAPS conferences is the responsibility of its Steering Committee, whose current membership is:

Vladimiro Sassone (Southampton, Chair), Roberto Amadio (Paris 7), Gilles Barthe (IMDEA-Software), David Basin (Zürich), Lars Birkedal (Copenhagen), Michael O'Boyle (Edinburgh), Giuseppe Castagna (CNRS Paris), Vittorio Cortellessa (L'Aquila), Koen De Bosschere (Gent), Pierpaolo Degano (Pisa), Matthias Felleisen (Boston), Bernd Finkbeiner (Saarbrücken), Cormac Flanagan (Santa Cruz), Philippa Gardner (Imperial College London), Andrew D. Gordon (MSR Cambridge and Edinburgh), Daniele Gorla (Rome), Joshua Guttman (Worcester USA), Holger Hermanns (Saarbrücken), Mike Hinchey (Lero, the Irish Software Engineering Research Centre), Ranjit Jhala (San Diego), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Jens Knoop (Vienna), Barbara König (Duisburg), Juan de Lara (Madrid), Gerald Lüttgen (Bamberg), Tiziana Margaria (Potsdam), Fabio Martinelli (Pisa), John Mitchell (Stanford), Catuscia Palamidessi (INRIA Paris), Frank Pfenning (Pittsburgh), Nir Piterman (Leicester), Don Sannella (Edinburgh), Helmut Seidl (TU Munich),

Scott Smolka (Stony Brook), Gabriele Taentzer (Marburg), Tarmo Uustalu (Tallinn), Dániel Varró (Budapest), Andrea Zisman (London), and Lenore Zuck (Chicago).

I would like to express my sincere gratitude to all of these people and organisations, the Programme Committee Chairs and PC members of the ETAPS conferences, the organisers of the satellite events, the speakers themselves, the many reviewers, all the participants, and Springer-Verlag for agreeing to publish the ETAPS proceedings in the ARCoSS subline.

Finally, I would like to thank the Organising Chair of ETAPS 2012, Tarmo Uustalu, and his Organising Committee, for arranging to have ETAPS in the most beautiful surroundings of Tallinn.

January 2012

Vladimiro Sassone
ETAPS SC Chair

Preface

FoSSaCS presents original papers on the foundations of software science. The Program Committee (PC) invited submissions on theories and methods to support analysis, synthesis, transformation, and verification of programs and software systems. We received 100 full-paper submissions; of these, 29 were selected for presentation at FoSSaCS and inclusion in the proceedings. Also included are two invited papers, one by Cali, Gottlob, Orsi, and Pieris on “Querying UML Class Diagrams,” presented by the ETAPS 2012 invited speaker Georg Gottlob; and one by Glynn Winskel, the FoSSaCS 2012 invited speaker, on “Bicategories of Concurrent Games.”

I thank all the authors of papers submitted to FoSSaCS 2012; the quality of the submissions was very high indeed, and the Program Committee had to reject several good papers. I thank also the members of the PC for their excellent work, as well as the external reviewers for the expert help and reviews they provided. Throughout the phases of submission, evaluation, and production of the proceedings, we relied on the invaluable assistance of the EasyChair system; we are very grateful to its developer Andrei Voronkov and his team. Last but not least, we would like to thank the ETAPS 2012 Local Organizing Committee (chaired by Tarmo Uustalu) and the ETAPS Steering Committee (chaired by Vladimiro Sassone) for their efficient coordination of all the activities leading up to FoSSaCS 2012.

January 2012

Lars Birkedal

Organization

Program Committee

Luca Aceto	Reykjavik University, Iceland
Roberto Amadio	University of Paris 7, France
Torben Amtoft	Kansas State University, USA
Lars Birkedal	IT University of Copenhagen, Denmark
Mikolaj Bojanczyk	Warsaw University, Poland
Thierry Coquand	Chalmers University, Sweden
Andrea Corradini	University of Pisa, Italy
Volker Diekert	University of Stuttgart, Germany
Maribel Fernandez	King's College London, UK
Kohei Honda	Queen Mary, University of London, UK
Bart Jacobs	Radboud University of Nijmegen, The Netherlands
Joost-Pieter Katoen	RWTH Aachen University, Germany
Olivier Laurent	ENS Lyon, France
Rupak Majumdar	Max Planck Institute for Software Systems, Germany
Markus Mueller-Olm	University of Muenster, Germany
Hanne Riis Nielson	Technical University of Denmark
Joachim Parrow	Uppsala University, Sweden
Dusko Pavlovic	University of Oxford, UK
Alex Simpson	University of Edinburgh, UK
Carolyn Talcott	SRI International, USA
Yde Venema	University of Amsterdam, The Netherlands
Thomas Vojnar	Brno University, Czech Republic

Additional Reviewers

Alves, Sandra	Berdine, Josh
Alvim, Mario S.	Bernardo, Marco
Andres, Miguel E.	Boker, Udi
Aranda, Jesus	Bollig, Benedikt
Asperti, Andrea	Boreale, Michele
Atkey, Robert	Borgstroem, Johannes
Bae, Kyungmin	Bouajjani, Ahmed
Bahr, Patrick	Bouyer, Patricia
Baillot, Patrick	Boyer, Benoit
Baldan, Paolo	Bradfield, Julian
Baltazar, Pedro	Brock-Nannestad, Taus
Barany, Vince	Brotherston, James

Bruggink, H.J. Sander
Bruni, Roberto
Bruyère, Véronique
Bucciarelli, Antonio
Caires, Luis
Carbone, Marco
Chlipala, Adam
Chockler, Hana
Ciancia, Vincenzo
Cimini, Matteo
Cirstea, Corina
Clairambault, Pierre
Clarkson, Michael
Compton, Kevin
De Liguoro, Ugo
De Nicola, Rocco
De Vries, Edsko
De-Falco, Marc
Debois, Soren
Demangeon, Romain
Deng, Yuxin
Di Gianantonio, Pietro
Donaldson, Robin
Dragoi, Cezara
Droste, Manfred
Dybjær, Peter
Faggian, Claudia
Fearnley, John
Ferrari, Gianluigi
French, Tim
Fu, Hongfei
Gadducci, Fabio
Galmiche, Didier
Gambino, Nicola
Gelderie, Marcus
Geuvers, Herman
Goldwurm, Massimiliano
Goubault-Larrecq, Jean
Groote, Jan Friso
Guerrini, Stefano
Gumm, H. Peter
Gutkovas, Ramunas
Habermehl, Peter
Hansen, Helle Hvid
Hasuo, Ichiro

Heindel, Tobias
Hemaspaandra, Edith
Herbelin, Hugo
Hermann, Frank
Hillston, Jane
Holik, Lukas
Hyvernât, Pierre
Iosif, Radu
Jansen, David N.
Jha, Susmit
Jibladze, Mamuka
Jonsson, Bengt
Jurdzinski, Marcin
Kaiser, Lukasz
Kara, Ahmet
Kartzow, Alexander
Klin, Bartek
Knapik, Teodor
Koenig, Barbara
Krishnaswami, Neelakantan
Kucera, Antonin
Kufleitner, Manfred
Kupferman, Orna
Kupke, Clemens
Kurz, Alexander
La Torre, Salvatore
Laird, Jim
Lange, Martin
Lasota, Slawomir
Lauser, Alexander
Leal, Raul
Lenisa, Marina
Leroux, Jerome
Leucker, Martin
Lluch Lafuente, Alberto
Longley, John
Loreti, Michele
Luo, Zhaohui
Luttenberger, Michael
Mardare, Radu
Marion, Jean-Yves
Markey, Nicolas
Meinecke, Ingmar
Merro, Massimo
Miculan, Marino

Mogelberg, Rasmus
Monreale, Giacoma
Montanari, Ugo
Moss, Larry
Mostrous, Dimitris
Myers, Robert
Negri, Sara
Neis, Georg
Nielson, Flemming
Noll, Thomas
Nordhoff, Benedikt
Ogawa, Mizuhito
Otop, Jan
Otto, Martin
Ouaknine, Joel
Palamidessi, Catuscia
Panangaden, Prakash
Parys, Pawel
Petersen, Rasmus Lerchedahl
Place, Thomas
Pollack, Randy
Polonsky, Andrew
Riba, Colin
Rypacek, Ondrej
Sack, Joshua
Salvati, Sylvain
Sangiorgi, Davide
Sangnier, Arnaud
Sasse, Ralf
Sewell, Peter
Sharma, Arpit
Sher, Falak
Silva, Alexandra
Simmons, Robert
Skrypnyuk, Nataliya
Skrzypczak, Michal
Sobocinski, Pawel
Staton, Sam
Stephan, Frank
Stirling, Colin
Streicher, Thomas
Svendsen, Kasper
Swierstra, Wouter
Tabareau, Nicolas
Ten Cate, Balder
Terui, Kazushige
Tiu, Alwen
Toruńczyk, Szymon
Turon, Aaron
Vaux, Lionel
Victor, Björn
Vogler, Walter
Wahl, Thomas
Walter, Tobias
Weiss, Armin
Wenner, Alexander
Winkel, Glynn
Wolter, Uwe
Yoshida, Nobuko
Zhang, Fuyuan
Zhang, Lijun

Table of Contents

Querying UML Class Diagrams (Invited Paper)	1
<i>Andrea Calì, Georg Gottlob, Giorgio Orsi, and Andreas Pieris</i>	
Bicategories of Concurrent Games (Invited Paper)	26
<i>Glynn Winskel</i>	
Fibrational Induction Meets Effects	42
<i>Robert Atkey, Neil Ghani, Bart Jacobs, and Patricia Johann</i>	
A Coalgebraic Perspective on Minimization and Determinization	58
<i>Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva</i>	
When Is a Container a Comonad?	74
<i>Danel Ahman, James Chapman, and Tarmo Uustalu</i>	
Well-Pointed Coalgebras (Extended Abstract)	89
<i>Jiří Adámek, Stefan Milius, Lawrence S. Moss, and Lurdes Sousa</i>	
Combining Interactive and Automatic Reasoning in First Order Theories of Functional Programs	104
<i>Ana Bove, Peter Dybjer, and Andrés Sicard-Ramírez</i>	
Applicative Bisimulations for Delimited-Control Operators	119
<i>Dariusz Biernacki and Sergueï Lenglet</i>	
Effective Characterizations of Simple Fragments of Temporal Logic Using Prophetic Automata	135
<i>Sebastian Preugschat and Thomas Wilke</i>	
Improved Ramsey-Based Büchi Complementation	150
<i>Stefan Breuers, Christof Löding, and Jörg Olschewski</i>	
Extending \mathcal{H}_1 -Clauses with Path Disequalities	165
<i>Helmut Seidl and Andreas Reuß</i>	
Brookes Is Relaxed, Almost!	180
<i>Radha Jagadeesan, Gustavo Petri, and James Riely</i>	
Revisiting Trace and Testing Equivalences for Nondeterministic and Probabilistic Processes	195
<i>Marco Bernardo, Rocco De Nicola, and Michele Loreti</i>	

Is It a “Good” Encoding of Mixed Choice?	210
<i>Kirstin Peters and Uwe Nestmann</i>	
Event Structure Semantics of Parallel Extrusion in the Pi-Calculus	225
<i>Silvia Crafa, Daniele Varacca, and Nobuko Yoshida</i>	
Narcissists Are Easy, Stepmothers Are Hard	240
<i>Daniel Gorín and Lutz Schröder</i>	
On Nominal Regular Languages with Binders	255
<i>Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto</i>	
Robustness of Structurally Equivalent Concurrent Parity Games	270
<i>Krishnendu Chatterjee</i>	
Subgame Perfection for Equilibria in Quantitative Reachability Games	286
<i>Thomas Brihaye, Véronique Bruyère, Julie De Pril, and Hugo Gimbert</i>	
Concurrent Games with Ordered Objectives	301
<i>Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels</i>	
Full Abstraction for Set-Based Models of the Symmetric Interaction Combinators	316
<i>Damiano Mazza and Neil J. Ross</i>	
On Distributability of Petri Nets (Extended Abstract)	331
<i>Rob van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann</i>	
Functions as Session-Typed Processes	346
<i>Bernardo Toninho, Luis Caires, and Frank Pfenning</i>	
Deriving Bisimulation Congruences for Conditional Reactive Systems	361
<i>Mathias Hülsbusch and Barbara König</i>	
First-Order Model Checking on Nested Pushdown Trees is Complete for Doubly Exponential Alternating Time	376
<i>Alexander Kartzow</i>	
Model Checking Languages of Data Words	391
<i>Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar</i>	
Branching-Time Model Checking of Parametric One-Counter Automata	406
<i>Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell</i>	

Synthesizing Probabilistic Composers	421
<i>Sumit Nain and Moshe Y. Vardi</i>	
On the Complexity of Computing Probabilistic Bisimilarity	437
<i>Di Chen, Franck van Breugel, and James Worrell</i>	
Probabilistic Transition System Specification: Congruence and Full Abstraction of Bisimulation	452
<i>Pedro Rubén D’Argenio and Matias David Lee</i>	
On the Complexity of the Equivalence Problem for Probabilistic Automata	467
<i>Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell</i>	
Author Index	483

Querying UML Class Diagrams

Andrea Cali^{2,3}, Georg Gottlob^{1,3,4}, Giorgio Orsi^{1,4}, and Andreas Pieris¹

¹ Department of Computer Science, University of Oxford, UK

² Dept. of Computer Science and Inf. Systems, Birkbeck University of London, UK

³ Oxford-Man Institute of Quantitative Finance, University of Oxford, UK

⁴ Institute for the Future of Computing, Oxford Martin School, UK


andrea@dcs.bbk.ac.uk,

{georg.gottlob,giorgio.orsi,andreas.pieris}@cs.ox.ac.uk

Abstract. UML Class Diagrams (UCDs) are the best known class-based formalism for conceptual modeling. They are used by software engineers to model the intensional structure of a system in terms of classes, attributes and operations, and to express constraints that must hold for every instance of the system. Reasoning over UCDs is of paramount importance in design, validation, maintenance and system analysis; however, for medium and large software projects, reasoning over UCDs may be impractical. Query answering, in particular, can be used to verify whether a (possibly incomplete) instance of the system modeled by the UCD, i.e., a snapshot, enjoys a certain property. In this work, we study the problem of querying UCD instances, and we relate it to query answering under guarded Datalog[±], that is, a powerful Datalog-based language for ontological modeling. We present an expressive and meaningful class of UCDs, named *Lean UCD*, under which conjunctive query answering is tractable in the size of the instances.

1 Introduction

Developing complex systems requires accurate design and early prototyping. To avoid the cost of fixing errors at later stages of a project, system designers use models of the final system to negotiate the system design, ensure all resulting requirements, and rule out unintended behavior manifesting itself during the system's lifetime. Models are also of paramount importance for software maintenance and for recovering the structure of a legacy and undocumented software.

UML Class Diagrams. The *Unified Modeling Language (UML)*  is one of the major tools for the design of complex systems such as software, business processes and even organizational structures. UML models were proposed by the *Object Modeling Group (OMG)* to diagrammatically represent the static and dynamic, e.g., behavioral aspects of a system, as well as the use cases the system will operate in. More specifically, *UML class diagrams (UCDs)* are widely used to represent the classes (types or entities) of a domain of interest, the associations (relationships) among them, their attributes (fields) and operations (methods).

¹ <http://www.omg.org/spec/UML/2.4.1/>

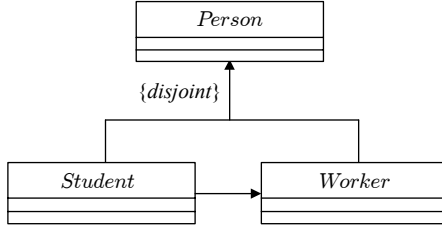


Fig. 1. A UCD which is not fully satisfiable

UCDs for complex projects become very large. Therefore, reasoning tasks such as verifying that a UCD is satisfiable, meaning that the model is realizable, can easily become unfeasible in practice. As a consequence, it is critical to adopt automated procedures to reason on the diagrams and to ensure that the final system will behave as planned.

When a UCD models a software system, the typical reasoning tasks include the following (see [7,37] for additional reasoning tasks):

1. Checking for *satisfiability* (or *consistency*) of the UCD, i.e., checking whether the class diagram admits at least one instantiation, that is, an instance of the system modeled by the UCD that satisfies the diagram.
2. Checking for *full satisfiability* (or *strong consistency*) of the UCD, i.e., checking whether the class diagram admits at least one instantiation where all classes and associations are non-empty. For example, consider the UCD \mathcal{G} of Figure 1, which expresses that each student is a worker, and also that students and workers are disjoint sets of persons. It is easy to see that \mathcal{G} is satisfiable, but not fully satisfiable since the class *Student* must be empty.
3. *Querying* the UCD, i.e., verifying whether a given property — expressed as a query — holds for a given instance of the system modeled by the UCD. This is the reasoning task that we address in this work.

(Full) Satisfiability of UCDs. The seminal work by Berardi et al. [7] established that reasoning on UCDs is hard. In fact, even satisfiability of UCDs is EXPTIME-complete w.r.t. the size of the given diagram. The EXPTIME-hardness is obtained by a reduction from the problem of checking the satisfiability of a concept in \mathcal{ALC} KBs [23,26]. The EXPTIME membership is obtained by providing a polynomial translation of UCD constructs to \mathcal{DLR}_{ifd} KBs [16].

The above results have been refined and extended to full satisfiability of UCDs by Artale et al. [2] and Kaneiwa et al. [30]; upper (resp., lower) bounds are obtained by a reduction to (resp., from) satisfiability of UCDs. In [2], classes of UCDs were identified for which satisfiability is NP-complete and NLOGSPACE-complete by restricting the constructs allowed in the diagrams. In [30], it has also been shown that there exists a fragment of UCDs for which full satisfiability is in PTIME. Moreover, in the same paper, a fragment of UCDs has been identified that allows to check satisfiability in constant time, since the expressive power of its constructs is not enough to capture any unsatisfiable UCD.

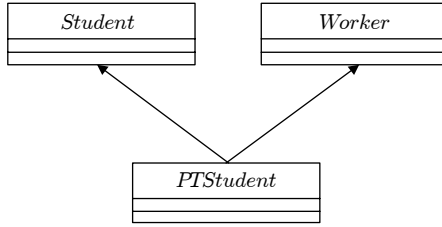


Fig. 2. The UCD of Example 1

Despite its expressiveness, the language of UCDs is often insufficient to express all constraints the designer would like to enforce. For this reason, the OMG consortium devised the *Object Constraint Language (OCL)*², which allows for expressing arbitrary constraints on UCDs. OCL is a powerful language, but is not widely adopted due to its complex syntax and ambiguous semantics.

Example 1. Consider the UCD shown in Figure 2 which represents the fact that part-time students are, at the same time, workers and regular students. However, the fact that whenever a student is also a worker, then necessarily (s)he must be a part-time student, cannot be expressed using UCDs. We can express such a constraint using the FOL expression

$$\forall X \text{ Student}(X) \wedge \text{Worker}(X) \rightarrow \text{PTStudent}(X).$$

Note that the FOL constraint above corresponds to the OCL expression

```

context PTStudent inv:
  Student.allInstances -> forAll ( s: Student |
    Worker.allInstances -> forAll ( w: Worker |
      s=w implies c.ocIsTypeOf(PTStudent)
    )
  )

```

For OCL syntax and semantics we refer the interested reader to [39]. ■

Reasoning on UML with OCL constraints is considerably harder. It is well known that satisfiability of UCDs extended with OCL constraints is undecidable since it amounts to checking satisfiability of arbitrary first-order formulas. Following Queralt et al. [36,38], the approaches can be classified into three families:

1. Unrestricted OCL constraints without guaranteeing termination, except for specific cases [24,37].
2. Unrestricted OCL constraints with terminating, but incomplete reasoning procedures [8,9,28,40].
3. Restricted classes of OCL constraints with both terminating and complete reasoning procedures [38].

² <http://www.omg.org/spec/OCL/2.3/>

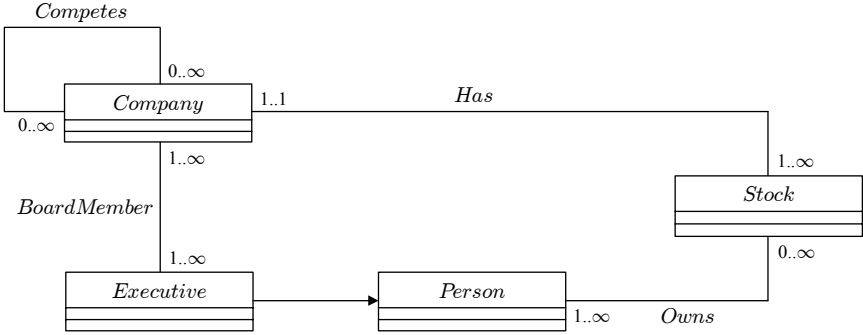


Fig. 3. UCD for the trading scenario

Querying UCDs. UCD satisfiability is an *intensional* property as it depends only on the class diagram and the OCL constraints, without involving any instance of the system modeled by the UCD. On the other hand, in many cases it is useful to reason over instances together with the diagram. For example, a typical task in specification recovery [20] is the reconstruction of a model of an unknown system involving low-level information obtained directly from a running instance of the system. The analyst usually starts from a partial specification (i.e., a UCD) and refines it based on information provided by instances, e.g., by verifying that they are consistent with the specification, and by adjusting the specification when they are not. However, instance-data collected during the analysis is partial but poorly structured. As a consequence they can easily become very large, emphasizing the importance of having procedures that are capable of handling very large instances. In this setting, query answering is a very useful tool for checking whether a property, not expressible diagrammatically, holds. More formally, given an instance D of a system modeled by a UCD \mathcal{G} , we can verify whether a property, represented as a query q , holds by checking whether q is a logical consequence of D and \mathcal{G} . This problem is known, in the knowledge representation (resp., database) community, as *ontological query answering* (resp., *query answering over incomplete databases*).

Example 2. Consider the UCD of Figure 3 representing a simplified high-frequency trading system. The diagram models companies and the associated stocks together with board members and stakeholders. The *conjunctive query*

$$\begin{aligned}
 \text{Conflict} \leftarrow & \text{Person}(P), \text{Company}(C_1), \text{Company}(C_2), \text{Stock}(S), \\
 & \text{BoardMember}(P, C_1), \text{Owns}(P, S), \text{Has}(C_2, S), \\
 & \text{Competes}(C_1, C_2)
 \end{aligned}$$

can be used to detect whether the system allows persons to be, at the same time, in the board of a company and owners of shares of a competing company, and are therefore operating in a conflict of interest. ■

Conjunctive query answering under UCDs and OCL constraints is undecidable in its general form, and has been addressed mainly by reducing it to answering

queries in known formalisms such as Prolog [21,41]. Decidable fragments of UCDs have been identified by comparing their expressive power with that of known description logics [3,14,18]. By leveraging on the results of Berardi et al. [7], Calvanese et al. [15] and Lutz [33], it is easy to see that the *combined complexity* of conjunctive query answering over UCDs (without OCL constraints), that is, the complexity w.r.t. the combined size of the the query, the system instance, and the corresponding diagram, is decidable and EXPTIME-complete; this is shown by a reduction from conjunctive query answering under \mathcal{ALC} KBs, and a reduction to conjunctive query answering under $\mathcal{DLR}_{f,id}$ KBs. Calvanese et al. [18] showed the coNP -completeness w.r.t. *data complexity*, i.e., the complexity calculated by considering only the system instance as part of the input, while the query and the diagram are considered fixed. Artale et al. [3] established that fragments of UCDs are captured by the description logic $DL\text{-}Lite_{horn}^{(HN)}$, for which query answering is NP-complete w.r.t. combined complexity, and in AC_0 w.r.t. data complexity. This result subsumes the tractability results for restricted classes of UCDs provided by [18].

Contributions. In this work, we study conjunctive query answering over UCDs and (a restricted class of) OCL constraints that are instrumental to the enforcement of specific assumptions that are commonly adopted for UCDs. We relate the problem to query answering under guarded Datalog $^\pm$ [11], a powerful Datalog-based ontological language under which query answering is not only decidable, but also tractable w.r.t. data complexity. In particular, we identify an expressive fragment of UCDs with a limited form of OCL constraints, named *Lean UCD*, which translates into guarded Datalog $^\pm$, that features tractable conjunctive query answering w.r.t. data complexity.

Roadmap. After providing some preliminary notions in Section 2, we introduce the UML class diagram formalism, and describe its semantics in terms of first-order logic in Section 3. In Section 4, we present Lean UCD, and we study query answering under the proposed formalism. Finally, Section 5 draws some conclusions and delineates future research directions.

2 Theoretical Background

As we shall see, query answering under UCDs can be reduced to query answering under relational constraints, in particular, tuple-generating dependencies. Therefore, in this section, we recall some basics on relational instances, (Boolean) conjunctive queries, tuple-generating dependencies, and the chase procedure relative to such dependencies.

Alphabets. We define the following pairwise disjoint (infinite) sets of symbols: a set Γ of *constants*, that constitute the “normal” domain of a database, a set Γ_N of *labeled nulls*, used as placeholders for unknown values, and thus can be also seen as (globally) existentially-quantified variables, and a set Γ_V of (regular) *variables*, used in queries and dependencies. Different constants represent different values (*unique name assumption*), while different nulls may represent the same value.

A lexicographic order is defined on $\Gamma \cup \Gamma_N$, such that every value in Γ_N follows all those in Γ . We denote by \mathbf{X} sequences (or sets, with a slight abuse of notation) of variables or constants X_1, \dots, X_k , with $k \geq 0$. Throughout, let $[n] = \{1, \dots, n\}$, for any integer $n \geq 1$.

Relational Model. A *relational schema* \mathcal{R} (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \dots, t_n)$, where r is an n -ary relation and t_1, \dots, t_n are terms. Conjunctions of atoms are often identified with the sets of their atoms. A *relational instance* (or simply *instance*) I for a schema \mathcal{R} is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where r is an n -ary predicate of \mathcal{R} and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. A *database* is a finite relational instance.

Substitutions and Homomorphisms. A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \rightarrow S_2$ defined as follows: \emptyset is a substitution (empty substitution), and if h is a substitution, then $h \cup \{X \rightarrow Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$. If $X \rightarrow Y \in h$, then we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2 is a substitution h from the set of terms of A_1 to the set of terms of A_2 such that: if $t \in \Gamma$, then $h(t) = t$, and if $r(t_1, \dots, t_n)$ is in A_1 , then $h(r(t_1, \dots, t_n)) = r(h(t_1), \dots, h(t_n))$ is in A_2 .

(Boolean) Conjunctive Queries. A *conjunctive query (CQ)* q of arity n over a schema \mathcal{R} , written as q/n , is an assertion the form $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over \mathcal{R} , and q is an n -ary predicate that does not occur in \mathcal{R} . $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q , denoted as $body(q)$. A *Boolean conjunctive query (BCQ)* is a CQ of arity zero. The *answer* to a CQ q/n over an instance I , denoted as $q(I)$, is the set of all n -tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \rightarrow \Gamma \cup \Gamma_N$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle \rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over I , denoted as $I \models q$, iff $\langle \rangle \in q(I)$, or, equivalently, $q(I) \neq \emptyset$.

Tuple-Generating Dependencies. A *tuple-generating dependency (TGD)* σ over a schema \mathcal{R} is a first-order formula $\forall \mathbf{X} \forall \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over \mathcal{R} , called the *body* and the *head* of σ , and denoted as $body(\sigma)$ and $head(\sigma)$, respectively. Such σ is satisfied by an instance I for \mathcal{R} , written as $I \models \sigma$, iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, then there exists an extension h' of h , i.e., $h' \supseteq h$, such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq I$. We write $I \not\models \sigma$ if I violates σ . Given a set of TGDs Σ , we say that I satisfies Σ , denoted as $I \models \Sigma$, iff I satisfies all the TGDs of Σ . Conversely, we say that I violates Σ , written as $I \not\models \Sigma$, iff I violates at least one TGD of Σ .

Query Answering under TGDs. Given a database D for a schema \mathcal{R} , and a set of TGDs Σ over \mathcal{R} , the answers we consider are those that are true in *all* models of D w.r.t. Σ , i.e., all instances that contain D and satisfy Σ . Formally,

the *models* of D w.r.t. Σ , denoted as $\text{mods}(D, \Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$. The *answer* to a CQ q/n w.r.t. D and Σ , denoted as $\text{ans}(q, D, \Sigma)$, is the set of n -tuples $\{\mathbf{t} \mid \mathbf{t} \in q(I), \text{ for each } I \in \text{mods}(D, \Sigma)\}$. The answer to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, iff $\langle \rangle \in \text{ans}(q, D, \Sigma)$, or, equivalently, $\text{ans}(q, D, \Sigma) \neq \emptyset$.

Given a CQ q/n over a schema \mathcal{R} , a database D for \mathcal{R} , a set Σ of TGDs over \mathcal{R} , and an n -tuple $\mathbf{t} \in \Gamma^n$, CQAns is defined as the problem whether $\mathbf{t} \in \text{ans}(q, D, \Sigma)$. In case that q is a BCQ (and thus, \mathbf{t} is the empty tuple $\langle \rangle$), the above problem is called BCQAns. Notice that these two problems under general TGDs are undecidable [6], even when the schema and the set of TGDs are fixed [10], or even when the set of TGDs is restricted to a single rule [4]. Following Vardi's taxonomy [42], the *data complexity* of the above problems is the complexity calculated taking only the database as input, while the query and the set of dependencies are considered fixed. The *combined complexity* is the complexity calculated considering as input, together with the database, also the query and the set of dependencies.

It is well-known that the above decision problems are LOGSPACE-equivalent; this result is implicit in [19], and stated explicitly in [10]. Henceforth, we thus focus only on BCQAns, and all complexity results carry over to CQAns.

The TGD Chase Procedure. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [34], and later for checking query containment [29]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted instance satisfies the dependencies. By abuse of terminology, we shall use the term “chase” interchangeably for both the procedure and its result. The building block of the chase procedure is the so-called *TGD chase rule*.

Definition 1 (TGD Chase Rule). *Consider an instance I for a schema \mathcal{R} , and a TGD $\sigma : \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$ over \mathcal{R} . If σ is applicable to I , i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, but there is no extension h' of h (i.e., $h' \supseteq h$) that maps $\psi(\mathbf{X}, \mathbf{Z})$ to I , then: (i) define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_N$ is a “fresh” labeled null not introduced before, and following lexicographically all those introduced so far, and (ii) add to I the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$, if not already in I .*

Given a database D and a set Σ of TGDs, the chase algorithm for D w.r.t. Σ consists of an exhaustive application of the TGD chase rule, which leads to a (possibly infinite) instance denoted as $\text{chase}(D, \Sigma)$. We assume that the chase algorithm is *fair*, i.e., each TGD that must be applied during the construction of $\text{chase}(D, \Sigma)$ is eventually applied.

Example 3. Consider the set Σ constituted by the TGDs $\sigma_1 : \forall X, Y r(X, Y) \wedge s(Y) \rightarrow \exists Z r(Z, X)$ and $\sigma_2 : \forall X, Y r(X, Y) \rightarrow s(X)$. Let D be the database $\{r(a, b), s(b)\}$. During the chase of D w.r.t. Σ , we first apply σ_1 and we add the atom $r(z_1, a)$, where $z_1 \in \Gamma_N$. Also, σ_2 is applicable and we add the atom $s(a)$. Now, σ_1 is applicable and the atom $r(z_2, z_1)$ is obtained, where $z_2 \in \Gamma_N$.

Then, σ_2 is applicable and the atom $s(z_1)$ is generated. It is straightforward to see that there is no finite chase. Satisfying both TGDs σ_1 and σ_2 would require to build the infinite instance $\{r(a, b), s(b), r(z_1, a), s(a), r(z_2, z_1), s(z_1), r(z_3, z_2), s(z_2), \dots\}$, where, for each $i > 0$, $z_i \in I_N$. ■

The fact that the chase algorithm is fair allows us to show that chase of a database D w.r.t. a set of TGDs Σ is a *universal model* of D w.r.t. Σ , i.e., for each $I \in \text{mods}(D, \Sigma)$, there exists a homomorphism from $\text{chase}(D, \Sigma)$ to I [22,25]. Using this fact it can be shown that the chase is a useful tool for query answering under TGDs. More precisely, the problem whether the answer to a BCQ q is positive w.r.t. a database D and a set of TGDs Σ , is equivalent to the problem whether q is entailed by the chase of D w.r.t. Σ .

Theorem 1 ([22,25]). *Consider a BCQ q over a schema \mathcal{R} , a database D for \mathcal{R} , and a set Σ of TGDs over \mathcal{R} . $D \cup \Sigma \models q$ iff $\text{chase}(D, \Sigma) \models q$.*

Guarded Datalog $^\pm$. Since query answering under TGDs is undecidable, several classes of TGDs have been proposed under which the problem becomes decidable, and even tractable w.r.t. data complexity (see, e.g., [4,25,31]). In particular, Datalog $^\pm$ [12] is a family of languages for ontological modeling based on classes of TGDs under which query answering is decidable and, in almost all cases, tractable w.r.t. data complexity. Datalog $^\pm$ proved to be a valid alternative to description logics in many database and knowledge management applications.

A member of the Datalog $^\pm$ family which is of special interest for our work is *guarded Datalog $^\pm$* [10,11]. A TGD σ is *guarded* if it has a body-atom which contains all the universally quantified variables of σ . Such atom is called the *guard atom* (or simply *guard*) of σ . The non-guard atoms are the *side atoms* of σ . For example, the TGD $r(X, Y), s(Y, X, Z) \rightarrow \exists W s(Z, X, W)$ is guarded (via the guard $s(Y, X, Z)$), while the TGD $r(X, Y), r(Y, Z) \rightarrow r(X, Z)$ is not guarded. Note that sets of guarded TGDs (with single-atom heads) are theories in the *guarded fragment* of first-order logic [1].

As shown in [10], the chase constructed under a set of guarded TGDs has finite treewidth, which, intuitively speaking, means that the chase is a tree-like structure. This is exactly the reason why query answering under guarded TGDs is decidable. The data and combined complexity of query answering under guarded TGDs have been investigated in [11] and [10], respectively.

Theorem 2 ([10,11]). *BCQAns under guarded TGDs is PTIME-complete w.r.t. data complexity, EXPTIME-complete in the case of bounded arity, and 2EXPTIME-complete w.r.t. combined complexity.*

3 UML Class Diagrams

As already mentioned in Section 1, UML class diagrams (UCDs) describe the static structure of a system by showing the system's classes, their attributes and operations, and the relationships among the classes. In this section, we describe

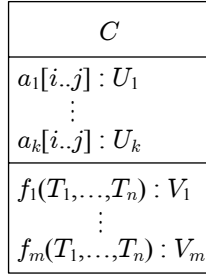


Fig. 4. Class representation

the semantics of each construct of UCDs in terms of first-order logic (FOL) generalized by counting quantifiers. The formalization adopted in this paper is based on the one presented in [30].

Classes. A *class* is graphically represented as shown in Figure 4, i.e., as a rectangle divided into three parts. The top part contains the *name* of the class which is unique in the diagram, the middle part contains the *attributes* of the class, and the bottom part contains the *operations* of the class, that is, the operations associated to the instances of the class. Note that both the middle and the bottom part are optional.

An *attribute assertion* of the form $a[i..j] : T$ states that the class C has an attribute a of *type*³ T , where the optional multiplicity $[i..j]$ specifies that a associates to each instance of C at least i and at most j instances of T . When there is no lower (resp., upper) bound on the multiplicity, the symbol 0 (resp., ∞) is used for i (resp., j). Notice that attributes are unique within a class. However, different classes may have attributes with the same name, possibly with different types.

An *operation* of a class C is a function from the instances of C , and possibly additional parameters, to objects and values. An *operation assertion* of the form $f(T_1, \dots, T_n) : T$ asserts that the class C has an operation f with $n \geq 0$ parameters, where its i -th parameter is of type T_i and its result is of type T . Let us clarify that the class diagram represents only the *signature*, that is, the name of the functions as well as the number and the types of their parameters, and the type of their result. The actual behavior of the function, which is not part of the diagram, can be represented using OCL constraints. Notice that operations are unique within a class. However, different classes may have operations with the same name, possibly with different signature, providing that they have the same number of parameters.

We are now ready to give the formal translation of a UML class definition into FOL. A class C is represented by a FOL unary predicate C . An attribute a for class C corresponds to a binary predicate a , and the attribute assertion

³ For simplicity, types, i.e., collections of values such as integers, are considered as classes, i.e., as collections of objects.

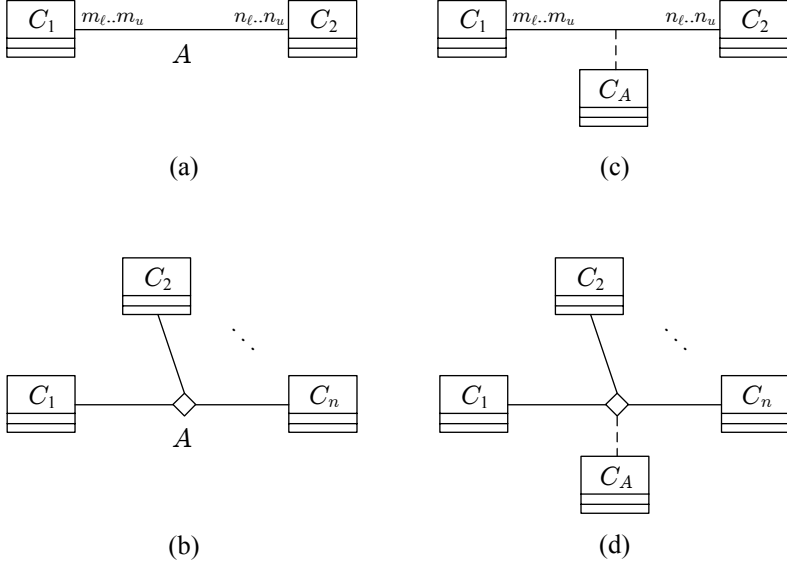


Fig. 5. Association representation

$a[i..j] : T$ is translated into two FOL assertions. The first one asserts that for each instance c of class C , an object c' related to c by the attribute a is an instance of T :

$$\forall X, Y C(X) \wedge a(X, Y) \rightarrow T(Y).$$

The second one asserts that for each instance c of class C , there exist at least i and at most j different objects related to c by a :

$$\forall X C(X) \rightarrow \exists_{\geq i} Z a(X, Z) \wedge \exists_{\leq j} Z a(X, Z).$$

If $i = 0$, which implies that there is no lower bound, then the corresponding FOL assertion is of the form $\forall X C(X) \rightarrow \exists_{\leq j} Z a(X, Z)$. Dually, if $j = \infty$, which implies that there is no upper bound, then the obtained assertion is of the form $\forall X C(X) \rightarrow \exists_{\geq i} Z a(X, Z)$.

An operation f , with $m \geq 0$ parameters, for class C corresponds to an $(m+2)$ -ary predicate f , and the operation assertion $f(T_1, \dots, T_m) : T$ is translated into the FOL assertions

$$\forall X, Y_1, \dots, Y_m, Z C(X) \wedge f(X, Y_1, \dots, Y_m, Z) \rightarrow \bigwedge_{i=1}^m T_i(Y_i) \wedge T(Z),$$

which imposes the correct typing for the parameters and the result, and

$$\begin{aligned} \forall X, Y_1, \dots, Y_m, Z_1, Z_2 C(X) \wedge f(X, Y_1, \dots, Y_m, Z_1) \\ \wedge f(X, Y_1, \dots, Y_m, Z_2) \rightarrow Z_1 = Z_2, \end{aligned}$$

i.e., the operation f is a function from the instances of C and the parameters to the result.

Associations. An *association* is a relation between the instances of two or more classes, that are said to *participate* in the association. Names of associations are unique in the diagram. A binary association A between two classes C_1 and C_2 is graphically represented as in Figure 5a. The multiplicity $n_\ell..n_u$ specifies that each instance of class C_1 can participate at least n_ℓ times and at most n_u times to A ; similarly we have the multiplicity $m_\ell..m_u$ for C_2 .

Clearly, we can have also n -ary associations which relate several classes as shown in Figure 5. As already discussed in [7], while multiplicity constraints in binary associations appear natural, for non-binary associations they do not correspond to an intuitive property of the multiplicity. Due to this fact, their presence in non-binary associations is awkward to a designer, and also they express a constraint which is (in general) too weak in practice. Therefore, in this paper, multiplicity in non-binary associations it is assumed to be always $0..∞$. Notice that in [30] arbitrary multiplicity constraints in non-binary associations are allowed.

An association can have an *association class* which describes properties of the association such as attributes and operations. A binary association between C_1 and C_2 with an association class C_A is graphically represented as in Figure 5c. An n -ary association can also have an association class as depicted in Figure 5d.

Let us now give the formal translation of an association definition into FOL. An n -ary association A corresponds to an n -ary predicate A . Assuming that A is among classes C_1, \dots, C_n , A is translated into the FOL assertion

$$\forall X_1, \dots, X_n A(X_1, \dots, X_n) \rightarrow \bigwedge_{i=1}^n C_i(X_i).$$

If A has a related association class C_A , then we have also the FOL assertions (in the sequel, r_n is an $(n + 1)$ -ary auxiliary predicate)

$$\forall X_1, \dots, X_n, Y A(X_1, \dots, X_n) \wedge r_n(X_1, \dots, X_n, Y) \rightarrow C_A(Y),$$

which types the association A ,

$$\forall X_1, \dots, X_n A(X_1, \dots, X_n) \rightarrow \exists Z r_n(X_1, \dots, X_n, Z),$$

i.e., for each instance $\langle x_1, \dots, x_n \rangle$ of A , there exists at least one object related to $\langle x_1, \dots, x_n \rangle$ by r_n ,

$$\begin{aligned} \forall X_1, \dots, X_n, Y_1, Y_2 A(X_1, \dots, X_n) \wedge r_n(X_1, \dots, X_n, Y_1) \\ \wedge r_n(X_1, \dots, X_n, Y_2) \rightarrow Y_1 = Y_2, \end{aligned}$$

that is, for each instance $\langle x_1, \dots, x_n \rangle$ of A , there exists at most one object related to $\langle x_1, \dots, x_n \rangle$ by r_n , and

$$\begin{aligned} \forall X_1, \dots, X_n, Y_1, \dots, Y_n, Z r_n(X_1, \dots, X_n, Z) \wedge r_n(Y_1, \dots, Y_n, Z) \\ \wedge C_A(Z) \rightarrow \bigwedge_{i=1}^n X_i = Y_i, \end{aligned}$$

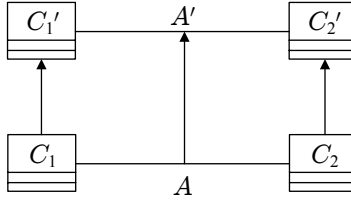


Fig. 6. Association generalization

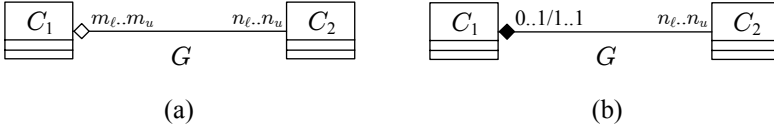


Fig. 7. Aggregation and composition

which imposes that there are no two different instances of C_A that are related by r_n with the same instance of A ; this rule guarantees the faithful representation of the association A by C_A , according to the original UML semantics.

Now, for a binary association A among C_1 and C_2 with multiplicities $m_\ell..m_u$ and $n_\ell..n_u$, we have also the FOL assertions

$$\begin{aligned} \forall X C_1(X) &\rightarrow \exists_{\geq n_\ell} Z A(X, Z) \wedge \exists_{\leq n_u} Z A(X, Z), \\ \forall X C_2(X) &\rightarrow \exists_{\geq m_\ell} Z A(Z, X) \wedge \exists_{\leq m_u} Z A(Z, X). \end{aligned}$$

We can also have *association generalization* such that an n -ary association A' between C'_1, \dots, C'_n generalizes the n -ary association A between C_1, \dots, C_n (see Figure 6 for the binary case). This feature is captured by the FOL assertion

$$\forall X_1, \dots, X_n A(X_1, \dots, X_n) \rightarrow A'(X_1, \dots, X_n).$$

A special kind of binary associations are *aggregations* and *compositions*, representing two different forms of *whole-part* or *part-of* relationship. An aggregation (see Figure 7a) between two classes C_1 and C_2 specifies that each instance of C_2 , called the *contained class*, is conceptually part of an instance of C_1 , called the *container class*; for example, a handle is part of a door. A composition (see Figure 7b) is more specific than aggregation. Composition has a strong *life cycle dependency* between instances of the container class and instances of the contained class. In particular, if the container is destroyed, then each instance that it contains is destroyed as well. Notice that the contained class of a composition must have a multiplicity of 0..1 or 1..1. Clearly, the life cycle dependency of a composition must be considered during the implementation phase of a system, however it is not relevant for query answering purposes. The translation of an aggregation and a composition into FOL is the same as the one given above for

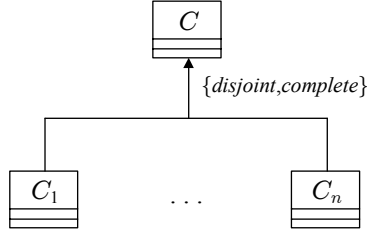


Fig. 8. A class hierarchy

binary associations without an association class (since aggregations and compositions have no association class).

Class Hierarchies. Similar to association generalization, one can use *class generalization* to assert that each instance of a child class is also an instance of the parent class. Several generalizations can be grouped together to form a class hierarchy, as shown in Figure 8. *Disjointness* and *completeness* constraints can also be enforced on a class hierarchy (graphically, by adding the labels $\{disjoint\}$ and $\{complete\}$). A class hierarchy, as the one in Figure 8, is translated into the FOL assertions

$$\begin{aligned} \forall X \ C_1(X) &\rightarrow C(X), \\ &\vdots \\ \forall X \ C_n(X) &\rightarrow C(X), \end{aligned}$$

i.e., each instance of C_i is also an instance of C ,

$$\forall X \ C_i(X) \rightarrow \bigwedge_{j=i+1}^n \neg C_j(X),$$

for each $i \in [n - 1]$, which specify the disjointness constraints, and

$$\forall X \ C(X) \rightarrow \bigvee_{i=1}^n C_i(X),$$

which specify the completeness constraints.

Sometimes, it is assumed that all classes in the same hierarchy are disjoint. However, we do not enforce this assumption, and we allow two classes to have common instances. When needed, disjointness can be enforced by means of FOL assertions, called *negative constraints*, of the form

$$\forall \mathbf{X} \ \varphi(\mathbf{X}) \rightarrow \perp,$$

where $\varphi(\mathbf{X})$ is a conjunction of atoms, and \perp denotes the truth constant *false*. Moreover, we do not enforce the *most specific class* assumption, stating that objects in a hierarchy must belong to a single most specific class. Therefore, two classes in a hierarchy may have common instances, even though they may not have a common subclass. When needed, the existence of the most specific class can be enforced by means of *multi-linear TGDs* [11] of the form

$$\forall X \ C_1(X) \wedge \dots \wedge C_n(X) \rightarrow C_{n+1}(X),$$

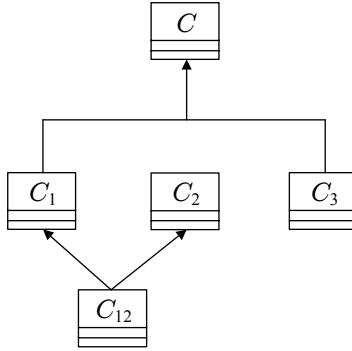


Fig. 9. A class hierarchy with most specific class assumption

where each C_i is a unary predicate representing a class. Observe that multi-linear TGDs are guarded TGDs where each body-atom is a guard. Notice that negative constraints and multi-linear TGDs can be represented using suitable OCL constraints.

For example, besides the assertions representing the hierarchy depicted in Figure 9 (taken from [7]), the most specific class assumption can be expressed by the following FOL assertions:

$$\begin{aligned}
 \forall X \ C_1(X) \wedge C_3(X) &\rightarrow \perp \\
 \forall X \ C_2(X) \wedge C_3(X) &\rightarrow \perp \\
 \forall X \ C_1(X) \wedge C_2(X) &\rightarrow C_{12}(X).
 \end{aligned}$$

4 Querying Lean UML Class Diagrams

The main goal of the present work is to study the problem of conjunctive query answering under UCDs. In particular, we are interested to identify an expressive fragment of UCDs which can be encoded in Datalog $^{\pm}$ so that chase-like techniques (for the chase algorithm see Section 2) and known complexity results can be employed.

In this section, we propose mild syntactic restrictions on the full version of UCDs, presented in Section 3, in order to get a fragment with the aforementioned desirable properties, called *Lean UCD*. We then study query answering under the proposed formalism. As we shall see, given a Lean UCD \mathcal{G} , by applying the translation of UCDs into FOL assertions given in the previous section — in the following we refer to this translation by τ — on \mathcal{G} , we get FOL assertions which have one of the following forms:

1. $\forall \mathbf{X} \forall \mathbf{Y} \ \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \ \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms, and $\varphi(\mathbf{X}, \mathbf{Y})$ has an atom that contains all the universally quantified variables; recall that an assertion of this form is called guarded TGD.

2. $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow X_i = X_j$, where $\varphi(\mathbf{X})$ is a conjunction of atoms, and both X_i and X_j are variables of \mathbf{X} ; assertions of this form are known as *equality-generating dependencies (EGDs)*.
3. $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow \perp$, where $\varphi(\mathbf{X})$ is a conjunction of atoms, and \perp denotes the truth constant *false*; recall that assertions of this form are called negative constraints.

The rest of this section is organized as follows. In Subsection 4.1, we formalize Lean UCDs by applying a series of syntactic restrictions on UCDs. Then, the problem of query answering under Lean UCDs is defined and studied in Subsection 4.2.

4.1 Formalizing Lean UCDs

Lean UCDs are obtained by restricting the multiplicity of attributes and binary associations, and by omitting completeness constraints. Formally, a UML class diagram \mathcal{G} is a Lean UCD if the following conditions hold:

1. For each attribute assertion of the form $a[i..j] : T$ of \mathcal{G} :
 - $i \in \{0, 1\}$ and $j \in \{1, \infty\}$.
2. For each binary association A of \mathcal{G} , either with or without an association class (consider the binary associations depicted in Figures 5a and 5c):
 - $n_\ell, m_\ell \in \{0, 1\}$ and $n_u, m_u \in \{1, \infty\}$,
 - if A generalizes some other binary association of \mathcal{G} , then $n_u = m_u = \infty$.
3. There are no completeness constraints in \mathcal{G} .

Example 4. The Lean UCD of Figure 10 describes members of a university department working in research groups. In particular, the class hierarchy specifies that students and professors, which are disjoint classes, are members of the department. A member of the department works in at least one research group, and at least one departmental member works in a research group. Moreover, a professor leads at most one research group, while a research group is led by exactly one professor; notice that a professor works in the same group that (s)he leads. Finally, a publication is authored by at least one member of the department. ■

Interestingly, the FOL assertions which represent multiplicities of attributes or multiplicities of binary associations, obtained by applying the translation τ on a Lean UCD \mathcal{G} , are guarded TGDs and EGDs. More precisely, from a class C with an attribute assertion $a[i..j] : T$ we get FOL assertions of the form

$$\begin{aligned} \forall X C(X) \rightarrow \exists_{\geq 1} Z a(X, Z) &\equiv \forall X C(X) \rightarrow \exists Z a(X, Z), \\ \forall X C(X) \rightarrow \exists_{\leq 1} Z a(X, Z) &\equiv \forall X, Y, Z C(X) \wedge a(X, Y) \wedge a(X, Z) \rightarrow Y = Z. \end{aligned}$$

From a binary association A among C_1 and C_2 with multiplicities $m_\ell..m_u$ and $n_\ell..n_u$, we get FOL assertions of the form

$$\begin{aligned} \forall X C_1(X) \rightarrow \exists_{\geq 1} Z A(X, Z) &\equiv \forall X C_1(X) \rightarrow \exists Z A(X, Z), \\ \forall X C_1(X) \rightarrow \exists_{\leq 1} Z A(X, Z) &\equiv \forall X, Y, Z C_1(X) \wedge A(X, Y) \wedge A(X, Z) \rightarrow Y = Z, \\ \forall X C_2(X) \rightarrow \exists_{\geq 1} Z A(Z, X) &\equiv \forall X C_2(X) \rightarrow \exists Z A(Z, X), \\ \forall X C_2(X) \rightarrow \exists_{\leq 1} Z A(Z, X) &\equiv \forall X, Y, Z C_2(X) \wedge A(Y, X) \wedge A(Z, X) \rightarrow Y = Z. \end{aligned}$$

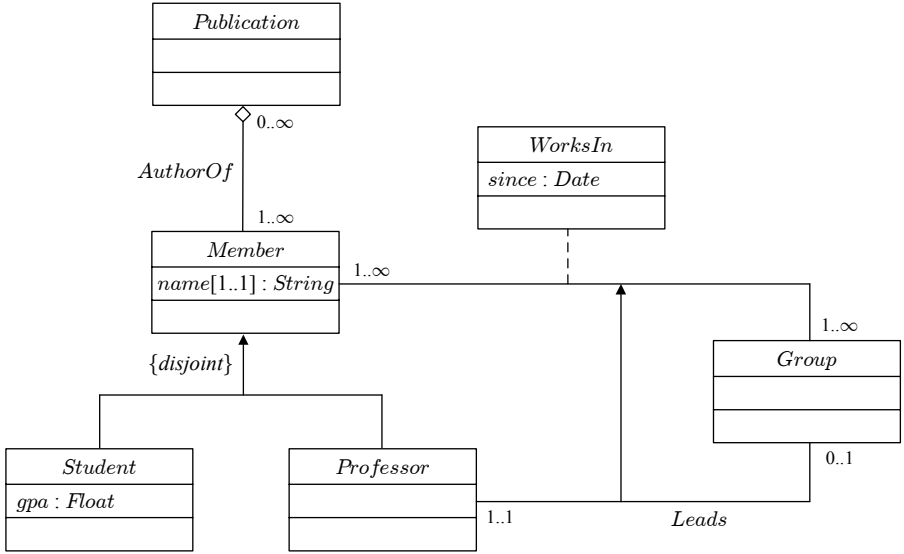


Fig. 10. Lean UCD of Example 4

It is an easy task to verify that, given a Lean UCD \mathcal{G} , the set of FOL assertions $\tau(\mathcal{G})$ is constituted by guarded TGDs, EGDs and negative constraints.

4.2 Query Answering under Lean UCDs

The problem of query answering under TGDs (discussed in Section 2) can be naturally extended to sets of TGDs, EGDs and negative constraints. An EGD $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow X_i = X_j$ is satisfied by a relational instance I if, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, then $h(X_i) = h(X_j)$. A negative constraint $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow \perp$ is satisfied by I if there is no homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$.

From the above discussion we immediately get that query answering under Lean UCDs is a well-defined problem. Formally, given a Lean UCD \mathcal{G} , a BCQ q (which represents a desirable property of the system; recall Example 2), and a relational database D (which is an instance of the system), the answer to q w.r.t. D and \mathcal{G} is positive, denoted as $D \cup \mathcal{G} \models q$, iff $\langle \rangle \in \text{ans}(q, D, \tau(\mathcal{G}))$, or, equivalently, $\text{ans}(q, D, \tau(\mathcal{G})) \neq \emptyset$.

In the rest of the paper, given a Lean UCD \mathcal{G} we denote by $\mathcal{R}_{\mathcal{G}}$ the relational schema associated to \mathcal{G} , i.e., the set of predicates occurring in $\tau(\mathcal{G})$, excluding the auxiliary predicates of the form r_i , where $i \geq 2$. Moreover, an instance of the system modeled by \mathcal{G} is considered as a (relational) database for $\mathcal{R}_{\mathcal{G}}$, while a property to be verified is encoded as a BCQ over $\mathcal{R}_{\mathcal{G}}$.

Elimination of EGDs and Negative Constraints. Recall that our main algorithmic tool for query answering is the chase algorithm. However, the chase

algorithm presented in Section 2 can treat only TGDs⁴, but in the set of FOL assertions that we obtain by applying τ on a Lean UCD \mathcal{G} we have also EGDs and negative constraints. Interestingly, as we shall see, providing that the logical theory constituted by the given database and the set of assertions $\tau(\mathcal{G})$ is consistent, we are allowed to eliminate the EGDs and the negative constraints, and proceed only with the TGDs.

Let us first concentrate on the EGDs. Consider a Lean UCD \mathcal{G} . For notational convenience, in the rest of this section, let $\tau_1(\mathcal{G})$ be the set of TGDs occurring in $\tau(\mathcal{G})$, $\tau_2(\mathcal{G})$ be the set of EGDs occurring in $\tau(\mathcal{G})$, and $\tau_3(\mathcal{G})$ be the set of negative constraints occurring in $\tau(\mathcal{G})$. The semantic notion of *separability* expresses a controlled interaction of TGDs and EGDs, so that the presence of EGDs does not play any role in query answering, and thus can be ignored [11,13]. Similarly, we can define separable Lean UCDs.

Definition 2 (Separability). *A Lean UCD \mathcal{G} is separable if, for every database D for $\mathcal{R}_{\mathcal{G}}$, either $D \not\models \tau_2(\mathcal{G})$, or $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \models q$, for every BCQ q over $\mathcal{R}_{\mathcal{G}}$.*

It is possible to show that *each* Lean UCD \mathcal{G} is separable. Lean UCDs, in fact, enjoy a stronger property than separability: during the construction of the chase of a database D for $\mathcal{R}_{\mathcal{G}}$ w.r.t. $\tau_1(\mathcal{G})$, it is not possible to violate an EGD of $\tau_2(\mathcal{G})$, and therefore, if D satisfies $\tau_2(\mathcal{G})$, then *chase*($D, \tau_1(\mathcal{G})$) is a universal model of D w.r.t. $\tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ which implies separability of \mathcal{G} .

Lemma 1. *Each Lean UCD is separable.*

Proof (sketch). Consider a Lean UCD \mathcal{G} , and a database D for $\mathcal{R}_{\mathcal{G}}$ such that $D \models \tau_2(\mathcal{G})$; if $D \not\models \tau_2(\mathcal{G})$, then the claim holds trivially. We need to show that $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \models q$, for every BCQ q over $\mathcal{R}_{\mathcal{G}}$. Observe that each model of the logical theory $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ is also a model of $D \cup \tau_1(\mathcal{G})$. Therefore, if $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$, then $D \cup \tau_1(\mathcal{G}) \models q$. It remains to establish the other direction. It suffices to show that, for each $i \geq 0$, *chase*^[i]($D, \tau_1(\mathcal{G})$), i.e., the initial segment of *chase*($D, \tau_1(\mathcal{G})$) obtained starting from D and applying i times the TGD chase rule (see Definition 1), does not violate any of the EGDs of $\tau_2(\mathcal{G})$. We proceed by induction on $i \geq 0$.

Base Step. Clearly, *chase*^[0]($D, \tau_1(\mathcal{G})$) = D , and the claim follows since $D \models \tau_2(\mathcal{G})$.

Inductive Step. Suppose that during the i -th application of the TGD chase rule we apply the TGD $\sigma \in \tau_1(\mathcal{G})$ with homomorphism λ , and the atom \underline{a} is obtained. Notice that each set Σ of guarded TGDs can be rewritten into an equivalent set Σ' of guarded TGDs, where each TGD of Σ' has just one head-atom [10]. Therefore, for technical clarity, we assume w.l.o.g. that each TGD of $\tau_1(\mathcal{G})$ has just one head-atom.

⁴ Notice that the chase algorithm can be extended to treat also EGDs (see, e.g., [25]). However, such an extended version of the chase algorithm is not needed for the purposes of the present paper.

Consider an EGD $\eta \in \tau_2(\mathcal{G})$. It suffices to show that there is no homomorphism which maps $body(\eta)$ to $chase^{[i]}(D, \tau_1(\mathcal{G}))$. Suppose that η is of the form $\forall X_1, \dots, X_n, Y_1, Y_2 A(X_1, \dots, X_n) \wedge r_n(X_1, \dots, X_n, Y_1) \wedge r_n(X_1, \dots, X_n, Y_2) \rightarrow Y_1 = Y_2$. We show that there is no homomorphism that maps the set of atoms $A = \{r_n(X_1, \dots, X_n, Y_1), r_n(X_1, \dots, X_n, Y_2)\} \subset body(\eta)$ to $chase^{[i]}(D, \tau_1(\mathcal{G}))$, and thus there is no homomorphism that maps $body(\eta)$ to $chase^{[i]}(D, \tau_1(\mathcal{G}))$. The critical case is when σ is of the form $\forall X_1, \dots, X_n A(X_1, \dots, X_n) \rightarrow \exists Z r_n(X_1, \dots, X_n, Z)$. In the sequel, let $\underline{a} = r_n(\mathbf{t})$, and consider an arbitrary atom $\underline{a}' = r_n(\mathbf{t}') \in chase^{[i-1]}(D, \tau_1(\mathcal{G}))$. Towards a contradiction, suppose that there exists a homomorphism that maps A to $\{\underline{a}, \underline{a}'\}$. This implies that there exists an extension of λ that maps $head(\sigma)$ to $\underline{a}' \in chase^{[i-1]}(D, \tau_1(\mathcal{G}))$, and hence σ is not applicable with homomorphism λ . But this contradicts the fact that σ has been applied during the i -th application of the TGD chase rule.

By providing a similar argument, we can establish the same fact for all the forms of EGDs that can appear in $\tau_2(\mathcal{G})$, and the claim follows. \square

Let us now focus on the negative constraints. Given a Lean UCD, checking whether $\tau_3(\mathcal{G})$ is satisfied by a database D for $\mathcal{R}_{\mathcal{G}}$ and the set of assertions $\tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ is tantamount to query answering; this is implicit in [11]. More precisely, for each negative constraint ν of the form $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow \perp$ of $\tau_3(\mathcal{G})$, we evaluate the BCQ $q_\nu : p \leftarrow \varphi(\mathbf{X})$ over $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$. If at least one of such queries answers positively, then the logical theory $D \cup \tau(\mathcal{G})$ is inconsistent, and thus $D \cup \mathcal{G} \models q$, for every BCQ q ; otherwise, given a BCQ q , it holds that $D \cup \tau(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$, i.e., we answer q by ignoring the negative constraints. The next lemma follows immediately.

Lemma 2. *Consider a Lean UCD \mathcal{G} , a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$. Then, $D \cup \mathcal{G} \models q$ iff (i) $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ or (ii) there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q_\nu$.*

By combining Lemmas 1 and 2, we immediately get the following useful technical result, which implies that query answering under Lean UCDs can be reduced to query answering under guarded TGDs, and thus chase-like techniques and existing complexity results can be employed.

Corollary 1. *Consider a Lean UCD \mathcal{G} , a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$. If $D \models \tau_2(\mathcal{G})$, then $D \cup \mathcal{G} \models q$ iff (i) $D \cup \tau_1(\mathcal{G}) \models q$ or (ii) there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$.*

Complexity of Query Answering. We are now ready to investigate the complexity of BCQAns under Lean UCDs. Before we proceed further, let us analyze the complexity of the problem of deciding whether a database violates the set of EGDs obtained from a Lean UCD.

Lemma 3. *Consider a Lean UCD \mathcal{G} , and a database D for $\mathcal{R}_{\mathcal{G}}$. The problem of deciding whether $D \not\models \tau_2(\mathcal{G})$ is feasible in PTIME if \mathcal{G} is fixed, and in NP in general.*

C
$a_1 : T_1$
$a_2 : T_2$
$a_3 : C$

Fig. 11. The construction in the proof of Theorem 3

Proof. It is possible to show that a database D' and a set of BCQs Q can be constructed in PTIME such that $D \not\models \tau_2(\mathcal{G})$ iff $D' \models q$, for some $q \in Q$. The database D' is constructed by adding to D an atom $neq(c_1, c_2)$, for each pair $\langle c_1, c_2 \rangle$ of distinct constants occurring in D , where neq is an auxiliary binary predicate not occurring in $\mathcal{R}_{\mathcal{G}}$. Clearly, the number of atoms that we need to add in D is at most n^2 , where n is the number of constants occurring in D , and thus D' can be constructed in polynomial time. The set Q of BCQs is defined as follows. For each EGD $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow X_i = X_j$ of $\tau_2(\mathcal{G})$, in Q there exists a BCQ $p \leftarrow \varphi(\mathbf{X}), neq(X_i, X_j)$, where p is a 0-ary auxiliary predicate. Clearly, Q can be constructed in linear time. Clearly, by construction, $D \not\models \tau_2(\mathcal{G})$ iff $D' \models q$, for some $q \in Q$. Since the evaluation of a BCQ over a database is feasible in AC₀ if the query is fixed [43], and in NP in general [19], the claim follows. \square

We continue to investigate the data complexity of BCQAns under Lean UCDs; recall that the data complexity is calculated by considering only the database as part of the input, while the query and the diagram are considered fixed.

Theorem 3. *BCQAns under Lean UCDs is PTIME-complete w.r.t. data complexity.*

Proof. By Corollary 1, given a Lean UCD \mathcal{G} , a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$, we can decide whether $D \cup \mathcal{G} \models q$ by applying the following simple algorithm: (1) if $D \not\models \tau_2(\mathcal{G})$, then answer *yes*; (2) if $D \cup \tau_1(\mathcal{G}) \models q$, then answer *yes*; (3) if there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$, then answer *yes*; (4) answer *no*. Since the diagram is fixed, Lemma 3 implies that the first step can be carried out in PTIME. Recall that $\tau_1(\mathcal{G})$ is a set of guarded TGDs. Since, by Theorem 2, BCQAns under guarded TGDs is feasible in PTIME w.r.t. data complexity, the claimed upper bound follows.

Let us now establish the PTIME-hardness of the problem under consideration. The proof is by reduction from *Path System Accessibility (PSA)* which is PTIME-hard [27]. An instance of PSA is a quadruple $\langle N, E, S, t \rangle$, where N is a set of nodes, $E \subseteq N \times N \times N$ is an accessibility relation (its elements are called *edges*), $S \subseteq N$ is a set of source nodes, and $t \in N$ is a terminal node. The question is whether t is accessible, where a node $v \in N$ is said to be accessible if $v \in S$ or there exist accessible nodes v_1 and v_2 s.t. $\langle v, v_1, v_2 \rangle \in E$.

Let \mathcal{G} be the Lean UCD depicted in Figure [III](#), which contains also the additional rule $\forall X T_1(X) \wedge T_2(X) \rightarrow C(X)$. Clearly, $\tau(\mathcal{G})$ is constituted by

$$\begin{aligned}\sigma_1 &: \forall X, Y C(X) \wedge a_1(X, Y) \rightarrow T_1(Y) \\ \sigma_2 &: \forall X, Y C(X) \wedge a_2(X, Y) \rightarrow T_2(Y) \\ \sigma_3 &: \forall X, Y C(X) \wedge a_3(X, Y) \rightarrow C(Y) \\ \sigma_4 &: \forall X T_1(X) \wedge T_2(X) \rightarrow C(X).\end{aligned}$$

For the construction of the database D we make use of the nodes and the edges of N and E , respectively, as constants. In particular, the domain of D is the set of constants $\{c_v \mid v \in N\} \cup \{c_e \mid e \in E\}$. For a node $v \in N$, let e_1^v, \dots, e_n^v , where $n \geq 0$, be all edges of E that have v as their first component (the order is not relevant). The database D contains, for each node $v \in N$, the following atoms:

- $a_3(c_{e_1^v}, c_v)$ and $a_3(c_{e_{i+1}^v}, c_{e_i^v})$, for each $i \in [n-1]$,
- $a_1(c_u, c_{e_i^v})$ and $a_2(c_w, c_{e_i^v})$, where $e_i^v = \langle v, u, w \rangle$, for each $i \in [n]$.

In addition, D contains an atom $C(c_v)$ for each $v \in S$. Finally, let q be the BCQ $p \leftarrow C(c_t)$. Intuitively speaking, T_1 keeps all the edges $\langle v_1, v_2, v_3 \rangle$ where v_2 is accessible, T_2 keeps all the edges $\langle v_1, v_2, v_3 \rangle$ where v_3 is accessible, and C keeps all the nodes which are accessible, and also all the edges $\langle v_1, v_2, v_3 \rangle$ where v_1 is accessible. Let us say that the above construction is similar to a construction given in [\[17\]](#), where the data complexity of query answering under description logics is investigated. It is easy to verify that \mathcal{G} , D and q can be constructed in LOGSPACE. Moreover, observe that only the database D depends on the given instance of PSA, while the diagram and the query are fixed. It remains to show that t is accessible iff $D \cup \mathcal{G} \models q$, or, equivalently, $\text{chase}(D, \tau(\mathcal{G})) \models q$. For brevity, in the rest of the proof, a constant c_x , where x is either a node of N or an edge of E , is denoted as x .

(\Rightarrow) Suppose first that t is accessible. It is possible to show, by induction on the derivation of accessibility, that if a node $v \in N$ is accessible, then $\text{chase}(D, \tau(\mathcal{G})) \models q_v$, where q_v is the BCQ $p \leftarrow C(v)$.

Base Step. Suppose that $v \in S$. By construction, $C(v) \in D$, and thus $\text{chase}(D, \tau(\mathcal{G})) \models q_v$.

Inductive Step. Clearly, there exists an edge $\langle v, v_1, v_2 \rangle \in E$, where both v_1 and v_2 are accessible. By induction hypothesis, both atoms $C(v_1)$ and $C(v_2)$ belong to $\text{chase}(D, \tau(\mathcal{G}))$. Let e_1, \dots, e_n be the edges of E that have v as their first component, where $e_n = \langle v, v_1, v_2 \rangle$; assume the same order used in the construction of the database D . Since the atoms $a_1(v_1, e_n)$ and $a_2(v_2, e_n)$ belong to D , and $\text{chase}(D, \tau(\mathcal{G})) \models \tau(\mathcal{G})$, the atoms $T_1(e_n)$ and $T_2(e_n)$ belong to $\text{chase}(D, \tau(\mathcal{G}))$. Therefore, due to the TGD σ_4 , $C(e_n) \in \text{chase}(D, \tau(\mathcal{G}))$. Now, by exploiting the atoms $\{a_3(e_{i+1}, e_i)\}_{i \in [n-1]} \subset D$ and the TGD σ_3 , it is easy to show that the atom $C(e_1)$ belongs to $\text{chase}(D, \tau(\mathcal{G}))$. Finally, since $a_3(e_1, v) \in \text{chase}(D, \tau(\mathcal{G}))$, we get that $C(v) \in \text{chase}(D, \tau(\mathcal{G}))$. Consequently, $\text{chase}(D, \tau(\mathcal{G})) \models q_v$.

(\Leftarrow) Suppose now that $\text{chase}(D, \tau(\mathcal{G})) \models q$. We need to show that t is accessible. We are going to show that, for each $v \in N$, if $C(v) \in \text{chase}(D, \tau(\mathcal{G}))$, then v

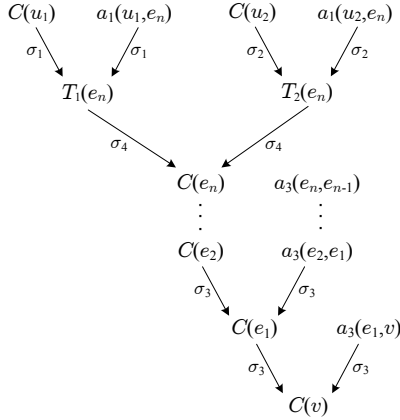


Fig. 12. The chase derivation in the proof of Theorem [3](#)

is accessible. Observe that the only way to obtain $C(v)$ during the construction of the chase is as depicted in Figure [12](#). Since $C(u_1)$ and $C(u_2)$ are database atoms, by construction, both u_1 and u_2 belong to S , and thus they are accessible. Since $e_n = \langle w, u_1, u_2 \rangle$, we immediately get that w is accessible. But observe that $w = v$, and hence v is accessible. This completes the proof. \square

We conclude this subsection by investigating the combined complexity of BCQAns under Lean UCDs; recall that the combined complexity is calculated by considering, apart from the database, also the query and the diagram as part of the input. As we shall see, query answering under Lean UCDs can be reduced to query answering under guarded TGDs of bounded arity, and thus (by Theorem [2](#)) we get an EXPTIME upper bound.

Theorem 4. *BCQAns under Lean UCDs is in EXPTIME w.r.t. combined complexity.*

Proof. As already discussed in the proof of Theorem [3](#), given a Lean UCD \mathcal{G} , a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$, we can decide whether $D \cup \mathcal{G} \models q$ by applying the following simple algorithm: (1) if $D \not\models \tau_2(\mathcal{G})$, then answer *yes*; (2) if $D \cup \tau_1(\mathcal{G}) \models q$, then answer *yes*; (3) if there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$, then answer *yes*; (4) answer *no*. By Lemma [3](#), we get that the first step can be carried out in NP. Now, in order to show that steps (2) and (3) can be carried out in EXPTIME, we are going to show that the problem of answering a BCQ under the set $\tau_1(\mathcal{G})$ can be reduced to query answering under a set of guarded TGDs of bounded arity, which is in EXPTIME (by Theorem [2](#)).

Any TGD of $\tau_1(\mathcal{G})$ which has one of the following forms is called *harmless*:

$$\begin{aligned}
 & \forall X_1, \dots, X_n \ A(X_1, \dots, X_n) \rightarrow C_1(X_1), \dots, C_n(X_n), \\
 & \forall X_1, \dots, X_n, Y \ A(X_1, \dots, X_n) \wedge r'(X_1, \dots, X_n, Y) \rightarrow C_A(Y), \\
 & \forall X_1, \dots, X_n \ A(X_1, \dots, X_n) \rightarrow \exists Z \ r'(X_1, \dots, X_n, Z), \\
 & \forall X_1, \dots, X_n \ A(X_1, \dots, X_n) \rightarrow A'(X_1, \dots, X_n).
 \end{aligned}$$

Let Σ_h be the set of harmless TGDs of $\tau_1(\mathcal{G})$. It is not difficult to see that, for every BCQ p over $\mathcal{R}_{\mathcal{G}}$, $D \cup \tau_1(\mathcal{G}) \models p$ iff $\text{chase}(D, \Sigma_h)_{\downarrow}, \tau_1(\mathcal{G}) \setminus \Sigma_h \models p$, where $\text{chase}(D, \Sigma_h)_{\downarrow}$ is the (finite) database obtained by *freezing* $\text{chase}(D, \Sigma_h)$, i.e., by replacing each null of Γ_N occurring in $\text{chase}(D, \Sigma_h)$ with a fresh constant of Γ . Now, observe that atoms of the form $f(c_1, \dots, c_n)$ are necessarily database atoms since the predicate f does not occur in the head of any of the TGDs of $\tau_1(\mathcal{G})$. Therefore, whenever a TGD of the form $\forall X, Y_1, \dots, Y_m, Z C(X) \wedge f(X, Y_1, \dots, Y_m, Z) \rightarrow T_1(Y_1), \dots, T_m(Y_m), T(Z)$ is applied during the chase, the variable X is mapped (via a homomorphism) to a constant. Thus, for each database atom $f(c_1, \dots, c_{m+2})$, we can replace the above rule with the rule $C(c_1) \rightarrow T_1(c_2), \dots, T_m(c_{m+1}), T(c_{m+2})$. Let Σ be the set obtained by applying the above transformation on the set $\tau_1(\mathcal{G}) \setminus \Sigma_h$. It holds that, $D \cup \tau_1(\mathcal{G}) \models p$ iff $\text{chase}(\text{chase}(D, \Sigma_h)_{\downarrow}, \Sigma) \models p$, for every BCQ p over $\mathcal{R}_{\mathcal{G}}$. It is not difficult to see that Σ is a set of guarded TGDs of bounded arity, and the claim follows. \square

Notice that the above theorem does not provide tight combined complexity bounds for the problem of query answering under Lean UCDs. The exact bound is currently under investigation, and the results will be presented in an upcoming work. Preliminary findings can be found in an online manuscript⁵.

Interestingly, both the data and combined complexity of query answering under Lean UCDs can be reduced by applying further restrictions. In particular, this can be achieved by assuming that each attribute and operation is associated to a unique class, i.e., different classes have disjoint sets of attributes and operations, and also an association A with an association class C_A does not generalize some other association A' . These assumptions allow us to establish the following: given a diagram \mathcal{G} , a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$, there exists a set Σ of multi-linear TGDs over a schema \mathcal{R} , where each predicate of \mathcal{R} has bounded arity, such that $D \cup \tau_1(\mathcal{G}) \models q$ iff $D \cup \Sigma \models q$; recall that a multi-linear TGD is a guarded TGD where each body-atom is a guard. Since query answering under multi-linear TGDs is in AC_0 w.r.t. data complexity and NP-complete in the case of bounded arity [11], we immediately get that query answering under Lean UCDs that satisfy the above assumptions is in AC_0 w.r.t. data complexity and NP-complete w.r.t. combined complexity.

5 Discussion and Future Work

In this work, we have studied the problem of query answering under UML class diagrams (UCDs) by relating it to the problem of query answering under guarded Datalog[±]. In particular, we have identified an expressive fragment of UCDs, called Lean UCD, under which query answering is PTIME-complete w.r.t. data complexity and in EXPTIME w.r.t. combined complexity.

In the immediate future we plan to investigate expressive, but still tractable, extensions of Lean UCD with additional constructs such as stratified negation.

⁵ <http://dl.dropbox.com/u/3185659/uml.pdf>

Moreover, we are planning to investigate the problem of query answering under Lean UCDs by considering only finite instantiations. This is an interesting and important research direction since, in practice, it is natural (for obvious reasons) to concentrate only on finite instances of a system.

Reasoning on UCDs by considering only finite instantiations has been studied first by Cadoli et al. [9] by building on results of Lenzerini and Nobili [32] established for the Entity-Relationship model. In particular, [9] proposes an encoding of satisfiability of UCDs under finite instantiations into a constraint satisfaction problem, showing that satisfiability of UCDs under finite instantiations is EXPTIME-complete. Later, more efficient algorithms based on linear programming were presented by Maraee and Balaban for specific instances of the problem [35]. Recently, Queralt et al. [38] extended the results of [9] to cope with a restricted form of OCL constraints, called *OCL-Lite*, and shown that satisfiability of UCDs and *OCL-Lite* constraints under finite instantiations is EXPTIME-complete.

An important notion related to query answering under UCDs and OCL constraints is the so-called *finite controllability*. A class of UCDs \mathcal{C} (possibly with OCL constraints) is finitely controllable if the following holds: given a diagram \mathcal{G} that falls in \mathcal{C} , an instance D of the system modeled by \mathcal{G} , and a query q , $D \cup \mathcal{G} \models q$ iff $D \cup \mathcal{G} \models_{fin} q$, i.e., q is entailed by $D \cup \mathcal{G}$ under arbitrary instantiations iff it is entailed by $D \cup \mathcal{G}$ under finite instantiations only. It is worthwhile to remark that by forbidding functional participation Lean UCD is finitely controllable. This is a consequence of the fact the guarded fragment of first-order logic, and thus guarded Datalog[±], is finitely controllable [5].

Acknowledgements. This research has received funding from the European Research Council under the European Community’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement DIADEM no. 246858 and from the Oxford Martin School’s grant no. LC0910-019.

References

1. Andr eka, H., van Benthem, J., N emeti, I.: Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic* 27, 217–274 (1998)
2. Artale, A., Calvanese, D., Ib a nez-Garc a, Y.A.: Full Satisfiability of UML Class Diagrams. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) *ER 2010. LNCS*, vol. 6412, pp. 317–331. Springer, Heidelberg (2010)
3. Artale, A., Calvanese, D., Kontchakov, R., Zakharyashev, M.: The DL-Lite family and relations. *J. Artificial Intelligence Res.* 36, 1–69 (2009)
4. Baget, J.-F., Lecl ere, M., Mugnier, M.-L., Salvat, E.: On rules with existential variables: Walking the decidability line. *Artif. Intell.* 175(9-10), 1620–1654 (2011)
5. B ar any, V., Gottlob, G., Otto, M.: Querying the guarded fragment. In: *Proc. of LICS*, pp. 1–10 (2010)
6. Beeri, C., Vardi, M.Y.: The Implication Problem for Data Dependencies. In: Even, S., Kariv, O. (eds.) *ICALP 1981. LNCS*, vol. 115, pp. 73–85. Springer, Heidelberg (1981)
7. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. *Artif. Intell.* 168(1-2), 70–118 (2005)

8. Cabot, J., Clariso, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: Proc. of ICSTW, pp. 73–80 (2008)
9. Cadoli, M., Calvanese, D., De Giacomo, G., Mancini, T.: Finite Model Reasoning on UML Class Diagrams Via Constraint Programming. In: Basili, R., Paziienza, M.T. (eds.) *AI*IA 2007*. LNCS (LNAI), vol. 4733, pp. 36–47. Springer, Heidelberg (2007)
10. Cali, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: Proc. of KR, pp. 70–80 (2008); Extended version available from the authors
11. Cali, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. In: Proc. of PODS, pp. 77–86 (2009); To appear in the *J. of Web Semantics*
12. Cali, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: Proc. of LICS, pp. 228–242 (2010)
13. Cali, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proc. of PODS, pp. 260–271 (2003)
14. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning* 39(3), 385–429 (2007)
15. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. PODS, pp. 149–158 (1998)
16. Calvanese, D., De Giacomo, G., Lenzerini, M.: Identification constraints and functional dependencies in description logics. In: Proc. of IJCAI, pp. 155–160 (2001)
17. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR, pp. 260–270 (2006)
18. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Conceptual Modeling for Data Integration. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) *Conceptual Modeling: Foundations and Applications*. LNCS, vol. 5600, pp. 173–197. Springer, Heidelberg (2009)
19. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of STOCs, pp. 77–90 (1977)
20. Chikofsky, E.J., Cross II, J.H.: Reverse engineering and design recovery: a taxonomy. *IEEE Software* 7(1), 13–17 (1990)
21. Chimia-Opoka, J., Felderer, M., Lenz, C., Lange, C.: Querying UML models using OCL and Prolog: A performance study. In: Proc. of ICSTW, pp. 81–88 (2008)
22. Deutsch, A., Nash, A., Rimmel, J.B.: The chase revisited. In: Proc. of PODS, pp. 149–158 (2008)
23. Donini, F.M., Massacci, F.: EXPTIME tableaux for \mathcal{ALC} . *Artif. Intell.* 124, 87–138 (2000)
24. Dupuy, S., Ledru, Y., Chabre-Peccoud, M.: An Overview of RoZ: A Tool for Integrating UML and Z Specifications. In: Wangler, B., Bergman, L.D. (eds.) *CAiSE 2000*. LNCS, vol. 1789, pp. 417–430. Springer, Heidelberg (2000)
25. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. *Theor. Comput. Sci.* 336(1), 89–124 (2005)
26. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. *J. Comput. System Sci.* 18(2), 194–211 (1979)
27. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)

28. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. *Sci. of Computer Progr.* 69(1-3), 27–34 (2007)
29. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.* 28(1), 167–189 (1984)
30. Kaneiwa, K., Satoh, K.: On the complexities of consistency checking for restricted UML class diagrams. *Theor. Comput. Sci.* 411(2), 301–323 (2010)
31. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: *Proc. of IJCAI*, pp. 963–968 (2011)
32. Lenzerini, M., Nobili, P.: On the satisfiability of dependency constraints in entity-relationship schemata. *Inf. Syst.* 15(4), 453–461 (1990)
33. Lutz, C.: The Complexity of Conjunctive Query Answering in Expressive Description Logics. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *IJCAR 2008*. LNCS (LNAI), vol. 5195, pp. 179–193. Springer, Heidelberg (2008)
34. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. *ACM Trans. Database Syst.* 4(4), 455–469 (1979)
35. Maraee, A., Balaban, M.: Efficient Reasoning about Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) *ECMDA-FA*. LNCS, vol. 4530, pp. 17–31. Springer, Heidelberg (2007)
36. Queralt, A., Teniente, E.: Reasoning on UML Class Diagrams with OCL Constraints. In: Embley, D.W., Olivé, A., Ram, S. (eds.) *ER 2006*. LNCS, vol. 4215, pp. 497–512. Springer, Heidelberg (2006)
37. Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL constraints. *ACM Trans. Softw. Eng. Meth.* 21(2) (2011) (in press)
38. Queralt, A., Teniente, E., Artale, A., Calvanese, D.: *OCL-Lite*: Finite reasoning on UML/OCL conceptual schemas. *Data and Know. Eng.* (2011) (in press)
39. Richters, M., Gogolla, M.: OCL: Syntax, Semantics, and Tools. In: Clark, A., Warmer, J. (eds.) *Object Modeling with the OCL*. LNCS, vol. 2263, pp. 447–450. Springer, Heidelberg (2002)
40. Snook, C., Butler, M.: UML-B: Formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Meth.* 15, 92–122 (2006)
41. Störrle, H.: A PROLOG-based approach to representing and querying software engineering models. In: *Proc. of VLL*, pp. 71–83 (2007)
42. Vardi, M.Y.: The complexity of relational query languages. In: *Proc. of STOC*, pp. 137–146 (1982)
43. Vardi, M.Y.: On the complexity of bounded-variable queries. In: *Proc. of PODS*, pp. 266–276 (1995)

Bicategories of Concurrent Games

(Invited Paper)

Glynn Winskel

University of Cambridge Computer Laboratory, England

Glynn.Winskel@cl.cam.ac.uk

<http://www.cl.cam.ac.uk/users/gw104>

Abstract. This paper summarises recent results on bicategories of concurrent games and strategies. Nondeterministic concurrent strategies, those nondeterministic plays of a game left essentially unchanged by composition with copy-cat strategies, have recently been characterized as certain maps of event structures. This leads to a bicategory of general concurrent games in which the maps are nondeterministic concurrent strategies. It is shown how the bicategory can be refined to a bicategory of winning strategies by adjoining winning conditions to games. Assigning “access levels” to moves addresses situations where Player or Opponent have imperfect information as to what has occurred in the game. Finally, a bicategory of deterministic “linear” strategies, a recently discovered model of MALL (multiplicative-additive linear logic), is described. All the bicategories become equivalent to simpler order-enriched categories when restricted to deterministic strategies.

Keywords: Games, strategies, concurrency, event structures, winning conditions, determinacy.

1 Introduction

Games and strategies are everywhere, in logic, philosophy, computer science, economics, leisure and in life. As abundant, but much less understood, are *concurrent* games in which a Player (or team of players) compete against an Opponent (or team of opponents) in a highly interactive and distributed fashion, especially when we recognize that the dichotomy Player vs. Opponent has several readings, as for example, process vs. environment, proof vs. refutation, or more ominously as ally vs. enemy. This paper summarises recent results on the mathematical foundations of concurrent games. It describes what it means to be a concurrent game, a concurrent strategy, a winning strategy, a concurrent game of imperfect information, and a linear strategy, and generally illustrates the rich mathematical structure concurrency brings to games.

Our primary motivation has come from the semantics of computation and the role of games in logic, although games are situated at a crossing point of several areas. In semantics it is becoming clear that we need an *intensional* theory to capture the *ways* of computing, to near operational and algorithmic

concerns. Sometimes unexpected intensionality is forced through the demands of compositionality, *e.g.* in *nondeterministic dataflow* [1]. More to the point we need to repair the artificial division between *denotational* and *operational* semantics. But what is to be our fundamental model of processes? Game semantics provides a possible answer: *strategies*. (There are others, *e.g.* profunctors as maps between presheaf categories [2,3].) Meanwhile in logic the well-known Curry-Howard correspondence “propositions as types, proofs as programs” is being recast as “propositions as games, proofs as strategies.”

However, in both semantics and logic, traditional definitions of strategies and games are not general enough: they do not adequately address the concurrent nature of computation and proof—see *e.g.* [4]. Game semantics has developed from simple sequential games, where only one move is allowed at a time and, for instance, it is often assumed that the moves of Player and Opponent alternate. Because of its history it is not obvious how to extend traditional game semantics to concurrent computation, or what relation it bears to other generalised domain theories such as those where domains are presheaf categories [2,3]. It is time to build game semantics on a broader foundation, one more squarely founded within a general model for concurrent processes. The standpoint of this paper is to base games and strategies on event structures, the analogue of trees but in a concurrent world; just as transition systems, an “interleaving” model, unfold to trees so do Petri nets, a “concurrent” model, unfold to event structures. In doing so we re-encounter earlier work of Abramsky and Melliès, first in their presentation of deterministic concurrent strategies as closure operators, and later in Melliès programme of *asynchronous games*, culminating in his definition with Mimram of ingenuous strategies; a consequence of the work described here is a characterization of Melliès and Mimram’s *receptive* ingenuous strategies [5] as precisely those deterministic pre-strategies for which copy-cat strategies behave as identities.

Our slogan: processes are nondeterministic concurrent strategies. For methodology we adopt ideas of Joyal who recognized that there was a category of games underlying Conway’s construction of the “surreal numbers” [6,7]. Like many 2-party games Conway’s games support two important operations: a form of parallel composition $G \parallel H$; a dualizing operation G^\perp which reverses the roles of Player and Opponent in G . Joyal defined a strategy σ from a game G to a game H , to be a strategy σ in $G^\perp \parallel H$. Following Conway’s method of proof, Joyal showed that strategies compose, with identities given by copy-cat strategies.

We shall transport the pattern established by Joyal to a general model for concurrent computation: games will be represented by event structures and strategies as certain maps into them. The motivation is to obtain: forms of generalised domain theory in which domains are replaced by concurrent games and continuous functions by nondeterministic concurrent strategies; operations, including higher-order operations via “function spaces” $G^\perp \parallel H$, within a model for concurrency; techniques for logic (via proofs as concurrent strategies), and possibly verification and algorithmics. However, first things first, here we will concentrate on the rich

algebra of concurrent strategies. Most proofs and background on stable families, on which proofs often rely, can be found in [8].

2 Event Structures

An *event structure* comprises (E, Con, \leq) , consisting of a set E , of *events* which are partially ordered by \leq , the *causal dependency relation*, and a nonempty *consistency relation* Con consisting of finite subsets of E , which satisfy

$$\begin{aligned} \{e' \mid e' \leq e\} \text{ is finite for all } e \in E, \\ \{e\} \in \text{Con for all } e \in E, \\ Y \subseteq X \in \text{Con} \implies Y \in \text{Con}, \text{ and} \\ X \in \text{Con} \ \& \ e \leq e' \in X \implies X \cup \{e\} \in \text{Con}. \end{aligned}$$

The *configurations*, $\mathcal{C}^\infty(E)$, of an event structure E consist of those subsets $x \subseteq E$ which are

$$\begin{aligned} \text{Consistent: } \forall X \subseteq x. X \text{ is finite} \implies X \in \text{Con}, \text{ and} \\ \text{Down-closed: } \forall e, e'. e' \leq e \in x \implies e' \in x. \end{aligned}$$

Often we shall be concerned with just the finite configurations of an event structure. We write $\mathcal{C}(E)$ for the *finite* configurations of an event structure E .

Two events which are both consistent and incomparable w.r.t. causal dependency in an event structure are regarded as *concurrent*. In games the relation of *immediate* dependency $e \rightarrow e'$, meaning e and e' are distinct with $e \leq e'$ and no event in between, will play a very important role. For $X \subseteq E$ we write $[X]$ for $\{e \in E \mid \exists e' \in X. e \leq e'\}$, the down-closure of X ; note if $X \in \text{Con}$, then $[X] \in \text{Con}$.

Notation 1. Let E be an event structure. We use $x \dashv\!\! \dashv y$ to mean y covers x in $\mathcal{C}^\infty(E)$, i.e. $x \subset y$ in $\mathcal{C}^\infty(E)$ with nothing in between, and $x \dashv\!\! \dashv^e y$ to mean $x \cup \{e\} = y$ for $x, y \in \mathcal{C}^\infty(E)$ and event $e \notin x$. We sometimes use $x \dashv\!\! \dashv^e$, expressing that event e is enabled at configuration x , when $x \dashv\!\! \dashv y$ for some y .

2.1 Maps of Event Structures

Let E and E' be event structures. A (*partial*) *map* of event structures $f : E \dashv\!\! \dashv E'$ is a partial function on events $f : E \dashv\!\! \dashv E'$ such that for all $x \in \mathcal{C}(E)$ its direct image $f x \in \mathcal{C}(E')$ and

$$\text{if } e_1, e_2 \in x \text{ and } f(e_1) = f(e_2) \text{ (with both defined), then } e_1 = e_2.$$

The map expresses how the occurrence of an event e in E induces the coincident occurrence of the event $f(e)$ in E' whenever it is defined. Partial maps of event structures compose as partial functions, with identity maps given by identity functions.

For any event e a map of event structures $f : E \rightarrow E'$ must send the configuration $[e]$ to the configuration $f[e]$. Partial maps preserve the concurrency relation, when defined.

We will say the map is *total* if the function f is total. Notice that for a total map f the condition on maps now says it is *locally injective*, in the sense that w.r.t. any configuration x of the domain the restriction of f to a function from x is injective; the restriction of f to a function from x to fx is thus bijective. A partial map of event structures which preserves causal dependency whenever it is defined, *i.e.* $e' \leq e$ implies $f(e') \leq f(e)$ whenever both $f(e')$ and $f(e)$ are defined, is called *partial rigid*. We reserve the term *rigid* for those total maps of event structures which preserve causal dependency.

2.2 Process Operations

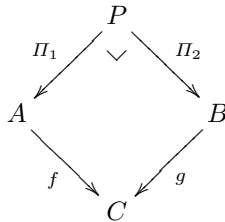
Products. The category of event structures with partial maps has *products* $A \times B$ with projections Π_1 to A and Π_2 to B . The effect is to introduce arbitrary synchronisations between events of A and events of B in the manner of process algebra.

Restriction. The restriction of an event structure E to a subset of events R , written $E \upharpoonright R$, is the event structure with events $E' = \{e \in E \mid [e] \subseteq R\}$ and causal dependency and consistency induced by E .

Synchronized Compositions and Pullbacks. Synchronized compositions play a central role in process algebra, with such seminal work as Milner's CCS and Hoare's CSP. Synchronized compositions of event structures A and B are obtained as restrictions $A \times B \upharpoonright R$. We obtain *pullbacks* as a special case. Let $f : A \rightarrow C$ and $g : B \rightarrow C$ be maps of event structures. Defining

$$P =_{\text{def}} A \times B \upharpoonright \{p \in A \times B \mid f\Pi_1(p) = g\Pi_2(p) \text{ with both defined}\}$$

we obtain a pullback square



in the category of event structures. When f and g are total the same construction gives the pullback in the category of event structures with *total* maps.

2.3 Projection

Let (E, \leq, Con) be an event structure. Let $V \subseteq E$ be a subset of ‘visible’ events. Define the *projection* of E on V , to be $E \downarrow V =_{\text{def}} (V, \leq_V, \text{Con}_V)$, where $v \leq_V v'$ iff $v \leq v'$ & $v, v' \in V$ and $X \in \text{Con}_V$ iff $X \in \text{Con}$ & $X \subseteq V$.

3 Event Structures with Polarities

Both a game and a strategy in a game are to be represented as an event structure with polarity, which comprises (E, pol) where E is an event structure with a polarity function $\text{pol} : E \rightarrow \{+, -\}$ ascribing a polarity + (Player) or - (Opponent) to its events. The events correspond to (occurrences of) moves. Maps of event structures with polarity are maps of event structures which preserve polarity.

3.1 Operations

Dual. The *dual*, E^\perp , of an event structure with polarity E comprises a copy of the event structure E but with a reversal of polarities.

Simple Parallel Composition. The operation $A \parallel B$ simply forms the disjoint juxtaposition of A, B , two event structures with polarity; a finite subset of events is consistent if its intersection with each component is consistent.

4 Pre-strategies

Let A be an event structure with polarity, thought of as a game; its events stand for the possible occurrences of moves of Player and Opponent and its causal dependency and consistency relations the constraints imposed by the game. A pre-strategy represents a nondeterministic play of the game—all its moves are moves allowed by the game and obey the constraints of the game; the concept will later be refined to that of *strategy* (and *winning strategy* in Section 7). A *pre-strategy* in A is defined to be a total map $\sigma : S \rightarrow A$ from an event structure with polarity S . Two pre-strategies $\sigma : S \rightarrow A$ and $\tau : T \rightarrow A$ in A will be essentially the same when they are isomorphic, *i.e.* there is an isomorphism $\theta : S \cong T$ such that $\sigma = \tau\theta$; then we write $\sigma \cong \tau$.

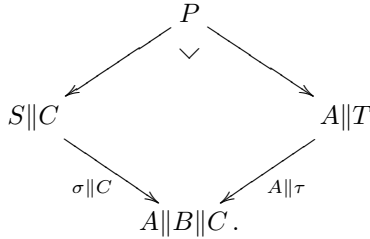
Let A and B be event structures with polarity. Following Joyal [7], a pre-strategy from A to B is a pre-strategy in $A^\perp \parallel B$, so a total map $\sigma : S \rightarrow A^\perp \parallel B$. It thus determines a span

$$\begin{array}{ccc}
 & S & \\
 \sigma_1 \swarrow & & \searrow \sigma_2 \\
 A^\perp & & B,
 \end{array}$$

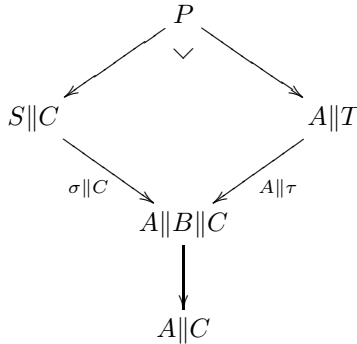
of event structures with polarity where σ_1, σ_2 are *partial* maps and for all $s \in S$ either, but not both, $\sigma_1(s)$ or $\sigma_2(s)$ is defined. Two pre-strategies from A to B will be isomorphic when they are isomorphic as pre-strategies in $A^\perp \parallel B$, or equivalently are isomorphic as spans. We write $\sigma : A \multimap B$ to express that σ is a pre-strategy from A to B . Note a pre-strategy σ in a game A coincides with a pre-strategy from the empty game $\sigma : \emptyset \multimap A$.

4.1 Composing Pre-strategies

We can present the composition of pre-strategies via pullbacks¹. Given two pre-strategies $\sigma : S \rightarrow A^\perp \parallel B$ and $\tau : T \rightarrow B^\perp \parallel C$, ignoring polarities we can consider the maps on the underlying event structures, *viz.* $\sigma : S \rightarrow A \parallel B$ and $\tau : T \rightarrow B \parallel C$. Viewed this way we can form the pullback in the category of event structures



There is an obvious partial map of event structures $A \parallel B \parallel C \rightarrow A \parallel C$ undefined on B and acting as identity on A and C . The partial map from P to $A \parallel C$ given by following the diagram (either way round the pullback square)



factors as the composition of the partial map $P \rightarrow P \downarrow V$, where V is the set of events of P at which the map $P \rightarrow A \parallel C$ is defined, and a total map $P \downarrow V \rightarrow A \parallel C$. The resulting total map gives us the composition $\tau \circ \sigma : P \downarrow V \rightarrow A^\perp \parallel C$ once we reinstate polarities.

¹ The construction here gives the same result as that via synchronized composition in [9]—I'm grateful to Nathan Bowler for this observation. Notice the analogy with the composition of relations $S \subseteq A \times B$, $T \subseteq B \times C$ which can be defined as $T \circ S = (S \times C \cap A \times T) \downarrow A \times C$, the image of $S \times C \cap A \times T$ under the projection of $A \times B \times C$ to $A \times C$.

4.2 Concurrent Copy-Cat

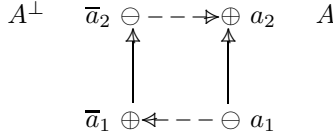
Identities w.r.t. composition are given by copy-cat strategies. Let A be an event structure with polarity. The copy-cat strategy from A to A is an instance of a pre-strategy, so a total map $\gamma_A : \mathbb{C}_A \rightarrow A^\perp \parallel A$. It describes a concurrent, or distributed, strategy based on the idea that Player moves, of +ve polarity, always copy previous corresponding moves of Opponent, of -ve polarity.

For $c \in A^\perp \parallel A$ we use \bar{c} to mean the corresponding copy of c , of opposite polarity, in the alternative component. Define \mathbb{C}_A to comprise the event structure with polarity $A^\perp \parallel A$ together with extra causal dependencies $\bar{c} \leq_{\mathbb{C}_A} c$ for all events c with $pol_{A^\perp \parallel A}(c) = +$.

Proposition 1. *Let A be an event structure with polarity. Then event structure with polarity \mathbb{C}_A is an event structure. Moreover, $x \in \mathcal{C}(\mathbb{C}_A)$ iff $x \in \mathcal{C}(A^\perp \parallel A)$ & $\forall c \in x. pol_{A^\perp \parallel A}(c) = + \implies \bar{c} \in x$.*

The *copy-cat* pre-strategy $\gamma_A : A \dashrightarrow A$ is defined to be the map $\gamma_A : \mathbb{C}_A \rightarrow A^\perp \parallel A$ where γ_A is the identity on the common set of events.

Example 1. We illustrate the construction of the copy-cat strategy for the event structure A comprising the single immediate dependency $a_1 \rightarrow a_2$ from an Opponent move a_1 to a Player move a_2 . The event structure \mathbb{C}_A is obtained from $A^\perp \parallel A$ by adjoining the additional immediate dependencies shown:



5 Strategies

The main result of [9] is that two conditions on pre-strategies, *receptivity* and *innocence*, are necessary and sufficient for copy-cat to behave as identity w.r.t. the composition of pre-strategies. Receptivity ensures an openness to all possible moves of Opponent. Innocence restricts the behaviour of Player; Player may only introduce new relations of immediate causality of the form $\ominus \rightarrow \oplus$ beyond those imposed by the game.

Receptivity. A pre-strategy σ is *receptive* iff $\sigma x \overset{a}{\dashleftarrow} \subset$ & $pol_A(a) = - \implies \exists ! s \in S. x \overset{s}{\dashrightarrow} \subset$ & $\sigma(s) = a$.

Innocence. A pre-strategy σ is *innocent* when it is both
 +-*innocent*: if $s \rightarrow s'$ & $pol(s) = +$ then $\sigma(s) \rightarrow \sigma(s')$, and
 --*innocent*: if $s \rightarrow s'$ & $pol(s') = -$ then $\sigma(s) \rightarrow \sigma(s')$.

Theorem 1. *Let $\sigma : A \dashrightarrow B$ be pre-strategy. Copy-cat behaves as identity w.r.t. composition, i.e. $\sigma \circ \gamma_A \cong \sigma$ and $\gamma_B \circ \sigma \cong \sigma$, iff σ is receptive and innocent. Copy-cat pre-strategies $\gamma_A : A \dashrightarrow A$ are receptive and innocent.*

5.1 The Bicategory of Concurrent Games and Strategies

Theorem [1](#) motivates the definition of a *strategy* as a pre-strategy which is receptive and innocent. In fact, we obtain a bicategory, **Games**, in which the objects are event structures with polarity—the games, the arrows from A to B are strategies $\sigma : A \dashrightarrow B$ and the 2-cells are maps of spans. The vertical composition of 2-cells is the usual composition of maps of spans. Horizontal composition is given by the composition of strategies \odot (which extends to a functor on 2-cells via the universality of pullback).

A strategy $\sigma : A \dashrightarrow B$ corresponds to a dual strategy $\sigma^\perp : B^\perp \dashrightarrow A^\perp$. This duality arises from the correspondence

$$\begin{array}{ccc}
 & S & \\
 \sigma_1 \swarrow & & \searrow \sigma_2 \\
 A^\perp & & B
 \end{array}
 \quad \longleftrightarrow \quad
 \begin{array}{ccc}
 & S & \\
 \sigma_2 \swarrow & & \searrow \sigma_1 \\
 (B^\perp)^\perp & & A^\perp
 \end{array}$$

The dual of copy-cat, γ_A^\perp , is isomorphic to the copy-cat of the dual, γ_{A^+} , for A an event structure with polarity. The dual of a composition of pre-strategies $(\tau \odot \sigma)^\perp$ is isomorphic to the composition $\sigma^\perp \odot \tau^\perp$. This duality is maintained in the major bicategories of games we shall consider.

One notable sub-bicategory of games, though one not maintaining duality, is obtained on restricting to objects which comprise purely +ve events: then we obtain the bicategory of stable spans, which have played a central role in the semantics of nondeterministic dataflow [\[1\]](#).

5.2 The Subcategory of Deterministic Strategies

Say an event structure with polarity S is *deterministic* iff

$$\forall X \subseteq_{\text{fin}} S. \text{Neg}[X] \in \text{Cons}_S \implies X \in \text{Cons}_S,$$

where $\text{Neg}[X] =_{\text{def}} \{s' \in S \mid \text{pol}(s') = - \ \& \ \exists s \in X. s' \leq s\}$. In other words, S is deterministic iff any finite set of moves is consistent when it causally depends only on a consistent set of opponent moves. Say a strategy $\sigma : S \rightarrow A$ is deterministic if S is deterministic.

Lemma 1. *An event structure with polarity S is deterministic iff*

$$\forall s, s' \in S, x \in \mathcal{C}(S). \quad x \xrightarrow{s} \mathcal{C} \ \& \ x \xrightarrow{s'} \mathcal{C} \ \& \ \text{pol}(s) = + \implies x \cup \{s, s'\} \in \mathcal{C}(S).$$

In general, a copy-cat strategy can fail to be deterministic, illustrated below.

Example 2. Take A to consist of two events, one +ve and one -ve event, inconsistent with each other (indicated by the wiggly line). The construction \mathbb{C}_A :

$$\begin{array}{ccc}
 A^\perp & \ominus & \text{---} \rightarrow \oplus & A \\
 & \{ & & \} \\
 & \{ & & \} \\
 & \{ & & \} \\
 & \oplus & \leftarrow \text{---} & \ominus
 \end{array}$$

To see \mathbb{C}_A is not deterministic, take x to be the singleton set consisting *e.g.* of the $-$ ve event on the left and s, s' to be the $+$ ve and $-$ ve events on the right.

Copy-cat γ_A is deterministic iff immediate conflict in A respects polarity, or equivalently that there is no immediate conflict between $+$ ve and $-$ ve events, a condition we call *race-free*.

Lemma 2. *Let A be an event structure with polarity. The copy-cat strategy γ_A is deterministic iff A is race-free, i.e. for all $x \in \mathcal{C}(A)$,*

$$x \xrightarrow{a} \mathcal{C} \ \& \ x \xrightarrow{a'} \mathcal{C} \ \& \ \text{pol}(a) = + \ \& \ \text{pol}(a') = - \implies x \cup \{a, a'\} \in \mathcal{C}(A).$$

Lemma 3. *The composition of deterministic strategies is deterministic.*

Lemma 4. *A deterministic strategy $\sigma : S \rightarrow A$ is injective on configurations (equivalently, σ is mono in the category of event structures with polarity).*

We obtain a sub-bicategory **DGames** of **Games** by restricting objects to race-free games and strategies to being deterministic. Via Lemma 4, deterministic strategies in a game correspond to certain subfamilies of configurations of the game. A characterization of those subfamilies which correspond to deterministic strategies shows them to coincide with the *receptive* ingenious strategies of Mirmam and Melliès [5]. This work grew out of Abramsky and Melliès early work in which deterministic concurrent strategies are presented, essentially, as partial closure operators on the domain of configurations of an event structure [4]. Via the presentation of deterministic strategies as families **DGames** is equivalent to an order-enriched category. There are notable subcategories: when the objects are countable event structures with polarity which consist of purely $+$ ve events we recover as a full subcategory the classical category of stable domain theory, *viz.* Berry's dI-domains and stable functions; this in turn has Girard's qualitative domains and coherence spaces, both with stable functions, as full subcategories [10]. The category of simple games [11,12], underlying both HO and AJM games, is a subcategory, though not full.

6 From Strategies to Profunctors

Let x and x' be configurations of an event structure with polarity. Write $x \subseteq^+ x'$ to mean $x \subseteq x'$ and $\text{pol}(x' \setminus x) \subseteq \{+\}$, *i.e.* the configuration x' extends the configuration x solely by events of $+$ ve polarity. Similarly $x \subseteq^- x'$ means configuration x' extends x solely by events of $-$ ve polarity. With this notation in place we can give an attractive characterization of concurrent strategies:

Lemma 5. *A strategy S in a game A comprises a total map of event structures with polarity $\sigma : S \rightarrow A$ such that*

(i) whenever $y \sqsubseteq^+ \sigma x$ in $\mathcal{C}(A)$ there is a (necessarily unique) $x' \in \mathcal{C}(S)$ so that $x' \sqsubseteq x$ & $\sigma x' = y$, i.e.

$$\begin{array}{ccc} x' & \cdots \sqsubseteq \cdots & x \\ \sigma \downarrow & & \downarrow \sigma \\ y & \sqsubseteq^+ & \sigma x, \end{array}$$

and

(ii) whenever $\sigma x \sqsubseteq^- y$ in $\mathcal{C}(A)$ there is a unique $x' \in \mathcal{C}(S)$ so that $x \sqsubseteq x'$ & $\sigma x' = y$, i.e.

$$\begin{array}{ccc} x & \cdots \sqsubseteq \cdots & x' \\ \sigma \downarrow & & \downarrow \sigma \\ \sigma x & \sqsubseteq^- & y. \end{array}$$

The above lemma tells us how to form a discrete fibration, so presheaf, from a strategy. For A , an event structure with polarity, we can define a new order, the *Scott order*, between configurations $x, y \in \mathcal{C}^\infty(A)$, by

$$x \sqsubseteq_A y \iff x \supseteq^- x \cap y \sqsubseteq^+ y.$$

Proposition 2. *Let $\sigma : S \rightarrow A$ be a pre-strategy in game A . The map σ “taking a finite configuration $x \in \mathcal{C}(S)$ to $\sigma x \in \mathcal{C}(A)$ is a discrete fibration from $(\mathcal{C}(S), \sqsubseteq_S)$ to $(\mathcal{C}(A), \sqsubseteq_A)$ iff σ is a strategy.*

As discrete fibrations correspond to presheaves, an alternative reading of Proposition 2 is that a pre-strategy $\sigma : S \rightarrow A$ is a strategy iff σ “determines a presheaf over $(\mathcal{C}(A), \sqsubseteq_A)$.”

Consequently, a strategy $\sigma : A \dashrightarrow B$ determines a discrete fibration over $(\mathcal{C}(A^\perp \parallel B), \sqsubseteq_{A^\perp \parallel B})$. But

$$(\mathcal{C}(A^\perp \parallel B), \sqsubseteq_{A^\perp \parallel B}) \cong (\mathcal{C}(A^\perp), \sqsubseteq_{A^\perp}) \times (\mathcal{C}(B), \sqsubseteq_B) \cong (\mathcal{C}(A), \sqsubseteq_A)^{\text{op}} \times (\mathcal{C}(B), \sqsubseteq_B),$$

so σ determines a presheaf over $(\mathcal{C}(A), \sqsubseteq_A)^{\text{op}} \times (\mathcal{C}(B), \sqsubseteq_B)$, i.e. a *profunctor*

$$\sigma^\circ : (\mathcal{C}(A), \sqsubseteq_A) \dashrightarrow (\mathcal{C}(B), \sqsubseteq_B).$$

The operation σ° , on a strategy σ , forms a *lax* functor from **Games** to **Prof**, the bicategory of profunctors: whereas it preserves identities, it is *not* the case that $(\tau \circ \sigma)^\circ$ and $\tau^\circ \circ \sigma^\circ$ coincide up to isomorphism; the profunctor composition $\tau^\circ \circ \sigma^\circ$ will generally contain extra “unreachable” elements.

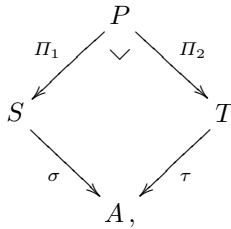
However, in special cases composition is preserved up to isomorphism. Say a strategy σ is *partial rigid* when the components σ_1, σ_2 are partial-rigid maps of event structures (with polarity). Partial-rigid strategies form a sub-bicategory of **Games**—see Section 9. For composable partial-rigid strategies σ and τ we do have $(\tau \circ \sigma)^\circ \cong \tau^\circ \circ \sigma^\circ$. Stable spans and simple games lie within the bicategory partial-rigid strategies.

7 Winning Strategies

A *game with winning conditions* comprises $G = (A, W)$ where A is an event structure with polarity and $W \subseteq \mathcal{C}^\infty(A)$ consists of the *winning configurations* for Player. We define the *losing conditions* to be $\mathcal{C}^\infty(A) \setminus W$ ².

A strategy in G is a strategy in A . A strategy in G is regarded as *winning* if it always prescribes Player moves to end up in a winning configuration, no matter what the activity or inactivity of Opponent. Formally, a strategy $\sigma : S \rightarrow A$ in G is *winning (for Player)* if $\sigma x \in W$ for all +-maximal configurations $x \in \mathcal{C}^\infty(S)$ —a configuration x is +-maximal if whenever $x \xrightarrow{s} _$ then the event s has -ve polarity. Any achievable position $z \in \mathcal{C}^\infty(S)$ of the game can be extended to a +-maximal, so winning, configuration (via Zorn’s Lemma). So a strategy prescribes Player moves to reach a winning configuration whatever state of play is achieved following the strategy. Note that for a game A , if winning conditions $W = \mathcal{C}^\infty(A)$, *i.e.* every configuration is winning, then any strategy in A is a winning strategy.

Informally, we can also understand a strategy as winning for Player if when played against any counter-strategy of Opponent, the final result is a win for Player. Suppose $\sigma : S \rightarrow A$ is a strategy in a game (A, W) . A counter-strategy is strategy of Opponent, so a strategy $\tau : T \rightarrow A^\perp$ in the dual game. We can view σ as a strategy $\sigma : \emptyset \dashrightarrow A$ and τ as a strategy $\tau : A \dashrightarrow \emptyset$. Their composition $\tau \circ \sigma : \emptyset \dashrightarrow \emptyset$ is not in itself so informative. Rather it is the status of the configurations in $\mathcal{C}^\infty(A)$ their full interaction induces which decides which of Player or Opponent wins. Ignoring polarities, we have total maps of event structures $\sigma : S \rightarrow A$ and $\tau : T \rightarrow A$. Form their pullback,



to obtain the event structure P resulting from the interaction of σ and τ . Because σ or τ may be nondeterministic there can be more than one maximal configuration z in $\mathcal{C}^\infty(P)$. A maximal configuration z in $\mathcal{C}^\infty(P)$ images to a configuration $\sigma \Pi_1 z = \tau \Pi_2 z$ in $\mathcal{C}^\infty(A)$. Define the set of *results* of the interaction of σ and τ to be

$$\langle \sigma, \tau \rangle =_{\text{def}} \{ \sigma \Pi_1 z \mid z \text{ is maximal in } \mathcal{C}^\infty(P) \}.$$

It can be shown that a strategy σ is a winning for Player iff all the results of the interaction $\langle \sigma, \tau \rangle$ lie within the winning configurations W , for any counter-strategy $\tau : T \rightarrow A^\perp$ of Opponent.

² It is fairly straightforward to generalize to the situation where configurations may be neutral, neither winning nor losing [13][8].

7.1 Operations

There is an obvious *dual* of a game with winning conditions $G = (A, W_G)$:

$$G^\perp = (A^\perp, \mathcal{C}^\infty(A) \setminus W_G),$$

reversing the role of Player and Opponent, and consequently that of winning and losing conditions.

The parallel composition of two games with winning conditions $G = (A, W_G)$, $H = (B, W_H)$ is

$$G \wp H =_{\text{def}} (A \parallel B, W_{G \wp H})$$

where, for $x \in \mathcal{C}^\infty(A \parallel B)$,

$$x \in W_{G \wp H} \text{ iff } x_1 \in W_G \text{ or } x_2 \in W_H$$

—a configuration x of $A \parallel B$ comprises the disjoint union of a configuration x_1 of A and a configuration x_2 of B . To win in $G \wp H$ is to win in either game. The unit of \parallel is (\emptyset, \emptyset) . Defining $G \otimes H =_{\text{def}} (G^\perp \parallel H^\perp)^\perp$ we obtain a game where to win is to win in both games G and H . The unit of \otimes is $(\emptyset, \{\emptyset\})$.

Defining $G \multimap H =_{\text{def}} G^\perp \wp H$, a win in $G \multimap H$ is a win in H conditional on a win in G : For $x \in \mathcal{C}^\infty(A^\perp \parallel B)$,

$$x \in W_{G \multimap H} \text{ iff } x_1 \in W_G \implies x_2 \in W_H.$$

7.2 The Bicategory of Winning Strategies

We can again follow Joyal and define strategies between games now with winning conditions: a (winning) strategy from G , a game with winning conditions, to another H is a (winning) strategy in $G \multimap H$. We compose strategies as before. The composition of winning strategies is winning. However, for a general game with winning conditions (A, W) the copy-cat strategy need not be winning, as shown in the following example.

Example 3. Let A be the event structure with polarity of Example 2. Take as winning conditions the set $\{\{\oplus\}\}$. To see \mathbb{C}_A is not winning consider the configuration x consisting of the two $-$ ve events in \mathbb{C}_A . Then x is $+$ -maximal as any $+$ -ve event is inconsistent with x . However, $x_1 \in W$ while $x_2 \notin W$, failing the winning condition of $(A, W) \multimap (A, W)$.

Recall from Section 6 that each event structure with polarity A possesses a Scott order on its configurations $\mathcal{C}^\infty(A)$: $x' \sqsubseteq x$ iff $x' \supseteq^- x \cap x' \sqsubseteq^+ x$. With it we can express a necessary and sufficient for copy-cat to be winning w.r.t. a game (A, W) :

$$\forall x, x' \in \mathcal{C}^\infty(A). \text{ if } x' \sqsubseteq x \text{ \& } x' \text{ is } +\text{-maximal \& } x \text{ is } --\text{-maximal,} \quad (\text{Cwins})$$

$$\text{then } x \in W \implies x' \in W.$$

The condition **(Cwins)** is assured for event structures with polarity which are race-free.

We can now refine the bicategory of strategies **Games** to the bicategory **WGames** with objects games with winning conditions G, H, \dots satisfying **(Cwins)** and arrows winning strategies $G \dashrightarrow H$; 2-cells, their vertical and horizontal composition are as before. Its restriction to deterministic strategies yields a bicategory equivalent to a simpler order-enriched category.

7.3 Applications

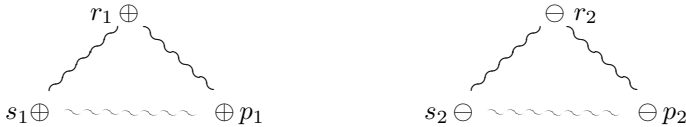
As an application of winning conditions we apply them to pick out a subcategory of “total strategies,” informally strategies in which Player can always answer a move of Opponent [14,11]—see [8] for details. Often problems can be reduced to whether Player or Opponent has a winning strategy, for which it is important to know when concurrent games are *determined*, *i.e.* either Player or Opponent has a winning strategy. As a first step, well-founded, race-free concurrent games have now been shown to be determined and have been applied to give a concurrent game semantics to predicate logic [8,15]. (A game A is well-founded if all configurations in $\mathcal{C}^\infty(A)$ are finite.) The game semantics extends to Hintikka’s “independence-friendly” logic, using ideas of the next section to associate ‘levels’ with quantified variables.

8 Imperfect Information

Consider the game “rock, scissors, paper” in which the two participants Player and Opponent independently sign one of r (“rock”), s (“scissors”) or p (“paper”). The participant with the dominant sign w.r.t. the relation

$$r \text{ beats } s, s \text{ beats } p \text{ and } p \text{ beats } r$$

wins. It seems sensible to represent this game by RSP , the event structure with polarity

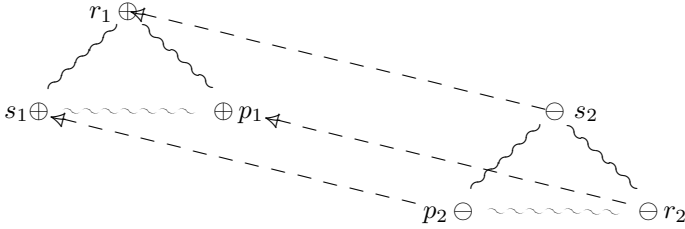


comprising the three mutually inconsistent possible signings of Player in parallel with the three mutually inconsistent signings of Opponent. In the absence of neutral configurations, a reasonable choice is to take the *losing* configurations (for Player) to be

$$\{s_1, r_2\}, \{p_1, s_2\}, \{r_1, p_2\}$$

and all other configurations as winning for Player. In this case there is a winning strategy for Player, *viz.* await the move of Opponent and then beat it with a

dominant move. Explicitly, the winning strategy $\sigma : S \rightarrow RSP$ is given as the obvious map from S , the following event structure with polarity:



But this strategy cheats. In “rock, scissors, paper” participants are intended to make their moves *independently*. The problem with the game RSP as it stands is that it is a game of *perfect information* in the sense that all moves are visible to both participants. This permits the winning strategy above with its unwanted dependencies on moves which should be unseen by Player. To adequately model “rock, scissors, paper” requires a game of *imperfect information* where some moves are masked, or inaccessible, and strategies with dependencies on unseen moves are ruled out.

We can extend concurrent games to games with imperfect information. To do so in way that respects the operations of the bicategory of games we suppose a fixed preorder of *levels* (Λ, \preceq) . The levels are to be thought of as levels of access, or permission. Moves in games and strategies are to respect levels: moves will be assigned levels in such a way that a move is only permitted to causally depend on moves at equal or lower levels; it is as if from a level only moves of equal or lower level can be seen.

A Λ -game (G, l) comprises a game $G = (A, W)$ with winning conditions together with a *level function* $l : A \rightarrow \Lambda$ such that

$$a \leq_A a' \implies l(a) \preceq l(a')$$

for all $a, a' \in A$. A Λ -strategy in the Λ -game (G, l) is a strategy $\sigma : S \rightarrow A$ for which

$$s \leq_S s' \implies l\sigma(s) \preceq l\sigma(s')$$

for all $s, s' \in S$.

For example, for “rock, scissors, paper” we can take Λ to be the discrete preorder consisting of levels 1 and 2 unrelated to each other under \preceq . To make RSP into a suitable Λ -game the level function l takes +ve events in RSP to level 1 and -ve events to level 2. The strategy above, where Player awaits the move of Opponent then beats it with a dominant move, is now disallowed because it is not a Λ -strategy—it introduces causal dependencies which do not respect levels. If instead we took Λ to be the unique preorder on a single level the Λ -strategies would coincide with all the strategies.

Fortunately the introduction of levels meshes smoothly with the bicategorical structure on games. For Λ -games (G, l_G) and (H, l_H) , define the dual $(G, l_G)^\perp$ to be (G^\perp, l_{G^\perp}) where $l_{G^\perp} = l_G$, and define the parallel composition $(G, l_G) \wp$

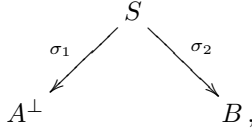
(H, l_H) to be $(G \wp H, l_{G \wp H})$ where $l_{G \wp H}(a) = l_G(a)$ for $a \in G$, $l_{G \wp H}(b) = l_H(b)$ for $b \in H$.

A Λ -strategy between Λ -games from (G, l_G) to (H, l_H) is a strategy in $(G, l_G)^\perp \wp (H, l_H)$. Let (G, l_G) be a Λ -game where G satisfies **(Cwins)**. The copy-cat strategy on G is a Λ -strategy. The composition of Λ -strategies is a Λ -strategy.

9 Linear Strategies

It has recently become clear that concurrent strategies support several refinements. For example, define a *partial-rigid strategy* to be a strategy σ in which both components σ_1 and σ_2 are partial rigid. Copy-cat strategies are partial rigid, and the composition of partial-rigid strategies is partial-rigid, so partial-rigid strategies form a sub-bicategory of **Games**. We can refine partial-rigid strategies further to *linear strategies*, where each +ve output event depends on a maximum +ve event of input, and dually, a -ve event of input depends on a maximum -ve event of output. By introducing this extra relevance, of input to output and output to input, we can recover coproducts and products lacking in **Games**.

Formally, a (nondeterministic) *linear strategy* is a strategy



where σ_1 and σ_2 are partial rigid maps such that

$$\begin{aligned}
 & \forall s \in S. \text{pol}_S(s) = + \ \& \ \sigma_2(s) \text{ is defined} \\
 & \implies \\
 & \exists s_0 \in S. \text{pol}_S(s_0) = - \ \& \ \sigma_1(s_0) \text{ is defined} \ \& \ s_0 \leq_S s \ \& \\
 & \forall s_1 \in S. \text{pol}_S(s_1) = - \ \& \ \sigma_1(s_1) \text{ is defined} \ \& \ s_1 \leq_S s \implies s_1 \leq_S s_0
 \end{aligned}$$

and

$$\begin{aligned}
 & \forall s \in S. \text{pol}_S(s) = + \ \& \ \sigma_1(s) \text{ is defined} \\
 & \implies \\
 & \exists s_0 \in S. \text{pol}_S(s_0) = - \ \& \ \sigma_2(s_0) \text{ is defined} \ \& \ s_0 \leq_S s \ \& \\
 & \forall s_1 \in S. \text{pol}_S(s_1) = - \ \& \ \sigma_2(s_1) \text{ is defined} \ \& \ s_1 \leq_S s \implies s_1 \leq_S s_0.
 \end{aligned}$$

Copy-cat strategies are linear and linear strategies are closed under composition. Linear strategies form a sub-bicategory **Games**. Its sub-bicategory **Lin** of deterministic subcategories is a model of MALL (multiplicative-additive linear logic) and a promising candidate in which to establish full-completeness—work in progress.

10 Conclusion

We have summarised the main results on concurrent strategies to date (December 2011). Two current research directions: One current is the development of an intensional semantics of processes and proofs. But games and concurrent strategies form a generalized *affine* domain theory. Does the bicategory **Lin** of deterministic linear strategies provide a fully-complete model of MALL? A next step is to extend concurrent games to allow *back-tracking* via “copying” monads in event structures with symmetry [16]. Another direction concerns the possible application of concurrent games for which we seek stronger determinacy results.

Acknowledgments. Thanks to Silvain Rideau and my coworkers on the project ECSYM, Pierre Clairambault and Julian Gutierrez who in particular assisted in the Aarhus course on which this summary is based. The support of Advanced Grant ECSYM of the European Research Council is acknowledged with gratitude.

References

1. Saunders-Evans, L., Winskel, G.: Event structure spans for nondeterministic dataflow. *Electr. Notes Theor. Comput. Sci.* 175(3), 109–129 (2007)
2. Hyland, M.: Some reasons for generalising domain theory. *Mathematical Structures in Computer Science* 20(2), 239–265 (2010)
3. Cattani, G.L., Winskel, G.: Profunctors, open maps and bisimulation. *Mathematical Structures in Computer Science* 15(3), 553–614 (2005)
4. Abramsky, S., Melliès, P.A.: Concurrent games and full completeness. In: *LICS 1999*. IEEE Computer Society (1999)
5. Melliès, P.A., Mimram, S.: Asynchronous Games: Innocence without Alternation. In: Caires, L., Vasconcelos, V.T. (eds.) *CONCUR 2007*. LNCS, vol. 4703, pp. 395–411. Springer, Heidelberg (2007)
6. Conway, J.: *On Numbers and Games*. A K Peters, Wellesley (2000)
7. Joyal, A.: Remarques sur la théorie des jeux à deux personnes. *Gazette des Sciences Mathématiques du Québec* 1(4) (1997)
8. Winskel, G.: Event structures, stable families and games. Lecture notes, Comp. Science Dept. Aarhus University (2011), <http://daimi.au.dk/~gwinskel>
9. Rideau, S., Winskel, G.: Concurrent strategies. In: *LICS 2011*. IEEE Computer Society (2011)
10. Girard, J.Y.: *The blind spot*. European Mathematical Society (2011)
11. Hyland, M.: Game semantics. In: Pitts, A., Dybjer, P. (eds.) *Semantics and Logics of Computation*. Publications of the Newton Institute (1997)
12. Harmer, R., Hyland, M., Melliès, P.A.: Categorical combinatorics for innocent strategies. In: *LICS 2007*. IEEE Computer Society (2007)
13. Winskel, G.: Winning, losing and drawing in games with perfect and imperfect information. In: *Festschrift for Dexter Kozen*. LNCS. Springer, Heidelberg (2012)
14. Abramsky, S.: Semantics of interaction. In: Pitts, A., Dybjer, P. (eds.) *Semantics and Logics of Computation*. Publications of the Newton Institute (1997)
15. Winskel, G., Gutierrez, J., Clairambault, P.: The winning ways of concurrent games (2011) (in preparation)
16. Winskel, G.: Event structures with symmetry. *Electr. Notes Theor. Comput. Sci.* 172, 611–652 (2007)

Fibrational Induction Meets Effects

Robert Atkey², Neil Ghani², Bart Jacobs¹, and Patricia Johann²

¹ Radboud University, The Netherlands
bart@cs.ru.nl

² University of Strathclyde, Scotland
{Robert.Atkey,Neil.Ghani,Patricia.Johann}@cis.strath.ac.uk

Abstract. This paper provides several induction rules that can be used to prove properties of effectful data types. Our results are semantic in nature and build upon Hermida and Jacobs' fibrational formulation of induction for polynomial data types and its extension to all inductive data types by Ghani, Johann, and Fumex. An effectful data type $\mu(TF)$ is built from a functor F that describes data, and a monad T that computes effects. Our main contribution is to derive induction rules that are generic over *all* functors F and monads T such that $\mu(TF)$ exists. Along the way, we also derive a principle of definition by structural recursion for effectful data types that is similarly generic. Our induction rule is also generic over the kinds of properties to be proved: like the work on which we build, we work in a general fibrational setting and so can accommodate very general notions of properties, rather than just those of particular syntactic forms. We give examples exploiting the generality of our results, and show how our results specialize to those in the literature, particularly those of Filinski and Støvring.

1 Introduction

Induction is a powerful principle for proving properties of data types and the programs that manipulate them. Probably the simplest induction rule is the familiar one for the set of natural numbers: For any property P of natural numbers, if $P0$ holds, and if $P(n+1)$ holds whenever Pn holds, then Pn holds for all natural numbers n . As early as the 1960s, Burstall [2] observed that induction rules are definable for various forms of tree-like data types as well. The data types he considered can all be modelled by polynomial functors on \mathbf{Set} , and even today induction is most often used to prove properties of these types. But while most treatments of induction for tree-like data types use a specific notion of predicate, other reasonable notions are possible. For example, a predicate on a set A is often taken by type theorists to be a function $P : A \rightarrow \mathbf{Set}$, by category theorists to be an object of the slice category \mathbf{Set}/A , and by logicians to be a subset of A . Thus, even just for tree-like data types, induction rules are typically derived on an *ad hoc* basis, with seemingly different results available for the different kinds of properties of data types and their programs to be proved.

Until fairly recently a comprehensive and general treatment of induction remained elusive. But in 1998 Hermida and Jacobs [9] showed how to replace *ad hoc*

treatments of induction by a unifying axiomatic approach based on fibrations. The use of fibrations was motivated by the facts that i) the semantics of data types in languages involving, say, non-termination usually involves categories other than **Set**; ii) in such circumstances, standard set-based notions of predicates are no longer germane; iii) even when working in **Set** there are many reasonable notions of predicate (e.g., the three mentioned above); and iv) when deriving induction rules for more sophisticated classes of data types, we do not want to have to develop a specialised theory of induction for each one; We hope instead to appropriately instantiate a single, generic, axiomatic theory of induction that is widely applicable and abstracts over the specific choices of category, functor, and predicate giving rise to different induction rules for specific classes of data types. Fibrations support precisely such an axiomatic approach.

Although Hermida and Jacobs derive their induction rules only for data types modelled by polynomial functors, this result was recently extended to all inductive data types — i.e., all data types that are fixed points of functors — by Ghani, Johann, and Fumex [78]. Examples of non-polynomial inductive data types include rose trees, inductive families (e.g., perfect trees), quotient types (e.g., finite power sets), and hyperfunctions. These data types are sophisticated, but nevertheless are still pure, i.e., effect-free. This leads us to ask:

How can we reason inductively about data types in the presence of effects?

In this situation, we are interested in *effectful data structures* — i.e., data structures whose constructors can perform effectful computations — and the (possibly effectful) programs that manipulate them. Such programs can fail to terminate, raise exceptions, alter state, perform non-deterministic computations, and so on.

Moggi’s landmark paper [13] suggests that one way to handle effectful programs is to model effects via a monad T , where TX represents effectful computations that return values of type X with effects described by T . In Haskell, for example, input/output effects are modelled by the monad **IO**, and we can define the following effectful data type of IO-lists:

```
type IOList a = IO (IOList' a)

data IOList' a = IONil | IOCons a (IO (IOList' a))
```

For any list of type **IOList** a , some **IO** action must be performed to discover whether or not there is an element at the head of the list. Additional **IO** actions must be performed to obtain any remaining elements. Such a data type could be used to read list data “on demand” from some file or input device, for instance. Recalling that the standard append function is associative on pure lists, and observing that standard induction techniques for lists do not apply to functions on effectful data types, we can ask whether or not it is possible to prove by induction that the following effectful append function is associative on IO-lists:

```
appIO :: IOList a -> IOList a -> IOList a
appIO s t = do z <- s
             case z of IONil           -> t
                    IOCons w u -> return (IOCons w (appIO u t))
```

In fact, an even more fundamental question must first be addressed: How do we know that `appIO` is well-defined? After all, it is not at all obvious that the argument `u` to the recursive call of `appIO` is smaller than the original input `s`.

More generally, we can consider *effectful* data types given by the following type definitions. These generalise IO-lists by abstracting, via `f`, over the data structure involved and, via `m`, over the monad involved.

```
type D f m = m (Mu f m)
```

```
data Mu f m = In (f (m (Mu f m)))
```

We can then ask whether structural recursion can be used to define functions on data structures of type `D f m` and induction can be used to prove their properties.

Filinski and Støvring [5] provide partial answers to these questions. Taking types to be interpreted in the category *CPO* of ω -complete partial orders and total continuous functions, and taking a predicate to be an admissible subset of a *CPO*, they give a mathematically principled induction rule for establishing the truth of predicates for effectful strictly positive data types that can be modelled in *CPO*. Their induction rule is modular, in that they separate the premises for inductive reasoning about data structures from those for inductive reasoning about effects, and a number of examples are given to illustrate its use. Filinski and Støvring also give a principle of definition by structural recursion for effectful data types. But because they restrict attention to *CPO*, to a syntactically restricted class of functors, and to a particular notion of predicate, their results are not as widely applicable as we might hope.

In this paper we show how the fibrational approach to induction can be extended to the effectful setting. We obtain a generalisation of Filinski and Støvring’s induction rule that is completely free of the above three restrictions. We also derive a principle of definition by structural recursion for effectful data types that is similarly restriction-free. Our principle of definition can be used to show that `appIO` is well-defined, and our induction rule can be used to show that it is associative on IO-lists (see Example [6]). These results lie outside the scope of Filinski and Støvring’s work. (Interestingly, while the standard reverse function is an involution on lists, a similarly effectful reverse function is *not* an involution on IO-lists, so not all results transfer from the pure setting to the effectful one.) When specialised to the fibration of subobjects of *CPO* implicitly used by Filinski and Støvring, Theorem [2] and Corollary [1] give precisely their definition principle and induction rule, respectively. But because we treat functors that are not strictly positive, we are able to derive results for data types they cannot handle. Moreover, even if we restrict to the same class of functors as Filinski and Støvring, our fibrational approach allows us to derive, in addition to our generalisation of their modular one, another more powerful effectful induction rule (see Theorem [4]). More specifically, our contributions are as follows:

- Given a functor F and a monad T , we first show in Theorem [2] that the carrier $\mu(TF)$ of the initial TF -algebra deserves to be thought of as the effectful data type built from data described by F and effects computed by T . In

fact, the effectful data types introduced by the Haskell code above are all of the form $\mu(TF)$. Informally, the data type $\mu(TF)$ contains all interleavings of F and T ; formally, it is the free structure which is the carrier of both an F -algebra and Eilenberg-Moore algebra for T . Theorem 2 also gives a principle of definition by structural recursion for effectful data types.

- We then turn to the question of proof by induction and show that there are a number of useful induction rules for effectful data types. (See Corollary 1 and Theorems 4, 6, and 7.) We consider the relative merits of these rules, and note that in the proof-irrelevant case, which includes that considered by Filinski and Støvring, Theorems 4 and 6 coincide. We note that each of our induction rules also gives us a definitional format for dependently typed functions whose domains are effectful data types. This generalises the elimination rules for (pure) inductive data types in Martin-Löf Type Theory.
- Finally, we consider effectful induction in fibrations having very strong sums. Examples include the codomain and families fibrations, used heavily by category theorists and type theorists, respectively. In such fibrations, we show that the key operation of *lifting* is a strong monoidal functor from the category of endofunctors on the base category to the category of endofunctors on the total category of the fibration, and thus that liftings preserve monads. This ensures that all of our inductive reasoning can be performed in the total category of the fibration (see Section 6).

The rest of this paper is structured as follows. Section 2 introduces some categorical preliminaries. Section 3 formalises the notion of an effectful data type. It also shows how to construct algebras for the composition of two functors from algebras for each of the component functors, gives a converse construction when one of the functors is a monad, and uses these two constructions to generalise Filinski and Støvring’s induction rule to arbitrary effectful data types. This is all achieved without the use of fibrations, although, implicitly, Section 3 takes place entirely within the subobject fibration over CPO. Section 4 reviews the fibrational approach to induction. Section 5 relaxes the restriction to the subobject fibration on CPOs and uses arbitrary fibrations explicitly to further abstract the notion of predicate under consideration. We capitalise on this abstraction to give a number of different induction rules derivable in the fibrational setting. Section 6 considers induction in the presence of very strong sums. Section 7 concludes and discuss directions for further research.

2 Categorical Preliminaries

We assume familiarity with basic category theory, initial algebra semantics of data types, and the representation of computational effects as monads.

Let \mathcal{B} be a category and $F : \mathcal{B} \rightarrow \mathcal{B}$ be a functor. Recall that an F -algebra is a morphism $h : FX \rightarrow X$ for some object X of \mathcal{B} , called the *carrier* of h . For any functor F , the collection of F -algebras itself forms the category Alg_F . In Alg_F , an F -algebra morphism from the F -algebra $h : FX \rightarrow X$ to the F -algebra $g : FY \rightarrow Y$ is a morphism $f : X \rightarrow Y$ such that $f \circ h = g \circ Ff$. When it exists,

the initial F -algebra $in : F(\mu F) \rightarrow \mu F$ is unique. We write U_F for the *forgetful functor* mapping each F -algebra to its carrier, and we suppress the subscript F on U and on in when convenient.

Let \mathcal{B} be a category and (T, η, μ) be a monad on \mathcal{B} .¹ We write T for the monad (T, η, μ) when no confusion may result. An *Eilenberg-Moore algebra* for T is a T -algebra $h : TX \rightarrow X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id$; we can think of such algebra as a T -algebra that respects the unit and multiplication of T . The collection of Eilenberg-Moore algebras for a monad T forms the category \mathbf{EM}_T . In \mathbf{EM}_T , an \mathbf{EM}_T -*morphism* from an Eilenberg-Moore algebra $h : TX \rightarrow X$ to an Eilenberg-Moore algebra $g : TY \rightarrow Y$ is just a T -algebra morphism from h to g . It is easy to check that \mathbf{EM}_T is a full subcategory of \mathbf{Alg}_T .

3 Effectful Data Types and an Effectful Induction Rule

The carrier μF of the initial F -algebra can thought of as the data type defined by F . But if we are in an effectful setting, with effects modelled by a monad (T, η, μ) , then what type should we consider the effectful data type defined by F and T together? Whatever it is, it should be the carrier of an F -algebra f that describes the data. It should also be the carrier of a Eilenberg-Moore algebra g for T so that it respects η and μ . Finally, it should be the carrier of an algebra constructed from f and g that is initial in the same way that μF is the carrier of the initial F -algebra. We therefore define the effectful data type generated by F and T to be $\mu(TF)$. Moreover, $in \circ \eta$ is an F -algebra, and $in \circ \mu_T \circ T(in^{-1})$ is an Eilenberg-Moore algebra for T , both with carrier $\mu(TF)$. It is easy to verify that no T -algebra structure exists on the other obvious choice, namely $\mu(FT)$.

Unfortunately, however, carriers of initial TF -algebras can be hard to work with. We therefore write $\mu(TF)$ for $T(\mu(FT))$ when convenient. This is justified by the “rolling lemma” [6], which entails that if F and G are functors such that $\mu(GF)$ exists, then $\mu(FG)$ exists and $\mu(GF) = G(\mu(FG))$. The data types $\mathbf{Df m}$ from the introduction all have the form $T(\mu(FT))$. We establish that $T(\mu(FT))$ satisfies the above specification by first showing in Lemma 1 how to construct FT -algebras from F -algebras and T -algebras, and then refining this construction in Theorem 1 to take into account that T is a monad and we are actually interested in its Eilenberg-Moore algebras. We begin with a definition.

Definition 1 *Let $F, G : \mathcal{B} \rightarrow \mathcal{B}$ be functors, and let $\mathbf{Alg}_F \times_{\mathcal{B}} \mathbf{Alg}_G$ be defined by the pullback of $U_F : \mathbf{Alg}_F \rightarrow \mathcal{B}$ and $U_G : \mathbf{Alg}_G \rightarrow \mathcal{B}$ in \mathbf{Cat} . An F -and- G -algebra is an object of $\mathbf{Alg}_F \times_{\mathcal{B}} \mathbf{Alg}_G$, i.e., a triple comprising an object A of \mathcal{B} , an F -algebra $f : FA \rightarrow A$, and a G -algebra $g : GA \rightarrow A$. Morphisms of $\mathbf{Alg}_F \times_{\mathcal{B}} \mathbf{Alg}_G$ are morphisms of \mathcal{B} that are simultaneously F -algebra and G -algebra morphisms.*

Lemma 1. *Let $F, G : \mathcal{B} \rightarrow \mathcal{B}$ be functors. There is a functor $\Phi : \mathbf{Alg}_F \times_{\mathcal{B}} \mathbf{Alg}_G \rightarrow \mathbf{Alg}_{FG}$ that sends F -and- G -algebras to FG -algebras.*

¹ We use μ to denote both least fixed points of functors and multiplication operators of monads as is traditional. Which is meant when will be made clear from context.

Proof. Define $\Phi(A, f, g) = f \circ Fg$. The action of Φ on morphisms is obvious. \square

In the setting of effectful data types, where G is a monad in whose Eilenberg-Moore algebras we are interested, Lemma [1](#) can be strengthened.

Definition 2 Let $F : \mathcal{B} \rightarrow \mathcal{B}$ be a functor, let (T, η, μ) be a monad on \mathcal{B} , and let $\mathbf{Alg}_F \times_B \mathbf{EM}_T$ be the category defined by the pullback of $U_F : \mathbf{Alg}_F \rightarrow \mathcal{B}$ and $U_T : \mathbf{EM}_T \rightarrow \mathcal{B}$ in \mathbf{Cat} . An F -and- T -Eilenberg-Moore algebra is an object of $\mathbf{Alg}_F \times_B \mathbf{EM}_T$, i.e., a triple comprising an object A of \mathcal{B} , an F -algebra $f : FA \rightarrow A$, and an Eilenberg-Moore algebra $g : TA \rightarrow A$ for T . Morphisms of $\mathbf{Alg}_F \times_B \mathbf{EM}_T$ are morphisms of \mathcal{B} that are simultaneously F -algebra morphisms and \mathbf{EM}_T -morphisms, i.e., F -algebra morphisms and T -algebra morphisms.

The key point about F -and- T -Eilenberg-Moore algebras is that the functor mapping them to FT -algebras has a left adjoint. Since every Eilenberg-Moore algebra for T is a T -algebra, we abuse notation and also call this functor Φ .

Theorem 1. Let $F : \mathcal{B} \rightarrow \mathcal{B}$ be a functor and (T, η, μ) be a monad on \mathcal{B} . The functor $\Phi : \mathbf{Alg}_F \times_B \mathbf{EM}_T \rightarrow \mathbf{Alg}_{FT}$ has a left adjoint $\Psi : \mathbf{Alg}_{FT} \rightarrow \mathbf{Alg}_F \times_B \mathbf{EM}_T$.

Proof. Define Ψ by $\Psi(k : FTA \rightarrow A) = (TA, \eta \circ k : FTA \rightarrow TA, \mu : T^2A \rightarrow TA)$ on objects and by $\Psi f = Tf$ on morphisms. For any FT -algebra morphism $f : A \rightarrow B$, naturality of η and μ ensure that Ψf is a morphism in $\mathbf{Alg}_F \times_B \mathbf{EM}_T$.

To see that Φ and Ψ are adjoint let $k : FTA \rightarrow A$ be an object of \mathbf{Alg}_{FT} and $(B, f : FB \rightarrow B, g : TB \rightarrow B)$ be an object of $\mathbf{Alg}_F \times_B \mathbf{EM}_T$. We construct a natural isomorphism between morphisms from Ψk to (B, f, g) and morphisms from k to $\Phi(B, f, g)$. Given $h : \Psi k \rightarrow (B, f, g)$, define $\phi(h) : k \rightarrow \Phi(B, f, g)$ by $\phi(h) = h \circ \eta$. Then $\phi(h)$ is an FT -algebra morphism because $f \circ Fg \circ FT\phi(h) = f \circ Fg \circ FT h \circ FT \eta = f \circ Fh \circ F\mu \circ FT \eta = f \circ Fh = h \circ \eta \circ k = \phi(h) \circ k$. Here, the first equality holds because FT is a functor, the second holds because h is a T -algebra morphism, the third by the monad laws, and the fourth because h is an F -algebra morphism. Conversely, given $h : k \rightarrow \Phi(B, f, g)$, define $\psi(h) : \Psi k \rightarrow (B, f, g)$ by $\psi(h) = g \circ Th$. Then $\psi(h)$ is an F -algebra morphism because $\psi(h) \circ \eta \circ k = g \circ Th \circ \eta \circ k = g \circ \eta \circ h \circ k = h \circ k = f \circ Fg \circ FT h = f \circ F\psi(h)$. Here, the second equality holds by naturality of η , the third because g is an Eilenberg-Moore algebra for T , and the fourth since h is an FT -algebra morphism. Moreover, $\psi(h)$ is a T -algebra morphism because $g \circ T\psi(h) = g \circ Tg \circ TTh = g \circ \mu \circ TTh = g \circ Th \circ \mu = \psi(h) \circ \mu$. Here, the second equality holds since g is an Eilenberg-Moore algebra for T and the third holds by naturality of μ .

To see that ϕ and ψ constitute an isomorphism, first note that $\phi(\psi(h)) = \phi(g \circ Th) = g \circ Th \circ \eta = g \circ \eta \circ h = h$ by naturality of η and the fact that g is an Eilenberg-Moore algebra for T . We also have that $\psi(\phi(h)) = \psi(h \circ \eta) = g \circ Th \circ T\eta = h \circ \mu \circ T\eta = h$ by the fact that h is a T -algebra morphism and the monad laws. Naturality of ϕ and ψ is easily checked. \square

A slightly slicker proof abstracts away from the category of Eilenberg-Moore algebras for T to any adjunction $L \dashv R : \mathcal{B} \rightarrow \mathcal{D}$ whose induced monad RL is T . In this setting, we define $\mathbf{Alg}_F \times_B \mathcal{D}$ to be the pullback of the forgetful

functor U_F and R . The adjunction $L \dashv R : \mathcal{B} \rightarrow \mathcal{D}$ then lifts to an adjunction $L^\dagger \dashv R^\dagger : \mathbf{Alg}_{FT} \rightarrow \mathbf{Alg}_F \times_{\mathcal{B}} \mathcal{D}$. Theorem [1](#) is the special case of this more general construction for which \mathcal{D} is \mathbf{EM}_T . Another special case takes \mathcal{D} to be the Kleisli category of T .

We can now give our principle of definition by effectful structural recursion.

Theorem 2. *Let F be a functor and T be a monad. Then $T(\mu(FT))$, if it exists, is the carrier of the initial F -and- T -Eilenberg-Moore algebra.*

Proof. Since in_{FT} is the initial FT -algebra, and since left adjoints preserve initial objects, we have that the initial F -and- T -Eilenberg-Moore algebra is $\Psi \mathit{in}_{FT}$, i.e., the triple $(T(\mu(FT)), \eta \circ \mathit{in}, \mu)$. \square

Given an F -and- T -Eilenberg-Moore algebra (A, f, g) , Theorem [2](#) ensures the existence of a unique F -and- T -Eilenberg-Moore algebra morphism from the initial such algebra to (A, f, g) . This gives a morphism from $T(\mu(FT))$ to A , and hence a principle of definition by effectful structural recursion. Indeed, when T is the identity monad, we recover precisely the standard principle of definition by structural recursion for (pure) carriers of initial algebras.

Example 1. We can place the definition of `appIO` from the introduction on a formal footing as follows. First note that `IOList a` is of the form $T(\mu(FT))$, where $FX = 1 + a \times X$ and T is the monad `IO`. The F -and- T -Eilenberg-Moore algebra whose F -algebra sends `inl *` to `ys` and `inr(z, zs)` to $(\eta \circ \mathit{in})(\mathit{inr}(z, zs))$, and whose Eilenberg-Moore algebra for T is μ , defines `appIO _ ys`. It further ensures that `appIO _ ys` is a T -algebra morphism between the T -algebra structure within the initial F -and- T -Eilenberg-Moore algebra and the T -algebra structure within the F -and- T -Eilenberg-Moore algebra just defined.

From Theorem [2](#) we also get the first of our effectful induction rules.

Corollary 1. *Let P be a subobject of $T(\mu(FT))$, as well as the carrier of an F -and- T -Eilenberg-Moore algebra such that the inclusion map from P to $T(\mu(FT))$ is an F -and- T -Eilenberg-Moore algebra morphism. Then $P = T(\mu(FT))$.*

Proof. There is an F -and- T -Eilenberg-Moore algebra morphism from the initial such algebra to any with carrier P , and this induces a morphism from $T(\mu(FT))$ to P . That this morphism is an inverse to the inclusion map from P to $T(\mu(FT))$ follows from initiality and the fact that the inclusion map is monic. \square

In Theorem [7](#) below we generalise Corollary [1](#) to handle more general notions of predicate than that given by subobjects. But first we argue that Corollary [1](#) specialises to recover Filinski and Støvring’s induction rule. This rule assumes a minimal T -invariant (TC, i) for F and a subset P of TC that is both T -admissible and F -closed, and concludes that $P = TC$. But i) the minimal T -invariant for F is precisely the initial F -and- T -algebra $T(\mu(FT))$ in `CPO`, ii) $P \subseteq T(\mu(FT))$ is T -admissible iff there exists a T -algebra $k : TP \rightarrow P$ such that the inclusion map ι from P to $T(\mu(FT))$ is a T -algebra morphism from k to $\mu : T^2(\mu(FT)) \rightarrow T(\mu(FT))$, and iii) $P \subseteq T(\mu(FT))$ is F -closed iff there exists

an F -algebra $h : FP \rightarrow P$ such that ι is an F -algebra morphism from h to $\eta \circ in$. Thus, P is the carrier of an F -and- T -Eilenberg-Moore algebra. Moreover, since k coincides with μ and h coincides with $\eta \circ in$ on P , ι is an F -and- T -Eilenberg-Moore algebra morphism. Thus, by Corollary [III](#) $P = T(\mu(FT))$. Of course this observation allows us to handle all of Filinski and Støvring's examples.

4 Induction in a Fibrational Setting

Thus far we have characterised effectful data types and given our first induction rule for them. This rule is generic over the category interpreting data types, as well as over both the monad interpreting the effects in question and the functor constructing the data type, and specialises to Filinski and Støvring's rule. On the other hand, it holds only for a specific notion of predicate, namely that given by subobjects. Since we seek an induction rule that is also generic over predicates, we turn to fibrations, which support an axiomatic approach to them. We begin by recalling the basics of fibrations. More details can be found in, e.g., [\[10\]](#).

Let $U : \mathcal{E} \rightarrow \mathcal{B}$ be a functor. A morphism $g : Q \rightarrow P$ in \mathcal{E} is *cartesian* over a morphism $f : X \rightarrow Y$ in \mathcal{B} if $Ug = f$, and for every $g' : Q' \rightarrow P$ in \mathcal{E} for which $Ug' = f \circ v$ for some $v : UQ' \rightarrow X$ there exists a unique $h : Q' \rightarrow Q$ in \mathcal{E} such that $Uh = v$ and $g \circ h = g'$. The cartesian morphism f_P^\S over a morphism f with codomain UP is unique. We write f^*P for the domain of f_P^\S .

Cartesian morphisms are the essence of fibrations. A functor $U : \mathcal{E} \rightarrow \mathcal{B}$ is a *fibration* if for every object P of \mathcal{E} and every morphism $f : X \rightarrow UP$ in \mathcal{B} there is a cartesian morphism $f_P^\S : f^*P \rightarrow P$ in \mathcal{E} such that $U(f_P^\S) = f$. If $U : \mathcal{E} \rightarrow \mathcal{B}$ is a fibration, we call \mathcal{B} the *base category* of U and \mathcal{E} the *total category* of U . Objects of \mathcal{E} can be thought of as predicates, objects of \mathcal{B} can be thought of as types, and U can be thought of as mapping each predicate P in \mathcal{E} to the type UP on which P is a predicate. We say that an object P in \mathcal{E} is *over* its image UP under U , and similarly for morphisms. For any object X of \mathcal{B} , we write \mathcal{E}_X for the *fibre over* X , i.e., for the subcategory of \mathcal{E} consisting of objects over X and vertical morphisms, i.e., morphisms over id_X . If $f : X \rightarrow Y$ is a morphism in \mathcal{B} , then the function mapping each object P of \mathcal{E} to f^*P extends to a functor $f^* : \mathcal{E}_Y \rightarrow \mathcal{E}_X$. We call the functor f^* the *reindexing functor induced by* f .

Example 2. The category $\text{Fam}(\text{Set})$ has as objects pairs (X, P) , where X is a set and $P : X \rightarrow \text{Set}$. We refer to (X, P) simply as P when convenient and call X its *domain*. A morphism from $P : X \rightarrow \text{Set}$ to $P' : X' \rightarrow \text{Set}$ is a pair $(f, f^\sim) : P \rightarrow P'$, where $f : X \rightarrow X'$ and $f^\sim : \forall x : X. Px \rightarrow P'(fx)$. The functor $U : \text{Fam}(\text{Set}) \rightarrow \text{Set}$ mapping (X, P) to X is called the *families fibration*.

Independently typed programmers typically work in the families fibration, in which induction amounts to defining dependently typed functions. It can be generalised (in an equivalent form) to the following fibration.

Example 3. Let \mathcal{B} be a category. The *arrow category* of \mathcal{B} , denoted \mathcal{B}^\rightarrow , has the morphisms of \mathcal{B} as its objects. A morphism in \mathcal{B}^\rightarrow from $f : X \rightarrow Y$ to

$f' : X' \rightarrow Y'$ is a pair (α_1, α_2) of morphisms in \mathcal{B} such that $f' \circ \alpha_1 = \alpha_2 \circ f$. The codomain functor $cod : \mathcal{B}^\rightarrow \rightarrow \mathcal{B}$ maps an object $f : X \rightarrow Y$ of \mathcal{B}^\rightarrow to the object Y of \mathcal{B} . If \mathcal{B} has pullbacks, then cod is a fibration, called the *codomain fibration over \mathcal{B}* : given an object $f : X \rightarrow Y$ in $(\mathcal{B}^\rightarrow)_Y$ and a morphism $f' : X' \rightarrow Y$ in \mathcal{B} , the pullback of f and f' gives a cartesian morphism over f at f' .

Example 4. If \mathcal{B} is a category, then the category of subobjects of \mathcal{B} , denoted $Sub(\mathcal{B})$, has (equivalence classes of) monomorphisms in \mathcal{B} as its objects. A monomorphism $f : X \hookrightarrow Y$ is called a *subobject* of Y . A morphism in $Sub(\mathcal{B})$ from $f : X \hookrightarrow Y$ to $f' : X' \hookrightarrow Y'$ is a map $\alpha_2 : Y \rightarrow Y'$ for which there exists a unique map $\alpha_1 : X \rightarrow X'$ such that $\alpha_2 \circ f = f' \circ \alpha_1$. The map $U : Sub(\mathcal{B}) \rightarrow \mathcal{B}$ sending $f : X \hookrightarrow Y$ to Y extends to a functor. If \mathcal{B} has pullbacks then U is a fibration since the pullback of a monomorphism is a monomorphism. In this case, U is called the *subobject fibration over \mathcal{B}* .

We also need the notion of an opfibration. Abstractly, $U : \mathcal{E} \rightarrow \mathcal{B}$ is an opfibration iff $U : \mathcal{E}^{op} \rightarrow \mathcal{B}^{op}$ is a fibration. More concretely, U is an opfibration if for every object P of \mathcal{E} and every morphism $f : UP \rightarrow Y$ in \mathcal{B} there is an *opcartesian morphism* $f_{\S}^P : P \rightarrow \Sigma_f P$ in \mathcal{E} over f . Moreover, if $f : X \rightarrow Y$ is a morphism in \mathcal{B} , then the function mapping each object P of \mathcal{E}_X to $\Sigma_f P$ extends to a functor $\Sigma_f : \mathcal{E}_X \rightarrow \mathcal{E}_Y$ which we call the *opreindexing functor*. A functor is a *bifibration* if it is both a fibration and an opfibration. The families and codomain fibrations are examples of bifibrations. More generally, a fibration is a bifibration iff, for every morphism $f : X \rightarrow Y$ in \mathcal{B} , f^* is left adjoint to Σ_f .

We can now give the key definitions and results for our fibrational approach to induction. If $U : \mathcal{E} \rightarrow \mathcal{B}$ is a fibration and $F : \mathcal{B} \rightarrow \mathcal{B}$ is a functor, then a *lifting* of F with respect to U is a functor $\hat{F} : \mathcal{E} \rightarrow \mathcal{E}$ such that $U\hat{F} = FU$. We say that U has *fibred terminal objects* if each fibre has a terminal object and reindexing functors preserve them. In this case, the functor $\top : \mathcal{B} \rightarrow \mathcal{E}$ mapping each object to the terminal object of the fibre over it is called the *truth functor* for U . A lifting \hat{F} of F is called *truth-preserving* if $\top F = \hat{F}\top$.

Example 5. A truth-preserving lifting F^\rightarrow of F with respect to the codomain fibration cod is given by the action of F on morphisms.

A *comprehension category with unit* (or *CCU*, for short) is a fibration $U : \mathcal{E} \rightarrow \mathcal{B}$ that has fibred terminal objects and is such that the terminal object functor \top has a right adjoint $\{-\}$. In this case, $\{-\}$ is called the *comprehension functor* for U . If ϵ is the counit of the adjunction $\top \dashv \{-\}$, then defining $\pi_P = U\epsilon_P$ gives a *projection* natural transformation from $\{P\}$ to UP . Truth-preserving liftings with respect to CCUs are used in [9] to give induction rules. The key result is:

Theorem 3. *Let $U : \mathcal{E} \rightarrow \mathcal{B}$ be a CCU and $F : \mathcal{B} \rightarrow \mathcal{B}$ be a functor such that μF exists, and let \hat{F} be a truth-preserving lifting of F . Then for every object P of \mathcal{E} and every algebra $\alpha : \hat{F}P \rightarrow P$, there is a unique morphism $ind_F \alpha : \mu F \rightarrow \{P\}$ such that $\pi_P \circ ind_F \alpha = fold(U\alpha)$.*

The proof consists of constructing a right adjoint $\langle - \rangle : \mathbf{Alg}_{\hat{F}} \rightarrow \mathbf{Alg}_F$ mapping \hat{F} -algebras with carrier P to F -algebras with carrier $\{P\}$. Given an \hat{F} -algebra

$\alpha : \hat{F}P \rightarrow P$, we can define $\text{ind}_F \alpha$ to be $\text{fold} \langle \alpha \rangle : \mu F \rightarrow \{P\}$. For second part of the theorem, first note that π_P is an F -algebra morphism from $\langle \alpha \rangle$ to $U\alpha$. Then, by the uniqueness of *folds*, we have that $\pi_P \circ \text{ind}_F \alpha = \pi_P \circ \text{fold} \langle \alpha \rangle = \text{fold}(U\alpha)$.

Fibrations thus provide just the right structure for defining induction rules for inductive data types. Although the truth-preserving liftings are given in [9] only for polynomial functors, this restriction was removed in [7,8], which showed that in a *Lawvere fibration* — i.e., a CCU that is also a bifibration — *every* functor has a truth-preserving lifting. Indeed, observing that π extends to a functor $\pi : \mathcal{E} \rightarrow \mathcal{B}^{\rightarrow}$ with left adjoint $I : \mathcal{B}^{\rightarrow} \rightarrow \mathcal{E}$ defined by $I(f : X \rightarrow Y) = \Sigma_f(\top X)$, we have that for any functor F , $\hat{F} = IF^{\rightarrow}\pi : \mathcal{E} \rightarrow \mathcal{E}$ is a truth-preserving lifting with respect to U , where F^{\rightarrow} is the lifting from Example 5. If μF exists, then Theorem 3 guarantees that it has an induction rule.

5 Effectful Induction

In the remainder of the paper we assume a Lawvere fibration $U : \mathcal{E} \rightarrow \mathcal{B}$, a functor $F : \mathcal{B} \rightarrow \mathcal{B}$, and a monad (T, η, μ) on \mathcal{B} . We further assume that $\mu(FT)$ exists. Our first effectful induction rule is obtained by recalling that $T(\mu(FT))$ is the initial TF -algebra and instantiating Theorem 3 for TF .

Theorem 4. *For every object P of \mathcal{E} and algebra $\alpha : (\widehat{TF})P \rightarrow P$ there is a unique morphism $\text{ind}_{TF} \alpha : T(\mu(FT)) \rightarrow \{P\}$ with $\pi_P \circ \text{ind}_{TF} \alpha = \text{fold}(U\alpha)$.*

Unfortunately, the induction rule in Theorem 4 is more complicated than we would like since the rule requires the user to supply a \widehat{TF} -algebra, and thus to deal with F and T at the same time, rather than separately as in Corollary 1. To produce a fibrational variant of Corollary 1, we therefore need to understand the relationship between \widehat{TF} and $\hat{T}\hat{F}$. We turn to this now.

Lemma 2. *If F and G are functors on \mathcal{B} , and $\alpha : F \rightarrow G$ is a natural transformation, then there is a natural transformation $\hat{\alpha} : \hat{F} \rightarrow \hat{G}$.*

Proof. Since I and π are functors, we can define $\hat{\alpha} = I\alpha^{\rightarrow}\pi$, where $\alpha^{\rightarrow} : F^{\rightarrow} \rightarrow G^{\rightarrow}$ maps $f : X \rightarrow Y$ in $\mathcal{B}^{\rightarrow}$ to the naturality square for α at f . \square

Theorem 5. *The lifting operation $\widehat{(-)}$ defines a lax monoidal functor mapping functors on \mathcal{B} to functors on \mathcal{E} .*

Proof. That $\widehat{(-)}$ preserves identity and composition of natural transformations is verified by simple calculation, so it is indeed a functor. To see that this functor is lax monoidal, we need natural transformations from \widehat{Id} to Id and from \widehat{FG} to $\hat{F}\hat{G}$. We take the former to be the counit of the adjunction $I \dashv \pi$. For the latter, define $\sigma : \widehat{FG} \rightarrow \hat{F}\hat{G}$ — i.e., $\sigma : I(FG)^{\rightarrow}\pi \rightarrow IF^{\rightarrow}\pi IG^{\rightarrow}\pi$ — by $\sigma = IF^{\rightarrow}\eta G^{\rightarrow}\pi$, where η is the unit of the adjunction $I \dashv \pi$.

We will use the natural transformation $\sigma : \widehat{FG} \rightarrow \hat{F}\hat{G}$ from the proof of Theorem 5 in the proof of Theorem 6 below. Note that if σ were oplax rather than

lax — i.e., if we had $\sigma : \widehat{FG} \rightarrow \widehat{FG}$ — then \widehat{T} would be a monad whenever T is. An induction rule for effectful data types that assumes \widehat{T} is a monad is discussed in Section 6. For now, we derive induction rules for effectful data types that hold even when \widehat{T} is not a monad. The first is a fibrational variant of Corollary 11.

Theorem 6. *Let P be an object of \mathcal{E} , and let $f : \widehat{FP} \rightarrow P$ and $g : \widehat{TP} \rightarrow P$ be morphisms of \mathcal{E} . Then there is a unique morphism $h : T(\mu(FT)) \rightarrow \{P\}$ in \mathcal{B} such that $\pi_P \circ h = \text{fold}(Ug \circ TUF)$. If P is over $T(\mu(FT))$, g is over μ , and f is over $\eta \circ \text{in}$, then $\pi_P \circ h = \text{id}$.*

Proof. From the algebra $\Phi(P, g, f) : \widehat{T}\widehat{FP} \rightarrow P$, where Φ is as in Lemma 11, we can construct the \widehat{TF} -algebra $\Phi(P, g, f) \circ \sigma_P$. By Theorem 4, there exists a morphism $h : T(\mu(FT)) \rightarrow \{P\}$ in \mathcal{B} such that $\pi_P \circ h = \text{fold}(U(\Phi(P, g, f) \circ \sigma_P))$. If $f : X \rightarrow Y$ is an object of \mathcal{B}^\rightarrow , then η_f is a pair whose second component is id . The second component of $F^\rightarrow \eta_f$ is thus also id , so by the definition of I we have that $IF^\rightarrow \eta_f$ is vertical. Since $\sigma_P = IF^\rightarrow \eta_{G \rightarrow \pi_P}$, σ_P is also vertical. Thus $\text{fold}(U(\Phi(P, g, f) \circ \sigma_P)) = \text{fold}(U(\Phi(P, g, f)))$, and by the definition of Φ and the fact that \widehat{T} is a lifting of T , we have that $\pi_P \circ h = \text{fold}(Ug \circ TUF)$ as desired. If g is over μ and f is over $\eta \circ \text{in}$, then $\pi_P \circ h = \text{fold}(Ug \circ TUF) = \text{fold}(\mu \circ T\eta \circ T(\text{in})) = \text{fold}(T(\text{in})) = \text{fold in} = \text{id}$. \square

The condition $\pi_P \circ h = \text{id}$ ensures that h maps every element t of $T(\mu(FT))$ to a proof that Pt holds. We will make good use of the following generalisation of Prop. 2.13 in 5, which shows how to build new \widehat{T} -algebras from old.

Lemma 3. *Let $k : TA \rightarrow A$ be an Eilenberg-Moore algebra for T .*

1. *Let δ be the natural transformation defined by $\delta_A : A \rightarrow A \times A$ and consider the equality predicate $\text{Eq}_A = \Sigma_\delta \top(A)$. Then $U(\text{Eq}_A) = A \times A$ and $\langle k \circ T\pi_1, k \circ T\pi_2 \rangle : T(A \times A) \rightarrow A \times A$ is an Eilenberg-Moore algebra for T . If $\{\text{Eq}_A\} = A$ for every $A \in \mathcal{B}$, then there exists a morphism $h : \widehat{T} \text{Eq}_A \rightarrow \text{Eq}_A$ such that $Uh = \langle k \circ T\pi_1, k \circ T\pi_2 \rangle$.*
2. *Let $h : \widehat{TP} \rightarrow P$ be a \widehat{T} -algebra such that $UP = A$ and $Uh = k$. Then for every Eilenberg-Moore algebra $k' : TB \rightarrow B$ for T and T -algebra morphism $f : B \rightarrow A$, there exists a morphism $h' : \widehat{T}(f^*P) \rightarrow f^*P$ such that $Uh' = k'$.*
3. *Let I be a set and suppose \mathcal{E} has I -indexed products in its fibres. Then for any I -indexed family $(P_i, h_i : \widehat{TP}_i \rightarrow P_i)$ of \widehat{T} -algebras with $UP_i = A$ and $Uh_i = k$ for all i , there is a morphism $h : \widehat{T}(\prod_{i \in I} P_i) \rightarrow \prod_{i \in I} P_i$ with $Uh = k$.*

Proof. The first part follows from a lemma in 10, and the third part follows from the universal property of products. For the second part, note that because h is over k , there is a vertical morphism $v : \widehat{TP} \rightarrow k^*P$. We construct $h' : \widehat{T}(f^*P) \rightarrow P$ by first noting that, since f is a T -Eilenberg-Moore algebra morphism, we have $(Tf)^*(k^*P) = k'^*(f^*P)$. We can then take h' to be the composition of $j : \widehat{T}(f^*P) \rightarrow (Tf)^*(\widehat{TP})$ and $(Tf)^*v : (Tf)^*(\widehat{TP}) \rightarrow (Tf)^*(k^*P)$ and $k'_{f^*P}^\S : k'^*(f^*P) \rightarrow f^*P$. Here, j is constructed by first observing that $\widehat{T}(f^*P) = \Sigma_{T\pi_{f^*P}} \top(T\{f^*P\})$ by definition of \widehat{T} , and that this is $\Sigma_{T\pi_{f^*P}}(T\{f_P^\S\})^* \top(T\{P\})$

because reindexing preserves terminal objects. We then construct a morphism from $\Sigma_{T\pi_{f^*P}}(T\{f_P^{\S}\})^*\top(T\{P\})$ to $(Tf)^*\Sigma_{T\pi_P}\top(T\{P\})$ using a morphism induced by a natural transformation from $\Sigma_{T\pi_{f^*P}}(T\{f_P^{\S}\})^*$ to $(Tf)^*\Sigma_{T\pi_P}$ that is itself constructed using i) the fact that f is a T -algebra morphism, ii) the unit of the adjunction $\Sigma_{T\pi_P} \dashv (T\pi_P)^*$, and iii) the counit of the adjunction $\Sigma_{T\pi_{f^*P}} \dashv (T\pi_{f^*P})^*$. Noting that $(Tf)^*\Sigma_{T\pi_P}\top(T\{P\}) = (Tf)^*(\hat{T}P)$ by the definition of \hat{T} completes the proof. \square

Example 6. We can now prove that `appIO` is associative. We use the families fibration, and so exploit the generality of our framework over that of [5] by working with a base category other than `CPO` and a notion of predicate other than that given by subobjects. The predicate $P : \text{IOList } \mathbf{a} \rightarrow \text{Set}$ sends \mathbf{x} s to 1 if, for all \mathbf{y} s \mathbf{z} s: `IOList a`, `appIO xs (appIO ys zs) = appIO (appIO xs ys) zs` holds and to \emptyset otherwise. Recalling that `IOList a` is $T(\mu(FT))$ for F and T as in Example [1], we can show that P is the carrier of a \hat{T} -algebra by observing that i) the equality predicate on `IOList a` is the carrier of a T -algebra over $\langle \mu \circ T\pi_1, \mu \circ T\pi_2 \rangle$ by the first part of Lemma [3]; ii) for all \mathbf{y} s \mathbf{z} s : `IOList a`, the predicate `appIO xs (appIO ys zs) = appIO (appIO xs ys) zs` on `IOList a` \times `IOList a` is the carrier of a \hat{T} -algebra over μ by the second part of Lemma [3]; and iii) P is thus the carrier of a \hat{T} -algebra over μ by the third part of Lemma [3]. In addition, P is the carrier of an \hat{F} -algebra: indeed by direct calculation we have that the predicate $\hat{F}P : FX \rightarrow \text{Set}$ sends `inl *` to 1 and sends `inr(x, xs)` to Pxs . An \hat{F} -algebra with carrier P over $\eta \circ \text{in}$ is therefore given by an element of $P(\text{return IONil})$ and, for every $\mathbf{x}:\mathbf{a}$ and \mathbf{x} s : `IOList a`, a function $P\mathbf{x}s \rightarrow P(\text{return IOCons } \mathbf{x} \mathbf{x}s)$. Both can be computed directly using the definition of `appIO`. By Theorem [6], we thus have that P holds for all elements of `IOList a`.

Example 7. We can exploit our ability to move beyond the effectful strictly positive data types treated by Filinski and Støvring to reason about *indexed effectful data types*. We work in the subobject fibration over $\text{Set}^{\mathbb{N}}$ (see Example [4]). For any set A and monad T on $\text{Set}^{\mathbb{N}}$, consider the \mathbb{N} -indexed data type of effectful perfect trees with data from A and effects from T given by $T(\mu(F_A T)) : \text{Set}^{\mathbb{N}}$, where, $F_A(X : \text{Set}^{\mathbb{N}}) = \lambda n. \{ * \mid n = 0 \} + \{ (a, x_1, x_2) \mid \exists n'. a \in A, x_1 \in Xn', x_2 \in Xn', n = n' + 1 \}$. By Theorem [2], if $f : A \rightarrow B$ then we can define a morphism $\text{map}(f, -) : T(\mu(F_A T)) \rightarrow T(\mu(F_B T))$ in $\text{Set}^{\mathbb{N}}$ by giving an F_A -and- T -Eilenberg-Moore algebra with carrier $T(\mu(F_B T))$. The F_A -algebra sends, when $n = 0$, `inl *` to $\eta(\text{in}(\text{inl } *))$, and, when $n = n' + 1$, `inr(a, x1, x2)` to $\eta(\text{in}(\text{inr}(f a, x_1, x_2)))$. The Eilenberg-Moore algebra for T is just the multiplication μ of T . Now define the subobject $i : P \hookrightarrow T(\mu(F_A T))$ in $\text{Sub}(\text{Set}^{\mathbb{N}})$ by $Pn = \{ t : T(\mu(F_A T))n \mid \forall f, g. \text{map}(f, \text{map}(g, t)) = \text{map}(f \circ g, t) \}$. To show that $P = T(\mu(F_A T))$, and hence that map preserves composition, we apply Theorem [6]. As in Example [6], we use Lemma [3] to give a \hat{T} -algebra on i over μ ; this uses the fact that $\text{map}(f, -)$ is a T -Eilenberg-Moore algebra morphism by construction. The existence of a \hat{F} -algebra on i over $\eta \circ \text{in}$ follows by direct calculation. By Theorem [6] there is a morphism $h : T(\mu(F_A T)) \rightarrow \{P\}$ such that

$\pi_P \circ h = id$. But since $\{P\} = P$ in $Sub(\mathbf{Set}^{\mathbb{N}})$, we have that i and h together give $P = T(\mu(F_A T))$.

Should we be satisfied with the effectful induction rule in Theorem 6? It is not as expressive as that in Theorem 4 since not all $\widehat{T}\widehat{F}$ -algebras arise from \widehat{T} -algebras and \widehat{F} -algebras individually, but it is easier to use since we can check whether or not predicates have \widehat{T} -algebra structures without considering the functor \widehat{F} at all and vice-versa. However, neither rule ensures that h respects the structure of the monad, i.e., is an F -and- T -Eilenberg-Moore algebra morphism. As we now see, this is the case if $\langle g \rangle$ is an Eilenberg-Moore algebra for T .

Theorem 7. *Let $f : \widehat{F}P \rightarrow P$ and $g : \widehat{T}P \rightarrow P$ be morphisms of \mathcal{E} such that $\langle g \rangle$ is an Eilenberg-Moore algebra for T . Then there is a unique $h : T(\mu(FT)) \rightarrow \{P\}$ in \mathcal{B} that is an F -and- T -Eilenberg-Moore algebra morphism. Further, $\pi_p \circ h = fold(Uf \circ F Ug)$. If g is over μ and f is over $\eta \circ in$, then $\pi_p \circ h = id$.*

Proof. Since $(\{P\}, \langle f \rangle, \langle g \rangle)$ is an F -and- T -Eilenberg-Moore algebra there is a unique F -and- T -Eilenberg-Moore algebra morphism from the initial such algebra to it. Since $T(\mu(FT))$ is the carrier of the initial F -and- T -Eilenberg-Moore algebra, this gives the required morphism $h : T(\mu(FT)) \rightarrow \{P\}$. Indeed, h is the morphism guaranteed by Theorem 6, and so $\pi_p \circ h = id$ as desired. \square

As expected, the effectful world contains the pure world. For example, if (T, η, μ) is a monad, then we can think of η as a family of functions $\eta_X : X \rightarrow TX$ mapping values of type X to the pure computations that just return those values. Similarly, an effectful data structure $T(\mu(FT))$ contains the pure data structure μF : indeed, the natural transformation $\eta F : F \rightarrow TF$ and the functoriality of the fixed point operator μ together give the inclusion $\mu(\eta F) : \mu F \rightarrow T(\mu(FT))$. Moreover, if P is a property over $T(\mu(FT))$, then $(\mu(\eta F))^* P$ is a property over μF . Thus given $f : \widehat{F}P \rightarrow P$ and $g : \widehat{T}P \rightarrow P$ such that $\langle g \rangle$ is an Eilenberg-Moore algebra for T , we may ask about the relationship between proofs of $(\mu(\eta F))^* P$ for μF obtained from f by Theorem 3 and proofs of P for $T(\mu(FT))$ obtained from f and g by Theorem 7. It is not hard to see that induction for $T(\mu(FT))$ specialises to induction for μF when $T(\mu(FT))$ is pure.

Finally, note that in a fibred preorder, for any \widehat{T} -algebra g , $\langle g \rangle$ is always an Eilenberg-Moore algebra for T . This is the case for the subobject fibration implicitly used by Filinski and Støvring since, there, admissible predicates are Eilenberg-Moore algebras over the multiplication μ of T . However, in the non-fibred preorder case, there are a variety of different induction rules which are possible. This reflects a trade-off: the more we assume, the better behaved our induction proof is! Our default preference is for structure and we believe that Theorem 7 provides the best rule. However, if we cannot establish the stronger premises so as to obtain the better behaved induction rules, it is comforting to know that the induction rules of Theorems 4 and 6 are still available.

6 A More Logical Treatment of Effectful Induction

The treatment we have given above of induction for effectful data types addresses many of our practical concerns. It is derived from the general theory of fibrational induction, it is axiomatic in terms of the functors, monads, and fibrations involved, and Theorems 6 and 7 allow us to separate the proof obligations in an induction proof into those pertaining only to the monad in question and those pertaining only to the functor in question. There is, however, one feature of Theorem 7 that is less than optimal. Underlying the fibrational methodology is the separation between logical structure in the total category of the fibration and type-theoretic structure in the base category of the fibration. Theorem 7 can thus be seen as converting logical structure in the form of \hat{F} -algebras and \hat{T} -algebras into type-theoretic structure in the form of F -and- T -Eilenberg-Moore algebras. This is, of course, completely valid, especially in light of the *propositions-as-types* interpretation, but this section shows there is a different approach which reasons solely in the total category of the fibration and is thus purely logical. The key idea is to deploy Theorem 2 in the total category of the fibration, and work directly with \hat{F} and \hat{T} -algebras on P rather than converting them to F -and- T -Eilenberg-Moore algebras on $\{P\}$. The major stumbling block to doing this is that, in general, \hat{T} is not a monad. In this section we investigate effectful induction in the case when \hat{T} is a monad. The condition we use to ensure this is that the Lawvere fibration in which we work has *very strong sums*.

Definition 3 *A Lawvere fibration $U : \mathcal{E} \rightarrow \mathcal{B}$ is said to have very strong sums if for all $f : X \rightarrow Y$ in \mathcal{B} and $P \in \mathcal{E}_X$, $\{f_P^\S\} : \{P\} \rightarrow \{\Sigma_f P\}$ is an isomorphism.*

The following important property of very strong sums is from 11: If $U : \mathcal{E} \rightarrow \mathcal{B}$ is a Lawvere fibration with very strong sums, $F : \mathcal{B} \rightarrow \mathcal{B}$ is a functor, $f : X \rightarrow Y$ is a morphism, and $P \in \mathcal{E}_X$, then $\hat{F}(\Sigma_f P) = \Sigma_{Ff} \hat{F}P$. Using this, we can prove that in a Lawvere fibration with very strong sums lifting is actually a strong monoidal functor, i.e., that lifting preserves functor composition.

Lemma 4. *Let $U : \mathcal{E} \rightarrow \mathcal{B}$ be a Lawvere fibration with very strong sums. If $F, G : \mathcal{B} \rightarrow \mathcal{B}$ are functors, then $\widehat{FG} = \hat{F}\hat{G}$. If T is a monad, then so is \hat{T} .*

Proof. For the first part of the lemma, note that $(\widehat{FG})P = \Sigma_{FG\pi_P} K_1 FG\{P\} = \Sigma_{FG\pi_P} \hat{F}K_1 G\{P\} = \hat{F}(\Sigma_{G\pi_P} K_1 G\{P\}) = \hat{F}\hat{G}P$. Here, the first equality is by the definition of \widehat{FG} , the second holds because \hat{F} is truth preserving, the third is by the aforementioned property from 11, and the last is by definition of \hat{G} . The essence of the proof of the second part is the observations that monads are just monoids in the monoidal category of endofunctors and strong monoidal functors map monoids to monoids. \square

Using Lemma 4 we can derive induction rules allowing us to work as much as possible in the total category of a Lawvere fibration with very strong sums. To do this, let (P, f, g) be an \hat{F} -and- \hat{T} -Eilenberg-Moore algebra. By Theorem 2, there is a unique morphism from the initial $\hat{T}\hat{F}$ -algebra to (P, f, g) . But since $\hat{T}\hat{F} = \widehat{TF}$

and $\mu(\widehat{TF}) = \top(\mu(TF))$ (see Corollary 4.10 of [8]), there is a morphism from $\top(T(\mu(FT)))$ to P , and thus one from $T(\mu(FT))$ to $\{P\}$ as desired.

The families and codomain fibrations both have very strong sums.

7 Conclusions, Related Work, and Future Work

We have investigated the interaction between induction and effects. We formalised the former using the recently developed fibrational interpretation of induction because it is axiomatic in the category interpreting types and programs, the functor representing the data type in question, and the category interpreting predicates. We formalised effects using monads because they are both simple to understand and widely used. We have shown, perhaps surprisingly, that several induction rules can be derived for effectful data types. These rules assume progressively more structure in their hypotheses but deliver progressively stronger inductive proofs. Ultimately, we hope this research will lay the foundation for a reasoning module for effectful data types in a proof system such as Coq.

The combination of monadic effects and inductive data types has previously been studied by Fokkinga [4] and Pardo [14]. They use distributive laws $\lambda : FT \rightarrow TF$ relating functors describing data types to monads modelling effects. Given a distributive law, it can be shown that μF is the carrier of an initial algebra in the Kleisli category of T . From this, a theory of effectful structural recursion over *pure* data is derived. By contrast, in this paper we have explored computation and reasoning with *effectful* data, where data and effects are interleaved.

Lehman and Smyth [12] give a generic induction rule for (pure) inductive data types in the case when predicates are taken to be subobjects in a category. Crole and Pitts [3] use this rule to give a fixpoint induction rule for effectful computations, generalising the usual notion of Scott induction. Filinski and Støvring's induction principle, which we have generalised in this paper, extends these rules to handle the interleaving of data and effects.

References

1. Atkey, R., Johann, P., Ghani, N.: When is a Type Refinement an Inductive Type? In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 72–87. Springer, Heidelberg (2011)
2. Burstall, R.: Proving Properties of Programs by Structural Induction. *Computer Journal* 12(1), 41–48 (1969)
3. Crole, R., Pitts, A.: New Foundations for Fixpoint Computations: FIX-Hyperdoctrines and the FIX-Logic. *Information and Computation* 98(2), 171–210 (1992)
4. Fokkinga, M.: Monadic Maps and Folds for Arbitrary Datatypes. Technical Report, University of Twente (1994)
5. Filinski, A., Støvring, K.: Inductive Reasoning About Effectful Data Types. In: Proc. International Conference on Functional Programming, pp. 97–110 (2007)

6. Gill, A., Hutton, G.: The worker/wrapper Transformation. *Journal of Functional Programming* 19(2), 227–251 (2009)
7. Ghani, N., Johann, P., Fumex, C.: Fibrational Induction Rules for Initial Algebras. In: Dawar, A., Veith, H. (eds.) *CSL 2010*. LNCS, vol. 6247, pp. 336–350. Springer, Heidelberg (2010)
8. Ghani, N., Johann, P., Fumex, C.: Generic Fibrational Induction (2011) (submitted)
9. Hermida, C., Jacobs, B.: Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation* 145, 107–152 (1998)
10. Jacobs, B.: *Categorical Logic and Type Theory*. *Studies in Logic and the Foundations of Mathematics*, vol. 141. Elsevier (1999)
11. Jacobs, B.: Comprehension Categories and the Semantics of Type Dependency. *Theoretical Computer Science* 107, 169–207 (1993)
12. Lehmann, D., Smyth, M.: Algebraic Specification of Data Types: A Synthetic Approach. *Theory of Computing Systems* 14(1), 97–139 (1981)
13. Moggi, E.: Computational Lambda-Calculus and Monads. In: *Proc. Logic in Computer Science*, pp. 14–23 (1989)
14. Pardo, A.: Combining Datatypes and Effects. In: Vene, V., Yu, H.-J. (eds.) *AFP 2004*. LNCS, vol. 3622, pp. 171–209. Springer, Heidelberg (2005)

A Coalgebraic Perspective on Minimization and Determinization*

Jiří Adámek¹, Filippo Bonchi², Mathias Hülsbusch³,
Barbara König³, Stefan Milius¹, and Alexandra Silva^{4,**}

¹ Technische Universität Braunschweig

² CNRS, ENS Lyon, Université de Lyon LIP (UMR 5668)

³ Universität Duisburg-Essen

⁴ Radboud University Nijmegen

Abstract. Coalgebra offers a unified theory of state based systems, including infinite streams, labelled transition systems and deterministic automata. In this paper, we use the coalgebraic view on systems to derive, in a uniform way, abstract procedures for checking behavioural equivalence in coalgebras, which perform (a combination of) minimization and determinization. First, we show that for coalgebras in categories equipped with *factorization structures*, there exists an abstract procedure for equivalence checking. Then, we consider coalgebras in categories without suitable factorization structures: under certain conditions, it is possible to apply the above procedure after transforming coalgebras with *reflections*. This transformation can be thought of as some kind of determinization. We will apply our theory to the following examples: conditional transition systems and (non-deterministic) automata.

1 Introduction

Finite automata are one of the most basic structures in computer science. One particularly interesting problem is that of minimization: given a (non-)deterministic finite automaton is there an equivalent one which has a minimal number of states?

Given a regular language L , minimal deterministic automata (DA) can be thought of as the canonical acceptors of the given language L . A minimal automaton is universal, in the sense that given any automaton which recognizes the same language (and where all states are reachable) there is a unique mapping into the minimal one. Similar notions exist for other kinds of transition systems such as Mealy machines or labelled transition systems. However, in many interesting cases, such as for non-deterministic automata (NDA) or for weighted automata, what it means to be a minimal system is not yet clear. Typically, for NDA one first determinizes the automaton and then minimizes it, since for DA minimization algorithms are well-known ([16]).

* The work of Mathias Hülsbusch and Barbara König was partially supported by the DFG project Behaviour-GT. The work of Alexandra Silva was partially supported by Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BPD/71956/2010.

** Also affiliated to Centrum Wiskunde & Informatica (Amsterdam, The Netherlands) and HASLab / INESC TEC, Universidade do Minho (Braga, Portugal).

It is the main aim of this paper to find a general notion of canonicity for a large class of transition systems, in a uniform manner. This encompasses two things: (i) casting the automata and the intended equivalence in a general framework; and (ii) using the general framework to devise algorithms to minimize (and determinize) the automata, yielding a canonical representative. To study all the types of automata mentioned above (and more) in a uniform setting, we use *coalgebras*.

For a functor $F: \mathbf{C} \rightarrow \mathbf{C}$, on a category \mathbf{C} , an F -coalgebra is a pair (X, α) , where X is an object of \mathbf{C} representing the “state space” of the system and $\alpha: X \rightarrow FX$ is a morphism of \mathbf{C} defining the “transitions” of the states. For instance, given an input alphabet A , DAs are coalgebras for the functor $2 \times (-)^A: \mathbf{Set} \rightarrow \mathbf{Set}$ and NDAs are coalgebras for the functor $A \times (-) + 1: \mathbf{Rel} \rightarrow \mathbf{Rel}$, where \mathbf{Set} is the category of sets and functions and \mathbf{Rel} the category of sets and relations.

The strength of the coalgebraic approach lies in the fact that many important notions, such as behavioural equivalence, are uniquely determined by the type of the system. Under mild conditions, functors F have a final coalgebra (unique up to isomorphism) into which every F -coalgebra can be mapped via a unique homomorphism. The final coalgebra can be viewed as the universe of all possible behaviours: the unique homomorphism into the final coalgebra maps every state of a coalgebra to its behaviour. This provides a general notion of behavioural equivalence: two states are equivalent iff they are mapped to the same element of the final coalgebra. In the case of DAs, the final coalgebra is $\mathcal{P}(A^*)$ (the set of all languages over input alphabet A) and the unique homomorphism is a *function* mapping each state to the language that it accepts. In the case of NDAs, as shown in [13], the final coalgebra is A^* (the set of all finite words over A) and the unique homomorphism is a *relation* linking each state with all the words that it accepts. In both cases, the induced behavioural equivalence is language equivalence. The base category chosen to model the system plays an important role in the obtained equivalence. For instance, NDAs can alternatively be modelled as coalgebras for the functor $2 \times \mathcal{P}(-)^A: \mathbf{Set} \rightarrow \mathbf{Set}$, where \mathcal{P} is the powerset functor, but then the induced behavioural equivalence is bisimilarity (which is finer than language equivalence).

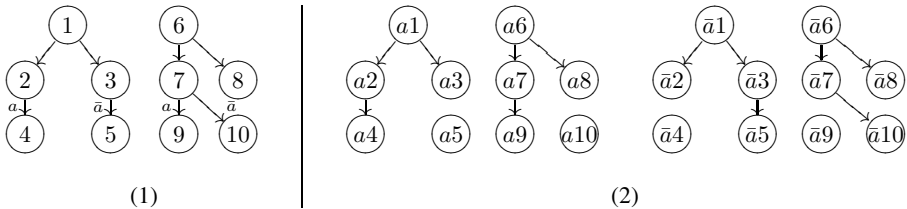
For a functor F on \mathbf{Set} , the *image* of an F -coalgebra under the unique morphism is its *minimal representative* (with respect to the induced behavioural equivalence) that, in the finite case, can be computed via ordinary partition refinement algorithms. For functors on categories not equipped with proper image factorization structures (such as \mathbf{Rel} , for instance) the situation is less clear-cut. This observation instantiates to the well-known fact that for every DA there exists an equivalent minimal automaton, while for NDAs the uniqueness of minimal automata is not guaranteed.

It is our aim to, on the one hand, offer a procedure to perform ordinary partition refinement for categories with suitable factorization structures (such as \mathbf{Set} , wherein DAs are modelled), yielding the *minimization* of a coalgebra. On the other hand, we want to offer an alternative procedure for categories without proper factorization structures: we describe a general setting for *determinizations* and show how to obtain a single algorithm that does determinization and minimization simultaneously. It is worth to note that the latter approach holds for functors for which a final coalgebra does not exist.

Our work was motivated by several examples, considering coalgebras in various underlying categories. In this paper, we take one example in \mathbf{Set} and two examples in

$\mathcal{Kl}(T)$, the Kleisli category for a monad T . More precisely, we consider DAs in **Set**, NDAs in **Rel**, which is $\mathcal{Kl}(\mathcal{P})$ where \mathcal{P} is the powerset monad, and conditional transition systems in $\mathcal{Kl}(T)$ where T is the input monad. For DAs, we recover the usual Hopcroft minimization algorithm [16]. Instantiation to NDAs gives us (a part of) Brzozowski’s algorithm [7]: the obtained automata coincide with *átomata*, that are a new kind of “canonical” NDAs recently introduced in [8].

Conditional Transition Systems (CTS). To better illustrate our work, we employ transition systems labelled with *conditions* that have similarly been studied in [14,10]. Consider the transition system (1) below where transitions are decorated with conditions a, \bar{a} , where intuitively \bar{a} stands for “not a ”. Labelled transitions are either present or absent, depending on whether a or \bar{a} hold. Unlabelled transitions are always present (they can be thought of as two transitions labelled a and \bar{a}).



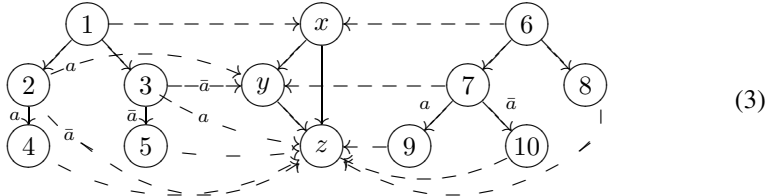
The environment can make one choice, which can not be changed later: it decides whether to take either a or \bar{a} . Regardless of the specific choice of the environment, the two states 1 and 6 in (1) above will be bisimilar. If a holds then the systems above would be instantiated to the left half of transition system (2) above. Instead if a does not hold then we obtain the right half. In both cases, the instances of the states 1 and 6 are bisimilar.

This shows that one possible way to solve the question whether two states are *always* bisimilar consists in enumerating all conditions and to create suitably many instantiations of the transition system. Then the resulting transition system can be minimized with respect to bisimilarity. This is analogous to the steps of determinization and minimization for NDAs. Indeed, the base category of coalgebras of CTSs, as **Rel** for NDAs, has no suitable factorization structures. In order to minimize (both NDAs and CTSs), coalgebras should be transformed via *reflections* that, in the case of NDAs means determinizing, while for CTS, means instantiating CTSs for all the conditions.

In this work, we will study both constructions in a general setting and also show how they can be combined into a single algorithm. For CTSs this mean that we will provide an algorithm that checks if two states are bisimilar under all the possible conditions, without performing all the possible instantiations.

Now, what would be a canonical representative of the systems above? In other words, is there a system into which CTS (1) can be mapped? In the example above, it is relatively easy to see that that system would be the transition system consisting of states x, y, z in (3) below. One would map both 1 and 6 to x , 7 to y , 4, 5, 9 and 10 to z . What about 2 and 3? We want to map 2 to y whenever a holds and to z whenever

\bar{a} holds, dually for 3. In order to do that we need to work in a category where we can represent such *conditional* maps. As we will show in the sequel, by modelling CTS as coalgebras in a Kleisli category this will be possible. The full mapping is represented below.



An extended version of this paper [2] contains all the proofs, further details and examples. In particular, there we instantiate our theory to the case of linear weighted automata [6].

2 Background Material on Coalgebras

We assume some prior knowledge of category theory (categories, functors, monads, limits and adjunctions). Definitions can be found in [3]. However, to establish some notation, we recall some basic definitions. We denote by Ord the class of all ordinals. Let Set be the category of sets and functions. Sets (and other objects) are denoted by capital letters X, Y, \dots and functions (and other morphisms) by lower case $f, g, \dots, \alpha, \beta, \dots$. We write \emptyset for the empty set, 1 for the singleton set, typically written as $1 = \{\bullet\}$, and 2 for the two elements set $2 = \{0, 1\}$. The collection of all subsets of a set X is denoted by $\mathcal{P}(X)$ and the collection of functions from a set X to a set Y is denoted by Y^X . We write $g \circ f$ for function composition, when defined. The product of two sets X, Y is written as $X \times Y$, while the coproduct, or disjoint union, as $X + Y$. These operations, defined on sets, can analogously be defined on functions, yielding (bi-)functors. A category \mathbf{C} is called *concrete* if a faithful functor $U: \mathbf{C} \rightarrow \text{Set}$ is given.

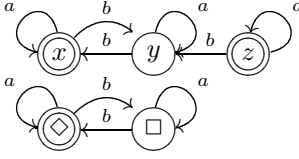
Definition 2.1 (Coalgebra). Given an endofunctor $F: \mathbf{C} \rightarrow \mathbf{C}$ an (F) -coalgebra is a pair (X, α) , where X is an object of \mathbf{C} and $\alpha: X \rightarrow FX$ a morphism in \mathbf{C} . A (coalgebra) homomorphism $f: (X, \alpha) \rightarrow (Y, \beta)$ between two coalgebras $\alpha: X \rightarrow FX$ and $\beta: Y \rightarrow FY$ is a \mathbf{C} -morphism $f: X \rightarrow Y$ such that $Ff \circ \alpha = \beta \circ f$.

An F -coalgebra (Ω, ω) is *final* if for any F -coalgebra (X, α) there exists a unique homomorphism $\text{beh}_X: (X, \alpha) \rightarrow (\Omega, \omega)$. If \mathbf{C} is concrete we can define behavioural equivalence. Given an F -coalgebra (X, α) and $x, y \in UX$, we say that x and y are *behaviourally equivalent*, written $x \approx y$, if and only if there exist an F -coalgebra (Z, γ) and a homomorphism $f: (X, \alpha) \rightarrow (Z, \gamma)$ such that $Uf(x) = Uf(y)$. If a final F -coalgebra exists, we have a simpler characterization of behavioural equivalence: $x \approx y$ iff $U\text{beh}_X(x) = U\text{beh}_X(y)$.

Example 2.2. (DA) A deterministic automaton over the alphabet A is a pair (X, α) , where X is a set of states and $\alpha: X \rightarrow 2 \times X^A$ is a function that to each state x associates a pair $\alpha(x) = \langle o_x, t_x \rangle$, where o_x , the output value, determines if a state

x is final ($o_x = 1$) or not ($o_x = 0$); and t_x , the transition function, returns for each $a \in A$ the next state. DAs are coalgebras for the functor $FX = 2 \times X^A$ on **Set**. The final coalgebra for this functor is $(\mathcal{P}(A^*), \omega)$ where $\mathcal{P}(A^*)$ is the set of languages over A and, for a language L , $\omega(L) = \langle \varepsilon_L, L_a \rangle$, where ε_L determines whether or not the empty word is in the language ($\varepsilon_L = 1$ or $\varepsilon_L = 0$, resp.) and, for each input letter a , L_a is the *derivative* of L : $L_a = \{w \in A^* \mid aw \in L\}$. From any DA (X, α) , there is a unique homomorphism beh_X into $\mathcal{P}(A^*)$ which assigns to each state its behaviour (that is, the language that the state recognizes). Two states are behaviourally equivalent iff they accept the same language.

Take $A = \{a, b\}$ and consider the DAs on the right. We call the topmost (X, α) where $X = \{x, y, z\}$ and $\alpha: X \rightarrow 2 \times X^A$ maps x to the pair $\langle 1, \{a \mapsto x, b \mapsto y\} \rangle$, y to $\langle 0, \{a \mapsto y, b \mapsto x\} \rangle$ and z to $\langle 1, \{a \mapsto z, b \mapsto y\} \rangle$. The bottom one is (Z, γ) where $Z = \{\diamond, \square\}$ and $\gamma: Z \rightarrow 2 \times Z^A$ maps \diamond to $\langle 1, \{a \mapsto \diamond, b \mapsto \square\} \rangle$ and \square to $\langle 0, \{a \mapsto \square, b \mapsto \diamond\} \rangle$.



As an example of a coalgebra homomorphism, take the function $e: X \rightarrow Z$ mapping x, z to \diamond and y to \square . □

Non-deterministic automata (NDA) can be described as coalgebras for the functor $2 \times \mathcal{P}(-)^A$ (on **Set**): to each input in A , we assign a set of possible successors states. Unfortunately, the resulting behavioural equivalence is not language equivalence (as for DAs), but bisimilarity (i.e., it only identifies states having the same branching structure). In [2113], it is shown that in order to retrieve language equivalence for NDAs, one should consider coalgebras in a Kleisli category. In what follows, we introduce Kleisli categories, in which we model NDAs and CTSS as coalgebras. While objects in a Kleisli category are sets, morphisms are generalized functions that incorporate side effects, such as non-determinism, specified by a monad (see [31313]).

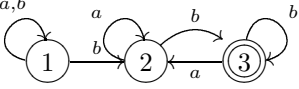
Definition 2.3 (Kleisli Category). Let $(T: \mathbf{Set} \rightarrow \mathbf{Set}, \eta, \mu)$ (or simply T) be a monad on **Set**. Its Kleisli category $\mathcal{Kl}(T)$ has sets as objects and a morphism $X \rightarrow Y$ in $\mathcal{Kl}(T)$ is a function $X \rightarrow TY$. The identity id_X is η_X and the composition $g \circ f$ of $f: X \rightarrow Y, g: Y \rightarrow Z$ (i.e., functions $f: X \rightarrow TY, g: Y \rightarrow TZ$) is $\mu_Z \circ Tg \circ f$.

In the following we will employ overloading and use the same letter to both denote a morphism in $\mathcal{Kl}(T)$ and the corresponding function in **Set**. Furthermore, note that **Set** can be seen as a (non-full) subcategory of $\mathcal{Kl}(T)$, where each function $f: X \rightarrow Y$ is identified with $\eta_Y \circ f$. Every Kleisli category $\mathcal{Kl}(T)$ is a concrete category where $UX = TX$ and $Uf = \mu_Y \circ Tf$ for an object X and a morphism $f: X \rightarrow Y$.

To define coalgebras over Kleisli categories we need the notion of lifting of a functor, which we define here directly, but could otherwise be specified via a distributive law (for details see [1319]): a functor $\overline{F}: \mathcal{Kl}(T) \rightarrow \mathcal{Kl}(T)$ is called a *lifting of $F: \mathbf{Set} \rightarrow \mathbf{Set}$* whenever it coincides with F on **Set**, seen as a subcategory of $\mathcal{Kl}(T)$.

Since F and \overline{F} coincide on objects, \overline{F} -coalgebras in $\mathcal{Kl}(T)$ are of the form $X \rightarrow TFX$, where intuitively the functor F describes the explicit branching, i.e. choices which are visible to the observer, and the monad T the implicit branching, i.e. side-effects, which are there but cannot be observed directly. In this way, the implicit branching is part of the underlying category and is also present in the morphism from any

coalgebra into the final coalgebra. As in functional programming languages such as Haskell, the idea is to “hide” computational effects underneath a monad and to separate them from the (functional) behaviour as much as possible.

Example 2.4. (NDA) Consider the powerset monad $TX = \mathcal{P}(X)$. The Kleisli category $\mathcal{Kl}(\mathcal{P})$ coincides with the category **Rel** of sets and relations. As an example of a lifting, take $FX = A \times X + 1$ in **Set** (with $1 = \{\bullet\}$). The functor F lifts to \overline{F} in **Rel** as follows: for any $f: X \rightarrow Y$ in **Rel** (that is $f: X \rightarrow \mathcal{P}(Y)$ in **Set**), $\overline{F}f: A \times X + 1 \rightarrow A \times Y + 1$ is defined as $\overline{F}f(\bullet) = \{\bullet\}$ and $\overline{F}f(\langle a, x \rangle) = \{\langle a, y \rangle \mid y \in f(x)\}$. Non-deterministic automata over the input alphabet A can be regarded as coalgebras in **Rel** for the functor \overline{F} . A coalgebra $\alpha: X \rightarrow \overline{F}X$ is a function $\alpha: X \rightarrow \mathcal{P}(A \times X + 1)$, which assigns to each state $x \in X$ a set which contains \bullet if x is final and $\langle a, y \rangle$ for all transitions $x \xrightarrow{a} y$. For instance,  the automaton on the right is the coalgebra (X, α) , where $X = \{1, 2, 3\}$ and $\alpha: X \rightarrow \mathcal{P}(\{a, b\} \times X + \{\bullet\})$ is defined as follows: $\alpha(1) = \{\langle a, 1 \rangle, \langle b, 1 \rangle, \langle b, 2 \rangle\}$, $\alpha(2) = \{\langle a, 2 \rangle, \langle b, 3 \rangle\}$ and $\alpha(3) = \{\bullet, \langle a, 2 \rangle, \langle b, 3 \rangle\}$. In [13], it is shown that the final \overline{F} -coalgebra (in **Rel**) is the set A^* of words. For an NDA (X, α) , the unique coalgebra homomorphism beh_X into A^* is the relation that links every state in X with all the words in A^* that it accepts.

Example 2.5. (CTS) We shortly discuss how to specify the example from the introduction in a Kleisli category. All the details can be found in [2].

We use the input monad $TX = X^A$, where A is a set of conditions or inputs (for the example of the introduction $A = \{a, \bar{a}\}$). Given a function $f: X \rightarrow Y$, $Tf: TX \rightarrow TY$ is $f^A: X^A \rightarrow Y^A$ defined for all $g \in X^A$ and $a \in A$ as $f^A(g)(a) = f(g(a))$.

Note that a morphism $f: X \rightarrow Y$ in the Kleisli category over the input monad is a function $f: X \rightarrow Y^A$. For instance, the dashed arrows in the introduction describe a morphism in $\mathcal{Kl}(T)$: state 2 is mapped to y if condition a holds and to z if \bar{a} holds.

We will use the countable powerset functor $FX = \mathcal{P}_c(X)$ as endofunctor, which is lifted to $\mathcal{Kl}(T)$ as follows: a morphism $f: X \rightarrow Y$ in $\mathcal{Kl}(T)$, which is a function of the form $f: X \rightarrow Y^A$, is mapped to $\overline{F}f: \mathcal{P}_c(X) \rightarrow \mathcal{P}_c(Y)$ with $\overline{F}f(X')(a) = \{f(x)(a) \mid x \in X'\}$ for $X' \subseteq X$, $a \in A$. Hence, CTS (I) from the introduction is modelled by a morphism $\alpha: X \rightarrow \mathcal{P}_c(X)$ in $\mathcal{Kl}(T)$ (i.e., a function $\alpha: X \rightarrow \mathcal{P}_c(X)^A$), where $X = \{1, \dots, 10\}$ and $A = \{a, \bar{a}\}$. For instance $\alpha(1)(a) = \alpha(1)(\bar{a}) = \{2, 3\}$, $\alpha(2)(a) = \{4\}$, $\alpha(2)(\bar{a}) = \emptyset$. The entire coalgebra α is represented by the matrix on the right.

α	1	2	3	4	5	6	7	8	9	10
a	{2, 3}	{4}	\emptyset	\emptyset	\emptyset	{7, 8}	{9}	\emptyset	\emptyset	\emptyset
\bar{a}	{2, 3}	\emptyset	{5}	\emptyset	\emptyset	{7, 8}	{10}	\emptyset	\emptyset	\emptyset

Note that the above $\alpha: X \rightarrow \mathcal{P}_c(X)^A$ can be seen as a coalgebra for the functor $FX = \mathcal{P}_c(X)^A$ in **Set**, which yields ordinary A -labelled transition systems. However, the resulting behavioural equivalence (that is, ordinary bisimilarity) would be inadequate for our intuition, since it would distinguish the states 1 and 6. In [2], we prove that behavioural equivalence of \overline{F} -coalgebras coincides with the expected one.

3 Minimization via $(\mathcal{E}, \mathcal{M})$ -Factorizations

We now introduce the notion of minimization of a coalgebra and its iterative construction that generalizes the minimization of transition systems via partition refinement.

This notion is parametrized by two classes \mathcal{E} and \mathcal{M} of morphisms that form a factorization structure for the considered category \mathbf{C} .

Definition 3.1 (Factorization Structures). Let \mathbf{C} be a category and let \mathcal{E}, \mathcal{M} be classes of morphisms in \mathbf{C} . The pair $(\mathcal{E}, \mathcal{M})$ is called a factorization structure for \mathbf{C} whenever

- \mathcal{E} and \mathcal{M} are closed under composition with isos.
- \mathbf{C} has $(\mathcal{E}, \mathcal{M})$ -factorizations of morphisms, i.e., each morphism f of \mathbf{C} has a factorization $f = m \circ e$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$.

- \mathbf{C} has the unique $(\mathcal{E}, \mathcal{M})$ -diagonalization property: for each commutative square as shown on the left-hand side with $e \in \mathcal{E}$ and $m \in \mathcal{M}$ there exists a unique diagonal, i.e., a morphism d such that the diagram on the right-hand side commutes (i.e., $d \circ e = f$ and $m \circ d = g$). If all morphisms in \mathcal{E} are epis we call $(\mathcal{E}, \mathcal{M})$ a right factorization structure.

$$\begin{array}{ccc}
 A & \xrightarrow{e} & B \\
 f \downarrow & & \downarrow g \\
 C & \xrightarrow{m} & D
 \end{array}
 \qquad
 \begin{array}{ccc}
 A & \xrightarrow{e} & B \\
 f \downarrow & \searrow^{d} & \downarrow g \\
 C & \xrightarrow{m} & D
 \end{array}$$

In any category with an $(\mathcal{E}, \mathcal{M})$ -factorization structure, the classes \mathcal{E}, \mathcal{M} are closed under composition and factorizations of morphisms are unique up to iso (see [3]). For **Set** we always consider below the factorization structure $(\mathcal{E}, \mathcal{M})$ with \mathcal{E} = epimorphisms (surjections) and \mathcal{M} = monomorphisms (injections); for the category **Set**^{op} we take the corresponding structure $(\mathcal{M}, \mathcal{E})$, i.e., where the epic part consists of functions that in **Set** are monomorphisms, analogously with \mathcal{E} . Morphisms from \mathcal{E} are drawn using double-headed arrows $A \twoheadrightarrow B$, whereas morphisms from \mathcal{M} are depicted using arrows of the form $A \rightarrow B$. Whenever the endofunctor F preserves \mathcal{M} -morphisms, which we assume in the following, the factorization structure can be straightforwardly lifted to coalgebra homomorphisms (see [17]).

Assumption 3.2. We assume that \mathbf{C} is a complete category with a right $(\mathcal{E}, \mathcal{M})$ -factorization structure and \mathbf{C} is \mathcal{E} -cowellpowered, i.e., every object X only has a set of \mathcal{E} -quotients (i.e., \mathcal{E} -morphisms with domain X up to isomorphism of the codomains). We also assume that $F: \mathbf{C} \rightarrow \mathbf{C}$ is a functor preserving \mathcal{M} , i.e., if $m \in \mathcal{M}$ then $Fm \in \mathcal{M}$.

Definition 3.3 (Minimization). The minimization of a coalgebra $\alpha: X \rightarrow FX$ is the greatest \mathcal{E} -quotient coalgebra. More precisely, the minimization is a coalgebra (Z, γ) with a homomorphism $e: (X, \alpha) \rightarrow (Z, \gamma)$ with $e \in \mathcal{E}$ such that for any other coalgebra homomorphism $e': (X, \alpha) \rightarrow (Y, \beta)$ with $e' \in \mathcal{E}$ there exists a (necessarily) unique coalgebra homomorphism $h: (Y, \beta) \rightarrow (Z, \gamma)$ such that $e = h \circ e'$.

$$\begin{array}{ccccc}
 X & & \xrightarrow{e} & & Z \\
 & \searrow^{e'} & & \searrow^{h} & \\
 & & Y & & \\
 \alpha \downarrow & & \downarrow \beta & & \downarrow \gamma \\
 FX & \xrightarrow{Fe'} & FY & \xrightarrow{Fh} & FZ \\
 & \searrow^{Fe} & & \searrow^{Fh} & \\
 & & FZ & &
 \end{array}$$

Remark 3.4. (1) Since \mathbf{C} is \mathcal{E} -cowellpowered and \mathcal{E} consists of epimorphisms, the \mathcal{E} -quotient coalgebras of a coalgebra (X, α) form a pre-ordered set: a quotient coalgebra $e': (X, \alpha) \rightarrow (Y', \beta')$ is larger than $e: (X, \alpha) \rightarrow (Y, \beta)$ iff there exists a coalgebra

homomorphism $h: (Y, \beta) \rightarrow (Y', \beta')$ with $e' = h \circ e$; notice that h is uniquely determined and $h \in \mathcal{E}$ by the properties of factorization systems. Thus, the minimization is simply the greatest element in the pre-order of \mathcal{E} -quotient coalgebras of (X, α) .

(2) While in **Set** the minimization is also determined by the strict minimality of the number of states, this is not necessarily true for other categories (see Example 4.10).

(3) We often speak about (Z, γ) (without explicitly referring to the morphism e) or even just the object Z as the minimization of the given coalgebra.

Theorem 3.8 will show that under Assumption 3.2 the minimization always exists, even when there is no final coalgebra. When the final coalgebra exists, minimization is the quotient of the unique morphism.

Proposition 3.5 (Minimization and Final Coalgebra). *If the final coalgebra $\omega: \Omega \rightarrow F\Omega$ exists, then – for a given coalgebra $\alpha: X \rightarrow FX$ – the minimization $\gamma: Z \rightarrow FZ$ can be obtained by factoring the unique coalgebra homomorphism $beh_X: (X, \alpha) \rightarrow (\Omega, \omega)$ into an \mathcal{E} -morphism and an \mathcal{M} -morphism.*

$$\begin{array}{ccccc}
 X & \xrightarrow{e} & Z & \xrightarrow{m} & \Omega \\
 \alpha \downarrow & & \downarrow \gamma & & \downarrow \omega \\
 FX & \xrightarrow{Fe} & FZ & \xrightarrow{Fm} & F\Omega \\
 & \dashrightarrow & & \dashrightarrow & \\
 & & Fbeh_X & &
 \end{array}$$

Note that whenever the concretization functor $U: \mathbf{C} \rightarrow \mathbf{Set}$ maps \mathcal{M} -morphisms to injections, $x, y \in UX$ are behaviourally equivalent ($x \approx y$) iff $Ue(x) = Ue(y)$.

Example 3.6 (DA, Minimal Automata). Recall that DAs are coalgebras for the functor $FX = 2 \times X^A$ on **Set** (Example 2.2). In this case, minimization corresponds to the well known minimization of deterministic automata. For instance, the minimization of the top automaton (X, α) in Example 2.2 yields the automaton (Z, γ) (on the bottom).

We now describe a construction that – given a coalgebra (X, α) – obtains the minimization γ without going via the final coalgebra. This closely resembles the partition refinement algorithm for minimizing deterministic automata or for computing bisimilarity. Whenever the construction below becomes stationary, we obtain the minimization. In many examples the constructed sequence might even become stationary after finitely many steps. The construction is reminiscent of the construction (in the dual setting) of the initial algebra by Adámek [1], for the coalgebraic version see Worrel [24] and Adámek and Koubek [4]. As in those papers, our construction works for ordinals beyond ω . Hereafter 1 denotes the final object of \mathbf{C} .

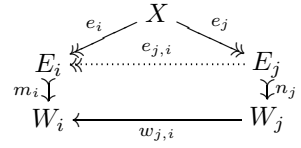
Construction 3.7 *Recall the final chain $W: \text{Ord} \rightarrow \mathbf{C}$ given by*

$$W_0 = 1, \quad W_{i+1} = FW_i, \quad W_j = \lim_{i < j} W_i \quad (j \text{ a limit ordinal.})$$

This is the unique chain, up to natural isomorphism, whose connecting morphisms $w_{i,j}$ fulfil (a) $w_{i+1,j+1} = Fw_{i,j}$ and (b) for limit ordinals they form a limit cone.

As we do not assume that F has a final coalgebra, the chain W need not converge. Every coalgebra $\alpha: X \rightarrow FX$ defines a unique canonical cone $(\alpha_i: X \rightarrow$

$W_i)_{i \in \text{Ord}}$ on W with the property that $\alpha_{i+1} = F\alpha_i \circ \alpha: X \rightarrow FW_i = W_{i+1}$. Let $e_i: X \twoheadrightarrow E_i$, $m_i: E_i \twoheadrightarrow W_i$ be an $(\mathcal{E}, \mathcal{M})$ -factorization of α_i . Then, we obtain an ordinal indexed chain (E_i) of quotients of X with the connecting morphisms $e_{j,i}$ obtained by diagonalization for $i < j$, as depicted on the right.



Theorem 3.8. For every F -coalgebra (X, α) , its minimization is E_i , for some $i \in \text{Ord}$.

More precisely, there exists an ordinal i such that E_i carries a coalgebra structure $\varepsilon: E_i \rightarrow FE_i$ such that $e_i: (X, \alpha) \rightarrow (E_i, \varepsilon)$ is the minimization; for details see the proof of Theorem 3.8 in the extended version of this paper [2].

By the above theorem, minimizations always exist even when there is no final coalgebra. Worrell [24] shows that for a finitary functor $F: \mathbf{Set} \rightarrow \mathbf{Set}$, the final chain W_i converges at the final coalgebra in $\omega + \omega$ iterations. The chain E_i , instead, converges at the minimization in ω iterations.

Theorem 3.9. Let $F: \mathbf{Set} \rightarrow \mathbf{Set}$ be a finitary functor. Then for every F -coalgebra (X, α) , its minimization is E_ω .

In our examples, we will use the following construction which is closer to the standard minimization algorithm and to any reasonable implementation of Construction 3.7.

Theorem 3.10. The chain $(E_i)_{i \in \text{Ord}}$ of Construction 3.7 can also be defined as follows:

- (a) Factor the unique morphism $d_0: X \rightarrow 1$ into $e_0: X \twoheadrightarrow E_0$ and $n_0: E_0 \twoheadrightarrow 1$.
- (b) Given $e_i: X \twoheadrightarrow E_i$, factor $d_{i+1} = Fe_i \circ \alpha$ into $e_{i+1}: X \twoheadrightarrow E_{i+1}$ and $n_{i+1}: E_{i+1} \twoheadrightarrow FE_i$.
- (c) For a limit ordinal j , form a limit of the preceding chain $(E_i)_{i < j}$, obtaining \hat{E}_j and $\hat{e}_j: X \twoheadrightarrow \hat{E}_j$ as mediating morphism. Factor \hat{e}_j into $e_j: X \twoheadrightarrow E_j$ and $n_j: E_j \twoheadrightarrow \hat{E}_j$.

By instantiating the above construction to the case of DAs, we obtain the standard minimization algorithm by Hopcroft [16].

4 Determinization via Reflections

For several categories there are no suitable factorization structures. This can for instance be observed in **Rel**, wherein we model non-deterministic automata as coalgebras. It is known that minimization of non-deterministic automata is not unique. The usual procedure is to first construct the corresponding deterministic automaton (via the powerset construction), which is then minimized in a second step. In this section, we will give a general framework for determinization-like constructions in the form of reflections, which can also be applied to other settings, such as conditional transition systems. For non-deterministic automata we will obtain an automaton which is “backward-deterministic”, i.e., for every state and each letter there is exactly one predecessor. Then we will show how reflections can be combined with the minimization.

Definition 4.1 (Reflective Subcategory). Let \mathbf{S} be a subcategory of \mathbf{C} . Let X be an object of \mathbf{C} . An \mathbf{S} -reflection for X is a morphism $\eta_X : X \rightarrow X'$, where X' is an \mathbf{S} -object, such that for every other morphism $f : X \rightarrow Y$ with Y in \mathbf{S} there exists a unique \mathbf{S} -morphism $f' : X' \rightarrow Y$ such that $f = f' \circ \eta_X$. \mathbf{S} is called a reflective subcategory of \mathbf{C} whenever each \mathbf{C} -object has an \mathbf{S} -reflection.

This definition is equivalent to saying that the functor embedding \mathbf{S} into \mathbf{C} has a left adjoint $L : \mathbf{C} \rightarrow \mathbf{S}$ called *reflector*. The morphisms η_X form the unit of this adjunction. In our examples in $\mathcal{K}\ell(T)$, the unit η of the reflection will *not* coincide with the natural transformation η of the monad T . It is well-known that for a monad $T : \mathbf{Set} \rightarrow \mathbf{Set}$ the category \mathbf{Set} is coreflective in $\mathcal{K}\ell(T)$, whereas here we need a reflective subcategory.

Example 4.2. (NDA) The category \mathbf{Set}^{op} is a reflective subcategory of \mathbf{Rel} . The reflector L is the contravariant powerset functor, i.e., for a relation $R : X \rightarrow Y$ we have $L(R) : \mathcal{P}(X) \rightarrow \mathcal{P}(Y)$ in \mathbf{Set}^{op} where $L(R)$ maps $Y' \subseteq Y$ to $R^{-1}(Y')$. The reflection $\eta_X : X \rightarrow \mathcal{P}(X)$ relates an element $x \in X$ with $X' \subseteq X$ if and only if $x \in X'$.

(CTS) For $\mathcal{K}\ell(T)$ where T is the input monad, we have the following situation: since every function $f : X \rightarrow Y^A$ corresponds to a function $f' : A \times X \rightarrow Y$ by currying, the category $\mathcal{K}\ell(T)$ is isomorphic to the co-Kleisli category over the comonad $VX = A \times X$ on \mathbf{Set} . Hence, \mathbf{Set} is both reflective and coreflective in $\mathcal{K}\ell(T)$. The reflection is the Kleisli morphism $\eta_X : X \rightarrow A \times X$ with $\eta_X(x)(a) = \langle a, x \rangle$. The reflector L coincides with V on objects and takes the product of the state set X with the label set A . More concretely, for a morphism $f : X \rightarrow Y$ in $\mathcal{K}\ell(T)$ we obtain a morphism $Lf : A \times X \rightarrow A \times Y$ in \mathbf{Set} with $Lf(\langle a, x \rangle) = \langle a, f(x)(a) \rangle$.

Definition 4.3 (Reflection of Coalgebras). Let \mathbf{S} be a reflective subcategory of a category \mathbf{C} and let $L : \mathbf{C} \rightarrow \mathbf{S}$ be the reflector. Assume that \mathbf{S} is preserved by the endofunctor F . Then, given a coalgebra $\alpha : X \rightarrow FX$ in \mathbf{C} we reflect it into \mathbf{S} , obtaining a coalgebra $\alpha' : LX \rightarrow FLX$ by the following construction:

$$\begin{array}{ccccc}
 X & \xrightarrow{\alpha} & FX & & \\
 \eta_X \downarrow & & \eta_{FX} \downarrow & \searrow F\eta_X & \\
 LX & \xrightarrow{L\alpha} & LFX & \cdots \cdots \rightarrow & FLX \\
 & \searrow \alpha' & & &
 \end{array}$$

Note that the existence of a unique morphism ζ_X is guaranteed by Definition 4.3 since F preserves \mathbf{S} and hence FLX is an object of \mathbf{S} .

Whenever \mathbf{C} is a concrete category (with concretization functor U) and $x, y \in UX$ it holds that $x \approx y$ iff $U\eta_X(x) \approx U\eta_Y(y)$. Hence two states in UX are behaviourally equivalent if and only if this holds for their images in the reflected coalgebra.

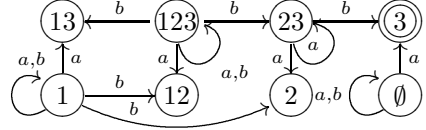
That the above construction indeed gives a reflection of coalgebras for F is a special instance of a known result (see for instance Hermida and Jacobs [15], Corollary 2.15).

Proposition 4.4. Let \mathbf{S} be a reflective subcategory of \mathbf{C} , which is preserved by the endofunctor F . The category of F -coalgebras in \mathbf{S} is a reflective subcategory of the category of F -coalgebras in \mathbf{C} .

A limit in a reflective subcategory \mathbf{S} is also a limit in \mathbf{C} . Hence, if the final coalgebra exists in the subcategory \mathbf{S} , it is also the final coalgebra in \mathbf{C} . In particular, whenever \mathbf{S} is complete, the chain (W_i) (Construction 3.7) in \mathbf{S} will coincide with the chain in \mathbf{C} .

Example 4.5. (NDA) We will first study the effect of a reflection on a non-deterministic automaton, for which we use the reflective subcategory \mathbf{Set}^{op} of \mathbf{Rel} (see Example 4.2). The effect of the reflection on coalgebras is a powerset automaton which is however “backwards-deterministic”: more specifically, given a coalgebra $\alpha: X \rightarrow A \times X + 1$ in \mathbf{Rel} , the reflected coalgebra $\alpha': \mathcal{P}(X) \rightarrow A \times \mathcal{P}(X) + 1$ is a relation which lives in \mathbf{Set}^{op} and, when seen as a function, maps $\langle a, X' \rangle$ with $X' \subseteq X$ to

$\{x \in X \mid \exists x' \in X': \langle a, x' \rangle \in \alpha(x)\}$ (the set of a -predecessors of X') and \bullet to $\{x \in X \mid \bullet \in \alpha(x)\}$ (the set of final states, the unique final state of the new automaton). For

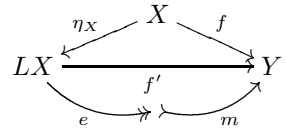


instance, the reflection of the NDA (X, α) in Example 2.4 is the above backwards-deterministic automaton. Note that the above automaton has a single final state (consisting of the set of final states of the original automaton) and every state has a unique predecessor for each alphabet letter. Hence, it can be seen as a function $\alpha': A \times Y + 1 \rightarrow Y$ (i.e., an algebra for the functor $FY = A \times Y + 1$). Note that \mathbf{Set} is not a reflective subcategory of \mathbf{Rel} – it is instead coreflective – and hence both categories have different final coalgebras. However for the reflective subcategory \mathbf{Set}^{op} , we have exactly the same final coalgebra as for \mathbf{Rel} , which, as shown in [13], is the initial algebra in \mathbf{Set} .

(CTS) Now we come back to the Kleisli category $\mathcal{Kl}(T)$ over the input monad T (see Example 2.5) and coalgebras with endofunctor \mathcal{P}_c . As discussed in Example 4.2, \mathbf{Set} is a reflective subcategory of $\mathcal{Kl}(T)$. On coalgebras reflection has the following effect: given a coalgebra $\alpha: X \rightarrow \mathcal{P}_c(X)$ in $\mathcal{Kl}(T)$ we obtain a reflected coalgebra $\alpha': A \times X \rightarrow \mathcal{P}_c(A \times X)$ in \mathbf{Set} with $\alpha'(\langle a, x \rangle) = \{\langle a, x' \rangle \mid x' \in \alpha(x)(a)\}$. That is, we generate the disjoint union of $|A|$ different transition systems, each of which describes the behaviour for some $a \in A$. For instance, the reflection of CTS (1) (formally introduced in Example 2.5; see also the introduction) is CTS (2) from the introduction.

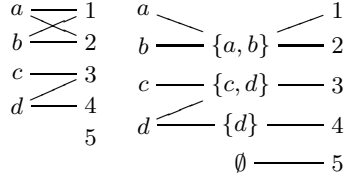
We now consider other forms of factorizations that do not conform to Definition 3.1

Definition 4.6 (Pseudo-Factorization). Let \mathbf{C} be a category and let \mathbf{S} be a reflective subcategory with a factorization structure $(\mathcal{E}, \mathcal{M})$. Let $f: X \rightarrow Y$ be a morphism of \mathbf{C} where Y is an object of \mathbf{S} . Take the unique morphism $f': LX \rightarrow Y$ with $f' \circ \eta_X = f$ (which exists due to the reflection) and factor $f' = m \circ e$ with $m \in \mathcal{M}, e \in \mathcal{E}$. Then the decomposition $f = m \circ c$ with $c = e \circ \eta_X$ is called the $(\mathcal{E}, \mathcal{M})$ -pseudo-factorization of f .



Example 4.7. (NDA) Consider \mathbf{Set}^{op} as the reflective subcategory of \mathbf{Rel} (Example 4.2). Given a relation $R: X \rightarrow Y$, let $Z = \{R^{-1}(y) \mid y \in Y\} \subseteq \mathcal{P}(X)$ be the

set of pre-images of elements of Y under R . Now define relations $R_c: X \rightarrow \mathcal{Z}$ with $R_c(x) = \{Z \in \mathcal{Z} \mid x \in Z\}$ and $R_m: \mathcal{Z} \rightarrow Y$ with $R_m(Z) = \{y \in Y \mid Z = R^{-1}(y)\}$. Note that $R_m \circ R_c = R$. As an example consider the relation R between sets $X = \{a, b, c, d\}$ and $Y = \{1, 2, 3, 4, 5\}$ visualized on the left (where $R(a) = R(b) = \{1, 2\}$, $R(c) = \{3\}$, $R(d) = \{3, 4\}$). Its pseudo-factorization into R_c and R_m is shown on the right. Here R_m maps elements of Y to their preimage under R in $\mathcal{P}(X)$.



(CTS) For **Set**, the reflective subcategory of $\mathcal{Kl}(T)$, where T is the input monad, we use the classical factorization structure with surjective and injective functions. Given a morphism $f: X \rightarrow Y$ in $\mathcal{Kl}(T)$, seen as a function $f: X \rightarrow Y^A$, we define $Y' = \{y \in Y \mid \exists x \in X, a \in A: f(x)(a) = y\}$. Then $f_c: X \rightarrow Y'^A$ with $f_c(x)(a) = f(x)(a)$ and $f_m: Y' \rightarrow Y^A$ with $f_m(y)(a) = y$ for all $a \in A$, i.e., f_m is simply an injection without side-effects. Note that $f_m \circ f_c = f$ in $\mathcal{Kl}(T)$.

Note that pseudo-factorizations enjoy the diagonalization property as in Definition 3.1 whenever g is a morphism of **S**. However pseudo-factors are not necessarily closed under composition with the isos of **C**.

Assumption 4.8. We assume that **S** is a reflective subcategory of **C**. We also assume that an endofunctor F of **C** is given preserving **S**. And **S** and F fulfil Assumption 3.2

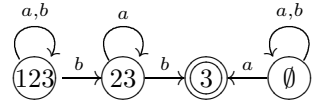
Theorem 4.9. Given a coalgebra $\alpha: X \rightarrow FX$ in **C**, the following four constructions obtain the same result (we also call this result the minimization):

- (i) Apply Construction 3.7 using the $(\mathcal{E}, \mathcal{M})$ -pseudo-factorizations of Definition 4.6
- (ii) Reflect α into the subcategory **S** according to Definition 4.3 and then apply Construction 3.7 using $(\mathcal{E}, \mathcal{M})$ -factorizations.
- (iii) Apply the construction of Theorem 3.10 using $(\mathcal{E}, \mathcal{M})$ -pseudo-factorizations.
- (iv) Reflect α into the subcategory **S** and then apply the construction of Theorem 3.10 using $(\mathcal{E}, \mathcal{M})$ -factorizations.

Note that we do not have to require here that **C** is complete. As it is clear from the proof of Theorem 4.9 (see the extended version of this paper [2]) Construction 3.7 and the construction in Theorem 3.10 can be straightforwardly adapted to pseudo-factorizations instead of factorizations: The quotients E_i and the chain $e_{j,i}$ of connecting morphisms obtained in variants (i)–(iv) are identical and live in the subcategory **S**. Since **S** is reflective in **C** we obtain the same results when taking the limit in **C** or in **S**, respectively.

Variant (iii) allows to tightly integrate minimization with a determinization-like construction, i.e., to do both simultaneously instead of sequentially. For practical purposes it is usually the most efficient solution, since it avoids building the final chain of Construction 3.7 and the reflected coalgebra of Definition 4.3 which both usually involve significant combinatorial explosion.

Example 4.10. (NDA) Theorem 4.9 suggests two ways to build the minimization of an NDA (and thus checking the equivalence of its states). We first apply Construction (iv) to the NDA (X, α) in Example 2.4 and then we illustrate Construction (iii). Recall that the reflection of (X, α) into \mathbf{Set}^{op} is $(\mathcal{P}(X), \alpha')$ in Example 4.5. By applying Construction 3.7 (with the factorization structure of \mathbf{Set}^{op}), we remove from $(\mathcal{P}(X), \alpha')$ the states that are not related to any word in the final coalgebra or, in other words, those states from which there is no path to the final state. Intuitively, we perform a backwards breadth-first search and the factorizations make sure that unreachable states are discarded. The resulting automaton is illustrated above.



Construction (iii) can be understood as an efficient implementation of Construction (iv): we do not build the entire $(\mathcal{P}(X), \alpha')$, but we construct directly the above automaton by iteratively adding states and transitions. We start with state 3, then we add 23 and \emptyset and finally we add 123. All the details are shown in [2].

The minimized NDA can be thought of as a *canonical* representative of its equivalence class. The quest for canonical NDAs (also referred to as “universal”) started in the seventies and, recently, an interesting kind of canonical NDAs (called *átomata*) has been proposed in [8]. In [2], we show that our minimized NDAs coincide with *átomata* of [8]. This provides a universal property that uniquely characterizes *átomata* (up to isomorphism), namely the *átomaton* of a regular language is the minimization of any NDA accepting the language.

It is worth noting that the automaton obtained above is precisely the automaton in the third step of the well-known Brzozowski algorithm for minimization of non-deterministic automata [7], which, in a nutshell, works as follows: 1) given an NDA reverse it, by reversing all arrows and exchanging final and initial states; 2) determinize it, using the subset construction, and remove unreachable states; 3) reverse it again; 4) determinize it, using the subset construction, and remove unreachable states. In our example, we are doing steps 1)–3) but without the explicit reversal. Our automata do not have initial states, but steps 1)–3) are independent on the specific choice of initial states, because of the two reversals.

Example 4.11. (CTS) Recall the coalgebraic description of CTS given in Example 2.5: the base category is $\mathcal{Kl}(T)$, where T is the input monad and $\overline{F} = \mathcal{P}_c$ is the countable powerset functor. CTS (1) of the introduction is the coalgebra $\alpha: X \rightarrow \mathcal{P}_c(X)$ represented by the table in Example 2.5.

We describe the algorithm in Theorem 4.9 (iii) with the pseudo-factorization of Example 4.7 (Construction (iv) only consists in the standard minimization of the reflected coalgebra α' , that is CTS (2) of the introduction). We start by taking the unique morphism $d_0: X \rightarrow 1$ into the final object of $\mathcal{Kl}(T)$, that is $1 = \{\bullet\}$. At the iteration i , we obtain e_i via the pseudo-factorization of $d_i = n_i \circ e_i$, and then we build $d_{i+1} = \overline{F}e_i \circ \alpha$. The iterations of the algorithm are shown in the following tables below.

$$d_0: X \rightarrow 1 = \{\bullet\} = E_0$$

d_0, e_0	1	2	3	4	5	6	7	8	9	10
a	•	•	•	•	•	•	•	•	•	•
\bar{a}	•	•	•	•	•	•	•	•	•	•

$$d_2: X \rightarrow \mathcal{P}_c(E_1), E_2 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\bullet\}\}\}$$

d_2, e_2	1	2	3	4	5	6	7	8	9	10
a	$\{\emptyset, \{\bullet\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset	$\{\emptyset, \{\bullet\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset
\bar{a}	$\{\emptyset, \{\bullet\}\}$	\emptyset	$\{\emptyset\}$	\emptyset	\emptyset	$\{\emptyset, \{\bullet\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset

$$d_1: X \rightarrow \mathcal{P}_c(E_0) = \{\emptyset, \{\bullet\}\} = E_1$$

d_1, e_1	1	2	3	4	5	6	7	8	9	10
a	$\{\bullet\}$	$\{\bullet\}$	\emptyset	\emptyset	$\{\bullet\}$	$\{\bullet\}$	\emptyset	\emptyset	\emptyset	\emptyset
\bar{a}	$\{\bullet\}$	\emptyset	$\{\bullet\}$	\emptyset	$\{\bullet\}$	$\{\bullet\}$	\emptyset	\emptyset	\emptyset	\emptyset

$$d_3: X \rightarrow \mathcal{P}_c(E_2), E_3 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$$

d_3, e_3	1	2	3	4	5	6	7	8	9	10
a	$\{\emptyset, \{\emptyset\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset	$\{\emptyset, \{\emptyset\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset
\bar{a}	$\{\emptyset, \{\emptyset\}\}$	\emptyset	$\{\emptyset\}$	\emptyset	\emptyset	$\{\emptyset, \{\emptyset\}\}$	$\{\emptyset\}$	\emptyset	\emptyset	\emptyset

Each table represents both d_i and $e_i: X \rightarrow E_i$ (the morphisms n_i such that $d_i = n_i \circ e_i$ are just the obvious injections). At the iterations 0 and 1, $E_0 = 1$ and $E_1 = \mathcal{P}_c(E_0)$. At the iteration 2 instead, $E_2 \neq \mathcal{P}_c(E_1)$, since nothing maps to $\{\{\bullet\}\} \in \mathcal{P}_c(E_1)$.

The algorithm reaches a fixed-point at iteration 3, since there is an iso $\iota: E_2 \rightarrow E_3$. The minimization ($E_3, \mathcal{P}_c(\iota) \circ n_3$) is depicted below.

$$\{\emptyset, \{\emptyset\}\} \xrightarrow{\quad} \{\emptyset\} \xrightarrow{\quad} \emptyset$$

It is easy to see that the above transition system is isomorphic to the one from the introduction having states x, y, z . Moreover, the coalgebra morphism $e_3: (X, \alpha) \rightarrow (E_3, \mathcal{P}_c(\iota) \circ n_3)$, illustrated in the table above, corresponds to the dashed arrow of the introduction, where 2 is mapped to $\{\emptyset\}$ ($= y$) if a holds, and to \emptyset ($= z$) if \bar{a} holds.

5 Conclusion, Related and Future Work

In this work, we have introduced a notion of minimization, which encompasses several concepts of “canonical” systems in the literature, and abstract procedures to compute it. Our approach only relies on *(pseudo-)factorization structures* and it is completely independent of the base category and of the endofunctor F . Together with appropriate reflections, this allows to compute minimizations of interesting types of systems that, for the purpose of minimization, cannot be regarded as coalgebras over **Set**, such as non-deterministic automata and conditional transition systems.

For non-deterministic automata, which we model as coalgebras in **Rel** following [13], the result of the proposed algorithm coincides with the one of the third step of Brzozowski’s algorithm [7]. The resulting automata are not minimal in the number of states (it is well-known that there exists no unique minimal non-deterministic automata), but they correspond to *átomata*, recently introduced in [8].

The example of conditional transition systems is completely original, but it has been motivated by the work in [14, 10], which introduces notions of bisimilarity depending on conditions (which are fixed once and for all). The setting of [10] is closer to ours, but no algorithm is given there. Our algorithm can be made more efficient by considering CTSS where conditions are boolean expressions. We already have a prototype implementation performing the fixed-point iteration based on binary decision diagrams. Moreover, our coalgebraic model of CTSS provides a notion of *quantitative bisimulations* that can

be seen as a behavioural (pseudo-)metric. We plan to study how our approach can be integrated to define and compute behavioural metrics.

As related work, we should also mention that the notion of minimization generalizes simple [22] and minimal [12] coalgebras in the case where the base category is **Set** with epi-mono factorizations. Moreover, several previous studies (e.g. [17][9][23]) have pointed out the relationship between the construction of the final coalgebra (via the final chain [24][4]) and the minimization algorithm. For instance, in case of regular categories the chain of quotients $e_i: X \rightarrow E_i$ (Construction 3.7) corresponds to the chain $K_i \rightrightarrows X \times X$ of their kernel pairs, which is precisely the relation refinement sequence of Staton [23, Section 5.1]. However, none of these works employed reflections for determinization-like constructions, that is exactly what allows us to minimize coalgebras in categories not equipped with a proper factorization structure, such as non-deterministic automata and conditional transition systems.

In future work we will study general conditions ensuring finite convergence: it is immediate to see that for any functor on **Set** with epi-mono factorizations, the sequence E_i of a finite coalgebra converges in a finite number of iterations. However, discovering general conditions encompassing all the examples of this paper seems to be non-trivial.

Preliminary research suggests that by integrating our approach with well-pointed coalgebras [5], we might obtain an explicit account of initial states. Indeed, given the reachable part of a pointed coalgebra for a set functor (which is defined through the canonical graph of Gumm [11]), the result of its minimization is a well-pointed coalgebra, i. e., a pointed coalgebra with no proper subcoalgebra and no proper quotient.

In addition we plan to study how our work is related to [20], which also recovers Brzozowski's algorithm in an abstract categorical setting.

Acknowledgements. We would like to thank Ana Sokolova, Paolo Baldan and Walter Tholen for answering our questions and giving generous feedback. We also thank the reviewers for their valuable comments.

References

1. Adámek, J.: Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolin.* 15, 589–602 (1974)
2. Adámek, J., Bonchi, F., Hülsbusch, M., König, B., Milius, S., Silva, A.: A coalgebraic perspective on minimization and determinization (extended version), <http://alexandrasilva.org/files/fossacs12-extended.pdf>
3. Adámek, J., Herrlich, H., Strecker, G.E.: *Abstract and Concrete Categories – The Joy of Cats*. Wiley (1990)
4. Adámek, J., Koubek, V.: On the greatest fixed point of a set functor. *TCS* 150, 57–75 (1995)
5. Adámek, J., Milius, S., Moss, L.S., Sousa, L.: Well-pointed coalgebras. In: Birkeedal, L. (ed.) *FOSSACS 2012*. LNCS, vol. 7213, Springer, Heidelberg (2012)
6. Boreale, M.: Weighted Bisimulation in Linear Algebraic Form. In: Bravetti, M., Zavattaro, G. (eds.) *CONCUR 2009*. LNCS, vol. 5710, pp. 163–177. Springer, Heidelberg (2009)
7. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata* 12(6), 529–561 (1962)
8. Brzozowski, J., Tamm, H.: Theory of Automata. In: Mauri, G., Leporati, A. (eds.) *DLT 2011*. LNCS, vol. 6795, pp. 105–116. Springer, Heidelberg (2011)

9. Ferrari, G.L., Montanari, U., Tuosto, E.: Coalgebraic minimization of HD-automata for the pi-calculus using polymorphic types. *TCS* 331(2-3), 325–365 (2005)
10. Fitting, M.: Bisimulations and boolean vectors. In: *Advances in Modal Logic*, vol. 4, pp. 1–29. World Scientific Publishing (2002)
11. Gumm, H.P.: From T -Coalgebras to Filter Structures and Transition Systems. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) *CALCO 2005*. LNCS, vol. 3629, pp. 194–212. Springer, Heidelberg (2005)
12. Gumm, H.P.: On minimal coalgebras. *Applied Categorical Structures* 16, 313–332 (2008)
13. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. *LMCS* 3(4:11), 1–36 (2007)
14. Hennessy, M., Lin, H.: Symbolic bisimulations. *TCS* 138(2), 353–389 (1995)
15. Hermida, C., Jacobs, B.: Structural induction and coinduction in a fibrational setting. *Information and Computation* 145, 107–152 (1998)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*, 3rd edn. Wesley (2006)
17. Kurz, A.: *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Ludwigs-Maximilians-Universität München (2000)
18. Mac Lane, S.: *Categories for the Working Mathematician*. Springer, Heidelberg (1971)
19. Mulry, P.S.: Lifting Theorems for Kleisli Categories. In: Main, M.G., Melton, A.C., Mislove, M.W., Schmidt, D., Brookes, S.D. (eds.) *MFPS 1993*. LNCS, vol. 802, pp. 304–319. Springer, Heidelberg (1994)
20. Panangaden, P.: Duality in probabilistic automata Slides (May 19, 2011), http://www.cs.mcgill.ca/~prakash/Talks/duality_talk.pdf
21. Power, J., Turi, D.: A coalgebraic foundation for linear time semantics. In: *Proc. of CTCS 1999*. ENTCS, vol. 29, pp. 259–274 (1999)
22. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. *TCS* 249, 3–80 (2000)
23. Staton, S.: Relating coalgebraic notions of bisimulation. *LMCS* 7(1) (2011)
24. Worrell, J.: On the final sequence of a finitary set functor. *TCS* 338(1-3), 184–199 (2005)

When Is a Container a Comonad?

Danel Ahman^{1,*}, James Chapman², and Tarmo Uustalu²

¹ Computer Laboratory, University of Cambridge,
15 J. J. Thomson Avenue, Cambridge CB3 0FD, United Kingdom
`danel.ahman@cl.cam.ac.uk`

² Institute of Cybernetics, Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia
`{james,tarmo}@cs.ioc.ee`

Abstract. Abbott, Altenkirch, Ghani and others have taught us that many parameterized datatypes (set functors) can be usefully analyzed via container representations in terms of a set of shapes and a set of positions in each shape. This paper builds on the observation that datatypes often carry additional structure that containers alone do not account for. We introduce directed containers to capture the common situation where every position in a datastructure determines another datastructure, informally, the sub-datastructure rooted by that position. Some natural examples are non-empty lists and node-labelled trees, and datastructures with a designated position (zippers). While containers denote set functors via a fully-faithful functor, directed containers interpret fully-faithfully into comonads. But more is true: every comonad whose underlying functor is a container is represented by a directed container. In fact, directed containers are the same as containers that are comonads. We also describe some constructions of directed containers. We have formalized our development in the dependently typed programming language Agda.

1 Introduction

Containers, as introduced by Abbott, Altenkirch and Ghani [1] are a neat representation for a wide class of parameterized datatypes (set functors) in terms of a set of shapes and a set of positions in each shape. They cover lists, colists, streams, various kinds of trees, etc. Containers can be used as a “syntax” for programming with these datatypes and reasoning about them, as can the strictly positive datatypes and polynomial functors of Dybjer [8], Moerdijk and Palmgren [16], Gambino and Hyland [9], and Kock [15]. The theory of this class of datatypes is elegant, as they are well-behaved in many respects.

This paper proceeds from the observation that datatypes often carry additional structure that containers alone do not account for. We introduce directed containers to capture the common situation in programming where every position in a datastructure determines another datastructure, informally, the

* The first author was a summer intern at the Institute of Cybernetics, Tallinn University of Technology when the bulk of this work was carried out.

sub-datastructure rooted by that position. Some natural examples of such datastructures are non-empty lists and node-labelled trees, and datastructures with a designated position or focus (zippers). In the former case, the sub-datastructure is a sublist or a subtree. In the latter case, it is the whole datastructure but with the focus moved to the given position.

We show that directed containers are no less neat than containers. While containers denote set functors via a fully-faithful functor, directed containers interpret fully-faithfully into comonads. They admit some of the constructions that containers do, but not others: for instance, two directed containers cannot be composed in general. Our main result is that every comonad whose underlying functor is the interpretation of a container is the interpretation of a directed container. So the answer to the question in the title of this paper is: a container is a comonad exactly when it is a directed container. In more precise terms, the category of directed containers is the pullback of the forgetful functor from the category of comonads to that of set functors along the interpretation functor of containers. This also means that a directed container is the same as a comonoid in the category of containers.

In Sect. 2, we review the basic theory of containers, showing also some examples. We introduce containers and their interpretation into set functors. We show some constructions of containers such as the coproduct of containers. In Sect. 3, we revisit our examples and introduce directed containers as a specialization of containers and describe their interpretation into comonads. We look at some constructions, in particular the focussed container (zipper) construction. Our main result, that a container is a comonad exactly when it is directed, is the subject of Sect. 4. In Sect. 5, we ask whether a similar characterization is possible for containers that are monads and hint that this is the case. We briefly summarize related work in Sect. 6 and conclude with outlining some directions for future work in Sect. 7.

We spend a section on the background theory of containers as they are central for our paper but relatively little known, but assume that the reader knows about comonads, monoidal categories, monoidal functors and comonoids.

In our mathematics, we use syntax similar to the dependently typed functional programming language Agda [18]. If some function argument will be derivable in most contexts, we mark it as implicit by enclosing it/its type in braces in the function's type declaration and either give this argument in braces or omit it in the definition and applications of the function.

For lack of space, we have omitted all proofs from the paper. We have formalised our proofs in Agda; the development is available at <http://cs.ioc.ee/~danel/dcont.html>.

2 Containers

We begin with a recap of containers. We introduce the category of containers and the fully-faithful functor into the category of set functors defining the interpretation of containers and show that these are monoidal. We also recall some

basic constructions of containers. For proofs of the propositions in this section and further information, we refer the reader to Abbott et al. [1, 4].

2.1 Containers

Containers are a form of “syntax” for datatypes. A *container* $S \triangleleft P$ is given by a set $S : \mathbf{Set}$ of *shapes* and a shape-indexed family $P : S \rightarrow \mathbf{Set}$ of *positions*.

Intuitively, shapes are “templates” for datastructures and positions identify “blanks” in these templates that can be filled with data. The datatype of lists is represented by $S \triangleleft P$ where the shapes $S = \mathbf{Nat}$ are the possible lengths of lists and the positions $P s = \mathbf{Fin} s = \{0, \dots, s-1\}$ provide s places for data in lists of length s . Non-empty lists are obtained by letting $S = \mathbf{Nat}$ and $P s = \mathbf{Fin} (s+1)$ (so that shape s has $s+1$ rather than s positions). Streams are characterized by a single shape with natural number positions: $S = 1 = \{*\}$ and $P * = \mathbf{Nat}$. The singleton datatype has one shape and one position: $S = 1, P * = 1$.

A *morphism* between containers $S \triangleleft P$ and $S' \triangleleft P'$ is a pair $t \triangleleft q$ of maps $t : S \rightarrow S'$ and $q : \Pi\{s : S\}. P'(t s) \rightarrow P s$ (the shape map and position map). Note how the positions are mapped backwards. The intuition is that, if a function between two datatypes does not look at the data, then the shape of a datastructure given to it must determine the shape of the datastructure returned and the data in any position in the shape returned must come from a definite position in the given shape.

For example, the head function, sending a non-empty list to a single data item, is determined by the maps $t : \mathbf{Nat} \rightarrow 1$ and $q : \Pi\{s : \mathbf{Nat}\}. 1 \rightarrow \mathbf{Fin} (s+1)$ defined by $t _ = *$ and $q * = 0$. The tail function, sending a non-empty list to a list, is represented by $t : \mathbf{Nat} \rightarrow \mathbf{Nat}$ and $q : \Pi\{s : \mathbf{Nat}\}. \mathbf{Fin} s \rightarrow \mathbf{Fin} (s+1)$ defined by $t s = s$ and $q p = p+1$. For the function dropping every second element of a non-empty list, the shape and position maps $t : \mathbf{Nat} \rightarrow \mathbf{Nat}$ and $q : \Pi\{s : \mathbf{Nat}\}. \mathbf{Fin} (s \div 2 + 1) \rightarrow \mathbf{Fin} (s+1)$ are $t s = s \div 2$ and $q \{s\} p = p * 2$. For reversal of non-empty lists, they are $t : \mathbf{Nat} \rightarrow \mathbf{Nat}$ and $q : \Pi\{s : \mathbf{Nat}\}. \mathbf{Fin} (s+1) \rightarrow \mathbf{Fin} (s+1)$ defined by $t s = s$ and $q \{s\} p = s - p$. (See Prince et al. [19] for more similar examples.)

The *identity* morphism $\text{id}^c \{C\}$ on a container $C = S \triangleleft P$ is defined by $\text{id}^c = \text{id} \{S\} \triangleleft \lambda \{s\}. \text{id} \{P s\}$. The *composition* $h \circ^c h'$ of container morphisms $h = t \triangleleft q$ and $h' = t' \triangleleft q'$ is defined by $h \circ^c h' = t \circ t' \triangleleft \lambda \{s\}. q' \{s\} \circ q \{t' s\}$. Composition of container morphisms is associative, identity is the unit.

Proposition 1. *Containers form a category* **Cont**.

2.2 Interpretation of Containers

To map containers into datatypes made of datastructures that have the positions in some shape filled with data, we must equip containers with a “semantics”.

For a container $C = S \triangleleft P$, we define its *interpretation* $\llbracket C \rrbracket^c : \mathbf{Set} \rightarrow \mathbf{Set}$ on sets by $\llbracket C \rrbracket^c X = \Sigma s : S. P s \rightarrow X$, so that $\llbracket C \rrbracket^c X$ consists of pairs of a shape and an assignment of an element of X to each of the positions in this shape, reflecting

the “template-and-blanks” idea. The interpretation $\llbracket C \rrbracket^c : \forall \{X\}, \{Y\}. (X \rightarrow Y) \rightarrow (\Sigma s : S.P s \rightarrow X) \rightarrow \Sigma s : S.P s \rightarrow Y$ of C on functions is defined by $\llbracket C \rrbracket^c f (s, v) = (s, f \circ v)$. It is straightforward that $\llbracket C \rrbracket^c$ preserves identity and composition of functions, so it is a set functor (as any datatype should be).

Our example containers denote the datatypes intended. If we let C be the container of lists, we have $\llbracket C \rrbracket^c X = \Sigma s : \text{Nat}. \text{Fin } s \rightarrow X \cong \text{List } X$. The container of streams interprets into $\Sigma * : 1. \text{Nat} \rightarrow X \cong \text{Nat} \rightarrow X \cong \text{Str } X$. Etc.

A morphism $h = t \triangleleft q$ between containers $C = S \triangleleft P$ and $C' = S' \triangleleft P'$ is interpreted as a natural transformation between $\llbracket C \rrbracket^c$ and $\llbracket C' \rrbracket^c$, i.e., as a polymorphic function $\llbracket h \rrbracket^c : \forall \{X\}. (\Sigma s : S.P s \rightarrow X) \rightarrow \Sigma s' : S'.P' s' \rightarrow X$ that is natural. It is defined by $\llbracket h \rrbracket^c (s, v) = (t s, v \circ q \{s\})$. $\llbracket - \rrbracket^c$ preserves the identities and composition of container morphisms.

The interpretation of the container morphism h corresponding to the list head function $\llbracket h \rrbracket^c : \forall \{X\}. (\Sigma s : \text{Nat}. \text{Fin } (s + 1) \rightarrow X) \rightarrow \Sigma * : 1. 1 \rightarrow X$ is defined by $\llbracket h \rrbracket^c (s, v) = (*, \lambda *. v 0)$.

Proposition 2. $\llbracket - \rrbracket^c$ is a functor from **Cont** to **[Set, Set]**.

Every natural transformation between container interpretations is the interpretation of some container morphism. For containers $C = S \triangleleft P$ and $C' = S' \triangleleft P'$, a natural transformation τ between $\llbracket C \rrbracket^c$ and $\llbracket C' \rrbracket^c$, i.e., a polymorphic function $\tau : \forall \{X\}. (\Sigma s : S.P s \rightarrow X) \rightarrow \Sigma s' : S'.P' s' \rightarrow X$ that is natural, can be “quoted” to a container morphism $\ulcorner \tau \urcorner^c = (t \triangleleft q)$ between C and C' where $t : S \rightarrow S'$ and $q : \Pi \{s : S\}. P' (t s) \rightarrow P s$ are defined by $\ulcorner \tau \urcorner^c = (\lambda s. \text{fst } (\tau \{P s\} (s, \text{id}))) \triangleleft (\lambda \{s\}. \text{snd } (\tau \{P s\} (s, \text{id})))$.

For any container morphism h , $\ulcorner \llbracket h \rrbracket^c \urcorner^c = h$, and, for any natural transformation τ and τ' between container interpretations, $\ulcorner \tau \urcorner^c = \ulcorner \tau' \urcorner^c$ implies $\tau = \tau'$.

Proposition 3. $\llbracket - \rrbracket^c$ is fully faithful.

2.3 Monoidal Structure

We have already seen the *identity* container $\text{Id}^c = 1 \triangleleft \lambda *. 1$. The *composition* $C_0 \cdot^c C_1$ of containers $C_0 = S_0 \triangleleft P_0$ and $C_1 = S_1 \triangleleft P_1$ is the container $S \triangleleft P$ defined by $S = \Sigma s : S_0.P_0 s \rightarrow S_1$ and $P (s, v) = \Sigma p_0 : P_0 s.P_1 (v p_0)$. It has as shapes pairs of an outer shape s and an assignment of an inner shape to every position in s . The positions in the composite container are pairs of a position p in the outer shape and a position in the inner shape assigned to p . The (horizontal) composition $h_0 \cdot^c h_1$ of container morphisms $h_0 = t_0 \triangleleft q_0$ and $h_1 = t_1 \triangleleft q_1$ is the container morphism $t \triangleleft q$ defined by $t (s, v) = (t_0 s, t_1 \circ v \circ q_0 \{s\})$ and $q \{s, v\} (p_0, p_1) = (q_0 \{s\} p_0, q_1 \{v (q_0 \{s\} p_0)\} p_1)$. The horizontal composition preserves the identity container morphisms and the (vertical) composition of container morphisms, which means that $- \cdot^c -$ is a bifunctor.

Cont has isomorphisms $\rho : \forall \{C\}. C \cdot^c \text{Id}^c \rightarrow C$, $\lambda : \forall \{C\}. \text{Id}^c \cdot^c C \rightarrow C$ and $\alpha : \forall \{C\} \{C'\} \{C''\}. (C \cdot^c C') \cdot^c C'' \rightarrow C \cdot^c (C' \cdot^c C'')$, defined by $\rho = \lambda (s, v). s \triangleleft \lambda \{s, v\}. \lambda p. (p, *)$, $\lambda = \lambda (*, v). v * \triangleleft \lambda \{*, v\}. \lambda p. (*, p)$, $\alpha = \lambda ((s, v), v'). (s, \lambda p. (v p, \lambda p'. v' (p, p'))) \triangleleft \lambda \{(s, v), v'\}. \lambda (p, (p', p'')). ((p, p'), p'')$.

Proposition 4. *The category **Cont** is a monoidal category.*

There are also natural isomorphisms $\mathbf{e} : \mathbf{ld} \rightarrow \llbracket \mathbf{ld}^c \rrbracket^c$ and $\mathbf{m} : \forall \{C_0\}, \{C_1\}. \llbracket C_0 \rrbracket^c \cdot \llbracket C_1 \rrbracket^c \rightarrow \llbracket C_0 \cdot C_1 \rrbracket^c$ that are defined by $\mathbf{e} x = (*, \lambda *. x)$ and $\mathbf{m}(s, v) = ((s, \lambda p. \mathbf{fst}(v p)), \lambda(p, p'). \mathbf{snd}(v p) p')$ and are coherent.

Proposition 5. *The functor $\llbracket - \rrbracket^c$ is a monoidal functor.*

2.4 Constructions of Containers

Containers are closed under various constructions such as products, coproducts and constant exponentiation, preserved by interpretation.

- For two containers $C_0 = S_0 \triangleleft P_0$ and $C_1 = S_1 \triangleleft P_1$, their *product* $C_0 \times C_1$ is the container $S \triangleleft P$ defined by $S = S_0 \times S_1$ and $P(s_0, s_1) = P_0 s_0 + P_1 s_1$. It holds that $\llbracket C_0 \times C_1 \rrbracket^c \cong \llbracket C_0 \rrbracket^c \times \llbracket C_1 \rrbracket^c$.
- The *coproduct* $C_0 + C_1$ of containers $C_0 = S_0 \triangleleft P_0$ and $C_1 = S_1 \triangleleft P_1$ is the container $S \triangleleft P$ defined by $S = S_0 + S_1$, $P(\text{inl } s) = P_0 s$ and $P(\text{inr } s) = P_1 s$. It is the case that $\llbracket C_0 + C_1 \rrbracket^c \cong \llbracket C_0 \rrbracket^c + \llbracket C_1 \rrbracket^c$.
- For a set $K \in \mathbf{Set}$ and a container $C_0 = S_0 \triangleleft P_0$, the *exponential* $K \rightarrow C_0$ is the container $S \triangleleft P$ where $S = K \rightarrow S_0$ and $P f = \Sigma k : K. P(f k)$. We have that $\llbracket K \rightarrow C_0 \rrbracket^c \cong K \rightarrow \llbracket C_0 \rrbracket^c$.

3 Directed Containers

We now proceed to our contribution, directed containers. We define the category of directed containers and a fully-faithful functor interpreting directed containers as comonads, and discuss some examples and constructions.

3.1 Directed Containers

Datatypes often carry some additional structure that is worth making explicit. For example, each node in a list or non-empty list defines a sublist (a suffix). In container terms, this corresponds to every position in a shape determining another shape, the subshape corresponding to this position. The theory of containers alone does not account for such additional structure. Directed containers, studied in the rest of this paper, axiomatize subshapes and translation of positions in a subshape into the global shape.

A *directed container* is a container $S \triangleleft P$ together with three operations

- $\downarrow : \Pi s : S. P s \rightarrow S$ (the subshape for a position),
- $\circ : \Pi \{s : S\}. P s$ (the root),
- $\oplus : \Pi \{s : S\}. \Pi p : P s. P(s \downarrow p) \rightarrow P s$ (translation of subshape positions into positions in the global shape).

satisfying the following two shape equations and three position equations:

1. $\forall \{s\}. s \downarrow \circ = s,$
2. $\forall \{s, p, p'\}. s \downarrow (p \oplus p') = (s \downarrow p) \downarrow p',$
3. $\forall \{s, p\}. p \oplus \{s\} \circ = p,$
4. $\forall \{s, p\}. \circ \{s\} \oplus p = p,$
5. $\forall \{s, p, p', p''\}. (p \oplus \{s\} p') \oplus p'' = p \oplus (p' \oplus p'').$

(Using \oplus as an infix operation, we write the first, implicit, argument next to the operation symbol when we want to give it explicitly.) Modulo the fact that the positions involved come from different sets, laws 3-5 are the laws of a monoid.

To help explain the operations and laws, we sketch in Fig. 1 a datastructure with nested sub-datastructures.

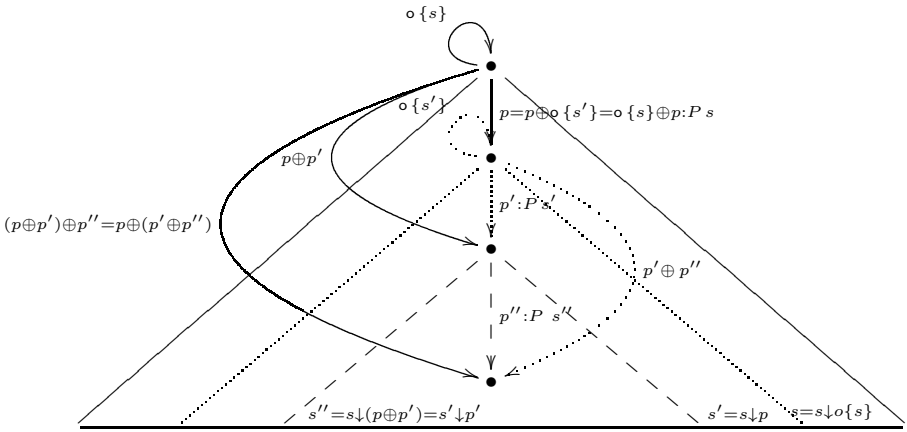


Fig. 1. A datastructure with two nested sub-datastructures

The global shape s is marked with a solid boundary and has a root position $\circ\{s\}$. Then, any position p in s determines a shape $s' = s \downarrow p$, marked with a dotted boundary, to be thought of as the subshape of s given by this position. The root position in s' is $\circ\{s'\}$. Law 3 says that its translation $p \oplus \circ\{s'\}$ into a position in shape s is p , reflecting the idea that the subshape given by a position should have that position as the root.

By law 1, the subshape $s \downarrow \circ\{s\}$ corresponding to the root position $\circ\{s\}$ in the global shape s is s itself. Law 4, which is only well-typed thanks to law 1, stipulates that the translation of position p in $s \downarrow \circ\{s\}$ into a position in s is just p (which is possible, as $P(s \downarrow \circ\{s\}) = P s$).

A further position p' in s' determines a shape $s'' = s' \downarrow p'$. But p' also translates into a position $p \oplus p'$ in s and that determines a shape $s \downarrow (p \oplus p')$. Law 2 says that s'' and $s \downarrow (p \oplus p')$ are the same shape, which is marked by a dashed boundary in the figure. Finally, law 5 (well-typed only because of law 2) says that the two alternative ways to translate a position p'' in shape s'' into a position in shape s agree with each other.

Lists cannot form a directed container, as the shape 0 (for the empty list), having no positions, has no possible root position. But the container of *non-empty lists* (with $S = \mathbf{Nat}$ and $P s = \mathbf{Fin}(\mathbf{succ} s)$) is a directed container with respect to *suffixes* as (non-empty) sublists. The subshape given by a position p in a shape s (for lists of length $s + 1$) is the shape of the corresponding suffix, given by $s \downarrow p = s - p$. The root $\mathbf{o}\{s\}$ is the position 0 of the head node. A position in the global shape is recovered from a position p' in the subshape of the position p by $p \oplus p' = p + p'$.

The “template” of non-empty lists of shape $s = 5$ (length 6) is given in Fig. 2. This figure also shows that the subshape determined by a position $p = 2$ in the global shape s is $s' = s \downarrow p = 5 - 2 = 3$ and a position $p' = 1$ in s' is rendered as the position $p \oplus p' = 2 + 1 = 3$ in the initial shape. Clearly one could

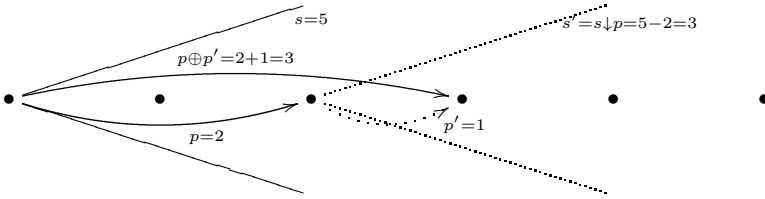


Fig. 2. The “template” of non-empty lists of shape 5 (length 6)

also choose prefixes as subshapes and the last node of a non-empty list as the root, but this gives an isomorphic directed container. Non-empty lists also give rise to an entirely different directed container structure that has *cyclic shifts* as “sublists” (this example was suggested to us by Jeremy Gibbons). The subshape at each position is the global shape ($s \downarrow p = s$). The root is still $\mathbf{o}\{s\} = 0$. The interesting part is that translation into the global shape of a subshape position is defined by $p \oplus \{s\} p' = (p + p') \bmod s$, satisfying all the required laws.

The container of *streams* ($S = 1$, $P * = \mathbf{Nat}$) carries a very trivial directed container structure given by $* \downarrow p = *$, $\mathbf{o} = 0$ and $p \oplus p' = p + p'$. Fig. 3 shows how a position $p = 2$ in the only possible global shape $s = *$ and a position $p' = 2$ in the equal subshape $s' = s \downarrow p = *$ give back a position $p + p = 4$ in the global shape.

Similarly to the theory of containers, one can also define morphisms between directed containers. A *morphism* between directed containers $(S \triangleleft P, \downarrow, \mathbf{o}, \oplus)$ and $(S' \triangleleft P', \downarrow', \mathbf{o}', \oplus')$ is a morphism $t \triangleleft q$ between the containers $S \triangleleft P$ and $S' \triangleleft P'$ that satisfies three laws:

- $\forall \{s, p\}. t (s \downarrow q p) = t s \downarrow' p,$
- $\forall \{s\}. \mathbf{o} \{s\} = q (\mathbf{o}' \{t s\}),$
- $\forall \{s, p, p'\}. q p \oplus \{s\} q p' = q (p \oplus' \{t s\} p').$

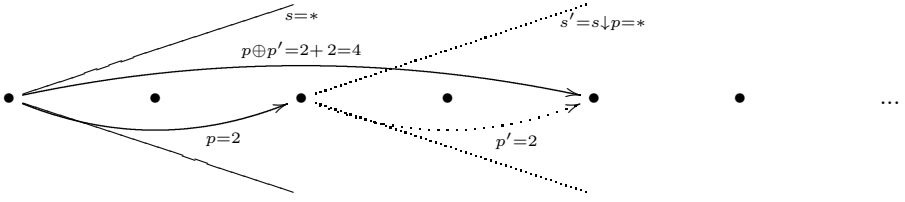


Fig. 3. The template of streams

Recall the intuition that t determines the shape of the datastructure that some given datastructure is sent to and q identifies for every position in the datastructure returned a position in the given datastructure. These laws say that the positions in the sub-datastructure for any position in the resulting datastructure must map back to positions in the corresponding sub-datastructure of the given datastructure. This means that they can receive data only from those positions, other flows are forbidden.

The container representations of the head and drop-even functions for non-empty lists are directed container morphisms. But that of reversal is not.

The identities and composition of **Cont** can give the identities and composition for directed containers, since for every directed container $E = (C, \downarrow, \circ, \oplus)$, the identity container morphism $\text{id}^c \{C\}$ is a directed container morphism and the composition $h \circ^c h'$ of two directed container morphisms is also a directed container morphism.

Proposition 6. *Directed containers form a category **DCont**.*

3.2 Interpretation of Directed Containers

As directed containers are containers with some operations obeying some laws, a directed container should denote not just a set functor, but a set functor with operations obeying some laws. The correct domain of denotation for directed containers is provided by comonads on sets.

Given a directed container $E = (S \triangleleft P, \downarrow, \circ, \oplus)$, we define its *interpretation* $\llbracket E \rrbracket^{\text{dc}}$ to be the set functor $D = \llbracket S \triangleleft P \rrbracket^c$ (i.e., the interpretation of the underlying container) together with two natural transformations

$$\begin{aligned} \varepsilon &: \forall \{X\}. (\Sigma s : S.P s \rightarrow X) \rightarrow X \\ \varepsilon(s, v) &= v(\circ \{s\}) \\ \delta &: \forall \{X\}. (\Sigma s : S.P s \rightarrow X) \rightarrow \Sigma s' : S.P s' \rightarrow X \\ \delta(s, v) &= (s, \lambda p. (s \downarrow p, \lambda p'. v(p \oplus \{s\} p'))) \end{aligned}$$

The directed container laws ensure that the natural transformations ε, δ make the counit and comultiplication of a comonad structure on D .

Intuitively, the counit extracts the data at the root position of a datastructure (e.g., the head of a non-empty list), the comultiplication, which produces a datastructure of datastructures, replaces the data at every position with the sub-datastructure corresponding to this position (e.g., the corresponding suffix or cyclic shift).

The interpretation $\llbracket h \rrbracket^{\text{dc}}$ of a morphism h between directed containers $E = (C, \downarrow, \circ, \oplus)$, $E' = (C', \downarrow', \circ', \oplus')$ is defined by $\llbracket h \rrbracket^{\text{dc}} = \llbracket h \rrbracket^{\text{c}}$ (using that h is a container morphism between C and C'). The directed container morphism laws ensure that this natural transformation between $\llbracket C \rrbracket^{\text{c}}$ and $\llbracket C' \rrbracket^{\text{c}}$ is also a comonad morphism between $\llbracket E \rrbracket^{\text{dc}}$ and $\llbracket E' \rrbracket^{\text{dc}}$.

Since **Comonads(Set)** inherits its identities and composition from **[Set, Set]**, $\llbracket - \rrbracket^{\text{dc}}$ also preserves the identities and composition.

Proposition 7. $\llbracket - \rrbracket^{\text{dc}}$ is a functor from **DCont** to **Comonads(Set)**.

Similarly to the case of natural transformations between container interpretations, one can also “quote” comonad morphisms between directed container interpretations into directed container morphisms. For any directed containers $E = (C, \downarrow, \circ, \oplus)$, $E' = (C', \downarrow', \circ', \oplus')$ and any morphism τ between the comonads $\llbracket E \rrbracket^{\text{dc}}$ and $\llbracket E' \rrbracket^{\text{dc}}$ (which is a natural transformation between $\llbracket C \rrbracket^{\text{c}}$ and $\llbracket C' \rrbracket^{\text{c}}$), the container morphism $\ulcorner \tau \urcorner^{\text{dc}} = \ulcorner \tau \urcorner^{\text{c}}$ between the underlying containers C and C' is also a directed container morphism between E and E' . The directed container morphism laws follow from the comonad morphism laws.

From what we already know about interpretation and quoting of container morphisms, it is immediate that $\ulcorner \llbracket h \rrbracket^{\text{dc}} \urcorner^{\text{dc}} = h$ for any directed container morphism h and that $\ulcorner \tau \urcorner^{\text{dc}} = \ulcorner \tau' \urcorner^{\text{dc}}$ implies $\tau = \tau'$ for any comonad morphisms τ and τ' between directed container interpretations.

Proposition 8. $\llbracket - \rrbracket^{\text{dc}}$ is fully faithful.

The *identity container* $\text{Id}^{\text{c}} = 1 \triangleleft \lambda *. 1$ extends trivially to an identity directed container whose denotation is isomorphic to the identity comonad. But, similarly to the situation with functors and comonads, composition of containers fails to yield a composition monoidal structure on **DCont**.

3.3 Constructions of Directed Containers

We now show some constructions of directed containers. While some standard constructions of containers extend to directed containers, others do not.

Coproducts. Given two directed containers $E_0 = (S_0 \triangleleft P_0, \downarrow_0, \circ_0, \oplus_0)$, $E_1 = (S_1 \triangleleft P_1, \downarrow_1, \circ_1, \oplus_1)$, their coproduct is $(S \triangleleft P, \downarrow, \circ, \oplus)$ whose underlying container $S \triangleleft P$ is the coproduct of containers $S_0 \triangleleft P_0$ and $S_1 \triangleleft P_1$. All of the directed container operations are defined either using $\downarrow_0, \circ_0, \oplus_0$ or $\downarrow_1, \circ_1, \oplus_1$ depending on the given shape. This means that the subshape is given by $\text{inl } s \downarrow p = \text{inl } (s \downarrow_0 p)$ and $\text{inr } s \downarrow p = \text{inr } (s \downarrow_1 p)$, the root position is given by $\circ \{\text{inl } s\} = \circ_0 \{s\}$ or $\circ \{\text{inr } s\} = \circ_1 \{s\}$ and the position in the initial shape is given by $p \oplus \{\text{inl } s\} p' = p \oplus_0 \{s\} p'$ and $p \oplus \{\text{inr } s\} p' = p \oplus_1 \{s\} p'$. Its interpretation is isomorphic to the coproduct of comonads $\llbracket E_0 \rrbracket^{\text{dc}}$ and $\llbracket E_1 \rrbracket^{\text{dc}}$.

Directed containers from monoids. Any monoid (M, e, \bullet) gives rise to a directed container $E = (S \triangleleft P, \downarrow, \mathfrak{o}, \oplus)$ where there is only one shape $*$ (with $S = 1$) whose positions $P * = M$ are the elements in the carrier set. The subshape operation $* \downarrow p = *$ thus becomes trivial as there is only one shape to return. Furthermore, the root position $\mathfrak{o} \{*\} = e$ in the shape $*$ is the unit of the monoid and the position in the initial shape is given by using the monoid operation $p \oplus \{*\} p' = p \bullet p'$. The interpretation of this directed container is the comonad (D, ε, δ) where $D X = M \rightarrow X$, $\varepsilon = \lambda f. f e$, $\delta = \lambda f. \lambda p. p'. f (p \bullet p')$.

Cofree directed containers. The cofree directed container on a container $C = S_0 \triangleleft P_0$ is $E = (S \triangleleft P, \downarrow, \mathfrak{o}, \oplus)$ where the underlying container is defined as $S = \nu Z. \Sigma s : S_0. P_0 s \rightarrow Z$ and $P = \mu Z. \lambda(s, v). 1 + \Sigma p : P_0 s. Z (vp)$. The subshapes are defined by $(s, v) \downarrow \text{inl} * = (s, v)$ and $(s, v) \downarrow \text{inr} (p, p') = vp \downarrow p'$. The root position is defined by $\mathfrak{o} \{s, v\} = \text{inl} *$ and subshape positions by $\text{inl} * \oplus \{s, v\} p'' = p''$ and $\text{inr} (p, p') \oplus \{s, v\} p'' = \text{inr} (p, p' \oplus \{vp\} p'')$. The interpretation $\llbracket E \rrbracket^{\text{dc}} = (D, \varepsilon, \delta)$ of this directed container has its underlying functor given by $D X = \nu Z. X \times \llbracket C \rrbracket^c Z$ and is the cofree comonad on the functor $\llbracket C \rrbracket^c$.

A different directed container, the cofree recursive directed container on C is obtained by replacing the ν in the definition of S with μ . The interpretation has its underlying functor given by $D X = \mu Z. X \times \llbracket C \rrbracket^c Z$ and is the cofree recursive comonad on $\llbracket C \rrbracket^c$.

There is no general way to endow the product of the underlying containers of two directed containers $E_0 = (S_0 \triangleleft P_0, \downarrow_0, \mathfrak{o}_0, \oplus_0)$ and $E_1 = (S_1 \triangleleft P_1, \downarrow_1, \mathfrak{o}_1, \oplus_1)$ with the structure of a directed container. One can define $S = S_0 \times S_1$ and $P(s_0, s_1) = P_0 s_0 + P_1 s_1$, but there are two choices \mathfrak{o}_0 and \mathfrak{o}_1 for \mathfrak{o} . Moreover, there is no general way to define $p \oplus p'$. But this should not be surprising, as the product of the underlying functors of two comonads is not generally a comonad. Also, the product of two comonads would not be a comonad structure on the product of the underlying functors.

3.4 Focussing

Another interesting construction turning any container into a directed container is “focussing”.

Datastructures with a focus. Any container $C = S_0 \triangleleft P_0$ defines a directed container $(S \triangleleft P, \downarrow, \mathfrak{o}, \oplus)$ as follows. We take $S = \Sigma s : S_0. P_0 s$, so that a shape is a pair of a shape s , the “shape proper”, and an arbitrary position p in that shape, the “focus”. We take $P(s, p) = P_0 s$, so that a position in the shape (s, p) is a position in the shape proper s , irrespective of the focus. The subshape determined by position p' in shape (s, p) is given by keeping the shape proper but changing the focus: $(s, p) \downarrow p' = (s, p')$. The root in the shape (s, p) is the focus p such that $\mathfrak{o} \{s, p\} = p$. Finally, we take the translation of positions from the subshape (s, p') given by position p' to shape (s, p) to be the identity, by defining

$p' \oplus \{s, p\} p'' = p''$. All directed container laws are satisfied. This directed container interprets into the canonical comonad structure on the functor $\partial[[C]]^c \times \text{Id}$ where ∂F denotes the derivative of the functor F .

Zippers. Inductive (tree-like) datatypes with a designated focus position are isomorphic to the zipper types of Huet [13]. A zipper datastructure encodes a tree with a focus as a pair of a context and a tree. The tree is the subtree of the global tree rooted by the focus and the context encodes the rest of the global tree. On zippers, changing the focus is supported via local navigation operations for moving one step down into the tree or up or aside into the context.

Zipper datatypes are directly representable as directed containers. We illustrate this on the example of zippers for non-empty lists. Such a zipper is a pair of a list (the context) and a non-empty list (the suffix determined by the focus position). Accordingly, by defining $S = \text{Nat} \times \text{Nat}$, the shape of a zipper is a pair (s_0, s_1) where s_0 is the shape of the context and s_1 is the shape of the suffix. For positions, it is convenient to choose $P(s_0, s_1) = \{-s_0, \dots, s_1\}$ by allocating the negative numbers in the interval for positions in the context and non-negative numbers for positions in the suffix. The root position is $\circ\{s_0, s_1\} = 0$, i.e., the focus. The subshape for each position is given by $(s_0, s_1) \downarrow p = (s_0 + p, s_1 - p)$ and translation of subshape positions by $p \oplus \{s_0, s_1\} p' = p + p'$.

Fig. 4 gives an example of a non-empty list with focus with its shape fixed to $s = (5, 6)$. It should be clear from the figure how the \oplus operation works on positions $p = 4$ and $p' = -7$ to get back the position $p \oplus p' = -3$ in the initial shape. The subshape operation \downarrow works as follows: $s \downarrow p$ gives back a subshape $s' = (9, 2)$ and $s \downarrow (p \oplus p')$ gives $s'' = (2, 9)$.

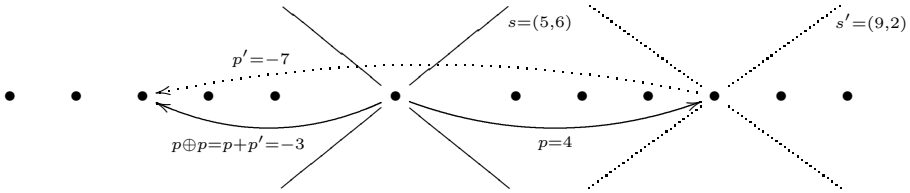


Fig. 4. The template for non-empty lists of length 12 focussed at position 5

4 Containers \cap Comonads = Directed Containers

Since not every functor can be represented by a container, there is no point in asking whether every comonad can be represented as a directed container. An example of a natural comonad that is not a directed container is the cofree comonad on the finite powerset functor \mathcal{P}_f (node-labelled nonwellfounded strongly-extensional trees) where the carrier of this comonad is not a container

(\mathcal{P}_f is also not a container). But, what about those comonads whose underlying functor is an interpretation of a container? It turns out that any such comonad does indeed define a directed container that is obtained as follows.

Given a comonad (D, ε, δ) and a container $C = S \triangleleft P$ such that $D = \llbracket C \rrbracket^c$, the counit ε and comultiplication δ induce container morphisms

$$\begin{aligned} h^\varepsilon &: C \rightarrow \mathbf{Id}^c \\ h^\varepsilon &= t^\varepsilon \triangleleft q^\varepsilon = \ulcorner \mathbf{e} \circ \varepsilon \urcorner^c \\ h^\delta &: C \rightarrow C \cdot^c C \\ h^\delta &= t^\delta \triangleleft q^\delta = \ulcorner \mathbf{m} \{C\} \{C\} \circ \delta \urcorner^c \end{aligned}$$

using that $\llbracket - \rrbracket^c$ is fully faithful. From (D, ε, δ) satisfying the laws of a comonad we can prove that $(C, h^\varepsilon, h^\delta)$ satisfies the laws of a comonoid in **Cont**. Further, we can define

$$\begin{aligned} s \downarrow p &= \mathbf{snd} (t^\delta s) p \\ \circ \{s\} &= q^\varepsilon \{s\} * \\ p \oplus \{s\} p' &= q^\delta \{s\} (p, p') \end{aligned}$$

and the comonoid laws further enforce the laws of the directed container for $(C, \downarrow, \circ, \oplus)$.

It may seem that the maps t^ε and $\mathbf{fst} \circ t^\delta$ are not used in the directed container structure, but $t^\varepsilon : S \rightarrow 1$ contains no information ($\forall \{s\}. t^\varepsilon s = *$) and the comonad/comonoid right unit law forces that $\forall \{s\}. \mathbf{fst} (t^\delta s) = s$, which gets used in the proof of each of the five directed container laws. The latter fact is quite significant. It tells us that the comultiplication δ of any comonad whose underlying functor is the interpretation of a container preserves the shape of a given datastructure as the outer shape of the datastructure returned.

The situation is summarized as follows.

Proposition 9. *Any comonad (D, ε, δ) and container $C = S \triangleleft P$ such that $D = \llbracket C \rrbracket^c$ determine a directed container $\llbracket (D, \varepsilon, \delta), C \rrbracket$.*

Proposition 10. $\llbracket \llbracket C, \downarrow, \circ, \oplus \rrbracket^{\mathbf{dc}}, C \rrbracket = (C, \downarrow, \circ, \oplus)$.

Proposition 11. $\llbracket \llbracket (D, \varepsilon, \delta), C \rrbracket^{\mathbf{dc}} \rrbracket = (D, \varepsilon, \delta)$.

These observations suggest the following theorem.

Proposition 12. *The following is a pullback in **CAT**:*

$$\begin{array}{ccc} \mathbf{DCont} & \xrightarrow{U} & \mathbf{Cont} \\ \llbracket - \rrbracket^{\mathbf{dc}} \downarrow \text{f.f.} & & \llbracket - \rrbracket^c \downarrow \text{f.f.} \\ \mathbf{Comonads}(\mathbf{Set}) & \xrightarrow{U} & \llbracket \mathbf{Set}, \mathbf{Set} \rrbracket \end{array}$$

It is proved by first noting that a pullback is provided by **Comonoids(Cont)** and then verifying that **Comonoids(Cont)** is isomorphic to **DCont**.

Sam Staton pointed it out to us that the proof of the first part only hinges on **Cont** and $\llbracket \mathbf{Set}, \mathbf{Set} \rrbracket$ being monoidal categories and $\llbracket - \rrbracket^c : \mathbf{Cont} \rightarrow \llbracket \mathbf{Set}, \mathbf{Set} \rrbracket$

being a fully faithful monoidal functor. Thus we actually establish a more general fact, viz., that for any two monoidal categories \mathcal{C} and \mathcal{D} and a fully-faithful monoidal functor $F : \mathcal{C} \rightarrow \mathcal{D}$, the pullback of F along the forgetful functor $U : \mathbf{Comonoids}(\mathcal{D}) \rightarrow \mathcal{D}$ is $\mathbf{Comonoids}(\mathcal{C})$.

In summary, we have seen that the interpretation of a container carries the structure of a comonad exactly when it extends to a directed container.

5 Containers \cap Monads = ?

Given that comonads whose underlying functor is the interpretation of a container are the same as directed containers, it is natural to ask whether a similar characterization is possible for monads whose underlying functor can be represented as a container. The answer is “yes”, but the additional structure is more involved than that of directed containers.

Given a container $C = S \triangleleft P$, the structure (η, μ) of a monad on the functor $T = \llbracket C \rrbracket^c$ is interdefinable with the following structure on C

- $e : S$ (for the shape map for η),
- $\bullet : \Pi s : S.(P s \rightarrow S) \rightarrow S$ (for the shape map for μ),
- $\wedge : \Pi \{s : S\}. \Pi v : P s \rightarrow S.P (s \bullet v) \rightarrow P s$ and
- $\uparrow : \Pi \{s : S\}. \Pi v : P s \rightarrow S. \Pi p : P (s \bullet v). P (v (v \wedge \{s\} p))$ (both for the position map for μ)

subject to three shape equations and five position equations. Perhaps not unexpectedly, this amounts to having a monoid structure on C .

To get some intuition, consider the monad structure on the datatype of lists. The unit is given by singleton lists and multiplication is flattening a list of lists by concatenation. For the list container $S = \mathbf{Nat}$, $P s = \mathbf{Fin} s$, we get that $e = 1$, $s \bullet v = \sum_{p : \mathbf{Fin} s} v p$, $v \wedge \{s\} p = [\text{greatest } p' : \mathbf{Fin} s \text{ such that } \sum_{p'' : \mathbf{Fin} p'} v p'' \leq p]$ and $v \uparrow \{s\} p = p - \sum_{p'' : \mathbf{Fin} (v \wedge \{s\} p)} v p''$. The reason is that the shape of singleton lists is e while flattening a list of lists with outer shape s and inner shape $v p$ for every position p in s results in a list of shape $s \bullet v$. For a position p in the shape of the flattened list, the corresponding positions in the outer and inner shapes of the given list of lists are $v \wedge \{s\} p$ and $v \uparrow \{s\} p$.

For lack of space, we refrain from a more detailed discussion of this variation of the concept of containers.

6 Related Work

We build on the theory of containers as developed by Abbott, Altenkirch and Ghani [1, 4] to analyze strictly positive datatypes. Some generalizations of the concept of containers are the indexed containers of Altenkirch and Morris [5, 17] and the quotient containers of Abbott et al. [2]. In our work we look at a specialization of containers rather than a generalization. Simple/indexed containers are intimately related to strongly positive datatypes/families and simple/dependent

polynomial functors as appearing in the works of Dybjer [8], Moerdijk and Palmgren [16], Gambino and Hyland [9], Kock [15]. Girard's normal functors [11] and Joyal's analytic functors [14] functors are similar to containers resp. quotient containers, but only allow for finitely many positions in a shape.

Gambino and Kock [10] treat polynomial monads.

Abbott, Altenkirch, Ghani and McBride [3] have investigated derivatives of datatypes that provide a systematic way to explain Huet's zipper type [13].

Brookes and Geva [6] and later Uustalu with coauthors [20, 21, 12, 7] have used comonads to analyze notions of context-dependent computation such as dataflow computation, attribute grammars, tree transduction and cellular automata. Uustalu and Vene's [22] observation of a connection between bottom-up tree relabellings and containers with extra structure started our investigation into directed containers.

7 Conclusions and Future Work

We introduced directed containers as a specialization of containers for describing a certain class of datatypes (datastructures where every position determines a sub-datastructure) that occur very naturally in programming. It was a pleasant discovery for us that directed containers are an entirely natural concept also from the mathematical point of view: they are the same as containers whose interpretation carries the structure of a comonad.

In this paper, we could not discuss the equivalents of distributive laws between comonads, the composition of comonads, strict comonads and the product of (strict) comonads in the directed container world. We have already done some work around these concepts and constructions and plan to report our results in an extended version of this paper and elsewhere.

Acknowledgments. We are indebted to Thorsten Altenkirch, Jeremy Gibbons, Peter Morris, and Sam Staton for comments and suggestions. This work was supported by the European Regional Development Fund (ERDF) through Estonian Centre of Excellence in Computer Science (EXCS) project.

References

- [1] Abbott, M., Altenkirch, T., Ghani, N.: Containers: Constructing strictly positive types. *Theor. Comput. Sci.* 342(1), 3–27 (2005)
- [2] Abbott, M., Altenkirch, T., Ghani, N., McBride, C.: Constructing Polymorphic Programs with Quotient Types. In: Kozen, D. (ed.) *MPC 2004*. LNCS, vol. 3125, pp. 2–15. Springer, Heidelberg (2004)
- [3] Abbott, M., Altenkirch, T., Ghani, N., McBride, C.: δ for data: Differentiating data structures. *Fund. Inform.* 65(1-2), 1–28 (2005)
- [4] Abbott, M.: *Categories of Containers*. Ph.D. thesis, University of Leicester (2003)
- [5] Altenkirch, T., Morris, P.: Indexed containers. In: *Proc. of 24th Ann. IEEE Symp. on Logic in Computer Science, LICS 2009*, pp. 277–285. IEEE CS Press (2009)

- [6] Brookes, S., Geva, S.: Computational comonads and intensional semantics. In: Fourman, M.P., Johnstone, P.T., Pitts, A.M. (eds.) *Applications of Categories in Computer Science*, London. Math. Society Lect. Note Series, vol. 77, pp. 1–44. Cambridge Univ. Press (1992)
- [7] Capobianco, S., Uustalu, T.: A categorical outlook on cellular automata. In: Kari, J. (ed.) *Proc. of 2nd Symp. on Cellular Automata*, JAC 2010. TUCS Lecture Note Series, vol. 13, pp. 88–89. Turku Centre for Comput. Sci. (2011)
- [8] Dybjer, P.: Representing inductively defined sets by wellorderings in Martin-Löf’s type theory. *Theor. Comput. Sci.* 176(1-2), 329–335 (1997)
- [9] Gambino, N., Hyland, M.: Wellfounded Trees and Dependent Polynomial Functors. In: Berardi, S., Coppo, M., Damiani, F. (eds.) *TYPES 2003*. LNCS, vol. 3085, pp. 210–225. Springer, Heidelberg (2004)
- [10] Gambino, N., Kock, J.: Polynomial functors and polynomial monads. Tech. Rep. 867, Centre de Recerca Matemàtica, Barcelona (2009)
- [11] Girard, J.Y.: Normal functors, power series and lambda-calculus. *Ann. of Pure and Appl. Logic* 37(2), 129–177 (1988)
- [12] Hasuo, I., Jacobs, B., Uustalu, T.: Categorical Views on Computations on Trees. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *ICALP 2007*. LNCS, vol. 4596, pp. 619–630. Springer, Heidelberg (2007)
- [13] Huet, G.: The zipper. *J. of Funct. Program.* 7, 549–554 (1997)
- [14] Joyal, A.: Foncteurs analytiques et espèces de structures. In: Labelle, G., Leroux, P. (eds.) *Combinatoire énumérative*. Lect. Notes in Math., vol. 1234, pp. 126–159. Springer, Heidelberg (1987)
- [15] Kock, J.: Polynomial functors and trees. *Int. Math. Research Notices* 2011(3), 609–673 (2011)
- [16] Moerdijk, I., Palmgren, E.: Wellfounded trees in categories. *Ann. of Pure and Appl. Logic* 104(1-3), 189–218 (2000)
- [17] Morris, P.: *Constructing Universes for Generic Programming*. Ph.D. thesis, University of Nottingham (2007)
- [18] Norell, U.: *Towards a Practical Programming Language Based on Dependent type Theory*. Ph.D. thesis, Chalmers University of Technology (2007)
- [19] Prince, R., Ghani, N., McBride, C.: Proving Properties about Lists using Containers. In: Garrigue, J., Hermenegildo, M. (eds.) *FLOPS 2008*. LNCS, vol. 4989, pp. 97–112. Springer, Heidelberg (2008)
- [20] Uustalu, T., Vene, V.: The Essence of Dataflow Programming. In: Yi, K. (ed.) *APLAS 2005*. LNCS, vol. 3780, pp. 2–18. Springer, Heidelberg (2005)
- [21] Uustalu, T., Vene, V.: Attribute evaluation is comonadic. In: van Eekelen, M. (ed.) *Trends in Functional Programming*, vol. 6, pp. 145–162. Intellect (2007)
- [22] Uustalu, T., Vene, V.: Comonadic notions of computation. In: Adámek, J., Kupke, C. (eds.) *Proc. of 9th Int. Wksh. on Coalgebraic Methods in Computer Science*, CMCS 2008. *Electron. Notes in Theor. Comput. Sci.*, vol. 203(5), pp. 263–284. Elsevier (2008)

Well-Pointed Coalgebras (Extended Abstract)*

Jiří Adámek¹, Stefan Milius¹, Lawrence S. Moss², and Lurdes Sousa^{3,**}

¹ Institut für Theoretische Informatik,
Technische Universität Braunschweig, Germany
adamek@iti.cs.tu-bs.de, mail@stefan-milius.eu

² Department of Mathematics, Indiana University, Bloomington, IN, USA
lsm@indiana.edu

³ Departamento de Matemática, Instituto Politécnico de Viseu, Portugal
sousa@mat.estv.ipv.pt

Abstract. For set functors preserving intersections, a new description of the final coalgebra and the initial algebra is presented: the former consists of all well-pointed coalgebras. These are the pointed coalgebras having no proper subobject and no proper quotient. And the initial algebra consists of all well-pointed coalgebras that are well-founded in the sense of Taylor [16]. Finally, the initial iterative algebra consists of all finite well-pointed coalgebras. Numerous examples are discussed e.g. automata, graphs, and labeled transition systems.

1 Introduction

Initial algebras are known to be of primary interest in denotational semantics, where abstract data types are often presented as initial algebras for an endofunctor H expressing the type of the constructor operations of the data type. For example, finite binary trees are the initial algebra for the functor $HX = X \times X + 1$ on sets. Analogously, final coalgebras for an endofunctor H play an important role in the theory of systems developed by Rutten [13]: H expresses the system type, i.e., which kind of one-step reactions states can exhibit (input, output, state transitions etc.), and the elements of a final coalgebra represent the behavior of all states in all systems of type H (and the unique homomorphism from a system into the final one assigns to every state its behavior). For example, deterministic automata with input alphabet I are coalgebras for $HX = X^I \times \{0, 1\}$, the final coalgebra is the set of all languages on I .

In this paper a unified description is presented for (a) initial algebras, (b) final coalgebras and (c) initial iterative algebras (in the automata example this is the set of all regular languages on I). We also demonstrate that this new description

* The full version containing all proofs is available at <http://www.iti.cs.tu-bs.de/ITI-INFO/milius/research/wellS.full.pdf>

** Financial support by the Centre for Mathematics of the University of Coimbra is acknowledged.

provides a unifying view of a number of other important examples. We work with set functors H preserving intersections. This is a requirement that many “everyday” set functors satisfy. We prove that the final coalgebra of H can then be described as the set of all *well-pointed coalgebras*, i.e., pointed coalgebras not having any proper subobject and also not having any proper quotient. Moreover, the initial algebra can be described as the set of all well-pointed coalgebras which are well-founded in the sense of Taylor [16]. A coalgebra (A, α) is *well-founded* if no proper subcoalgebra (A', α') of (A, α) forms a pullback

$$\begin{array}{ccc}
 A' & \xrightarrow{\alpha'} & HA' \\
 \downarrow m & & \downarrow Hm \\
 A & \xrightarrow{\alpha} & HA
 \end{array} \tag{1.1}$$

This concept was first studied by Osius [12] for graphs considered as coalgebras of the power-set functor \mathcal{P} : a graph is well-founded iff it has no infinite paths. Taylor [16,17] introduced well-founded coalgebras for general endofunctors, and he proved that for endofunctors preserving inverse images the concepts of initial algebra and final well-founded coalgebra coincide.

We are going to prove that this result holds for every set functor H ; the step towards making no assumptions on H is non-trivial. And if H preserves intersections, we describe its final coalgebra, initial algebra, and initial iterative algebra using well-pointed coalgebras as above. The first result will be proved in a much more general context, working with an endofunctor of a locally finitely presentable category preserving finite intersections, but this extra assumption can be dropped in the case of set functors.

2 Well-Founded Coalgebras

Throughout this section \mathcal{A} denotes an LFP category with a simple initial object. And H is an endofunctor preserving monomorphisms. Let us recall these concepts:

- Definition 2.1.** 1. A category \mathcal{A} is *locally finitely presentable (LFP)* if
- (a) \mathcal{A} is complete, and
 - (b) there is a set of finitely presentable objects whose closure under filtered colimits is all of \mathcal{A} .
2. An object A is called **simple** if it has no proper quotients. That is, every epimorphism with domain A is invertible.

Example 2.2. The categories of sets, graphs, posets, and semigroups are locally finitely presentable. The initial objects of these categories are empty, hence simple. The category of rings is LFP but the initial object \mathbb{Z} is not simple.

Notation 2.3. For every endofunctor H denote by **Coalg** H the category of coalgebras $\alpha: A \longrightarrow HA$ and coalgebra homomorphisms, where a homomorphism h from (A, α) to (B, β) is a morphism $h: A \longrightarrow B$ such that $\beta \cdot h = Hh \cdot \alpha$.

Remark 2.4. There are some consequences of the LFP assumption that play an important role in our development. These pertain to *monomorphisms*.

1. \mathcal{A} has (strong epi, mono)-factorizations; see 1.16 in [5]. Recall that an epimorphism is *strong* iff it has the diagonal fill-in property w. r. t. all monomorphisms.
2. \mathcal{A} is wellpowered, see 1.56 in [5]. This implies that for every object A the poset $\mathbf{Sub}(A)$ of all subobjects of A is a complete lattice.
3. Monomorphisms are closed under filtered colimits; see 1.62 in [5].

Since subcoalgebras play a basic role in the whole paper, and quotients are important from Section 3 onwards, we need to make clear what we mean by those. We use the term *subcoalgebra* of a coalgebra (A, α) to mean a subobject $m: (A', \alpha') \longrightarrow (A, \alpha)$ represented by a monomorphism m in \mathcal{A} . Then m is clearly a monomorphism of $\mathbf{Coalg} H$; however, in general, monomorphisms in $\mathbf{Coalg} H$ need not be carried by monomorphisms from \mathcal{A} . As usual, if a subcoalgebra m is not invertible, it is said to be *proper*. What about quotient coalgebras? A *quotient* of a coalgebra (A, α) is represented by $e: (A, \alpha) \longrightarrow (A', \alpha')$, where e is a strong epimorphism in \mathcal{A} . Since H is assumed to preserve monomorphisms, $\mathbf{Coalg} H$ has factorizations of morphisms $f: (A, \alpha) \longrightarrow (B, \beta)$ into homomorphisms $e: (A, \alpha) \longrightarrow (C, \gamma)$ and $m: (C, \gamma) \longrightarrow (B, \beta)$, i. e., such that (C, γ) is a quotient of (A, α) and a subcoalgebra of (B, β) .

Definition 2.5. A *cartesian subcoalgebra* of a coalgebra (A, α) is a subcoalgebra (A', α') forming a pullback (1.1). A coalgebra is called **well-founded** if it has no proper cartesian subcoalgebra.

- Example 2.6.* (1) The concept of well-founded coalgebra was introduced originally by Osius [12] for the power set functor \mathcal{P} . A graph is a coalgebra (A, a) for \mathcal{P} , where $a(x)$ is the set of neighbors of x in the graph. Then a subcoalgebra of A is an (induced) subgraph A' with the property that every neighbor of a vertex of A' lies in A' . The subgraph A' is cartesian iff it contains every vertex all of whose neighbors lie in A' . The graph A is a well-founded coalgebra iff it has no infinite path.
- (2) Let A be a deterministic automaton considered as a coalgebra for $HX = X^I \times \{0, 1\}$. A subcoalgebra A' is cartesian iff it contains every state all whose successors (under the inputs from I) lie in A' . This holds, in particular, for $A' = \emptyset$. Thus, no nonempty automaton is well-founded.
- (3) Coalgebras for $HX = X + 1$ are dynamical systems with deadlocks, see [13]. A subcoalgebra A' of a dynamical system A is cartesian iff it contains all deadlocks and every state whose next state lies in A' . A dynamical system is well-founded iff it has no infinite computation.

Definition 2.7. Every coalgebra $\alpha: A \longrightarrow HA$ induces an endofunctor of $\mathbf{Sub}(A)$ (see Remark 2.4.2) assigning to a subobject $m: A' \longrightarrow A$ the inverse image $\bigcirc m$ of Hm under α , i. e., we have a pullback square:

$$\begin{array}{ccc}
 \bigcirc A' & \xrightarrow{\alpha[m]} & HA' \\
 \downarrow \text{\scriptsize } \bigcirc m & \lrcorner & \downarrow Hm \\
 A & \xrightarrow{\alpha} & HA
 \end{array} \tag{2.1}$$

This function $m \mapsto \bigcirc m$ is obviously order-preserving. By the Knaster-Tarski fixed point theorem, this function has a least fixed point.

Incidentally, the notation $\bigcirc m$ comes from modal logic, especially the areas of temporal logic where one reads $\bigcirc\phi$ as “ ϕ is true in the next moment,” or “next time ϕ ” for short.

Example 2.8. Recall our discussion of graphs from Example 2.6 (1). The pullback $\bigcirc A'$ of a subgraph A' is the set of vertices of A all of whose neighbors belong to A' .

Remark 2.9. As we mentioned in the introduction, the concept of well-founded coalgebra was introduced by Taylor [16,17]. Our formulation is a bit simpler. In [17, Definition 6.3.2] he calls a coalgebra (A, α) well-founded if for every pair of monomorphisms $m: U \rightarrow A$ and $h: H \rightarrow U$ such that $h \cdot m$ is the inverse image of Hm under α it follows that m is an isomorphism. Thus in lieu of fixed points of $m \mapsto \bigcirc m$ he uses pre-fixed points.

In addition, our overall work has a *methodological* difference from Taylor’s that is worth mentioning at this point. Taylor is giving a general account of recursion and induction, and so he is concerned with general principles that underlie these phenomena. Indeed, he is interested in settings like non-boolean toposes where classical reasoning is not necessarily valid. On the other hand, in this paper we are studying initial algebras, final coalgebras, and similar concepts, using standard classical mathematical reasoning. In particular, we make free use of transfinite recursion. The definitions in Notation 2.10 just below would look out of place in Taylor’s paper. But we believe they are an important step in our development.

Notation 2.10. (a) For every coalgebra $\alpha: A \rightarrow HA$ denote by

$$a^*: A^* \rightarrow A \tag{2.2}$$

the least fixed point of the function $m \mapsto \bigcirc m$ of Definition 2.7. (Thus, (A, a) is well-founded iff a^* is invertible.) Since a^* is a fixed point we have a coalgebra structure $\alpha^*: A^* \rightarrow HA^*$ making a^* a coalgebra homomorphism.

(b) For every coalgebra $a: A \rightarrow HA$ we define a chain of subobjects

$$a_i^*: A_i^* \rightarrow A \quad (i \in \mathbf{Ord})$$

of A in \mathcal{A} by transfinite recursion: $a_0^*: 0 \rightarrow A$ is the unique morphism; $a_{i+1}^* = \bigcirc a_i^*$ and for limit ordinals $a_i^* = \bigcup_{j < i} a_j^*$. Since 0 is simple, a_0^* is a

monomorphisms. Moreover, what we have is nothing else than the construction of the least fixed point of $m \mapsto \bigcirc m$ (cf. Remark 2.9) in the proof of the Knaster-Tarski Theorem in [15]. Thus, $a^* = \bigcup_{i \in \text{Ord}} a_i^*$. Also, there exists an ordinal i with $A^* = A_i^*$ (due to wellpoweredness). Henceforth, we call A^* the *smallest cartesian subcoalgebra* of A .

Proposition 2.11. *For every coalgebra (A, α) , the smallest cartesian subcoalgebra (A^*, α^*) is its coreflection in the full subcategory of well-founded coalgebras.*

Remark. We thus prove that (A^*, α^*) is well-founded, and for every homomorphism $f: (B, \beta) \longrightarrow (A, \alpha)$ with (B, β) well-founded there exists a unique homomorphism $\bar{f}: (B, \beta) \longrightarrow (A^*, \alpha^*)$ with $f = a^* \cdot \bar{f}$.

Corollary 2.12. *The full subcategory of $\mathbf{Coalg} H$ consisting of the well-founded coalgebras is closed under quotients and colimits in $\mathbf{Coalg} H$.*

For endofunctors preserving inverse images the above corollary is Exercise VI.16 in Taylor [17] and the following theorem is Corollary 9.9 of [16]. As we mentioned in the introduction, it is non-trivial to relax the assumption on the endofunctor, and so our proof is different from Taylor's.

Theorem 2.13. *If H preserves finite intersections, then*

$$\text{initial algebra} = \text{final well-founded coalgebra}.$$

That is, an algebra $\varphi: HI \longrightarrow I$ is initial iff $\varphi^{-1}: I \longrightarrow HI$ is the final well-founded coalgebra.

Proof (Sketch). (a) Let I be an initial algebra. It follows from [20] that I is obtained as $H^i 0$ for some ordinal i for the initial chain introduced in [2] defined by $H^0 0 = 0$, $H^{i+1} 0 = H(H^i 0)$ and $H^i 0 = \text{colim}_{j < i} H^j 0$ for limit ordinals i . We prove by transfinite induction that if $I = H^k 0$ then the connecting morphisms $H^i 0 \longrightarrow H^k 0$ for $i \leq k$ are precisely a_i^* of Notation 2.10. Consequently, I is well-founded. We next use the concept of recursive coalgebra of Capretta et al [6]: It is a coalgebra from which a unique coalgebra-to-algebra morphism into every algebra exists. Initial algebras are proved there to be precisely the final recursive coalgebras. We prove that every well-founded coalgebra is recursive. We thus derive that I is a final well-founded coalgebra.

(b) Let $\psi: I \longrightarrow HI$ be a final well-founded coalgebra. Factorize $\psi = m \cdot e$ where e is a strong epimorphism and m a monomorphism (Remark 2.4). By diagonal fill-in we obtain a quotient $e: (I, \psi) \longrightarrow (I', \psi')$ which, by Corollary 2.12, is well-founded, thus recursive. Consequently, a coalgebra homomorphism $f: (I', \psi') \longrightarrow (I, \psi)$ exists. Then $f \cdot e$ is an endomorphism of the final well-founded coalgebra, hence, $f \cdot e = \text{id}_I$. This proves that e is an isomorphism, thus, ψ is a monomorphism. This fact is used to prove that the coalgebra $(HI, H\psi)$ is well-founded. Using an argument similar to Lambek's Lemma we derive that ψ is invertible. Therefore results of [20] imply that the initial chain

above converges, and for some ordinal k , $H^k 0$ is an initial algebra. Moreover, $H^k 0$ is by (a) a final well-founded coalgebra, thus, isomorphic to $\psi: I \longrightarrow HI$. Thus (I, ψ^{-1}) is isomorphic to the initial algebra. \square

Theorem 2.14. *For every endofunctor of **Set** we have:*

$$\text{initial algebra} = \text{final well-founded coalgebra}.$$

Proof (Sketch). There exists an endofunctor H^* preserving finite intersections and agreeing on nonempty sets with H , see [19]. Given H , we know from Theorem 2.13 that the equation above holds for H^* . From this one can prove it for H . The proof is quite technical because we need to compare well-foundedness of coalgebras for H and H^* , and the empty set plays a substantial role here. \square

This last result and Corollary 2.12 serve as a basis for a description of initial algebras in Theorem 3.15.

3 Well-Pointed Coalgebras

We arrive at the centerpiece of this paper, characterizations of the initial algebra, final coalgebra, and initial iterative algebra for set functors.

Throughout this section H denotes an endofunctor of **Set** which preserves (wide) intersections. Many endofunctors of interest satisfy this condition, for example:

- (a) the power-set functor, all polynomial functors, the finite distribution functor,
- (b) products, coproducts, quotients, and subfunctors of functors preserving intersections, and
- (c) “almost” all finitary functors: if H is finitary then H^* in Theorem 2.14 preserves intersections.

An example of a set functor not preserving intersections is the continuation monad $HX = R^{(R^X)}$, where R is the set of results. A simpler example is the one taking every nonempty set to the terminal object and the empty set to itself.

By a *pointed coalgebra* is meant a triple (A, a, x) , where (A, a) is a coalgebra and x an element of A called *initial state*. When speaking about morphisms between pointed coalgebras we mean those preserving the initial state. In particular, given a pointed coalgebra $1 \xrightarrow{x} A \longrightarrow HA$ by a *subobject* is meant a subcoalgebra containing the initial state x .

Definition 3.1. *A well-pointed coalgebra is a pointed coalgebra which has no proper subobjects and no proper quotients.*

Remark 3.2. Recall that a *simple coalgebra* (called *minimal coalgebra* by Gumm [8]) is a coalgebra (A, a) with no nontrivial quotient. That is, a coalgebra such that every homomorphism $h: (A, a) \longrightarrow (B, b)$ has h monic. Gumm observed that

- (a) The full subcategory of **Coalg** H given by all simple coalgebras is reflective: the reflection of a coalgebra (A, a) is the simple quotient

$$e_{(A,a)}: (A, a) \longrightarrow (\bar{A}, \bar{a})$$

obtained as the wide pushout of all quotients of (A, a) .

- (b) Every subcoalgebra of a simple coalgebra is simple.
- (c) The coalgebra map $a: A \longrightarrow HA$ of a simple coalgebra is monic.

Remark 3.3. Thus, $1 \xrightarrow{x} A \xrightarrow{a} HA$ is a well-pointed coalgebra iff (A, a) is simple and is generated by x . We call the latter condition *reachability*. That is, a pointed coalgebra is *reachable* if it has no proper pointed subcoalgebra. It is easy to see that this holds iff the canonical graph (see Definition 3.11 below) is reachable: every state has a directed path from the initial state.

Examples 3.4. (a) A deterministic automaton with a given initial state is a pointed coalgebra for $HX = X^I \times \{0, 1\}$. Reachability means that every state can be reached (in finitely many steps) from the initial state. Simplicity means that the automaton is *observable*, i.e., for every pair of different states there exists an input word leading one of them to an accepting state and the other to a non-accepting state. The usual terminology is that reachability and observability together are called *minimality*.

- (b) For the power-set functor the pointed coalgebras are the pointed graphs. Well-pointed means reachable and simple, where simplicity states that no pair of different vertices is bisimilar.

Notation 3.5. Since H preserves intersections, there is a canonical process of turning an arbitrary pointed coalgebra (A, a, x) into a well-pointed one: form the simple quotient, see Remark 3.2(a) pointed by $e_{(A,a)} \cdot x: 1 \longrightarrow \bar{A}$, then form the least subcoalgebra containing that point:

$$\begin{array}{ccccc}
 & & \bar{A}_0 & \xrightarrow{\bar{a}_0} & H\bar{A}_0 \\
 & \nearrow x_0 & \downarrow m & & \downarrow Hm \\
 1 & \xrightarrow{x} & A & \xrightarrow{e_{(A,a)}} & \bar{A} & \xrightarrow{\bar{a}} & H\bar{A}
 \end{array}$$

That is, m is the intersection of all subcoalgebras of (\bar{A}, \bar{a}) through which $e_{(A,a)} \cdot x$ factorizes. Then $(\bar{A}_0, \bar{a}_0, x_0)$ is well-pointed due to Remark 3.2(b).

Example 3.6. For deterministic automata our process $A \longmapsto \bar{A}_0$ above means that we first merge the states that are observably equivalent and then discard the states that are not reachable. A more efficient way is first discarding the unreachable states and then merging observably equivalent pairs. Both ways are possible since our functor preserves inverse images: this implies that a quotient of a reachable pointed coalgebra is reachable.

Notation 3.7. The collection of all well-pointed coalgebras up to isomorphism is denoted by

$$\nu H.$$

For every coalgebra $a : A \longrightarrow HA$ we have a function $a^+ : A \longrightarrow \nu H$ assigning to every element $x : 1 \longrightarrow A$ the well-pointed coalgebra of Notation 3.5:

$$a^+(x) = (\bar{A}_0, \bar{a}_0, x_0). \tag{3.1}$$

Theorem 3.8. *A set functor H preserving intersections has a final coalgebra iff it has only a set of well-pointed coalgebras up to isomorphism. And, if it is the case, νH is a final coalgebra.*

Remark. Whenever νH is a set, it carries a canonical coalgebra structure $\psi : \nu H \longrightarrow H(\nu H)$. It assigns to every member (A, a, x) of νH the following element of $H(\nu H)$:

$$1 \xrightarrow{x} A \xrightarrow{a} HA \xrightarrow{Ha^+} H(\nu H). \tag{3.2}$$

We prove below that this is a final coalgebra.

Proof. (1) If H has a final coalgebra, then due to Remark 3.2 every simple coalgebra is its subcoalgebra, since the unique homomorphism is monic. The final coalgebra has only a set of subcoalgebras, consequently, there exists up to isomorphism only a set of simple coalgebras. Consequently, only a set of well-pointed coalgebras.

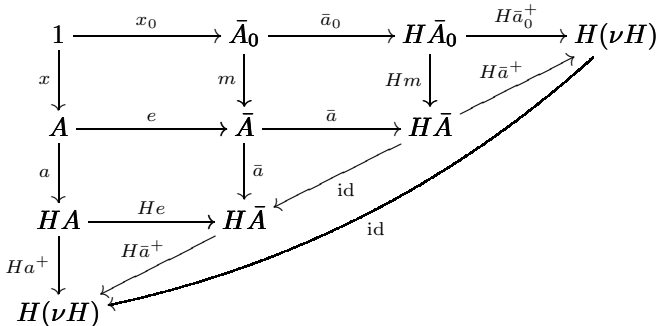
(2) Let H have a set νH of representative well-pointed coalgebras. We prove that νH with the coalgebra structure ψ from (3.2) is final.

(2a) We first prove that for every coalgebra homomorphism $h : (A, a) \longrightarrow (B, b)$ we have

$$a^+ = b^+ \cdot h. \tag{3.3}$$

Given $x : 1 \longrightarrow A$, then $b^+ \cdot h$ assigns to it the well-pointed coalgebra $(\bar{B}_0, \bar{b}_0, y_0)$ obtained from (B, b, y) , where $y = h \cdot x$, as in Notation 3.5. It is not difficult, using Remark 3.2, to prove that this well-pointed coalgebra is isomorphic to $(\bar{A}_0, \bar{a}_0, x)$.

(2b) νH is a weakly final coalgebra because for every coalgebra (A, a) we have a coalgebra homomorphism $a^+ : (A, a) \longrightarrow (\nu H, \psi)$. Indeed, by 3.2 we have $\psi \cdot a^+(x) = H\bar{a}_0^+ \cdot \bar{a}_0(x_0)$ and the diagram below shows that this is equal to $Ha^+ \cdot a(x)$:



Notice that the upper and lower triangles commute since m and e are homomorphisms, see (2a).

(2c) We next prove that for the coalgebra $\psi: \nu H \longrightarrow H(\nu H)$ we have $\psi^+ = \text{id}_{\nu H}$. Indeed, given a well-pointed coalgebra $(A, a, x) \in \nu H$, consider the equality (3.3) with $h = a^+$ (which is a homomorphism by (2b)) and $b = \psi$. Of course, $a^+(x) = (A, a, x)$, since (A, a) is simple and (A, a, x) is reachable. Then $\psi^+(A, a, x) = (A, a, x)$.

Finally, to prove uniqueness of the homomorphism a^+ , suppose that $h: (A, a) \longrightarrow (\nu H, \psi)$ is any homomorphism. Then we have

$$a^+ \stackrel{(2a)}{=} \psi^+ \cdot h \stackrel{(2b)}{=} h. \quad \square$$

Examples 3.9. (a) For deterministic automata the final coalgebra (for $HX = X^I \times \{0, 1\}$) consists of all minimal (i.e., reachable and observable) automata. The more usual description is: the set $\mathcal{P}I^*$ of all formal languages. However, this is isomorphic: every formal language is accepted by a minimal automaton, unique up to isomorphism.

(b) The final coalgebra for the finite power-set functor is the coalgebra of all finitely branching well-pointed graphs. See Section 4 for more details.

Remark 3.10. If νH is not a set, then H does not have a (small) final coalgebra. However, νH is its large final coalgebra: in the above proof smallness was not used.

Definition 3.11. For every coalgebra $a: A \longrightarrow HA$ define the **canonical graph** on A : the neighbors of $x \in A$ are precisely those elements of A which lie in the least subset $m: M \hookrightarrow A$ with $a(x) \in Hm[HM]$.

Proposition 3.12. A coalgebra for H is well-founded iff its canonical graph is well-founded.

Remark. For functors H preserving inverse images this fact is proved by Taylor, see 6.3.4 in [17]. Our proof is essentially the same.

Corollary 3.13. Subcoalgebras of a well-founded coalgebra are well-founded.

Notation 3.14. The collection of all well-founded, well-pointed coalgebras (up to isomorphism) is denoted by

$$\mu H.$$

For every well-founded coalgebra $a: A \longrightarrow HA$ we have a function $a^+: A \longrightarrow \mu H$ assigning to every element $x: 1 \longrightarrow A$ the well-founded, well-pointed coalgebra (3.1). Indeed, (\bar{A}_0, \bar{a}_0) is well-founded due to Corollaries 2.12 and 3.13.

Theorem 3.15. A set functor H preserving intersections has an initial algebra iff it has only a set of well-founded, well-pointed coalgebras up to isomorphism. And, if it is the case, μH is an initial algebra.

Remark. Whenever μH is a set, it carries a canonical coalgebra structure $\bar{\psi}: \mu H \longrightarrow H(\mu H)$ defined by (3.2). We prove below that this is a final well-founded coalgebra. Thus, by Theorem 2.14, μH is an initial algebra with the structure given by the inverse of $\bar{\psi}$.

Proof. (1) If H has an initial algebra I , then by Theorem 2.14 this is a final well-founded coalgebra. Every well-founded, well-pointed coalgebra is simple, whence a subcoalgebra of I since the unique homomorphism into I is monomorphic by Remark 3.2. Consequently, μH is a set.

(2) Let H have a set μH of representatives of well-founded, well-pointed coalgebras. The proof that for every well-founded coalgebra (A, a) the map $a^+ : A \longrightarrow \mu H$ is a unique coalgebra homomorphism into $\bar{\psi}: \mu H \longrightarrow H(\mu H)$ is completely analogous to the proof of finality of $\psi: \nu H \longrightarrow H(\nu H)$ in Theorem 3.8. Just recall that subcoalgebras and quotients of a well-founded coalgebra are all well-founded (by Corollaries 2.12 and 3.13).

It remains to prove that $(\mu H, \bar{\psi})$ is a well-founded coalgebra. To this end notice that for every well-pointed, well-founded coalgebra (A, a, x) in μH we have $a^+(x) = (A, a, x)$. Now take the coproduct (in **Coalg** H) of all (A, a) for which there is an $x \in A$ such that (A, a, x) lies in μH . This coproduct is a well-founded coalgebra by Corollary 2.12, and, as we have just seen, the unique induced homomorphism from the coproduct into $(\mu H, \bar{\psi})$ is epimorphic, whence μH is a quotient coalgebra of the coproduct. Thus, another application of Corollary 2.12 shows that μH is a well-founded coalgebra as desired. \square

Remark 3.16. (a) Recall from [4] that an algebra $a: HA \longrightarrow A$ is *iterative* provided that every (equation) morphism $e: X \longrightarrow HX + A$, where X is a finite set, has a unique solution, i.e., $e^\dagger: X \longrightarrow A$ such that $e^\dagger = [a, A] \cdot (He^\dagger + A) \cdot e$. It was proved in [10] that the initial iterative algebra is precisely the final locally finite coalgebra, where a coalgebra is called *locally finite* if every element of it lies in a finite subcoalgebra.

Example 3.17 (see [4]). The initial iterative algebra for $HX = X^I \times \{0, 1\}$ consists of all finite minimal automata. This is isomorphic to its description as all regular languages.

Notation 3.18. For every finitary set functor denote by

$$\varrho H$$

the set of all finite well-pointed coalgebras up to isomorphism.

Given a finite coalgebra $a: A \longrightarrow HA$ we define a function $a^+ : A \longrightarrow \varrho H$ by (3.1).

Theorem 3.19. *Every finitary set functor H has an initial iterative algebra ϱH formed by all finite well-pointed coalgebras.*

Remark. ϱH has the canonical coalgebra structure $\tilde{\psi}: \varrho H \longrightarrow H(\varrho H)$ given by (3.2). The proof that this is the final locally finite coalgebra is analogous to the proof of Theorem 3.8.

4 Examples of Well-Pointed Coalgebras

Example 4.1. Deterministic automata, $HX = X^I \times \{0, 1\}$. In Example 3.9 we saw that νH consists of all minimal automata, or, equivalently, all languages over I . The initial iterative algebra ρH consists of all finite minimal automata, this is isomorphic to

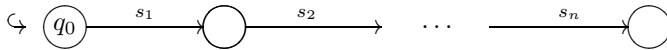
$$\rho H = \text{all regular languages.}$$

Finally, no well-pointed coalgebra is well-founded because the empty subcoalgebra is cartesian, thus,

$$\mu H = \emptyset.$$

Example 4.2. Streams. Consider the coalgebras for $HX = X \times I + 1$. Jan Rutten [13] interprets them as dynamical systems with outputs in I and with terminating states (where no next state is given). Every state q yields a stream, finite or infinite, over I by starting in q and traversing the dynamic system as long as possible. We call it the *response* of q . It is an element of $I^\omega + I^*$.

(a) For every word $s_1 \cdots s_n$ in I^* we have a well-pointed dynamic system



(b) For every *eventually periodic* stream in I^ω ,

$$w = uv^\omega \quad \text{for } u, v \in I^*,$$

we have a well-pointed dynamic system which uses u as in (a) and adds a cycle repeating v .

The following was already proved by Arbib and Manes [9, 10.2.5].

Corollary 4.3. *For $HX = X \times I + 1$ we have*

$$\nu H \cong I^* + I^\omega, \quad \text{all finite and infinite streams,}$$

$$\rho H \cong \text{all finite and eventually periodic streams,}$$

$$\mu H \cong I^*, \quad \text{all finite streams.}$$

Example 4.4. Binary trees. Coalgebras for the functor

$$HX = X \times X + 1$$

are given, as observed by Jan Rutten [13], by a set Q of states which are either terminating or have precisely two next states according to a binary input, say $\{l, r\}$. Every state $q \in Q$ yields an ordered binary tree T_q (i.e. nodes that are not leaves have a left-hand child and a right-hand one) by *tree expansion*: the root is q and a node is either a leaf, if it is a terminating state, or has the two next states as children (left-hand for input l , right-hand for input r). Binary trees are considered up to isomorphism.

Lemma 4.5. *For every coalgebra of the functor $HX = X \times X + 1$ the largest congruence merges precisely the pairs of states having the same tree expansion.*

Proof. Let \sim be the equivalence with $q \sim q'$ iff $T_q = T_{q'}$. There is an obvious structure of a coalgebra on Q/\sim showing that \sim is a congruence. For every coalgebra homomorphism $h: Q \rightarrow \bar{Q}$ the tree expansion of $q \in Q$ is always the same as the tree expansion of $h(q)$ in \bar{Q} . Thus, \sim is the largest congruence. \square

Corollary 4.6. *A well-pointed coalgebra of the functor $HX = X \times X + 1$ is a coalgebra with an initial state q_0 which is reachable (every state can be reached from q_0) and simple (different states have different tree expansions).*

Moreover, tree expansion of the initial state is a bijection between well-pointed coalgebras and binary trees. The coalgebra is finite iff the tree expansion is *rational*, i.e., it has only finitely many subtrees up to isomorphism. And the well-founded coalgebras are precisely those yielding a finite tree expansion.

The following result was proved by Arbib and Manes [9], 10.2.5 (description of νH) and in [4] (description of ρH).

Corollary 4.7. *For the functor $HX = X \times X + 1$ we have*

$$\begin{aligned} \nu H &\cong \text{all binary trees,} \\ \varrho H &\cong \text{all rational binary trees,} \\ \mu H &\cong \text{all finite binary trees.} \end{aligned}$$

Example 4.8. Graphs. Here we investigate coalgebras for the power-set functor \mathcal{P} (that is, graphs) and for the finitary power-set functor \mathcal{P}_ω (that is, finitely branching graphs). In the rest of Section 4 all trees are understood to be non-ordered. That is, a tree is a directed graph with a node (root) from which every node can be reached by a unique path.

Recall the concept of a *bisimulation* between graphs X and Y : it is a relation $R \subseteq X \times Y$ such that whenever $x R y$ then every child of x is related to a child of y , and vice versa. Two nodes of a graph X are called *bisimilar* if they are related by a bisimulation $R \subseteq X \times X$.

Lemma 4.9. *The greatest congruence on a graph merges precisely the bisimilar pairs of states.*

This follows, since \mathcal{P} preserves weak pullbacks, from general results of Rutten [13].

Corollary 4.10. *A pointed graph (G, q_0) is well-pointed iff it is reachable (every vertex can be reached from q_0 by a directed path) and simple (all distinct pairs of states are non-bisimilar).*

Example 4.11. Peter Aczel introduced in [1] the *canonical picture* of a (well-founded) set X . It is the graph with vertices all sets Y such that a sequence

$$Y = Y_0 \in Y_1 \in \dots \in Y_n = X$$

of sets exists. The neighbors of a vertex Y are all of its elements. When pointed by X , this is a well-pointed graph which is, due to the Foundation Axiom, well-founded. Conversely, every well-founded well-pointed graph is isomorphic to the canonical picture of a set.

Corollary 4.12. $\mu\mathcal{P} = \text{all sets}$.

This was proved by Rutten and Turi in [14]. The bijection between well-founded, well-pointed graphs and sets (given by the canonical picture) takes the finite well-founded graphs to the *hereditarily finite sets* X , i.e., finite sets with finite elements which also have finite elements, etc. More precisely: a set is hereditarily finite if all sets in the canonical picture of X are finite:

Corollary 4.13. $\mu\mathcal{P}_\omega = \text{all hereditarily finite sets}$.

In order to describe the final coalgebra for \mathcal{P} in a similar set-theoretic manner, we must move from the classical theory to the non-well-founded set theory of Peter Aczel [1]. Recall that a *decoration* of a graph is a coalgebra homomorphism from this graph into the large coalgebra (\mathbf{Set}, \in) . Non-well-founded set theory is obtained by swapping the axiom of foundation, telling us that (\mathbf{Set}, \in) is well-founded, with the following

Anti-Foundation Axiom. Every graph has a unique decoration.

Example 4.14. The decoration of a single loop is a set Ω such that $\Omega = \{\Omega\}$.

The coalgebra (\mathbf{Set}, \in) where now \mathbf{Set} is the class of all non-well-founded sets, is of course final: the decoration of G is the unique homomorphism $d: G \longrightarrow \mathbf{Set}$.

Corollary 4.15. *In the non-well-founded set theory: $\nu\mathcal{P} = \text{all sets}$.*

Let us turn to the finite power-set functor \mathcal{P}_ω .

Remark 4.16. Worrell introduced in [21] the notion of a *tree-bisimulation* between trees T_1 and T_2 ; this is a graph bisimulation $R \subseteq T_1 \times T_2$ which relates the roots and such that $x_1 R x_2$ implies that x_1 and x_2 are the roots or have related parents.

A tree T is called *strongly extensional* iff every tree bisimulation $R \subseteq T \times T$ is trivial: $R \subseteq \Delta_T$. The tree expansion is a bijection between all well-pointed finitely branching graphs and strongly extensional finitely branching trees.

Corollary 4.17. *For the finite power-set functor \mathcal{P}_ω we have*

$$\begin{aligned} \nu\mathcal{P}_\omega &= \text{all finitely branching, strongly extensional trees,} \\ \varrho\mathcal{P}_\omega &= \text{all finitely branching, rational, strongly extensional trees,} \\ \mu\mathcal{P}_\omega &= \text{all finite strongly extensional trees.} \end{aligned}$$

Example 4.18. Labeled transition systems. Here we consider, for a set A of actions, coalgebras for $\mathcal{P}_\omega(-\times A)$. A *bisimulation* between two finitely branching labeled transition systems (LTS) G and G' is a relation $R \subseteq G \times G'$ such that

if $x R y$ then for every transition $x \xrightarrow{a} x'$ in G there exists
a transition $y \xrightarrow{a} y'$ with $x' R y'$, and vice versa.

States x, y of an LTS are called *bisimilar* if $x R y$ for some bisimulation $R \subseteq G \times G$.

A well-pointed LTS is an LTS together with an initial state q_0 which is reachable (every state can be reached from q_0) and simple (distinct states are non-bisimilar).

The *tree expansion* of a state q is a (non-ordered) tree with edges labeled in A , shortly, an A -labeled tree. For A -labeled trees we modify Definition [4.16](#) in an obvious manner.

Corollary 4.19. *For the finitely branching LTS we have*

$\nu\mathcal{P}_\omega(-\times A)$ = all finitely branching, strongly extensional A -labeled trees,
 $\rho\mathcal{P}_\omega(-\times A)$ = all rational, finitely branching, strongly extensional
 A -labeled trees,
 $\mu\mathcal{P}_\omega(-\times A)$ = all finite extensional A -labeled trees.

5 Conclusions

For set functors H satisfying the (mild) assumption of preservation of intersections we described (a) the final coalgebra as the set of all well-pointed coalgebras, (b) the initial algebra as the set of all well-pointed coalgebras that are well-founded, and (c) in the case where H is finitary, the initial iterative algebra as the set of all finite well-pointed coalgebras. This is based on the observation that given an element of a final coalgebra, the subcoalgebra it generates has no proper subcoalgebras nor proper quotients—shortly, this subcoalgebra is well-pointed. And different elements define nonisomorphic well-pointed subcoalgebras. We then combined this with our result that for all set functors the initial algebra is precisely the final well-founded coalgebra. This resulted in the above description of the initial algebra. Numerous examples demonstrate that this view of final coalgebras and initial algebras is useful in applications.

Whereas our result about well-founded coalgebras was proved in locally finitely presentable categories, the description of the final coalgebra was formulated for set functors only. In future research we intend to generalize this result to a wider class of base categories.

References

1. Aczel, P.: Non-well-founded Sets. CSLI Lect. Notes, vol. 14. Stanford CSLI Publications, Stanford (1988)
2. Adámek, J.: Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae* 15, 589–602 (1974)
3. Adámek, J., Herrlich, H., Strecker, G.E.: *Abstract and Concrete Categories*. John Wiley and Sons, New York (1990)
4. Adámek, J., Milius, S., Velebil, J.: Iterative algebras at work. *Math. Structures Comput. Sci.* 16, 1085–1131 (2006)
5. Adámek, J., Rosický, J.: *Locally Presentable and Accessible Categories*. Cambridge University Press (1994)
6. Capretta, V., Uustalu, T., Vene, V.: Recursive coalgebras from comonads. *Inform. and Comput.* 204, 437–468 (2006)
7. Gabriel, P., Ulmer, F.: *Lokal präsentierbare Kategorien*. Lecture Notes in Math., vol. 221. Springer, Berlin (1971)
8. Gumm, H.-P.: On minimal coalgebras. *Appl. Categ. Structures* 16, 313–332 (2008)
9. Manes, E.G., Arbib, M.A.: *Algebraic Approaches to Program Semantics*. Springer, New York (1986)
10. Milius, S.: A sound and complete calculus for finite stream circuits. In: *Proc. 25th Annual Symposium on Logic in Computer Science (LICS 2010)*, pp. 449–458. IEEE Computer Society (2010)
11. Nelson, E.: Iterative algebras. *Theoret. Comput. Sci.* 25, 67–94 (1983)
12. Osius, G.: Categorical set theory: a characterization of the category of sets. *J. Pure Appl. Algebra* 4, 79–119 (1974)
13. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. *Theoret. Comput. Sci.* 249, 3–80 (2000)
14. Rutten, J.J.M.M., Turi, D.: On the Foundations of Final Semantics: Non-Standard Sets, Metric Spaces, Partial Orders. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) *REX 1992. LNCS*, vol. 666, pp. 477–530. Springer, Heidelberg (1993)
15. Tarski, A.: A lattice theoretical fixed point theorem and its applications. *Pacific J. Math.* 5, 285–309 (1955)
16. Taylor, P.: Towards a unified treatment of induction I: the general recursion theorem, preprint (1995–6), <http://www.paultaylor.eu/ordinals/#towuti>
17. Taylor, P.: *Practical Foundations of Mathematics*. Cambridge University Press (1999)
18. Tiurin, J.: Unique fixed points vs. least fixed points. *Theoret. Comput. Sci.* 12, 229–254 (1980)
19. Trnková, V.: On a descriptive classification of set functor I. *Comment. Math. Univ. Carolinae* 12, 323–352 (1971)
20. Trnková, V., Adámek, J., Koubek, V., Reiterman, J.: Free algebras, input processes and free monads. *Comment. Math. Univ. Carolinae* 16, 339–351 (1975)
21. Worrell, J.: On the final sequence of a finitary set functor. *Theoret. Comput. Sci.* 338, 184–199 (2005)

Combining Interactive and Automatic Reasoning in First Order Theories of Functional Programs

Ana Bove¹, Peter Dybjer¹, and Andrés Sicard-Ramírez^{2,*}

¹ Chalmers University of Technology, Gothenburg, Sweden

² EAFIT University, Medellín, Colombia

Abstract. We propose a new approach to the computer-assisted verification of functional programs. We work in first order theories of functional programs which are obtained by extending Aczel’s first order theory of combinatory formal arithmetic with positive inductive and coinductive predicates. Rather than building a special purpose system we implement our theories in Agda, a proof assistant for dependent type theory which can be used as a generic theorem prover. Agda provides support for interactive reasoning by encoding first order theories using the formulae-as-types principle. Further support is provided by off-the-shelf automatic theorem provers for first order logic which can be called by a program which translates Agda representations of first order formulae into the TPTP language understood by the provers. We show some examples where we combine interactive and automatic reasoning, covering both proof by induction and coinduction.

1 Introduction

The goal of this paper is to show a simple way to build a system for reasoning about programs in functional languages with higher order functions, general recursion and lazy evaluation in the style of Haskell [23]. Building a mature proof assistant from scratch for this purpose is a daunting task, although there are some attempts in this direction [15,20]. Here we suggest to achieve this goal by building on existing state-of-the-art systems in interactive and automatic theorem proving. Our solution combines the following three strands of research:

- Using a logic for *general recursive* functional programs [9,10,11] which is based on Aczel’s *first order theory of combinatory arithmetic* [3]; we extend this theory to deal in a seamless way with full general recursion, higher order functions, termination proofs, and inductive and coinductive predicates.
- Using *automatic theorem provers* for proving properties of functional programs by translating them into *first order logic* as proposed by Claessen and Hamon in their work on “The Cover Translator” (Chalmers, 2003).
- Using automatic theorem provers for first order logic for proof assistants based on *dependent type theory*, see Tammet and Smith’s Gandalf [27], and Abel, Coquand, and Norell’s AgdaLight [1].

* Part of this research was performed during a research visit to Chalmers University of Technology which was funded by the ALFA network LERnet and EAFIT University.

We use the Agda system [28] as our interactive theorem prover. It is simultaneously a dependently typed functional programming language and a proof assistant. It is an extension of Martin-Löf type theory with numerous programming language features which facilitate programming and interactive proof construction.

Like Martin-Löf type theory, Agda has the strong normalisation property. This property is ensured by only allowing restricted forms of recursion. A consequence is that one cannot write programs by arbitrary general recursion. It is the goal of the *dependently typed programming* community to turn this restricted discipline of programming into a practical methodology.

In this paper we directly verify mainstream general recursive functional programs. To this end we use Agda as a *logical framework* in much the same way as the Edinburgh logical framework [14], that is, as a meta-logical system which is used as a basis for the implementation of a range of special purpose logics. Our logic is a first order theory of combinators (FOTC) based on Aczel's theory [3]. When implementing FOTC in Agda we get access to advanced features for interactively building proofs in the proof assistant, such as, commands for refining proof terms, definition by pattern matching, flexible mixfix syntax accepting Unicode, etc.

Furthermore, we provide a translation of Agda representations of formulae in the FOTC into the TPTP language [26] so that we can call off-the-shelf automatic theorem provers (ATPs) when proving properties of our programs.

A key point of our approach is that Martin-Löf type theory is a subsystem of our theory through a natural interpretation [3]. However, our theory is strictly more general; in particular, we can write arbitrary general recursive functional programs. This extra generality comes at a price: since we can now reason about programs which do not terminate, we can no longer make use of the automatic type-checking in the same way as before. To compensate for this loss we use automatic first order theorem proving, although it does not fully replace the type-checking algorithm as we shall see. On the other hand, the ATPs can prove theorems automatically which would otherwise require manual proofs.

Overview of the paper. Section 2 introduces our FOTC for Plotkin's PCF language. In Section 3 we explain how to encode first order theories in Agda and how to instruct the proof assistant to call the ATPs. Section 4 shows how to encode FOTC for PCF in Agda and how this enables us to combine interactive and automatic theorem proving. In Section 5 we extend FOTC by adding inductive and coinductive predicates and we present an example using both. Finally, Section 6 contains some discussion of future and related work.

The programs and the examples described in the paper are available at www1.eafit.edu.co/asicard/code/fossacs-2012/.

2 First Order Theories of Combinators

As we mentioned before, Aczel showed how to interpret Martin-Löf type theory in traditional first order logic. He gave an *abstract realisability interpretation*, where

the *proof objects* are interpreted as terms in combinatory logic and *types* are interpreted as unary predicates. Aczel’s first order theory only has two constants (K and S) and one binary function symbol (for application). This is because all the term formers of Martin-Löf type theory can be encoded in the usual way using bracket abstraction, Church encodings, and fixed point operators. The theory also has three unary predicate symbols \mathcal{N} , \mathcal{P} , and \mathcal{T} meaning that a combinatory term encodes a natural number, an internal proposition, and an internal true proposition, respectively. Aczel’s paper was the first of several papers on realisability interpretations of Martin-Löf type theory; see for example Aczel [4] and Smith [25].

A Logic for PCF with Totality Predicates. Dybjer [9] showed that one of these logics for realisability interpretations, the so called *Logical Theory of Constructions (LTC)* is appropriate for practical verification of functional programs. This logic is closely related to Aczel’s first order theory, but is based on the λ -calculus, and is hence not a first order theory.

For the purpose of this paper we begin by considering an LTC-style logic for Plotkin’s PCF language [24]. PCF does not have internal propositions, hence we do not need the predicate symbols \mathcal{P} and \mathcal{T} . On the other hand, we have two unary predicate symbols $\mathcal{B}ool$ and \mathcal{N} , where $\mathcal{B}ool(t)$ means that t is a *total* boolean value (**true** or **false**), and $\mathcal{N}(t)$ that t is a *total* natural number. We will use these predicates to assert that a certain (possibly non-terminating) PCF program terminates with a total boolean value or a total natural number, respectively.

In a previous paper [7] we showed how to use Agda for implementing this LTC-style logic. The aim of the present paper is to make use of off-the-shelf automatic theorem provers for first order logic. Hence, we must make our logic first order by removing λ -abstraction. Instead, we work in an extensible theory and add a new function symbol for each recursive function definition of the form

$$f x_1 \cdots x_n = e[f, x_1, \dots, x_n].$$

It is well-known how to translate such definitions into terms using λ -abstraction and fixed point operators. For convenience, we might actually define function symbols by pattern matching, whenever it is clear that this pattern matching can be replaced by a single recursive equation by using `if`, `pred` and `iszero`.

The grammar for terms is now first order:

$$t ::= x \mid tt \mid \text{true} \mid \text{false} \mid \text{if} \mid 0 \mid \text{succ} \mid \text{pred} \mid \text{iszero} \mid f$$

where f ranges over new combinators defined by recursive equations as above. The axioms can be classified into three groups: (i) conversion rules for the combinators, (ii) discrimination rules expressing that terms beginning with different constructors are not convertible, and (iii) introduction and elimination rules for $\mathcal{B}ool$ and \mathcal{N} . We show these axioms in Section 4.

3 Combining Interactive and Automatic Proofs in First Order Logic

3.1 First Order Logic in Agda

The encoding of intuitionistic first order logic in dependent type theory using the formulae-as-types principle is of course well-known; below we briefly show what it looks like in Agda. For example, to implement disjunction we encode it as the disjoint union; note that below, we declare the constants as *postulates*.

```
postulate _∨_ : Set → Set → Set
  inl : {A B : Set} → A → A ∨ B
  inr : {A B : Set} → B → A ∨ B
  case : {A B C : Set} → (A → C) → (B → C) → A ∨ B → C
```

The first constant declares the syntax of disjunction as an infix binary set former. The second and third constants declare the introduction rules, and the fourth the elimination rule. Note that these rules are *axiom schemata* that is, they are sets of first order formulae, one for each instance of A , B and C . Agda is a *higher order* logic; to express the schematic nature of these rules we use (implicit) quantification over `Set`. Curly brackets `{,}` declare *implicit* arguments, that is, arguments that do not appear explicitly in the proof terms.

The proof of commutativity of disjunction can now be written as

```
commOr : {A B : Set} → A ∨ B → B ∨ A
commOr c = case inr inl c
```

By using postulates we can encode all of classical first order logic. The *adequacy problem*—the question of whether such an encoding gives rise to exactly the same provable formulae as the original theory—is studied by Gardner [12].

However, to make the most of the proof assistant it is preferable to use Agda's *data* declarations for inductively defined types, whenever appropriate. Hence, we declare the syntax and the introduction rules for disjunction as follows:

```
data _∨_ (A B : Set) : Set where
  inl : A → A ∨ B
  inr : B → A ∨ B
```

We can now write proofs by pattern matching; for example the proof of commutativity of disjunction becomes

```
commOr : {A B : Set} → A ∨ B → B ∨ A
commOr (inl a) = inr a
commOr (inr b) = inl b
```

When we encode our theory using `data` rather than `postulate` we get a new adequacy problem, since we have a more general language where we can write proofs by pattern matching. Here, we should only use pattern matching in ways which are reducible to the `case` combinator, encoding disjunction elimination.

We shall use `data` for all logical constants, the equality relation (denoted as \equiv), and the totality predicates in our FOTC (with the same remark as above).

Furthermore, to define the quantifiers we postulate a domain of individuals:

```
postulate D : Set
```

The *universal quantifier* is implemented by the dependent function type $(x : D) \rightarrow P$. If the domain D can be deduced by the type checker, we use the alternative notation $\forall x \rightarrow P$ for this type.

Finally, since the automatic theorem provers implement *classical* first order logic we need to include (a postulate for) the law of excluded middle:

```
postulate lem : {A : Set} → A ∨ ¬ A
```

3.2 Combining Agda with Automatic Theorem Provers

We have modified Agda by adding *pragmas* containing information to be used by the ATPs. These pragmas instruct the system to add information in an interface file which is generated after type-checking a file. In this way we tell the ATPs to prove a certain formula, or that a certain formula is an axiom or a general hint, or that a certain constant is a definition.

We tell the ATPs that the formula `name` is an axiom by the pragma

```
{-# ATP axiom name #-}
```

To prove a property automatically we first postulate it and add the pragma that instructs the ATPs to prove this conjecture. For example, to prove commutativity of disjunction automatically we write

```
postulate comm0r : {A B : Set} → A ∨ B → B ∨ A
{-# ATP prove comm0r #-}
```

After type-checking we run the program `agda2atp`, which first translates all axioms, definitions and conjectures in the generated interface file into the TPTP language, and then tries to prove the conjectures calling independently the automatic theorem provers E, Equinox, SPASS, Metis, or Vampire. In the terminal, we get information about which property is being proved and which ATP was able to prove a property first, if any.

```
Proving the conjecture in /tmp/Examples.comm0r_7.tptp ...
```

```
Vampire 0.6 (...) proved the conjecture in /tmp/Examples.comm0r_7.tptp
```

If no ATP could prove a conjecture within five minutes (by default), the process is cancelled and the ATPs will continue and try to prove the next conjecture.

It is possible to specify local hints in the pragma `{-# ATP prove ... #-}` by giving their names after the name of the conjecture to be proved.

4 Implementing FOTC for PCF in Agda

We first declare the syntax of PCF terms as the following postulates:

```
postulate if_then_else_ : D → D → D → D
         _' _ : D → D → D
         succ pred isZero : D → D
         zero true false : D
```

Note that if we were faithful to the syntax of PCF given in Section 2, `if`, `succ`, `pred` and `isZero` would have type `D`. However, the above versions are definable, and easier to use with the theorem prover (and easier to read for humans).

We now postulate the conversion rules, and add a pragma which declare them to be axioms for the ATPs:

```
postulate if-true : ∀ d1 {d2} → if true then d1 else d2 ≡ d1
         if-false : ∀ {d1} d2 → if false then d1 else d2 ≡ d2
         pred-S   : ∀ d → pred (succ d) ≡ d
         isZero-0 : isZero zero ≡ true
         isZero-S : ∀ d → isZero (succ d) ≡ false
{-# ATP axiom if-true if-false pred-S isZero-0 isZero-S #-}
```

We omit the discrimination rules.

Then we define a predicate for total natural numbers as a data type, and the induction schema for natural numbers by pattern matching:

```
data N : D → Set where
  zN : N zero
  sN : ∀ {n} → N n → N (succ n)
{-# ATP axiom zN sN #-}

indN : (P : D → Set) → P zero →
       (∀ {n} → P n → P (succ n)) → ∀ {n} → N n → P n
indN P P0 h zN = P0
indN P P0 h (sN Nn) = h (indN P P0 h Nn)
```

Note that since induction is a *schema* we cannot declare it as an axiom until it is instantiated. There are analogous rules for total Booleans.

Let us now add a combinator for addition. We postulate a binary infix operation on `D` and the (recursive) equations as axioms for the ATPs.

```
postulate _+_ : D → D → D
         +-0x : ∀ d → zero + e ≡ e
         +-Sx : ∀ d e → succ d + e ≡ succ (d + e)
{-# ATP axiom +-0x +-Sx #-}
```

We can show that addition is a total function on natural numbers by induction on the first argument. If we manually instantiate the induction schema, then both cases can be proved automatically (using a hint in the proof of `+-N1`):

```
indN-instance : ∀ x → N (zero + x) →
               (∀ {n} → N (n + x) → N (succ n + x)) →
               ∀ {n} → N n → N (n + x)
indN-instance x = indN (λ i → N (i + x))

postulate +-N1 : ∀ {m n} → N m → N n → N (m + n)
{-# ATP prove +-N1 indN-instance #-}
```

A more convenient way to instantiate the induction schema is to instruct Agda to do pattern matching on the first argument:

```

+-N : ∀ {m n} → N m → N n → N (m + n)
+-N {n = n} zN Nn = prf
  where postulate prf : N (zero + n)
        {-# ATP prove prf #-}
+-N {n = n} (sN {m} Nm) Nn = prf (+-N Nm Nn)
  where postulate prf : N (m + n) → N (succ m + n)
        {-# ATP prove prf #-}

```

To prove commutativity of addition we proceed in the same way: we do pattern matching on one of the arguments, then we prove the base case and the step case of the induction automatically.

```

+-comm : ∀ {m n} → N m → N n → m + n ≡ n + m
+-comm {n = n} zN Nn = prf
  where postulate prf : zero + n ≡ n + zero
        {-# ATP prove prf +-rightIdentity #-}
+-comm {n = n} (sN {m} Nm) Nn = prf (+-comm Nm Nn)
  where postulate prf : m + n ≡ n + m → succ m + n ≡ n + succ m
        {-# ATP prove prf x+Sy≡S[x+y] #-}

```

Here we used the following hints, which both were proved automatically:

```

+-rightIdentity : ∀ {n} → N n → n + zero ≡ n
x+Sy≡S[x+y] : ∀ {m n} → N m → N n → m + succ n ≡ succ (m + n)

```

4.1 An Example with Nested Recursion

McCarthy's 91-function is defined by the following axiom:

```

postulate mc91 : D → D
          mc91-eq : ∀ n → mc91 n ≡
                    if n > 100 then n - 10 else mc91 (mc91 (n + 11))
{-# ATP axiom mc91-eq #-}

```

We shall show that it has the following property:

```

mc91-res≧100 : ∀ {n} → N n → n ≧ 100 → mc91 n ≡ 91

```

The proof is done interactively by well-founded induction on the relation $101 - m < 101 - n$. Most of the auxiliary properties are proved with the help of the ATPs. We show only a few of them.

First we show that $mc91\ 100 \equiv 91$ by using the ATPs

```

postulate mc91-res-100 : mc91 100 ≡ 91
{-# ATP prove mc91-res-100 100+11>100 100+11-10>100
          101≡100+11-10 91≡100+11-10-10 #-}

```

where the hints are arithmetic properties which are proved automatically. To prove the remaining cases, we use a lemma that is proved automatically:

```

postulate mc91x-res≧100 : ∀ m n → m ≧ 100 → mc91 (m + 11) ≡ n →
          mc91 n ≡ 91 → mc91 m ≡ 91
{-# ATP prove mc91x-res≧100 #-}

```

Let $m < 100$. To compute `mc91 m` we use `mc91-eq`, for which we first need to compute `mc91 (m + 11)`. Which branch of the definition of `mc91` we use for this computation depends of the value of m .

If $90 \leq m \leq 99$ then $m + 11 > 100$, so we apply the true-branch and obtain $(m + 11) \dot{-} 10$ and we apply `mc91` again to the result of `mc91 (m + 11)`. We now use `mc91x-res` $\not\geq 100$ to prove that `mc91 m` returns 91. For the case of 98 we have:

```
postulate mc91-res-109 : mc91 (98 + 11) ≡ 99
          mc91-res-99  : mc91 99 ≡ 91
{-# ATP prove mc91-res-109 98+11>100 x+11-10≡Sx #-}
{-# ATP prove mc91-res-99 mc91x-res<100 mc91-res-110 mc91-res-100 #-}
```

On the other hand, if $m \leq 89$ then $m + 11 \not\geq 100$. Hence, our inductive hypothesis tells us that `mc91 (m + 11) ≡ 91`. Using `mc91x-res` $\not\geq 100$ on the inductive hypothesis and on the proof that `mc91 91 ≡ 91` we obtain the desired result.

Additionally, using well-founded induction on the relation $101 \dot{-} m < 101 \dot{-} n$ and with the help of the ATPs, we proved that `mc91` is a total function, we prove that `mc91 n ≡ n \dot{-} 10` when $n > 100$, and we prove that $\forall n. n < (\text{mc91 } n + 11)$.

5 Adding Inductive and Coinductive Predicates

5.1 Inductive Predicates

Note that FOTC for PCF is not *one* first order theory; it is a family of first order theories. When we add a new recursive function, we extend the theory with a new function symbol and one (or several) equational axioms. As we already remarked, it is easy to extend the model accordingly, since the model is based on Scott domains with a fixed point operator.

Furthermore, in addition to our inductively defined totality predicates \mathcal{N} and Bool , we may add other inductively defined predicates. For example, we may add a new inductively defined unary predicate symbol Even with axioms stating the *introduction rules* that **zero** is an even number and that even numbers are closed under the function which adds 2 to a natural number; and the induction schema stating that Even is the least predicate with those properties.

A schema for (intuitionistically valid) inductive predicates in first order logic is given by Martin-Löf [18]. However, since we work in classical logic, nothing prohibits us from adding inductively generated predicates by arbitrary (not necessarily strictly) positive operators, since they can easily be modelled as least fixed points of monotone operators on subsets of the domain [2].

5.2 An Example with Higher-Order Recursion

Here we define the mirror function for general trees in FOTC. First we extend our language with constructors for lists and trees:

```
postulate [] : D
          _::_ node : D → D → D
```

Then we mutually define predicates for total forests and trees.

```
mutual data Forest : D → Set where
  nilF : Forest []
  consF : ∀ {t ts} → Tree t → Forest ts → Forest (t :: ts)
data Tree : D → Set where
  treeT : ∀ d {ts} → Forest ts → Tree (node d ts)
```

(For space reasons we will omit the pragmas instructing the ATPs about axioms.)

Furthermore, we define the map function for lists

```
postulate map : D → D → D
  map-[] : ∀ f → map f [] ≡ []
  map-:: : ∀ f d ds → map f (d :: ds) ≡ f · d :: map f ds
```

and the mirror function for trees:

```
postulate mirror : D
  mirror-eq : ∀ d ts → mirror · (node d ts) ≡
    node d (reverse (map mirror ts))
```

We prove the following property:

```
mirror2 : ∀ {t} → Tree t → mirror · (mirror · t) ≡ t
```

We do induction on the proof that the tree is total and then on its underlying forest; we obtain two cases depending on whether the forest is empty or not.

```
mirror2 (treeT d nilF) = prf
  where postulate prf : mirror · (mirror · node d []) ≡ node d []
    {-# ATP prove prf #-}
mirror2 (treeT d (consF {t} {ts} Tt Fts)) = prf
  where postulate prf : mirror · (mirror · node d (t :: ts)) ≡
    node d (t :: ts)
    {-# ATP prove prf helper #-}
```

The hint helper is the following lemma:

```
helper : ∀ {ts} → Forest ts →
  reverse (map mirror (reverse (map mirror ts))) ≡ ts
```

It follows by induction on forests. Both cases are proved automatically.

5.3 Coinductive Predicates

We shall now show how to prove the correctness of a functional programming version of the alternating bit protocol (ABP). The purpose of this protocol is to ensure safe communication over an unreliable transmission channel. The sender tags the message with an (alternating) bit which is checked by the receiver. In the case of proper transmission the receiver sends the bit back to the sender as an acknowledgment. Otherwise, it sends the opposite bit back to signal that the message needs to be resent.

We follow Dybjer and Sander [11] who showed how to represent the ABP as a *Kahn network*, that is, as a network of communicating stream transformers, written in the lazy functional programming language Miranda [30] (a precursor of Haskell). They proved it correct in Park's μ -calculus [21]. This is an extension of first order classical logic with a μ -operator: for any positive formula $\Phi[X]$ with a free predicate variable X , we can form $\mu X.\Phi[X]$, with axioms which express that (i) $\mu X.\Phi[X]$ is a *prefixed point* of $\Phi[X]$ (the introduction rule for the *inductive predicate*), and (ii) that it is the *least prefixed point* (the elimination rule or induction principle). Since we work in classical logic we automatically have *coinductive predicates*, since *greatest fixed points* of $\Phi[X]$ can be defined as special least fixed points by dualisation.

Dybjer and Sander implemented the μ -calculus in the Isabelle system [22], and the proof was mechanically checked using Isabelle's tactics.

We here show how to modify Dybjer and Sander's approach so that it fits within first order logic. Rather than using the μ -operator (a second order construct) for inductive and coinductive predicates, we add new predicate symbols to our first order theories with axioms and axiom schemata corresponding to the least and greatest fixed point properties, respectively. In the previous section we showed how to add some inductive predicates. Now we will also add some coinductive predicates which will be used in the proof of the correctness of the alternating bit protocol.

Our first example is the coinductive definition of the predicate expressing that a certain list is infinite or *productive*. We add a unary predicate symbol *Stream* and two axioms expressing (i) that it is a *postfixed point* of a certain operator, and (ii) that it is the *greatest* such postfixed point:

$$\begin{aligned} \text{Stream-gfp}_1 & : \forall \{xs\} \rightarrow \text{Stream } xs \rightarrow \\ & \quad \exists [x'] \exists [xs'] \text{Stream } xs' \wedge xs \equiv x' :: xs' \\ \text{Stream-gfp}_2 & : (\text{P} : \text{D} \rightarrow \text{Set}) \rightarrow \\ & \quad (\forall \{xs\} \rightarrow \text{P } xs \rightarrow \exists [x'] \exists [xs'] \\ & \quad \quad \text{P } xs' \wedge xs \equiv x' :: xs') \rightarrow \\ & \quad \forall \{xs\} \rightarrow \text{P } xs \rightarrow \text{Stream } xs \end{aligned}$$

Similarly, we coinductively define when two streams are *bisimilar*:

$$\begin{aligned} \approx\text{-gfp}_1 & : \forall \{xs\} \forall \{ys\} \rightarrow xs \approx ys \rightarrow \exists [x'] \exists [xs'] \exists [ys'] xs' \approx ys' \\ & \quad \wedge xs \equiv x' :: xs' \wedge ys \equiv x' :: ys' \\ \approx\text{-gfp}_2 & : (_R : \text{D} \rightarrow \text{D} \rightarrow \text{Set}) \rightarrow (\forall \{xs\} \forall \{ys\} \rightarrow xs \text{ R } ys \rightarrow \\ & \quad \exists [x'] \exists [xs'] \exists [ys'] xs' \text{ R } ys' \\ & \quad \wedge xs \equiv x' :: xs' \wedge ys \equiv x' :: ys') \rightarrow \\ & \quad \forall \{xs\} \forall \{ys\} \rightarrow xs \text{ R } ys \rightarrow xs \approx ys \end{aligned}$$

In order to express the correctness property of the ABP we need a certain fairness property of the unreliable transmission channels. This property will be encoded in terms of oracle bit streams, where the bits T and F represent proper and improper transmission, respectively. Fairness here means that the bit stream contains an infinite number of Ts and is defined as follows:

$$\begin{aligned} \text{Fair-gfp}_1 & : \forall \{fs\} \rightarrow \text{Fair } fs \rightarrow \\ & \quad \exists [ft] \exists [fs'] \text{F*T } ft \wedge \text{Fair } fs' \wedge fs \equiv ft ++ fs' \end{aligned}$$

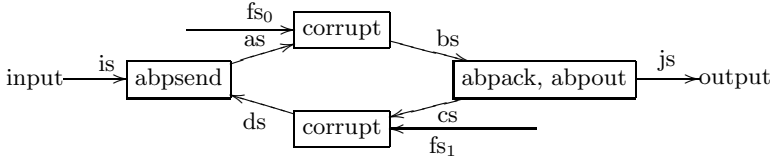


Fig. 1. The alternating bit protocol

$$\begin{aligned} \text{Fair-gfp}_2 : (P : D \rightarrow \text{Set}) \rightarrow (\forall \{fs\} \rightarrow P \ fs \rightarrow \\ \exists [ft] \exists [fs'] \ F * T \ ft \wedge P \ fs' \wedge fs \equiv ft ++ fs') \rightarrow \\ \forall \{fs\} \rightarrow P \ fs \rightarrow \text{Fair } fs \end{aligned}$$

Here $F * T \ ft$ is an inductive predicate expressing that ft is a finite list of F s followed by a final T . Note that we have added the constant symbols T and F for bits, and a binary infix function symbol $++$ for appending lists. In the proof below we will also make use of the predicate $\text{Bit} : D \rightarrow \text{Set}$. Moreover, we use $\langle _, _ \rangle$ for pairs, not for negation of bits, error for a corrupted message, and ok for a constructor for a proper message.

5.4 A Kahn Network for the Alternating Bit Protocol

Dybjer and Sander model the sender as a stream transformer abpsend and the receiver as a pair of stream transformers abpack , which returns the acknowledgement stream cs , and abpout , which returns the output stream js . Moreover, an unreliable transmission channel is modelled as a stream transformer, which non-deterministically corrupts the messages in the stream. To stay within the framework of deterministic lazy functional programming, we model the channels as a stream transformer $\text{corrupt} : D$ which accepts an oracle stream as an auxiliary argument as described above (see Fig 1). The axioms for corrupt are:

$$\begin{aligned} \text{corrupt-T} : \text{corrupt} \cdot (T :: fs) \cdot (x :: xs) &\equiv \text{ok } x :: \text{corrupt} \cdot fs \cdot xs \\ \text{corrupt-F} : \text{corrupt} \cdot (F :: fs) \cdot (x :: xs) &\equiv \text{error} :: \text{corrupt} \cdot fs \cdot xs \end{aligned}$$

(Note that for space reasons we have omitted the universal quantifiers. We will do so in the sequel as well. We will also omit the keyword *postulate*.)

The sender is written as a program which is mutually recursive with an auxiliary program await :

$$\begin{aligned} \text{abpsend-eq} : \text{abpsend} \cdot b \cdot (i :: is) \cdot ds &\equiv \langle i, b \rangle :: \text{await } b \ i \ is \ ds \\ \text{await-ok} &\equiv b \equiv b_0 \rightarrow \text{await } b \ i \ is \ (\text{ok } b_0 :: ds) \equiv \\ &\quad \text{abpsend} \cdot (\text{not } b) \cdot is \cdot ds \\ \text{await-ok} \neq &: \neg (b \equiv b_0) \rightarrow \text{await } b \ i \ is \ (\text{ok } b_0 :: ds) \equiv \\ &\quad \langle i, b \rangle :: \text{await } b \ i \ is \ ds \\ \text{await-error} : \text{await } b \ i \ is \ (\text{error} :: ds) &\equiv \langle i, b \rangle :: \text{await } b \ i \ is \ ds \end{aligned}$$

The first order axioms for the receiver programs `abpack` and `abpack` are

```

abpack-ok≡ : b ≡ b0 → abpack · b · (ok < i , b0 > :: bs) ≡
              b :: abpack · (not b) · bs
abpack-ok≠ : ¬ (b ≡ b0) → abpack · b · (ok < i , b0 > :: bs) ≡
              not b :: abpack · b · bs
abpack-error : abpack · b · (error :: bs) ≡ not b :: abpack · b · bs

```

```

abpack-ok≡ : b ≡ b0 → abpack · b · (ok < i , b0 > :: bs) ≡
              i :: abpack · (not b) · bs
abpack-ok≠ : ¬ (b ≡ b0) → abpack · b · (ok < i , b0 > :: bs) ≡
              abpack · b · bs
abpack-error : ∀ b bs → abpack · b · (error :: bs) ≡ abpack · b · bs

```

We can now write a function `abptransfer` which computes the output `js` from the input `is`, and accepts three more arguments: the initial bit `b`, and the two oracle streams `os0` and `os1`:

```

abptransfer-eq : abptransfer b fs0 fs1 is ≡
                 transfer (abpsend · b) (abpack · b) (abpack · b)
                 (corrupt · fs0) (corrupt · fs1) is

```

Here `transfer` is the general transfer function for the network topology of Fig. 2. It simultaneously computes the output `js` and the streams `as`, `bs`, `cs`, `ds` given

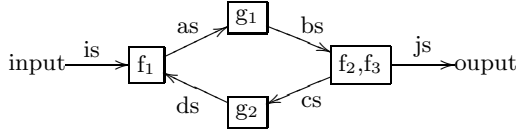


Fig. 2. Network topology for the alternating bit protocol

the stream transformers `f1`, `f2`, `f3`, `g1`, `g2`:

```

transfer-eq : transfer f1 f2 f3 g1 g2 is ≡ f3 · (hbs f1 f2 f3 g1 g2 is)
has-eq : has f1 f2 f3 g1 g2 is ≡ f1 · is · (hds f1 f2 f3 g1 g2 is)
hbs-eq : hbs f1 f2 f3 g1 g2 is ≡ g1 · (has f1 f2 f3 g1 g2 is)
hcs-eq : hcs f1 f2 f3 g1 g2 is ≡ f2 · (hbs f1 f2 f3 g1 g2 is)
hds-eq : hds f1 f2 f3 g1 g2 is ≡ g2 · (hcs f1 f2 f3 g1 g2 is)

```

To prove that the alternating bit protocol is correct means to prove that each message is eventually transmitted properly. Formally this means that the input stream is bisimilar to the output stream computed by `abptransfer`. This property can only hold if one assumes that the transmission channel(s) are “fair” in the sense described above. Formally we thus need to prove

```

spec : Bit b → Stream is → Fair fs0 → Fair fs1 →
       is ≈ abptransfer b fs0 fs1 is

```

The proof is by coinduction. We prove the `is` and `js` are in the greatest bisimulation $\approx_{_}$ by finding another bisimulation which they are in. This proof uses an auxiliary proof by induction on the predicate `F*T`.

As in the previous examples, we manually need to instantiate the axiom schemata (for induction and coinduction), but once this has been done a major part (but not all) equational and logical reasoning is done automatically by the ATPs. We do not have space here to present the details of this proof. The reader is referred to the paper’s website.

6 Conclusions and Related Work

What is unique about our approach is that its logical basis is Aczel’s first order theories of combinators. These theories were used for interpreting early versions of Martin-Löf’s intuitionistic type theory, and our approach can be summarised by saying that we work in models of type theory rather than in type theory itself. In particular we make essential use of totality predicates (which were used for interpreting types of type theory) and other inductive definitions.

A similar viewpoint has been exploited in the NuPrl project [29], where Martin-Löf type theory is also viewed through an interpretation in untyped computation systems. The difference is that in NuPrl the user still works in an extension of (extensional) Martin-Löf’s type theory, while we work in a setting which abandons most of the characteristics of Martin-Löf type theory. We work in classical rather than intuitionistic logic; we do not use the formulae-as-types principle; we have no dependent types, in fact our language is untyped rather than typed; and we deal with non-terminating as well as terminating programs. The advantage is that we can write our functional programs in the usual way as in mainstream functional languages. Although our term language is untyped, we may use polymorphic type inference during programming. However, the inferred types play no role during verification.

In this work we use the Agda system, but we could carry out similar work using another generic theorem prover such as Isabelle. However, Agda seems to work well as an interface to automatic first order theorem provers is positive: we have used it not only for FOTC but also for other first order theories such as Group Theory and Peano Arithmetic with encouraging results.

Future research. The present approach can be improved in several ways. The most obvious is to extend Agda so that it gives more support for FOTC and for interacting with ATPs. It would also be interesting to modify our program `agda2atp` and return witnesses for the automatically generated proofs so that they can be checked by Agda. Another interesting direction is to connect Agda to systems which can automatically do proof by induction; currently we only automate pure first order logic reasoning. In fact, Agda comes with its own automatic theorem prover *Agdy - the Agda Synthesiser* which can do proof by induction [17].

Related work. There is much related work on different aspects of this topic, and we only have space to mention a few. Perhaps most importantly, we should compare our approach with other systems which can be used for reasoning about general recursive programs, going back at least to the LCF-system [13]. We already mentioned some recent dedicated such systems [15,20]. Another interesting approach is the function package [16] built on top of the Isabelle system. The logical basis is here different from ours: the basic idea is to interpret first order functions as relations in Isabelle-HOL. The function package can also deal with higher order functions. Moreover, the Boyer-Moore theorem prover [8] is a powerful system for automatically proving properties of programs by induction. Logically, however it is based on primitive recursive arithmetic rather than untyped combinatory logic as ours. Yet another system in somewhat the same spirit as ours is Schwichtenberg's Minlog [5].

We will only mention some other related areas. One such area is concerned with methods for encoding general recursive functions in intuitionistic type theory, see for example [6]. Another area is concerned with connecting theorem provers with dependent type theory [1,27] or other generic theorem provers such as Isabelle [19].

References

1. Abel, A., Coquand, T., Norell, U.: Connecting a Logical Framework to a First-Order Logic Prover. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 285–301. Springer, Heidelberg (2005)
2. Aczel, P.: An introduction to inductive definitions. In: Barwise, J. (ed.) Handbook of Mathematical Logic, pp. 739–782. North-Holland Publishing Company (1977)
3. Aczel, P.: The strength of Martin-Löf's intuitionistic type theory with one universe. In: Miettinen, S., Väänänen, J. (eds.) Proc. of the Symposium on Mathematical Logic (Oulu, 1974). Report No. 2, Department of Philosophy, pp. 1–32. University of Helsinki, Helsinki (1977)
4. Aczel, P.: Frege structures and the notions of proposition, truth and set. In: Barwise, J., et al. (eds.) The Kleene Symposium, pp. 31–59. North-Holland, Amsterdam (1980)
5. Benl, H., et al.: Proof theory at work: Program development in the Minlog system. In: Bibel, W., et al. (eds.) Automated Deduction, vol. II, pp. 41–71. Kluwer Academic Publishers (1998)
6. Bove, A., Capretta, V.: Modelling general recursion in type theory. Math. Struct. in Comp. Science 15, 671–708 (2005)
7. Bove, A., Dybjer, P., Sicard-Ramírez, A.: Embedding a Logical Theory of Constructions in Agda. In: PLPV 2009, pp. 59–66 (2009)
8. Boyer, R.S., Kaufmann, M., Moore, J.S.: The Boyer-Moore theorem prover and its interactive enhancement. Computers & Mathematics with Applications 29(2), 27–62 (1995)
9. Dybjer, P.: Program Verification in a Logical Theory of Constructions. In: Jouanaud, J.-P. (ed.) FPCA 1985. LNCS, vol. 201, pp. 334–349. Springer, Heidelberg (1985)
10. Dybjer, P.: Comparing integrated and external logics of functional programs. Science of Computer Programming 14, 59–79 (1990)

11. Dybjer, P., Sander, H.P.: A functional programming approach to the specification and verification of concurrent systems. *Formal Aspects of Computing* 1, 303–319 (1989)
12. Gardner, P.: Representing Logics in Type Theory. Ph.D. thesis. University of Edinburgh, Department of Computer Science (1992)
13. Gordon, M., Wadsworth, C.P., Milner, R.: *Edinburgh LCF*. LNCS, vol. 78. Springer, Heidelberg (1979)
14. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. *JACM* 40(1), 143–184 (1993)
15. Harrison, W.L., Kieburtz, R.B.: The logic of demand in Haskell. *Journal of Functional Programming* 15(6), 837–891 (2005)
16. Krauss, A.: Partial and nested recursive function definitions in higher-order logic. *Journal of Automated Reasoning* 44(4), 303–336 (2010)
17. Lindblad, F., Benke, M.: A Tool for Automated Theorem Proving in Agda. In: Filliâtre, J.-C., Paulin-Mohring, C., Werner, B. (eds.) *TYPES 2004*. LNCS, vol. 3839, pp. 154–169. Springer, Heidelberg (2006)
18. Martin-Löf, P.: Hauptsatz for the intuitionistic theory of iterated inductive definitions. In: Fenstad, J.E. (ed.) *Proceedings of the Second Scandinavian Logic Symposium*, pp. 179–216. North-Holland Publishing Company (1971)
19. Meng, J., Quigley, C., Paulson, L.C.: Automation for interactive proof: First prototype. *Information and Computation* 204(10), 1575–1596 (2006)
20. de Mol, M., van Eekelen, M., Plasmeijer, R.: Theorem Proving for Functional Programmers. Sparkle: A Functional Theorem Prover. In: Arts, T., Mohnen, M. (eds.) *IFL 2001*. LNCS, vol. 2312, pp. 55–71. Springer, Heidelberg (2002)
21. Park, D.: Finitess is mu-ineffable. *Theoretical Computer Science* 3, 173–181 (1976)
22. Paulson, L.C.: Isabelle. A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994) (With a contribution by T. Nipkow)
23. Peyton Jones, S. (ed.): *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press (2003)
24. Plotkin, G.: LCF considered as a programming language. *Theoretical Computer Science* 5(3), 223–255 (1997)
25. Smith, J.: An interpretation of Martin-Löf’s type theory in a type-free theory of propositions. *The Journal of Symbolic Logic* 49(3), 730–753 (1984)
26. Sutcliffe, G.: The TPTP problem library and associated infrastructure. The FOT and CNF parts, v.3.5.0. *Journal of Automated Reasoning* 43(4), 337–362 (2009)
27. Tammet, T., Smith, J.M.: Optimized Encodings of Fragments of Type Theory in First Order Logic. In: Berardi, S., Coppo, M. (eds.) *TYPES 1995*. LNCS, vol. 1158, pp. 265–287. Springer, Heidelberg (1996)
28. The Agda development team: The Agda Wiki (2011), <http://wiki.portal.chalmers.se/agda>
29. The Nuprl development team: PRL Project (2011), <http://www.cs.cornell.edu/info/projects/nuprl/>
30. Turner, D.: An overview of Miranda. *SIGPLAN Notices* 21, 158–166 (1986)

Applicative Bisimulations for Delimited-Control Operators

Dariusz Biernacki and Sergueï Lenglet*

University of Wrocław

Abstract. We develop a behavioral theory for the untyped call-by-value λ -calculus extended with the delimited-control operators *shift* and *reset*. For this calculus, we discuss the possible observable behaviors and we define an applicative bisimilarity that characterizes contextual equivalence. We then compare the applicative bisimilarity and the CPS equivalence, a relation on terms often used in studies of control operators. In the process, we illustrate how bisimilarity can be used to prove equivalence of terms with delimited-control effects.

1 Introduction

Morris-style contextual equivalence [21] is usually regarded as the most natural behavioral equivalence for functional languages based on λ -calculi. Roughly, two terms are equivalent if we can exchange one for the other in a bigger program without affecting its behavior (i.e., whether it terminates or not). The quantification over program contexts makes contextual equivalence hard to use in practice and, therefore, it is common to look for more effective characterizations of this relation. One approach is to rely on coinduction, by searching for an appropriate notion of *bisimulation*. The bisimulation has to be defined in such a way that its resulting behavioral equivalence, called *bisimilarity*, is *sound* and *complete* with respect to contextual equivalence (i.e., it is included and contains contextual equivalence, respectively).

The problem of finding a sound and complete bisimilarity in the λ -calculus has been well studied and usually leads to the definition of an *applicative* bisimilarity [11,12,11] (or, more recently, *environmental* bisimilarity [23]). The situation is more complex in λ -calculi extended with *control operators* for first-class continuations—so far, only a few works have been conducted on the behavioral theory of such calculi. A first step can be found for the $\lambda\mu$ -calculus (a calculus that mimics abortive control operators such as *call/cc* [22]) in [3] and [9], where it is proved that the definition of contextual equivalence can be slightly simplified by quantifying over evaluation contexts only; such a result is usually called a *context lemma*. In [25], Støvring and Lassen define an *eager normal form bisimilarity* (based on the notion of Lévy-Longo tree equivalence) [15,16,17] which is sound for the $\lambda\mu$ -calculus, and which becomes sound and complete when a

* The author is supported by the Alain Bensoussan Fellowship Programme.

notion of state is added to the $\lambda\mu$ -calculus. In [19], Merro and Biasi define an applicative bisimilarity which characterizes contextual equivalence in the *CPS calculus* [26], a minimal calculus which models the control features of functional languages with imperative jumps. As for the λ -calculus extended with control only, however, no sound and complete bisimilarities have been defined.

In this article, we present a sound and complete applicative bisimilarity for a λ -calculus extended with Danvy and Filinski’s static delimited-control operators *shift* and *reset* [8]. In contrast to abortive control operators, delimited-control operators allow to delimit access to the current continuation and to compose continuations. The operators *shift* and *reset* were introduced as a direct-style realization of the traditional success/failure continuation model of backtracking otherwise expressible only in continuation-passing style. The numerous theoretical and practical applications of *shift* and *reset* (see, e.g., [5] for an extensive list) include the seminal result by Filinski showing that a programming language endowed with *shift* and *reset* is monadically complete [10].

The λ -calculi with static delimited-control operators have been an active research topic from the semantics as well as type- and proof-theoretic point of view (see, e.g., [5,4,2]). However, to our knowledge, no work has been carried out on the behavioral theory of such λ -calculi. In order to fill this void, we present a study of the behavioral theory of an untyped, call-by-value λ -calculus extended with *shift* and *reset* [8], called λ_S . In Section 2, we give the syntax and reduction semantics of λ_S , and discuss the possible observable behaviors for the calculus. In Section 3, we define an applicative bisimilarity, based on a labelled transition semantics, and prove it characterizes contextual equivalence, using an adaptation of Howe’s congruence proof method [12]. As a byproduct, we also prove a context lemma for λ_S . In Section 4, we study the relationship between applicative bisimilarity and an equivalence based on translation into continuation-passing style (CPS), a relation often used in works on control operators and CPS. In the process, we show how applicative bisimilarity can be used to prove equivalence of terms. Section 5 concludes the article and gives ideas for future work. Most of the proofs missing from the article are available in [7].

2 The Language λ_S

In this section, we present the syntax, reduction semantics, and contextual equivalence of the language λ_S used throughout this article.

2.1 Syntax

The language λ_S extends the call-by-value λ -calculus with the delimited-control operators *shift* and *reset* [8]. We assume we have a set of term variables, ranged over by x and k . We use two metavariables to distinguish term variables bound with a λ -abstraction from variables bound with a *shift*; we believe such distinction helps to understand examples and reduction rules. The syntax of terms and values is given by the following grammars:

Terms: $t ::= x \mid \lambda x.t \mid tt \mid Sk.t \mid \langle t \rangle$
 Values: $v ::= \lambda x.t$

The operator *shift* ($Sk.t$) is a capture operator, the extent of which is determined by the delimiter *reset* ($\langle \cdot \rangle$). A λ -abstraction $\lambda x.t$ binds x in t and a shift construct $Sk.t$ binds k in t ; terms are equated up to α -conversion of their bound variables. The set of free variables of t is written $\text{fv}(t)$; a term is *closed* if it does not contain any free variable. Because we work mostly with closed terms, we consider only λ -abstractions as values.

We distinguish several kinds of contexts, defined below, which all can be seen as terms with a hole.

Pure evaluation contexts: $E ::= \square \mid v E \mid E t$
 Evaluation contexts: $F ::= \square \mid v F \mid F t \mid \langle F \rangle$
 Contexts: $C ::= \square \mid \lambda x.C \mid t C \mid C t \mid Sk.C \mid \langle C \rangle$

Regular contexts are ranged over by C . The pure evaluation contexts¹ (abbreviated as pure contexts), ranged over by E , represent delimited continuations and can be captured by the shift operator. The call-by-value evaluation contexts, ranged over by F , represent arbitrary continuations and encode the chosen reduction strategy. Following the correspondence between evaluation contexts of the reduction semantics and control stacks of the abstract machine for shift and reset, established by Biernacka et al. [5], we interpret contexts inside-out, i.e., \square stands for the empty context, $v E$ represents the “term with a hole” $E[v []]$, $E t$ represents $E[[] t]$, $\langle F \rangle$ represents $F[\langle [] \rangle]$, etc. (This choice does not affect the results presented in this article in any way.) Filling a context C (respectively E , F) with a term t produces a term, written $C[t]$ (respectively $E[t]$, $F[t]$); the free variables of t can be captured in the process. A context is *closed* if it contains only closed terms.

2.2 Reduction Semantics

Let us first briefly describe the intuitive semantics of shift and reset by means of an example written in SML using Filinski’s implementation of shift and reset [10].

Example 1. The following function copies a list [6] (the SML expression `shift (fn k => t)` corresponds to $Sk.t$ and `reset (fn () => t)` corresponds to $\langle t \rangle$):

```
fun copy xs =
  let fun visit nil = nil
      | visit (x::xs) = visit (shift (fn k => x :: (k xs)))
  in reset (fn () => visit xs) end
```

This program illustrates the main ideas of programming with shift and reset:

¹ This terminology comes from Kameyama (e.g., in [13]).

- Reset delimits continuations. Control effects are local to `copy`.
- Shift captures delimited continuations. Each, but last, recursive call to `visit` abstracts the continuation `fn v => reset (fn () => visit v)` and binds it to `k`.
- Captured continuations are statically composed. When applied in the expression `k xs`, the captured continuation becomes the current delimited continuation that is isolated from the rest of the program by a control delimiter—witness the reset expression in the captured continuation.

Formally, the call-by-value reduction semantics of $\lambda_{\mathcal{S}}$ is defined by the following rules, where $t\{v/x\}$ is the usual capture-avoiding substitution of v for x in t :

$$\begin{array}{ll}
 (\beta_v) & F[(\lambda x.t) v] \rightarrow_v F[t\{v/x\}] \\
 (shift) & F[\langle E[\mathcal{S}k.t] \rangle] \rightarrow_v F[\langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle] \text{ with } x \notin \text{fv}(E) \\
 (reset) & F[\langle v \rangle] \rightarrow_v F[v]
 \end{array}$$

The term $(\lambda x.t) v$ is the usual call-by-value redex for β -reduction (rule (β_v)). The operator $\mathcal{S}k.t$ captures its surrounding context E up to the dynamically nearest enclosing `reset`, and substitutes $\lambda x.\langle E[x] \rangle$ for k in t (rule $(shift)$). If a `reset` is enclosing a value, then it has no purpose as a delimiter for a potential capture, and it can be safely removed (rule $(reset)$). All these reductions may occur within a metalevel context F . The chosen call-by-value evaluation strategy is encoded in the grammar of the evaluation contexts.

Example 2. Let $i = \lambda x.x$ and $\omega = \lambda x.x x$. We present the sequence of reductions initiated by $\langle\langle(\mathcal{S}k_1.i(k_1 i)) \mathcal{S}k_2.\omega\rangle\rangle(\omega \omega)$. The term $\mathcal{S}k_1.i(k_1 i)$ is within the pure context $E = (\square(\omega \omega)) \mathcal{S}k_2.\omega$ (remember that we represent contexts inside-out), enclosed in a delimiter $\langle \cdot \rangle$, so E is captured according to rule $(shift)$.

$$\langle\langle(\mathcal{S}k_1.i(k_1 i)) \mathcal{S}k_2.\omega\rangle\rangle(\omega \omega) \rightarrow_v \langle i((\lambda x.\langle(x \mathcal{S}k_2.\omega)(\omega \omega))\rangle) i) \rangle$$

The role of `reset` in $\lambda x.\langle E[x] \rangle$ becomes clearer after reduction of the (β_v) -redex $(\lambda x.\langle E[x] \rangle) i$.

$$\langle i((\lambda x.\langle(x \mathcal{S}k_2.\omega)(\omega \omega))\rangle) i) \rangle \rightarrow_v \langle i \langle (i \mathcal{S}k_2.\omega)(\omega \omega) \rangle \rangle$$

When the captured context E is reactivated, it is not *merged* with the context $i \square$, but *composed* thanks to the `reset` enclosing E . As a result, the capture triggered by $\mathcal{S}k_2.\omega$ leaves the term i outside the first enclosing `reset` untouched.

$$\langle i \langle (i \mathcal{S}k_2.\omega)(\omega \omega) \rangle \rangle \rightarrow_v \langle i \langle \omega \rangle \rangle$$

Because k_2 does not occur in ω , the context $i \square(\omega \omega)$ is discarded when captured by $\mathcal{S}k_2.\omega$. Finally, we remove the useless delimiter $\langle i \langle \omega \rangle \rangle \rightarrow_v \langle i \omega \rangle$ with rule $(reset)$, and we then (β_v) -reduce and remove the last delimiter $\langle i \omega \rangle \rightarrow_v \langle \omega \rangle \rightarrow_v \omega$. Note that, while the reduction strategy is call-by-value, some function arguments are not evaluated, like the non-terminating term $\omega \omega$ in this example.

There exist terms which are not values and which cannot be reduced any further; these are called *stuck terms*.

Definition 1. *A closed term t is stuck if t is not a value and $t \not\rightarrow_v$.*

For example, the term $E[Sk.t]$ is stuck because there is no enclosing reset; the capture of E by the shift operator cannot be triggered. In fact, closed stuck terms are easy to characterize.

Lemma 1. *A closed term t is stuck iff $t = E[Sk.t']$ for some E , k , and t' .*

We call *redexes* (ranged over by r) the terms of the form $(\lambda x.t)v$, $\langle E[Sk.t] \rangle$, and $\langle v \rangle$. Thanks to the following unique-decomposition property, the reduction \rightarrow_v is deterministic.

Lemma 2. *For all closed terms t , either t is a value, or it is a stuck term, or there exist a unique redex r and a unique context F such that $t = F[r]$.*

Given a relation \mathcal{R} on terms, we write \mathcal{R}^* for the transitive and reflexive closure of \mathcal{R} . We define the evaluation relation of λ_S as follows.

Definition 2. *We write $t \Downarrow_v t'$ if $t \rightarrow_v^* t'$ and $t' \not\rightarrow_v$.*

The result of the evaluation of a closed term, if it exists, is either a value or a stuck term. If a term t admits an infinite reduction sequence, we say it *diverges*, written $t \uparrow_v$. In the rest of the article, we use extensively $\Omega = (\lambda x.x x) (\lambda x.x x)$ as an example of such a term.

2.3 Contextual Equivalence

In this section, we discuss the possible definitions of a Morris-style contextual equivalence for the calculus λ_S . As usual, the idea is to express that two terms are equivalent iff they cannot be distinguished when put in an arbitrary context. The question is then what kind of behavior we want to observe. As in the regular λ -calculus we could observe only termination (i.e., does a term reduce to a value or not), leading to the following relation.

Definition 3. *Let t_0, t_1 be closed terms. We write $t_0 \approx_c^1 t_1$ if for all closed C , $C[t_0] \Downarrow_v v_0$ implies $C[t_1] \Downarrow_v v_1$, and conversely for $C[t_1]$.*

This definition does not mention stuck terms; as a result, they can be equated with diverging terms. For example, let $t_0 = (Sk.k \lambda x.x) \Omega$, $t_1 = \Omega$, and C be a closed context. If $C[t_0] \Downarrow_v v_0$, then we can prove that for all closed t , there exists v such that $C[t] \Downarrow_v v$ (roughly, because t is never evaluated; see [7] for further details). In particular, we have $C[t_1] \Downarrow_v v_1$. Hence, we have $t_0 \approx_c^1 t_1$.

A more fine-grained analysis is possible, by observing stuck terms.

Definition 4. Let t_0, t_1 be closed terms. We write $t_0 \approx_c^2 t_1$ if for all closed C ,

- $C[t_0] \Downarrow_v v_0$ implies $C[t_1] \Downarrow_v v_1$;
- $C[t_0] \Downarrow_v t'_0$, where t'_0 is stuck, implies $C[t_1] \Downarrow_v t'_1$, with t'_1 stuck as well;

and conversely for $C[t_1]$.

The relation \approx_c^2 distinguishes the terms t_0 and t_1 defined above. We believe \approx_c^2 is more interesting because it gives more information on the behavior of terms; consequently, we use it as the contextual equivalence for λ_S . Henceforth, we simply write \approx_c for \approx_c^2 .

The relation \approx_c , like the other equivalences on terms defined in this article, can be extended to open terms in the following way.

Definition 5. Let \mathcal{R} be a relation on closed terms. The open extension of \mathcal{R} , written \mathcal{R}° , is defined on open terms as: we write $t_0 \mathcal{R}^\circ t_1$ if for every substitution σ which closes t_0 and t_1 , $t_0\sigma \mathcal{R} t_1\sigma$ holds.

Remark 1. Contextual equivalence can be defined directly on open terms by requiring that the context C binds the free variables of the related terms. The resulting relation would be equal to \approx_c° [11].

3 Bisimilarity for λ_S

In this section, we define an applicative bisimilarity and prove it equal to contextual equivalence.

3.1 Labelled Transition System

To define the bisimilarity for λ_S , we propose a labelled transition system (LTS), where the possible interactions of a term with its environment are encoded in the labels. Figure 1 defines a LTS $t_0 \xrightarrow{\alpha} t_1$ with three kinds of transitions. An *internal action* $t \xrightarrow{\tau} t'$ is an evolution from t to t' without any help from the surrounding context; it corresponds to a reduction step from t to t' . The transition $v_0 \xrightarrow{v_1} t$ expresses the fact that v_0 needs to be applied to another value v_1 to evolve, reducing to t . Finally, the transition $t \xrightarrow{E} t'$ means that t is stuck, and when t is put in a context E enclosed in a reset, the capture can be triggered, the result of which being t' .

Most rules for internal actions (Fig. 1) are straightforward; the rules (β_v) and (reset) mimic the corresponding reduction rules, and the compositional rules (right_τ) , (left_τ) , and $(\langle \cdot \rangle_\tau)$ allow internal actions to happen within any evaluation context. The rule $(\langle \cdot \rangle_S)$ for context capture is explained later. Rule (val) defines the only possible transition for values. Note that while both rules (β_v) and (val) encode β -reduction, they are quite different in nature; in the former, the term $(\lambda x.t) v$ can evolve by itself, without any help from the surrounding context, while the latter expresses the possibility for $\lambda x.t$ to evolve only if a value v is provided by the environment.

$\frac{}{(\lambda x.t) v \xrightarrow{\tau} t\{v/x\}} \text{ (\beta}_v\text{)}$	$\frac{}{\langle v \rangle \xrightarrow{\tau} v} \text{ (reset)}$	$\frac{t_0 \xrightarrow{\tau} t'_0}{t_0 t_1 \xrightarrow{\tau} t'_0 t_1} \text{ (left}_\tau\text{)}$	
$\frac{t \xrightarrow{\tau} t'}{v t \xrightarrow{\tau} v t'} \text{ (right}_\tau\text{)}$	$\frac{t \xrightarrow{\tau} t'}{\langle t \rangle \xrightarrow{\tau} \langle t' \rangle} \text{ ((}\cdot\text{)}_\tau\text{)}$	$\frac{t \xrightarrow{\square} t'}{\langle t \rangle \xrightarrow{\tau} t'} \text{ ((}\cdot\text{)}_S\text{)}$	$\frac{}{\lambda x.t \xrightarrow{v} t\{v/x\}} \text{ (val)}$
$\frac{x \notin \text{fv}(E)}{Sk.t \xrightarrow{E} \langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle} \text{ (shift)}$	$\frac{t_0 \xrightarrow{E t_1} t'_0}{t_0 t_1 \xrightarrow{E} t'_0} \text{ (left}_S\text{)}$	$\frac{t \xrightarrow{v E} t'}{v t \xrightarrow{E} t'} \text{ (right}_S\text{)}$	

Fig. 1. Labelled Transition System

The rules for context capture are built following the principles of complementary semantics developed in [18]. The label of the transition $t \xrightarrow{E} t'$ contains what the environment needs to provide (a context E , but also an enclosing reset, left implicit) for the stuck term t to reduce to t' . Hence, the transition $t \xrightarrow{E} t'$ means that we have $\langle E[t] \rangle \xrightarrow{\tau} t'$ by context capture. For example, in the rule (shift), the result of the capture of E by $Sk.t$ is $\langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle$.

In rule (left_S), we want to know the result of the capture of E by the term $t_0 t_1$, assuming t_0 contains an operator shift. Under this hypothesis, the capture of E by $t_0 t_1$ comes from the capture of $E t_1$ by t_0 . Therefore, as premise of the rule (left_S), we check that t_0 is able to capture $E t_1$, and the result t'_0 of this transition is exactly the result we want for the capture of E by $t_0 t_1$. The rule (right_S) follows the same pattern. Finally, a stuck term t enclosed in a reset is able to perform an internal action (rule ((\cdot)_S)); we obtain the result t' of the transition $\langle t \rangle \xrightarrow{\tau} t'$ by letting t capture the empty context, i.e., by considering the transition $t \xrightarrow{\square} t'$.

Example 3. With the same notations as in Example 2, we illustrate how the LTS handles capture by considering the transition from $\langle\langle i Sk.\omega \rangle (\omega \omega) \rangle$.

$$\frac{}{Sk.\omega \xrightarrow{i(\square(\omega \omega))} \langle \omega \rangle} \text{ (shift)}$$

$$\frac{Sk.\omega \xrightarrow{i(\square(\omega \omega))} \langle \omega \rangle}{i Sk.\omega \xrightarrow{\square(\omega \omega)} \langle \omega \rangle} \text{ (right}_S\text{)}$$

$$\frac{i Sk.\omega \xrightarrow{\square(\omega \omega)} \langle \omega \rangle}{(i Sk.\omega) (\omega \omega) \xrightarrow{\square} \langle \omega \rangle} \text{ (left}_S\text{)}$$

$$\frac{(i Sk.\omega) (\omega \omega) \xrightarrow{\square} \langle \omega \rangle}{\langle\langle i Sk.\omega \rangle (\omega \omega) \rangle \xrightarrow{\tau} \langle \omega \rangle} \text{ ((}\cdot\text{)}_S\text{)}$$

Reading the tree from bottom to top, we see that the rules ((\cdot)_S), (left_S), and (right_S) build the captured context in the label by deconstructing the initial term. Indeed, the rule ((\cdot)_S) removes the outermost reset, and initiates the context in the label with \square . The rules (left_S) and (right_S) then successively remove the outermost application and store it in the context. The process continues until

a shift operator is found; then we know the captured context is completed, and the rule (shift) computes the result of the capture. This result is then simply propagated from top to bottom by the other rules.

The LTS corresponds to the reduction semantics and exhibits the observable terms (values and stuck terms) of the language in the following way.

Lemma 3. *The following hold:*

- We have $\xrightarrow{\tau} = \rightarrow_v$.
- If $t \xrightarrow{E} t'$, then t is a stuck term, and $\langle E[t] \rangle \xrightarrow{\tau} t'$.
- If $t \xrightarrow{v} t'$, then t is a value, and $t v \xrightarrow{\tau} t'$.

3.2 Applicative Bisimilarity

We now define the notion of applicative bisimilarity for λ_S . We write \Rightarrow for the reflexive and transitive closure of $\xrightarrow{\tau}$. We define the weak delay² transition $\overset{\alpha}{\Rightarrow}$ as \Rightarrow if $\alpha = \tau$ and as \Rightarrow^{α} otherwise. The definition of the (weak delay) bisimilarity is then straightforward.

Definition 6. *A relation \mathcal{R} on closed terms is an applicative simulation if $t_0 \mathcal{R} t_1$ implies that for all $t_0 \xrightarrow{\alpha} t'_0$, there exists t'_1 such that $t_1 \overset{\alpha}{\Rightarrow} t'_1$ and $t'_0 \mathcal{R} t'_1$.*

A relation \mathcal{R} on closed terms is an applicative bisimulation if \mathcal{R} and \mathcal{R}^{-1} are simulations. Applicative bisimilarity \approx is the largest applicative bisimulation.

In words, two terms are equivalent if any transition from one is matched by a weak transition with the same label from the other. As in the λ -calculus [UUI], it is not mandatory to test the internal steps when proving that two terms are bisimilar, because of the following result.

Lemma 4. *If $t \xrightarrow{\tau} t'$ (respectively $t \downarrow_v t'$) then $t \approx t'$.*

Lemma 4 holds because $\{(t, t'), t \xrightarrow{\tau} t'\}$ is an applicative bisimulation. Consequently, applicative bisimulation can be defined in terms of big-step transitions as follows.

Definition 7. *A relation \mathcal{R} on closed terms is a big-step applicative simulation if $t_0 \mathcal{R} t_1$ implies that for all $t_0 \overset{\alpha}{\Rightarrow} t'_0$ with $\alpha \neq \tau$, there exists t'_1 such that $t_1 \overset{\alpha}{\Rightarrow} t'_1$ and $t'_0 \mathcal{R} t'_1$.*

A relation \mathcal{R} on closed terms is a big-step applicative bisimulation if \mathcal{R} and \mathcal{R}^{-1} are big-step applicative simulations. Big-step applicative bisimilarity \approx is the largest big-step applicative bisimulation.

Henceforth, we drop the adjective “applicative” and refer to the two kinds of relations simply as “bisimulation” and “big-step bisimulation”.

² Where internal steps are allowed before, but not after a visible action.

Lemma 5. *We have $\approx = \approx^\bullet$.*

The proof is by showing that \approx is a big step bisimulation, and that \approx^\bullet is a bisimulation (using a variant of Lemma 4 involving \approx^\bullet). As a result, if \mathcal{R} is a big-step bisimulation, then $\mathcal{R} \subseteq \approx^\bullet \subseteq \approx$. We work with both styles (small-step and big-step), depending on which one is easier to use in a given proof.

Example 4. Assuming we add lists and recursion to the calculus, we informally prove that the function `copy` defined in Example 1 is bisimilar to its effect-free variant, defined below.

```
fun copy2 nil = nil
  | copy2 (x::xs) = x::(copy2 xs)
```

To this end, we define the relations (where we let l range over lists, and e over their elements)

$$\begin{aligned} \mathcal{R}_1 &= \{ \langle (e_1 :: \langle e_2 :: \dots \langle e_n :: \langle \text{visit } l \rangle \rangle) \rangle, e_1 :: (e_2 :: \dots e_n :: (\text{copy2 } l)) \rangle \} \\ \mathcal{R}_2 &= \{ \langle (e_1 :: \langle e_2 :: \dots \langle e_n :: \langle l \rangle \rangle) \rangle, e_1 :: (e_2 :: \dots e_n :: l) \rangle \} \end{aligned}$$

and we prove that $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \{(l, l)\}$ is a bisimulation. First, let $t_0 \mathcal{R}_1 t_1$. If l is empty, then both `visit l` and `copy2 l` reduce to the empty list, and we obtain two terms related by \mathcal{R}_2 . Otherwise, we have $l = e_{n+1} :: l'$, `visit l` reduces to `visit l'`, `copy2 l` reduces to `copy2 l'`, and therefore t_0 and t_1 reduce to terms that are still in \mathcal{R}_1 . Now, consider $t_0 \mathcal{R}_2 t_1$; the transition from t_0 removes the delimiter surrounding l , giving a term related by \mathcal{R}_2 to t_1 if there are still some delimiters left, or equal to t_1 if all the delimiters are removed. Finally, two identical lists are clearly bisimilar.

3.3 Soundness

To prove soundness of \approx w.r.t. contextual equivalence, we show that \approx is a congruence using *Howe's method*, a well-known congruence proof method initially developed for the λ -calculus [12, 11]. We briefly sketch the method and explain how we apply it to \approx ; the complete proof can be found in [7].

The idea of the method is as follows: first, prove some basic properties of *Howe's closure* \approx^\bullet , a relation which contains \approx and is a congruence by construction. Then, prove a simulation-like property for \approx^\bullet . From this result, prove that \approx^\bullet and \approx coincide on closed terms. Because \approx^\bullet is a congruence, it shows that \approx is a congruence as well. The definition of \approx^\bullet relies on the notion of *compatible refinement*; given a relation \mathcal{R} on open terms, the compatible refinement $\widehat{\mathcal{R}}$ relates two terms iff they have the same outermost operator and their immediate subterms are related by \mathcal{R} . Formally, it is inductively defined by the following rules.

$$\frac{}{x \widehat{\mathcal{R}} x} \quad \frac{t_0 \mathcal{R} t_1}{\lambda x.t_0 \widehat{\mathcal{R}} \lambda x.t_1} \quad \frac{t_0 \mathcal{R} t_1 \quad t'_0 \mathcal{R} t'_1}{t_0 t'_0 \widehat{\mathcal{R}} t_1 t'_1} \quad \frac{t_0 \mathcal{R} t_1}{Sk.t_0 \widehat{\mathcal{R}} Sk.t_1} \quad \frac{t_0 \mathcal{R} t_1}{\langle t_0 \rangle \widehat{\mathcal{R}} \langle t_1 \rangle}$$

Howe's closure \approx^\bullet is inductively defined as the smallest congruence containing \approx° and closed under right composition with \approx° .

Definition 8. *Howe's closure \approx^\bullet is the smallest relation satisfying:*

$$\frac{t_0 \approx^\circ t_1}{t_0 \approx^\bullet t_1} \qquad \frac{t_0 \approx^\bullet \approx^\circ t_1}{t_0 \approx^\bullet t_1} \qquad \frac{t_0 \widehat{\approx}^\bullet t_1}{t_0 \approx^\bullet t_1}$$

By construction, \approx^\bullet is a congruence (by the third rule of the definition), and composing on the right with \approx° gives some transitivity properties to \approx^\bullet . In particular, it helps in proving the following classical results (see [11] for the proofs).

Lemma 6 (Basic properties of \approx^\bullet). *The following hold:*

- For all t_0, t_1, v_0 , and v_1 , $t_0 \approx^\bullet t_1$ and $v_0 \approx^\bullet v_1$ implies $t_0\{v_0/x\} \approx^\bullet t_1\{v_1/x\}$.
- The relation $(\approx^\bullet)^*$ is symmetric.

The first item states that \approx^\bullet is substitutive. This property helps in establishing the simulation-like property of \approx^\bullet (second step of the method). Let $(\approx^\bullet)^c$ be the restriction of \approx^\bullet to closed terms. We cannot prove directly that $(\approx^\bullet)^c$ is a bisimulation, so we prove a stronger result instead. We extend \approx^\bullet to labels, by defining $E \approx^\bullet E'$ as the smallest congruence extending \approx^\bullet with the relation $\square \approx^\bullet \square$, and by adding the relation $\tau \approx^\bullet \tau$.

Lemma 7 (Simulation-like property). *If $t_0 (\approx^\bullet)^c t_1$ and $t_0 \xrightarrow{\alpha} t'_0$, then for all $\alpha (\approx^\bullet)^c \alpha'$, there exists t'_1 such that $t_1 \xrightarrow{\alpha'} t'_1$ and $t'_0 (\approx^\bullet)^c t'_1$.*

Using Lemma 7 and the fact that $((\approx^\bullet)^c)^*$ is symmetric (by the second item of Lemma 6), we can prove that $((\approx^\bullet)^c)^*$ is a bisimulation. Therefore, we have $((\approx^\bullet)^c)^* \subseteq \approx$, and because $\approx \subseteq ((\approx^\bullet)^c)^*$ holds by construction, we can deduce $\approx = ((\approx^\bullet)^c)^*$. Because $(\approx^\bullet)^c$ is a congruence, we have the following result.

Theorem 1. *The relation \approx is a congruence.*

As a corollary, \approx is sound w.r.t. contextual equivalence.

Theorem 2. *We have $\approx \subseteq \approx_c$.*

3.4 Completeness and Context Lemma

In this section, we prove that \approx is complete w.r.t. \approx_c . To this end, we use an auxiliary relation $\tilde{\approx}_c$, defined below, which refines contextual equivalence by testing terms with evaluation contexts only. While proving completeness, we also prove $\tilde{\approx}_c = \approx_c$, which means that testing with evaluation contexts is as discriminative as testing with any contexts. Such a simplification result is similar to Milner's context lemma [20].

Definition 9. *Let t_0, t_1 be closed terms. We write $t_0 \tilde{\approx}_c t_1$ if for all closed F ,*

- $F[t_0] \Downarrow_v v_0$ implies $F[t_1] \Downarrow_v v_1$;
- $F[t_0] \Downarrow_v t'_0$, where t'_0 is stuck, implies $F[t_1] \Downarrow_v t'_1$, with t'_1 stuck as well;

and conversely for $F[t_1]$.

Clearly we have $\approx_c \subseteq \tilde{\approx}_c$ by definition. The relation \approx is complete w.r.t. $\tilde{\approx}_c$.

$$\begin{aligned}
\bar{x} &= \lambda k_1 k_2. k_1 x k_2 \\
\overline{\lambda x.t} &= \lambda k_1 k_2. k_1 (\lambda x.\bar{t}) k_2 \\
\overline{t_0 t_1} &= \lambda k_1 k_2. \bar{t}_0 (\lambda x_0 k'_2. \bar{t}_1 (\lambda x_1 k'_2. x_0 x_1 k_1 k'_2)) k'_2 k_2 \\
\overline{\langle t \rangle} &= \lambda k_1 k_2. \bar{t} \theta (\lambda x. k_1 x k_2) \\
\overline{Sk.t} &= \lambda k_1 k_2. \bar{t} \{ (\lambda x_1 k'_1 k'_2. k_1 x_1 (\lambda x_2. k'_1 x_2 k'_2)) / k \} \theta k_2 \\
&\text{with } \theta = \lambda x k_2. k_2 x
\end{aligned}$$

Fig. 2. CPS translation

Theorem 3. *We have $\approx_c \subseteq \approx$.*

The proof of Theorem 3 is the same as in λ -calculus [11]; we prove that \approx_c is a big-step bisimulation, using Lemmas 3, 4, and Theorem 2. The complete proof can be found in [7]. We can now prove that all the relations defined so far coincide.

Theorem 4. *We have $\approx_c = \approx_c = \approx$.*

Indeed, we have $\approx_c \subseteq \approx$ (Theorem 3), $\approx \subseteq \approx_c$ (Theorem 2), and $\approx_c \subseteq \approx_c$ (by definition).

4 Relation to CPS Equivalence

In this section we study the relationship between our bisimilarity (and thus contextual equivalence) and an equivalence relation based on translating terms with shift and reset into continuation-passing style (CPS). Such an equivalence has been characterized in terms of direct-style equations by Kameyama and Hasegawa who developed an axiomatization of shift and reset [13]. We show that all but one of their axioms are validated by the bisimilarity of this article, which also provides several examples of use of the bisimilarity. We also pinpoint where the two relations differ.

4.1 Axiomatization of Delimited Continuations

The operators shift and reset have been originally defined by a translation into continuation-passing style [8] that we present in Fig. 2. Translated terms expect two continuations: the delimited continuation representing the rest of the computation up to the dynamically nearest enclosing delimiter and the meta-continuation representing the rest of the computation beyond this delimiter.

It is natural to relate any other theory of shift and reset to their definitional CPS translation. For example, the reduction rules $t \rightarrow_v t'$ given in Section 2.2 are sound w.r.t. the CPS because CPS translating t and t' yields $\beta\eta$ -convertible terms in the λ -calculus. More generally, the CPS translation for shift and reset induces the following notion of equivalence on terms:

$\langle \lambda x.t \rangle v = t\{v/x\}$	β_v	$\langle \lambda x.E[x] \rangle t = E[t]$ if $x \notin \text{fv}(E)$	β_Ω
$\langle E[\mathcal{S}k.t] \rangle = \langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle$	$\langle \cdot \rangle \mathcal{S}$	$\langle \langle \lambda x.t_0 \rangle \langle t_1 \rangle \rangle = \langle \lambda x.\langle t_0 \rangle \rangle \langle t_1 \rangle$	$\langle \cdot \rangle \text{lift}$
$\langle v \rangle = v$	$\langle \cdot \rangle \text{val}$	$\mathcal{S}k.\langle t \rangle = \mathcal{S}k.t$	$\mathcal{S} \langle \cdot \rangle$
$\lambda x.v x = v$ if $x \notin \text{fv}(v)$	η_v	$\mathcal{S}k.k t = t$ if $k \notin \text{fv}(t)$	$\mathcal{S} \text{elim}$

Fig. 3. Axiomatization of λ_S

Definition 10. *Terms t and t' are CPS equivalent if their CPS translations are $\beta\eta$ -convertible.*

In order to relate the bisimilarity of this article and the CPS equivalence, we use Kameyama and Hasegawa's axioms [13], which characterize the CPS equivalence in a sound and complete way: two terms are CPS equivalent iff one can derive their equality using the equations of Fig. 3. Kameyama and Hasegawa's axioms relate not only closed, but arbitrary terms and they assume variables as values.

4.2 Kameyama and Hasegawa's Axioms through Bisimilarity

We show that closed terms related by all the axioms except for $\mathcal{S} \text{elim}$ are bisimilar. In the following, we write \mathcal{I} for the bisimulation $\{(t, t)\}$.

Proposition 1. *We have $\langle \lambda x.t \rangle v \approx t\{v/x\}$, $\langle E[\mathcal{S}k.t] \rangle \approx \langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle$, and $\langle v \rangle \approx v$.*

Proof. These are direct consequences of the fact that $\xrightarrow{\tau} \subseteq \approx$ (Lemma 4). □

Proposition 2. *If $x \notin \text{fv}(v)$, then $\lambda x.v x \approx v$.*

Proof. We prove that $\mathcal{R} = \{(\lambda x.(\lambda y.t) x, \lambda y.t), x \notin \text{fv}(t)\} \cup \approx$ is a bisimulation. To this end, we have to check that $\lambda x.(\lambda y.t) x \xrightarrow{v_0} (\lambda y.t) v_0$ is matched by $\lambda y.t \xrightarrow{v_0} t\{v_0/y\}$, i.e., that $(\lambda y.t) v_0 \mathcal{R} t\{v_0/y\}$ holds for all v_0 . We have $(\lambda y.t) v_0 \xrightarrow{\tau} t\{v_0/y\}$, and because $\xrightarrow{\tau} \subseteq \approx \subseteq \mathcal{R}$, we have the required result. □

Proposition 3. *We have $\mathcal{S}k.\langle t \rangle \approx \mathcal{S}k.t$.*

Proof. Let $\mathcal{R} = \{(\langle \langle t \rangle \rangle, \langle t \rangle)\}$. We prove that $\{(\mathcal{S}k.\langle t \rangle, \mathcal{S}k.t)\} \cup \mathcal{R} \cup \mathcal{I}$ is a big-step bisimulation. The transition $\mathcal{S}k.\langle t \rangle \xrightarrow{E} \langle \langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle \rangle$ is matched by $\mathcal{S}k.t \xrightarrow{E} \langle t\{\lambda x.\langle E[x] \rangle/k\} \rangle$, and conversely. Let $\langle \langle t \rangle \rangle \mathcal{R} \langle t \rangle$. It is straightforward to check that $\langle \langle t \rangle \rangle \xrightarrow{v_0} v$ iff $t \xrightarrow{v_0} v$ iff $\langle t \rangle \xrightarrow{v_0} v$. Therefore, any $\xrightarrow{v_0}$ transition from $\langle \langle t \rangle \rangle$ is matched by $\langle t \rangle$, and conversely. If $\langle t \rangle \xrightarrow{\tau} t'$, then t' is a value or $t' = \langle t'' \rangle$ for some t'' ; consequently, neither $\langle t \rangle$ nor $\langle \langle t \rangle \rangle$ can perform a \xrightarrow{E} transition. □

Proposition 4. *We have $\langle\langle\lambda x.t_0\rangle\langle t_1\rangle\rangle \approx (\lambda x.\langle t_0\rangle)\langle t_1\rangle$.*

Proof. We prove that $\{\langle\langle\lambda x.t_0\rangle\langle t_1\rangle\rangle, (\lambda x.\langle t_0\rangle)\langle t_1\rangle\} \cup \mathcal{I}$ is a big-step bisimulation. A transition $\langle\langle\lambda x.t_0\rangle\langle t_1\rangle\rangle \xrightarrow{\alpha} t'$ (with $\alpha \neq \tau$) is possible only if $\langle t_1\rangle$ evaluates to some value v . In this case, we have $\langle\langle\lambda x.t_0\rangle\langle t_1\rangle\rangle \xrightarrow{\tau} \langle(\lambda x.t_0)v\rangle \xrightarrow{\tau} \langle t_0\{v/x\}\rangle$ and $(\lambda x.\langle t_0\rangle)\langle t_1\rangle \xrightarrow{\tau} \langle t_0\{v/x\}\rangle$. From this, it is easy to see that $\langle\langle\lambda x.t_0\rangle\langle t_1\rangle\rangle \xrightarrow{\alpha} t'$ (with $\alpha \neq \tau$) implies $(\lambda x.\langle t_0\rangle)\langle t_1\rangle \xrightarrow{\alpha} t'$, and conversely. \square

Proposition 5. *If $x \notin \text{fv}(E)$, then $(\lambda x.E[x])t \approx E[t]$.*

Proof (Sketch). The complete proof, quite technical, can be found in [7]. Let E_0 be such that $\text{fv}(E_0) = \emptyset$. Given two families of contexts $(E_1^i)_i, (E_2^i)_i$, we write $\sigma_i^{E_0}$ (resp. $\sigma_i^{\lambda x.E_0[x]}$) the substitution mapping k_i to $\lambda y.\langle E_1^i[E_0[E_2^i[y]]]\rangle$ (resp. $\lambda y.\langle E_1^i[(\lambda x.E_0[x])E_2^i[y]]\rangle$). We define

$$\begin{aligned} \mathcal{R}_1 &= \{(F[(\lambda x.E_0[x])t]\sigma_0^{\lambda x.E_0[x]} \dots \sigma_n^{\lambda x.E_0[x]}, F[E_0[t]]\sigma_0^{E_0} \dots \sigma_n^{E_0}), \\ &\quad \text{fv}(t, F) \subseteq \{k_0 \dots k_n\}\} \\ \mathcal{R}_2 &= \{(t\sigma_0^{\lambda x.E_0[x]} \dots \sigma_n^{\lambda x.E_0[x]}, t\sigma_0^{E_0} \dots \sigma_n^{E_0}), \text{fv}(t) \subseteq \{k_0 \dots k_n\}\} \end{aligned}$$

and we prove that $\mathcal{R}_1 \cup \mathcal{R}_2$ is a bisimulation. The relation \mathcal{R}_1 contains the terms related by Proposition 5. The transitions from terms in \mathcal{R}_1 give terms in \mathcal{R}_1 , except if a capture happens in t ; in this case, we obtain terms in \mathcal{R}_2 . Similarly, most transitions from terms in \mathcal{R}_2 give terms in \mathcal{R}_2 , except if a term $\lambda y.\langle E_1^i[E_0[E_2^i[y]]]\rangle$ (resp. $\lambda y.\langle E_1^i[(\lambda x.E_0[x])E_2^i[y]]\rangle$) is applied to a value (i.e., if $t = F[k_i v]$). In this case, the β -reduction generates terms in \mathcal{R}_1 . \square

4.3 Bisimilarity and CPS Equivalence

In Section 4.2, we have considered all the axioms of Fig. 3, except $\mathcal{S} \text{ elim}$. The terms $\mathcal{S}k.k t$ (with $k \notin \text{fv}(t)$) and t are not bisimilar in general, as we can see in the following result.

Proposition 6. *We have $\mathcal{S}k.k v \not\approx v$.*

The \xrightarrow{E} transition from $\mathcal{S}k.k v$ cannot be matched by v . In terms of contextual equivalence, it is not possible to equate a stuck term and a value (it is also forbidden by the relation \approx_c^1 of Section 2.3). The CPS equivalence cannot distinguish between stuck terms and values, because the CPS translation turns all $\lambda_{\mathcal{S}}$ terms into λ -calculus terms of the form $\lambda k_1 k_2.t$, where k_1 is the continuation up to the first enclosing reset, and k_2 is the continuation beyond this reset. Therefore, the CPS translation (and CPS equivalence) assumes that there is always an enclosing reset, while contextual equivalence does not. To be in accordance with CPS, the contextual equivalence should be changed, so that it tests terms only in contexts with an outermost delimiter. We conjecture that the CPS equivalence is included in such a modified contextual equivalence. Note that stuck terms can no longer be observed in such modified relation, because a term within a reset

cannot become stuck (see the proof of Proposition 3). Therefore, the bisimilarity of this article is too discriminative w.r.t. to this modified equivalence, and a new complete bisimilarity has to be found.

Conversely, there exist bisimilar terms that are not CPS equivalent:

Proposition 7. 1. We have $\Omega \approx \Omega\Omega$, but Ω and $\Omega\Omega$ are not CPS equivalent.
 2. Let $\Theta = \theta\theta$, where $\theta = \lambda xy.y(\lambda z.xxy z)$, and $\Delta = \lambda x.\delta_x\delta_x$, where $\delta_x = \lambda y.x(\lambda z.yyz)$. We have $\Theta \approx \Delta$, but Θ and Δ are not CPS equivalent.

Contextual equivalence puts all diverging terms in one equivalence class, while CPS equivalence is more discriminating. Furthermore, as is usual with equational theories for λ -calculi, CPS equivalence is not strong enough to equate Turing's and Curry's (call-by-value) fixed point combinators.

5 Conclusion

In this article, we propose a first study of the behavioral theory of a λ -calculus with delimited-control operators. We discuss various definitions of contextual equivalence, and we define an LTS-based applicative bisimilarity which is sound and complete w.r.t. the chosen contextual equivalence. Finally, we point out some differences between bisimilarity and CPS equivalence. We believe this work can be pursued in the following directions.

Up-to techniques. Up-to techniques [24,14,23] have been introduced to simplify the use of bisimulation in proofs of program equivalences. The idea is to prove terms equivalences using relations that are usually not bisimulations, but are included in bisimulations. The validity of some applicative bisimilarities up-to context remains an open problem in the λ -calculus [14]; nevertheless, we want to see if some up-to techniques can be applied to the bisimulations of this article.

Other forms of bisimilarity. Applicative bisimilarity is simpler to prove than contextual equivalence, but its definition still involves some quantification over values and pure contexts in labels. *Normal bisimilarities* are easier to use because their definitions do not feature such quantification. Lassen has developed a notion of normal bisimilarity, sound in various λ -calculi [15,16,17], and also complete in the $\lambda\mu$ -calculus with state [25]. It would be interesting to see if this equivalence can be defined in a calculus with delimited control, and if it is complete in this setting. Another kind of equivalence worth exploring is *environmental bisimilarity* [23].

Other calculi with control. Defining an applicative bisimilarity for the call-by-name variant of λ_S and for the hierarchy of delimited-control operators [8] should be straightforward. We plan to investigate applicative bisimilarities for a typed λ_S as well [4]. The problem seems more complex in calculi with abortive operators, such as call/cc. Because there is no delimiter for capture, these languages are not compositional (i.e., $t \rightarrow_v t'$ does not imply $E[t] \rightarrow_v E[t']$), which makes the definition of a compositional LTS more difficult.

Acknowledgments. We thank Małgorzata Biernacka, Daniel Hirschhoff, Damien Pous, and the anonymous referees for many helpful comments on the presentation of this work.

References

1. Abramsky, S., Ong, C.-H.L.: Full abstraction in the lazy lambda calculus. *Information and Computation* 105, 159–267 (1993)
2. Ariola, Z.M., Herbelin, H., Sabry, A.: A type-theoretic foundation of delimited continuations. *Higher-Order and Symbolic Computation* 20(4), 403–429 (2007)
3. Bierman, G.: A Computational Interpretation of the $\lambda\mu$ -Calculus. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) *MFCS 1998*. LNCS, vol. 1450, pp. 336–345. Springer, Heidelberg (1998)
4. Biernacka, M., Biernacki, D.: Context-based proofs of termination for typed delimited-control operators. In: López-Fraguas, F.J. (ed.) *PPDP 2009*, pp. 289–300. ACM Press, Coimbra (2009)
5. Biernacka, M., Biernacki, D., Danvy, O.: An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science* 1(2:5), 1–39 (2005)
6. Biernacki, D., Danvy, O., Millikin, K.: A dynamic continuation-passing style for dynamic delimited continuations. Technical Report BRICS RS-05-16, DAIMI, Department of Computer Science. Aarhus University, Aarhus, Denmark (May 2005)
7. Biernacki, D., Lenglet, S.: Applicative bisimulations for delimited-control operators (January 2012), <http://arxiv.org/abs/1201.0874>
8. Danvy, O., Filinski, A.: Abstracting control. In: Wand, M. (ed.) *LFP 1990*, pp. 151–160. ACM Press, Nice (1990)
9. David, R., Py, W.: $\lambda\mu$ -calculus and Böhm’s theorem. *Journal of Symbolic Logic* 66(1), 407–413 (2001)
10. Filinski, A.: Representing monads. In: Boehm, H.-J. (ed.) *POPL 1994*, pp. 446–457. ACM Press, Portland (1994)
11. Gordon, A.D.: Bisimilarity as a theory of functional programming. *Theoretical Computer Science* 228(1-2), 5–47 (1999)
12. Howe, D.J.: Proving congruence of bisimulation in functional programming languages. *Information and Computation* 124(2), 103–112 (1996)
13. Kameyama, Y., Hasegawa, M.: A sound and complete axiomatization of delimited continuations. In: Shivers, O. (ed.) *ICFP 2003*. SIGPLAN Notices, vol. 38(9), pp. 177–188. ACM Press, Uppsala (2003)
14. Lassen, S.B.: Relational reasoning about contexts. In: Gordon, A.D., Pitts, A.M. (eds.) *Higher Order Operational Techniques in Semantics*, pp. 91–135. Cambridge University Press (1998)
15. Lassen, S.B.: Eager normal form bisimulation. In: Panangaden, P. (ed.) *LICS 2005*, pp. 345–354. IEEE Computer Society Press, Chicago (2005)
16. Lassen, S.B.: Normal form simulation for McCarthy’s amb. In: Escardó, M., Jung, A., Mislove, M. (eds.) *MFPS 2005*. ENTCS, vol. 155, pp. 445–465. Elsevier Science Publishers, Birmingham (2005)
17. Lassen, S.B., Levy, P.B.: Typed normal form bisimulation for parametric polymorphism. In: Pfenning, F. (ed.) *LICS 2008*, pp. 341–352. IEEE Computer Society Press, Pittsburgh (2008)

18. Lenglet, S., Schmitt, A., Stefani, J.-B.: Howe's Method for Calculi with Passivation. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 448–462. Springer, Heidelberg (2009)
19. Merro, M., Biasi, C.: On the observational theory of the CPS-calculus: (extended abstract). ENTCS 158, 307–330 (2006)
20. Milner, R.: Fully abstract models of typed λ -calculi. Theoretical Computer Science 4(1), 1–22 (1977)
21. Morris, J.H.: Lambda Calculus Models of Programming Languages. PhD thesis, Massachusetts Institute of Technology (1968)
22. Parigot, M.: $\lambda\mu$ -Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS (LNAI), vol. 624, pp. 190–201. Springer, Heidelberg (1992)
23. Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. In: Marcinkowski, J. (ed.) LICS 2007, pp. 293–302. IEEE Computer Society Press, Wroclaw (2007)
24. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press (2001)
25. Støvring, K., Lassen, S.B.: A complete, co-inductive syntactic theory of sequential control and state. In: Felleisen, M. (ed.) POPL 2007. SIGPLAN Notices, vol. 42(1), pp. 161–172. ACM Press, New York (2007)
26. Thielecke, H.: Categorical Structure of Continuation Passing Style. PhD thesis. University of Edinburgh, Edinburgh, Scotland (1997) ECS-LFCS-97-376

Effective Characterizations of Simple Fragments of Temporal Logic Using Prophetic Automata

Sebastian Preugschat and Thomas Wilke

Christian-Albrechts-Universität zu Kiel
{preugschat,wilke}@ti.informatik.uni-kiel.de

Abstract. We present a framework for obtaining effective characterizations of simple fragments of future temporal logic (LTL) with the natural numbers as time domain. The framework is based on prophetic automata (also known as complete unambiguous Büchi automata), which enjoy strong structural properties, in particular, they separate the “finitary fraction” of a regular language of infinite words from its “infinitary fraction” in a natural fashion. Within our framework, we provide characterizations of all natural fragments of temporal logic, where, in some cases, no effective characterization had been known previously, and give lower and upper bounds for their computational complexity.

1 Introduction

Ever since propositional linear-time temporal logic (LTL) was introduced into computer science by Amir Pnueli in [18] it has been a major object of research. The particular line of research we are following here is motivated by the question how each individual temporal operator contributes to the expressive power of LTL. More precisely, our objective is to devise decision procedures that determine whether a given LTL property can be expressed using a given subset of the set of all temporal operators, for instance, the subset that includes “next” and “eventually”, but not “until”.

As every LTL formula interpreted in the natural numbers (the common time domain) defines a regular language of infinite words (ω -language), the aforementioned question can be viewed as part of a larger program: classifying regular ω -languages, that is, finding effective characterizations of subclasses of the class of all regular ω -languages. Over the years, many results have been established and specific tools have been developed in this program, the most fundamental result being the one that says that a regular ω -language is star-free or, equivalently, expressible in first-order logic or in LTL if, and only if, its syntactic semigroup is aperiodic [10,23,16].

The previous result is a perfect analogue of the same result for regular languages of finite words, that is, of the classical theorems by Schützenberger [19], McNaughton and Papert [13], and Kamp [10]. In general, the situation with infinite words is more complicated as with finite words; a good example for this is given in [5], where, for instance, tools from topology and algebra are used to settle characterization problems for ω -languages.

The first characterization of a fragment of LTL over finite linear orderings was given in [4], another one followed in [7], both following a simple and straightforward approach: to determine whether a formula is equivalent to a formula in a certain fragment, one computes the minimum reverse DFA for the corresponding regular language and verifies certain structural properties of this automaton, more precisely, one checks whether certain “forbidden patterns” do not occur. The first characterization for infinite words (concerning stutter-invariant temporal properties) [15] used sequential relations on ω -words; the second (concerning the nesting depth in the until/since operator) [22] used heavy algebraic machinery and did not shed any light on the computational complexity of the decision procedures involved.

In this paper, we describe a general, conceptually simple paradigm for characterizing fragments of LTL when interpreted in the natural numbers, combining ideas from [4,7] for finite words with the work by Carton and Michel on unambiguous Büchi automata [23]. The approach works roughly as follows. To determine whether a given formula is equivalent to a formula in a given fragment, convert the formula into what is called a “prophetic automaton” in [17], check that the automaton, when viewed as an automaton on finite words, satisfies certain properties, and check that languages of finite words derived from the accepting loops (“loop languages”) satisfy certain other properties. In other words, we reduce the original problem for ω -languages to problems for languages of finite words. We show that the approach works for all reasonable fragments of future LTL and yields optimal upper bounds for the complexity of the corresponding decision procedures for all but one fragment.

A note on terminology. As just explained, we work with a variant (for details, see below) of the automaton model introduced by Carton and Michel in [23] and named CUBA model (**C**omplete **U**nambiguous **B**üchi **A**utomata). In [17], Pin uses “prophetic automata” to refer to CUBA’s. We suggest to henceforth refer to these automata as “Carton–Michel automata” (CMA).

Structure of this extended abstract. In Section 2, we provide background on the topics relevant to this paper, in particular, CMA’s and propositional linear-time temporal logic. In Section 3, we explain that a translation from temporal logic into CMA’s is straightforward. In Section 4, we present our characterizations. In Section 5, we give a proof of the correctness of one of our characterizations, and in Section 6, we explain how our characterizations can be used effectively and deal with complexity issues. We conclude with open problems.

2 Basic Notation and Background

2.1 Reverse Deterministic Büchi Automata

A *Büchi automaton with a reverse deterministic transition function* is a tuple (A, Q, I, \cdot, F) where A is a finite set of symbols, Q is a finite set of states, $I \subseteq Q$ is a set of initial states, \cdot is a reverse transition function $A \times Q \rightarrow Q$, and $F \subseteq Q$ is

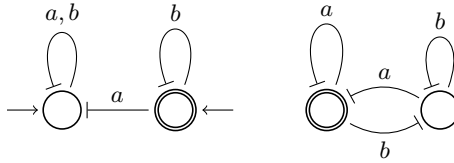


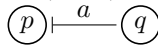
Fig. 1. CMA which recognizes $(a + b)^*b^\omega$

a set of final states. As usual, the transition function is extended to finite words by setting $\epsilon \cdot q = q$ and $au \cdot q = a \cdot (u \cdot q)$ for $q \in Q$, $a \in A$, and $u \in A^*$. For ease in notation, we write uq for $u \cdot q$ when the transition function \cdot is clear from the context.

A run of an automaton as above on an ω -word u over A is an ω -word r over Q satisfying the condition $r(i) = u(i)r(i + 1)$ for every $i < \omega$. Such a run is called *initial* if $r(0) \in I$; it is *final* if there exist infinitely many i such that $r(i) \in F$; it is *accepting* if it is initial and final. The language of ω -words recognized by such an automaton, denoted $L(\mathbf{A})$ when \mathbf{A} stands for the automaton, is the set of ω -words for which there exists an accepting run.

An automaton as above is called a *Carton–Michel automaton (CMA)* if for every ω -word over A there is exactly one final run. Such an automaton is *trim*, if every state occurs in some final run.— The original definition of Carton and Michel in [2,3] is slightly different, but for trim automata—the interesting ones—the definitions coincide.

As an example, consider the automaton depicted in Figure 1, which is a CMA for the language denoted by $(a + b)^*b^\omega$. Note that we depict $p = aq$ as



An initial state has an incoming edge \longrightarrow , a final state has a double circle \odot .

The fundamental result obtained by Carton and Michel is the following.

Theorem 1 (Carton and Michel [2,3]). *Every regular ω -language is recognized by some CMA. More precisely, every Büchi automaton with n states can be transformed into an equivalent CMA with at most $(12n)^n$ states.*

Let \mathbf{A} be a CMA over an alphabet A and $u \in A^+$. The word u is a *loop at q* if $q = uq$ and there exist $v, w \in A^*$ satisfying $vw = u$ and $wq \in F$. The set of loops at q is denoted $S(q)$. What Carton and Michel prove about loops is:

Lemma 1 (Carton and Michel [2,3]). *Let \mathbf{A} be a CMA over some alphabet A . Then, for every $u \in A^+$, there is exactly one state q , denoted $u\}$ and called anchor of u , such that u is a loop at q .*

In other words, the $S(q)$'s are pairwise disjoint and $\bigcup_{q \in Q} S(q) = A^+$.

A *generalized Carton–Michel automaton (GCMA)* is defined as expected. It is the same as a CMA except that the set F of final states is replaced by a set $\mathfrak{F} \subseteq 2^Q$ of final sets, just as with ordinary generalized Büchi automata. For such

an automaton, a run r is final if for every $F \in \mathfrak{F}$ there exist infinitely many i such that $r(i) \in F$.

The above definitions for CMA's can all be adapted to GCMA's in a natural fashion and all the statements hold accordingly. For instance, a word u is a loop at some state q in a GCMA if $q = uq$ and for every $F \in \mathcal{F}$ there exist $v, w \in A^*$ such that $u = vw$ and $wq \in F$. It is a theorem by Carton and Michel that every GCMA can be converted into an equivalent CMA.

2.2 Temporal Logic

In the following, it is understood that temporal logic refers to propositional linear-time future temporal logic where the natural numbers are used as the domain of time. For background on temporal logic, we refer to [6] and [8].

Given an alphabet A , the set of *temporal formulas over A* , denoted TL_A , is typically inductively defined by:

- (i) for every $a \in A$, the symbol a is an element of TL_A ,
- (ii) if $\varphi \in \text{TL}_A$, so is $\neg\varphi$,
- (iii) if $\varphi, \psi \in \text{TL}_A$, so are $\varphi \vee \psi$ and $\varphi \wedge \psi$,
- (iv) if $\varphi \in \text{TL}_A$, so is $X\varphi$ (“next φ ”),
- (v) if $\varphi \in \text{TL}_A$, so are $F\varphi$ and $G\varphi$ (“eventually φ ” and “always φ ”),
- (vi) if $\varphi, \psi \in \text{TL}_A$, so are $\varphi U \psi$ and $\varphi R \psi$ (“ φ until ψ ” and “ φ releases ψ ”).

Often, the operators $X\mathcal{F}$ (“strictly eventually”) and $X\mathcal{G}$ (“strictly always”) are part of the syntax of temporal logic; we view them as abbreviations of $X\mathcal{F}$ and $X\mathcal{G}$. (Obviously, F and G can be viewed as abbreviations of $(a \vee \neg a)U$ and $(a \wedge \neg a)R$, respectively.)

Formulas of TL_A are interpreted in ω -words over A . For every such word u , we define what it means for a formula to hold in u , denoted $u \models \varphi$, where we omit the straightforward rules for Boolean connectives:

- $u \models a$ if $u(0) = a$,
- $u \models X\varphi$ if $u[1, *) \models \varphi$, where, as usual, $u[1, *)$ denotes the word $u(1)u(2)\dots$,
- $u \models F\varphi$ if there exists $i \geq 0$ such that $u[i, *) \models \varphi$, similarly, $u \models G\varphi$ if $u[i, *) \models \varphi$ for all $i \geq 0$,
- $u \models \varphi U \psi$ if there exists $j \geq 0$ such that $u[j, *) \models \psi$ and $u[i, *) \models \varphi$ for all $i < j$, similarly, $u \models \varphi R \psi$ if there exists $j \geq 0$ such that $u[j, *) \models \varphi$ and $u[i, *) \models \psi$ for all $j \leq i$ or if $u[i, *) \models \psi$ for all $i \geq 0$.

Clearly, a formula of the form $\neg F\varphi$ is equivalent to $G\neg\varphi$, and a formula of the form $\neg(\varphi U \psi)$ is equivalent to $\neg\varphi R \neg\psi$, which means F and G as well as U and R are dual to each other; X is self-dual.

Given a TL_A formula φ , we write $L(\varphi)$ for the set of ω -words over A where φ holds, that is, $L(\varphi) = \{u \in A^\omega : u \models \varphi\}$. This ω -language is called the *language defined by φ* .

Fragments of LTL. An *operator set* is a subset of the set of all basic temporal operators, $\{X, F, X\mathcal{F}, U\}$. If A is an alphabet and O an *operator set*, then $\text{TL}_A[O]$ denotes all LTL formulas that can be built from A using Boolean connectives

and the operators from O . We say a language $L \subseteq A^\omega$ is O -expressible if there is a formula $\varphi \in \text{TL}_A[O]$ such that $L(\varphi) = L$. The O -fragment is the set of all LTL-formulas φ such that $L(\varphi)$ is O -expressible.

Observe that several operator sets determine the same fragment: $\{\mathbf{XF}\}$ and $\{\mathbf{F}, \mathbf{XF}\}$; $\{\mathbf{U}\}$ and $\{\mathbf{F}, \mathbf{U}\}$; $\{\mathbf{XF}, \mathbf{U}\}$ and $\{\mathbf{F}, \mathbf{XF}, \mathbf{U}\}$; $\{\mathbf{X}, \mathbf{F}\}$, $\{\mathbf{X}, \mathbf{XF}\}$ and $\{\mathbf{X}, \mathbf{F}, \mathbf{XF}\}$; $\{\mathbf{X}, \mathbf{U}\}$ and every superset of this.

What we are aiming at are decision procedures for each fragment except for the one determined by $\{\mathbf{XF}, \mathbf{U}\}$, as this is kind of unnatural: a strict operator combined with a non-strict one.

Ehrenfeucht–Fraïssé Games for LTL. The statements of our results (Section 4.2) do not involve Ehrenfeucht–Fraïssé games (EF games), but we use them extensively in our proofs. In this extended abstract, we make use of them in Section 5.

In the following, we recall the basics of EF games for temporal logic, see [7] for details.

A play of a temporal logic EF game is played by two players, Spoiler and Duplicator, on two ω -words over some alphabet A , say u and v . The game is played in rounds, where in every round, Spoiler moves first and Duplicator replies. The basic idea is that Spoiler is trying to reveal a difference between u and v which can be expressed in temporal logic, while Duplicator is trying to show—by somehow imitating the moves of Spoiler—that there is no such difference.

There are different types of rounds, corresponding to the temporal operators considered. We explain the ones that we need:

X-round. Spoiler chooses either u or v , say v , and chops off the first letter of v , that is, he replaces v by $v[1, *)$. Duplicator does the same for u .

F-round. Spoiler chooses either u or v , say v , and chops off an arbitrary finite (possibly empty) prefix, that is, he replaces v by $v[i, *)$ for some $i \geq 0$. Duplicator replaces u (the other word) by $u[j, *)$ for some $j \geq 0$.

XF-round. Spoiler chooses either u or v , say v , and chops off an arbitrary non-empty finite prefix, that is, he replaces v by $v[i, *)$ for some $i > 0$. Duplicator replaces u (the other word) by $u[j, *)$ for some $j > 0$.

Before the first round, $u(0)$ and $v(0)$ are compared. If they are distinct, then this is a win (an early win) for Spoiler. After each round, the same condition is verified, and, again, if the two symbols are distinct, then this is a win for Spoiler. If, by the end of a play, Spoiler hasn't won, then this play is a win for Duplicator. For a fixed n , Duplicator wins the n -round game, if Duplicator has a strategy to win it.

When only rounds are allowed that correspond to operators in a temporal operator set $O \subseteq \{\mathbf{X}, \mathbf{F}, \mathbf{XF}\}$, then we speak of an O -game.

The fundamental property of EF games we are going to use is the following, which was essentially proved in [7].

Theorem 2. *Let L be a language of ω -words over some alphabet A and $O \subseteq \{\mathbf{X}, \mathbf{F}, \mathbf{XF}\}$ a temporal operator set. Then the following are equivalent:*

(A) L is O -expressible.

(B) There is some k such that for all words $u, v \in A^\omega$ with $u \in L \Leftrightarrow v \in L$, Spoiler has a strategy to win the O -game on u and v within k rounds.

3 From Temporal Logic to CMA's

Several translations from temporal logic into Büchi and generalized Büchi automata are known, see, for instance, [24,21,9]. Here, we follow the same ideas and “observe” that the resulting automaton is a GCMA. This is supposed to be folklore¹ but—to the best of our knowledge—has not been made precise yet.

First note that every formula can be assumed to be in negation normal form, which means (ii) from above is not used. So, without loss of generality, we only work with formulas of this form in what follows.

Let $\varphi \in TL_A$ and let $\text{sub}(\varphi)$ denote the set of its subformulas. We define a GCMA $\mathbf{A}[\varphi] = (A, 2^{\text{sub}(\varphi)}, I, \cdot, \mathfrak{F})$. Our goal is to construct the automaton in such a way that in the unique final run r of this automaton on a given word u the following holds for every i and every $\psi \in \text{sub}(\varphi)$:

$$u[i, *) \models \psi \quad \text{iff} \quad \psi \in r(i) . \quad (1)$$

First, we set $I = \{\Phi \subseteq \text{sub}(\varphi) : \varphi \in \Phi\}$ (which is motivated directly by (I)).

Second, we define $a \cdot \Phi$ to be the smallest set Ψ satisfying the following conditions:

- (i) $a \in \Psi$,
- (ii) if $\psi \in \Psi$ and $\chi \in \Psi$, then $\psi \wedge \chi \in \Psi$,
- (iii) if $\psi \in \Psi$ or $\chi \in \Psi$, then $\psi \vee \chi \in \Psi$,
- (iv) if $\psi \in \Phi$, then $\mathbf{X}\psi \in \Psi$,
- (v) if $\psi \in \Psi$ or $\mathbf{F}\psi \in \Phi$, then $\mathbf{F}\psi \in \Psi$,
- (vi) if $\psi \in \Psi$ and $\mathbf{G}\psi \in \Phi$, then $\mathbf{G}\psi \in \Psi$,
- (vii) if $\chi \in \Psi$ or if $\psi \in \Psi$ and $\psi \mathbf{U}\chi \in \Phi$, then $\psi \mathbf{U}\chi \in \Psi$,
- (viii) if $\chi \in \Psi$ and if $\psi \in \Psi$ or $\psi \mathbf{R}\chi \in \Phi$, then $\psi \mathbf{R}\chi \in \Psi$.

This definition reflects the “local semantics” of temporal logic, for instance, $\mathbf{F}\psi$ is true now if, and only if, ψ is true now or $\mathbf{F}\psi$ is true in the next point in time. Observe, however, that the fulfillment of $\mathbf{F}\psi$ must not be deferred forever, which means that local conditions are not enough to capture the entire semantics of temporal logic. This is taken care of by the final sets.

Third, for every formula $\mathbf{F}\psi \in \text{sub}(\varphi)$ the set $\{\Phi \subseteq \text{sub}(\varphi) : \psi \in \Phi \text{ or } \mathbf{F}\psi \notin \Phi\}$ is a member of \mathfrak{F} . Similarly, for every formula $\psi \mathbf{U}\chi$ the set $\{\Phi \subseteq \text{sub}(\varphi) : \chi \in \Phi \text{ or } \psi \mathbf{U}\chi \notin \Phi\}$ is a member of \mathfrak{F} . No other set belongs to \mathfrak{F} .

Proposition 1. *Let A be an alphabet and $\varphi \in TL_A$. Then \mathbf{A}_φ is a GCMA and $L(\mathbf{A}_\varphi) = L(\varphi)$.*

¹ Personal communication of the second author with Olivier Carton: the observation can already be found in the notes by Max Michel which he handed over to Olivier Carton in the last millennium.

Proof. We first show that \mathbf{A}_φ is a GCMA. To this end, let u be an ω -word over A . We show that the word r defined by (II), for every i and every $\psi \in \text{sub}(\varphi)$, is a final run on u and the only one.

The ω -word r is a run on u . To see this, let $i \geq 0$ be arbitrary and observe that if we define Φ and Ψ by $\Phi = \{\psi \in \text{sub}(\varphi): u[i+1, *] \models \psi\}$ and $\Psi = \{\psi \in \text{sub}(\varphi): u[i, *] \models \psi\}$, then the implications (i)–(viii) not only hold, but also hold in the opposite direction. That is, $r(i) = u(i) \cdot r(i+1)$ for every i , in other words, r is a run on u .

The run r is final. Obvious from the semantics of temporal logic.

The run r is the only possible final run. Let s be a final run. We need to show $s = r$. To this end, one shows by an inductive proof on the structure of the elements of $\text{sub}(\varphi)$ that for every i and every $\psi \in \text{sub}(\varphi)$ condition (II) holds for s and hence $s = r$. The interesting part is the inductive step for formulas of the form $F\psi$ and $\psi U \chi$. We only show how the proof works for $F\psi$.

*If $F\psi \in s(i)$, then $u[i, *] \models F\psi$.* Suppose $F\psi \in s(i)$. Then, by definition of \cdot , $\psi \in s(i)$ or $F\psi \in s(i+1)$. In the first case, we have $u[i, *] \models \psi$ by the induction hypothesis, so $u[i, *] \models F\psi$ by the semantics of temporal logic, and we are done. In the second case, we can apply the same argument to the next point in time and get $u[i+1, *] \models F\psi$, hence $u[i, *] \models F\psi$, or we get $F\psi \in s(i+2)$. Continuing like this, we eventually (!) get $u[i, *] \models F\psi$ or that $F\psi \in s(j)$ for every $j \geq i$. Because s is final, one of the states of the final set for $F\psi$ must occur somewhere, that is, we get $\psi \in s(j)$ for some $j \geq i$, thus $u[j, *] \models \psi$ by induction hypothesis, hence $u[i, *] \models F\psi$.

*If $F\psi \notin s(i)$, then $u[i, *] \not\models F\psi$.* If $F\psi \notin s(i)$, then, because of the definition of \cdot , $\psi \notin s(i)$ and $F\psi \notin s(i+1)$. Continuing like this, we get $\psi \notin s(j)$ for every $j \geq i$, so, by induction hypothesis, $u[j, *] \not\models \psi$ for every $j \geq i$, hence $u[i, *] \not\models F\psi$.

Correctness of the construction, that is, $L(\varphi) = L(\mathbf{A}_\varphi)$. This follows immediately from what was shown before because of the way I is defined. \square

4 General Approach and Individual Results

This section has two purposes: it explains our general approach and presents the characterizations we have found.

4.1 The General Approach

To describe our general approach, we first need to explain what we understand by the left congruence of a (G)CMA.

Let \mathbf{A} be a (G)CMA. For every $q \in Q$, let L_q denote the set of words $u \in A^*$ such that $uq \in I$. The relation $\equiv_{\mathbf{A}}$ on Q , which we call the *left congruence of \mathbf{A}* , is defined by $q \equiv_{\mathbf{A}} q'$ when $L_q = L_{q'}$. The terminology is justified:

Remark 1. Let \mathbf{A} be a (G)CMA. Then $\equiv_{\mathbf{A}}$ is a left congruence, that is, $uq \equiv_{\mathbf{A}} uq'$ whenever $u \in A^*$ and $q, q' \in A$ are such that $q \equiv_{\mathbf{A}} q'$.

In other words, we can define the *left quotient of \mathbf{A}* with respect to $\equiv_{\mathbf{A}}$ to be the reverse semi DFA $\mathbf{A}/\equiv_{\mathbf{A}}$ given by

$$\mathbf{A}/\equiv_{\mathbf{A}} = (A, Q'/\equiv_{\mathbf{A}}, I/\equiv_{\mathbf{A}}, \circ) \tag{2}$$

where

- Q' is the set of all states that occur in some final run of \mathbf{A} (active states), and
- $a \circ (q/\equiv_{\mathbf{A}}) = (a \cdot q)/\equiv_{\mathbf{A}}$ for all $a \in A$ and $q \in Q'$.

As usual, the attribute “semi” refers to the fact that this automaton has no final states nor final sets.

Next, we combine the left congruence of a (G)CMA with its loops. The *loop language of a state q* of a (G)CMA \mathbf{A} is denoted $\text{LL}(q)$ and defined by

$$\text{LL}(q) = \bigcup_{q':q' \equiv_{\mathbf{A}} q} S(q') \ , \tag{3}$$

that is, $\text{LL}(q)$ contains all loops at q and at congruent states.

Our general approach is to characterize a fragment of LTL as follows. To check whether a given formula φ is equivalent to a formula in a given fragment, we compute the GCMA \mathbf{A}_{φ} and check various conditions on its left quotient and its loop languages. It turns out that this is sufficient; intuitively, the left quotient accounts for the “finitary fraction” of $L(\mathbf{A}_{\varphi})$, whereas the loop languages account for its “infinitary fraction”.

4.2 Characterization of the Individual Fragments

The formal statement of our main result is as follows.

Theorem 3. *Let A be some alphabet, φ an LTL-formula, and O a temporal operator set as listed in Table 1. Then the following are equivalent:*

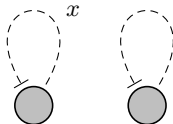
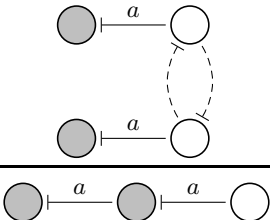
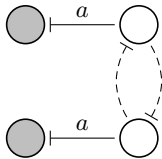
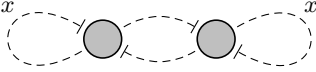

- (A) *The formula φ belongs to the O -fragment.*
- (B) *The left quotient of \mathbf{A}_{φ} and its loop languages satisfy the respective conditions listed in Table 1. (Information on how to read this table follows.)*

Conditions on the left quotient of \mathbf{A}_{φ} are phrased in terms of “forbidden patterns” (also called “forbidden configurations” in [4]). To explain this, let $\mathbf{A} = (A, Q, I, \circ)$ be any reverse semi DFA. Its *transition graph*, denoted $T(\mathbf{A})$, is the A -edge-labeled directed graph (Q, E) where $E = \{(a \circ q, a, q) : a \in A, q \in Q\}$.

Now, the conditions depicted in the second column of Table 1 are to be read as follows: the displayed graph(s) do not (!) occur as subgraphs of the transition graph of the left quotient of \mathbf{A}_{φ} , that is, as subgraphs of $T(\mathbf{A}_{\varphi}/\equiv_{\mathbf{A}_{\varphi}})$. Vertices filled gray must be distinct, the others may coincide (even with gray ones); dashed arrows stand for non-trivial paths.

For instance, the condition for the left quotient in the case of the $\{X\}$ -fragment requires that the following is not true for $T(\mathbf{A}_{\varphi}/\equiv_{\mathbf{A}_{\varphi}})$: there exist distinct states q and q' and a word $x \in A^+$ such that $q = x \circ q$ and $q' = x \circ q'$.

Table 1. Characterizations of the individual fragments of LTL

fragment	left quotient	loop languages
X		no condition
F		1-locally testable
XF		1-locally testable
X, F		locally testable
U		stutter-invariant

Note that for the {X}-fragment one forbidden pattern consisting of two strongly connected components is listed, whereas for the {F}-fragment two forbidden patterns (indicated by the horizontal line) are listed.

The conditions listed in the third column of Table 1 are conditions borrowed from formal language theory, which we explain in what follows. For a word $u \in A^*$ and $k \geq 0$, we let $\text{prf}_k(u)$, $\text{sffx}_k(u)$, and $\text{occ}_k(u)$ denote the set of prefixes, suffixes, and infixes of u of length $\leq k$, respectively. For words $u, v \in A^*$, we write $u \equiv_{k+1} v$ if $\text{prf}_k(u) = \text{prf}_k(v)$, $\text{occ}_{k+1}(u) = \text{occ}_{k+1}(v)$, and $\text{sffx}_k(u) = \text{sffx}_k(v)$. A language L is called $(k + 1)$ -locally testable if $u \in L \leftrightarrow v \in L$, whenever $u \equiv_k v$, and it is called *locally testable* if it is k -locally testable for some k , see [1].

A language $L \subseteq A^+$ is *stutter-invariant* if $uav \in L \leftrightarrow uaav \in L$ holds for all $a \in A, u, v \in A^*$.

4.3 Proof Techniques

For each fragment dealt with in Theorem 3, we have a separate proof, some of them are similar, others are completely different. In this section, we give a brief overview of our proofs.

For the operator set $\{X\}$, the proof is more or less a simple exercise, given that $\{X\}$ -expressibility means that there is some k such that $u \models \varphi$ is determined by $\text{prf}_k(u)$.

For the operator sets $\{F\}$, $\{XF\}$, and $\{X, F\}$, we use similar proofs. As an example, we treat the simplest case, $\{XF\}$, in the next section.

For $\{U\}$, we use a theorem from [14], which says that an LTL formula over some alphabet A is equivalent to a formula in $\text{TL}_A[U]$ if the language defined by the formula is stutter-invariant, where stutter invariance is defined using an appropriate notion of stutter equivalence on ω -words.

5 Characterization of the $\{XF\}$ -Fragment

We present the characterization of the $\{XF\}$ -fragment in detail. Since every GCMA can obviously be turned into an equivalent trim GCMA, all GCMA are assumed to be trim subsequently.

We start with a refined version of Theorem 3 for the $\{XF\}$ -fragment.

Theorem 4. *The following are equivalent for a given GCMA A :*

- (A) $L(A)$ is XF -expressible.
- (B) (a) The transition graph $T(A/\equiv_A)$ doesn't have a subgraph of the following form (in the above sense):



- (b) For all $u, v \in A^+$ with $\text{occ}(u) = \text{occ}(v)$, it holds that $u \vDash_A v$.
- (C) (a) The same as in (B)(a).
- (b) (i) For all $u, v \in A^*$, $a \in A$, it holds that $uav \vDash_A uav$.
- (ii) For all $u, v \in A^*$, $a, b \in A$, it holds that $uabv \vDash_A ubav$.

Observe that (B)(b) means that the loop languages are 1-locally testable. In other words, the above theorem implies that the characterization of the $\{XF\}$ -fragment given in Theorem 3 is correct.

Before we get to the proof of Theorem 4 we provide some more notation and prove some useful lemmas. For ease in notation, we often write \bar{q} for q/\equiv_A . When $u \in A^\omega$, then $u \cdot \infty$ denotes the first state of the unique final run of A on u , and $\text{inf}(u) = \{a \in A: \exists^\infty i(u(i) = a)\}$. For $a \in A$ and $u \in A^*$, $|u|_a$ denotes the number of occurrences of a in u .

Lemma 2. *Assume $\mathsf{T}(\mathbf{A}/\equiv_{\mathbf{A}})$ has a subgraph of type (T1). Then for every k there exist words $u, v \in A^\omega$ such that Duplicator wins the k -round XF -game on u and v , but $u \in \mathsf{L}(\mathbf{A}) \leftrightarrow v \in \mathsf{L}(\mathbf{A})$.*

Proof. Assume $\mathsf{T}(\mathbf{A}/\equiv_{\mathbf{A}})$ has a subgraph of type (T1). That is, there are states $\bar{p} \neq \bar{q}, \bar{r}, \bar{s}$, words $x, y \in A^+$, and a letter $a \in A$ such that $\bar{p} = a \cdot \bar{r}$, $\bar{q} = a \cdot \bar{s}$, $\bar{s} = y \cdot \bar{r}$ and $\bar{r} = x \cdot \bar{s}$. We find states r_0, r_1, \dots , and s_0, s_1, \dots such that

- $\bar{r}_i = \bar{r}$ and $\bar{s}_i = \bar{s}$ for all $i < \omega$, and
- $x \cdot s_i = r_i$ and $y \cdot r_i = s_{i+1}$ for all $i < \omega$.

Because Q is a finite set, we find $l > 0$ and i such that $r_i = r_{i+l}$. Since \mathbf{A} is trim, we find v such that $v \cdot \infty = r_i$ and u such that $ua \cdot r_i \in I$ iff $ua \cdot s_i \notin I$. This means that $ua(yx)^{lm}v \in L \leftrightarrow uax(yx)^{lm}v \in L$ for all $m \geq 1$.

Clearly, if we choose $lm > k$, then the two resulting words cannot be distinguished in the k -round XF -game. \square

Lemma 3. *Let \mathbf{A} be a GCMA such that $\mathsf{T}(\mathbf{A}/\equiv_{\mathbf{A}})$ doesn't have a subgraph of type (T1). Further, let r and s be the unique final runs of \mathbf{A} on words $u, v \in A^\omega$ and define \bar{r} and \bar{s} by $\bar{r}(i) = r(i)/\equiv_{\mathbf{A}}$ and $\bar{s}(i) = s(i)/\equiv_{\mathbf{A}}$ for all $i < \omega$.*

If $\bar{r}(0) \neq \bar{s}(0)$ and $\inf(\bar{r}) \cap \inf(\bar{s}) \neq \emptyset$, then Spoiler wins the k -round XF -game on u and v where k is twice the number of states of $\mathbf{A}/\equiv_{\mathbf{A}}$.

Proof. In the following, we use SCC as an abbreviation for strongly connected component. In our context, a state which is not reachable by a non-trivial path from itself is considered to be an SCC by itself. For every $i < \omega$, let R_i and S_i be the SCC's of $\bar{r}(i)$ and $\bar{s}(i)$ in $\mathbf{A}/\equiv_{\mathbf{A}}$, respectively. Observe that because of $\inf(\bar{r}) \cap \inf(\bar{s}) \neq \emptyset$ there is some l such that the R_i 's and S_j 's are all the same for $i, j \geq l$.

Let $\mathfrak{R} = \{R_i : i > 0\}$, $\mathfrak{S} = \{S_i : i > 0\}$, $m = |\mathfrak{R}| - 1$, and $n = |\mathfrak{S}| - 1$. We show that Spoiler wins the XF -game in at most $m + n$ rounds. The proof is by induction on $m + n$.

Base case. Let $m = n = 0$. Then $R_1 = S_1$. Because of the absence of (T1), we have $u(0) \neq v(0)$, and Spoiler wins instantly.

Induction step. Note that if r is the unique final run of \mathbf{A} on u , then $r[i, *)$ is the unique final run of \mathbf{A} on $u[i, *)$ for every i .

Let $m + n > 0$. If $u(0) \neq v(0)$, then Spoiler wins instantly. If $u(0) = v(0)$, we proceed by a case distinction as follows.

Case 1, $R_1 = S_1$. This is impossible because of the absence of (T1).

Case 2, $R_1 \neq S_1$, $R_1 \notin \mathfrak{S}$. Since $R_1 \notin \mathfrak{S}$ and $\inf(\bar{r}) \cap \inf(\bar{s}) \neq \emptyset$ we have $m > 0$. So there must be some $i \geq 1$ such that $\bar{r}(i) \in R_1$ and $\bar{r}(i+1) \notin R_1$. Spoiler chooses the word u and replaces u by $u[i, *)$.

Now Duplicator has to replace v by $v[j, *)$ for some $j > 0$. Since $R_1 \notin \mathfrak{S}$ we have $\bar{r}(i) \neq \bar{s}(j)$ and the induction hypothesis applies.

Case 3, $R_1 \neq S_1$, $S_1 \notin \mathfrak{R}$. Symmetric to Case 2.

Case 4, $R_1 \neq S_1$, $R_1 \in \mathfrak{S}$, and $S_1 \in \mathfrak{R}$. Impossible, because R_1 would be reachable from S_1 and vice versa, which would mean R_1 and S_1 coincide. \square

Lemma 4. *Let \mathbf{A} be a (G) CMA. Then the following are equivalent:*

- (A) *For all $u, v \in A^+$ with $\text{occ}(u) = \text{occ}(v)$, it holds that $u\mathfrak{J} \equiv_{\mathbf{A}} v\mathfrak{J}$.*
- (B) (a) *For all $u, v \in A^*$, $a \in A$, it holds that $uav\mathfrak{J} \equiv_{\mathbf{A}} uaav\mathfrak{J}$.*
 (b) *For all $u, v \in A^*$, $a, b \in A$, it holds that $uabv\mathfrak{J} \equiv_{\mathbf{A}} ubav\mathfrak{J}$.*

Proof. That (A) implies (B) is obvious. For the converse, let $u, v \in A^+$ with $\text{occ}(u) = \text{occ}(v)$. Let $\text{occ}(u) = \{a_0, a_1, \dots, a_n\}$. Now, we have

$$u\mathfrak{J} \equiv_{\mathbf{A}} a_0^{|u|_{a_0}} a_1^{|u|_{a_1}} \dots a_n^{|u|_{a_n}} \mathfrak{J} \equiv_{\mathbf{A}} a_0^{|v|_{a_0}} a_1^{|v|_{a_1}} \dots a_n^{|v|_{a_n}} \mathfrak{J} \equiv_{\mathbf{A}} v\mathfrak{J} ,$$

where the first and the last equivalence are obtained by iterated application of (b), and the second equivalence is obtained by iterated application of (a). \square

In what follows, we need more notation and terminology. A word $u \in A^\omega$ is an *infinite loop* at q if $q = u \cdot \infty$ and $q \in \text{inf}(r)$ where r is the unique final run of \mathbf{A} on u .

Proof of Theorem 4. The implication from (A) to (B)(a) is Lemma 2. We prove that (A) implies (B)(b) by contraposition. Assume (B)(b) does not hold, that is, there are $u, v \in A^+$ with $\text{occ}(u) = \text{occ}(v)$, and $u\mathfrak{J} \not\equiv_{\mathbf{A}} v\mathfrak{J}$. Then there exists $x \in A^*$ such that $x \cdot u\mathfrak{J} \in I \leftrightarrow x \cdot v\mathfrak{J} \in I$, that is, $xu^\omega \in L \leftrightarrow xv^\omega \in L$. It is easy to see that Duplicator wins the \mathbf{X} -game on xu^ω and xv^ω for any number of rounds, which, in turn, implies L is not \mathbf{X} -expressible.

For the implication from (B) to (A), let n be the number of states of $\mathbf{A}/\equiv_{\mathbf{A}}$. We show that whenever $u, v \in A^\omega$ such that $u \in L \leftrightarrow v \in L$, then Spoiler wins the $2n$ -round \mathbf{X} -game on u and v .

Assume $u, v \in A^\omega$ are such that $u \in L \leftrightarrow v \in L$ and let r and s be the unique final runs of \mathbf{A} on u and v , respectively, and \bar{r} and \bar{s} defined as in Lemma 3. We distinguish two cases.

First case, $\text{inf}(u) \neq \text{inf}(v)$. Then Spoiler wins within at most 2 rounds.

Second case, $\text{inf}(u) = \text{inf}(v)$. Then there are i, i' and j, j' such that

- $\text{occ}(u[i, j]) = \text{occ}(v[i', j'])$,
- $u[i, *] \cdot \infty$ is an infinite loop at $u[i, j]\mathfrak{J}$, and
- $v[i', *] \cdot \infty$ is an infinite loop at $v[i', j']\mathfrak{J}$.

From (B)(b), we conclude $u[i, j]\mathfrak{J} \equiv_{\mathbf{A}} v[i', j']\mathfrak{J}$. As a consequence, $\text{inf}(\bar{r}) \cap \text{inf}(\bar{s}) \neq \emptyset$. Since $\bar{r}(0) \neq \bar{s}(0)$, Lemma 3 applies: L is \mathbf{X} -expressible.

The equivalence between (B) and (C) follows directly from Lemma 4. \square

6 Effectiveness and Computational Complexity

To conclude, we explain how Theorem 3 can be used effectively. In general, we have:

Theorem 5. *Each of the fragments listed in Table 1 is decidable.*

Observe that for the fragment with operator set $\{\mathbf{F}, \mathbf{U}\}$, this is a result from [15], and for the fragment with operator set $\{\mathbf{X}, \mathbf{F}\}$, this is a result from [22].

Proof (of Theorem 5). First, observe that A_φ can be constructed effectively. Also, it is easy to derive the left quotient of A_φ from A_φ itself and DFA's for the loop languages, even minimum-state DFA's for them.

Second, observe that the presence of the listed forbidden patterns can be checked effectively. The test for the existence of a path between two states can be restricted to paths of length at most the number of states; the test for the existence of two paths with the same label (see forbidden patterns for $\{X\}$ and $\{X, F\}$) can be restricted to paths of length at most the number of states squared.

Third, the conditions on the loop languages can be checked effectively. For 1-local testability, this is because a language $L \subseteq A^*$ is not 1-locally testable if, and only if, one of the following conditions holds:

1. There are words $u, v \in A^*$ and there is a letter $a \in A$ such that $uav \in L \leftrightarrow uaav \in L$.
2. There are words $u \in A^*, v \in A^*$ and letters $a, b \in A$ such that $uabv \in L \leftrightarrow ubav \in L$.

Again, u and v can be bounded in length by the number of states. For local testability, we refer to [11], where it was shown this can be decided in polynomial time. For stutter invariance, remember that a language $L \subseteq A^*$ is not stutter-invariant if, and only if, the above condition 1. holds. So this can be checked effectively, too. (One could also use the forbidden pattern listed.) \square

As to the computational complexity of the problems considered, we first note:

Proposition 2. *Each of the fragments listed in Table 1 is PSPACE-hard.*

Proof. The proof is an adaptation of a proof for a somewhat weaker result given in [15]. First, recall that LTL satisfiability is PSPACE-hard for some fixed alphabet [20], hence LTL unsatisfiability for this alphabet is PSPACE-hard, too. Let A denote such an alphabet in the following.

Second, note that (because all fragments considered are proper fragments of LTL) there exists some alphabet B such that for each operator set O in question there exists an LTL formula α_O such that α_O is not O -expressible.

We can now describe a reduction from LTL unsatisfiability to the O -fragment using formulas over the alphabet $A \times B$. Let α'_O be the formula which is obtained from α by replacing every occurrence of a letter b by the formula $\bigvee_{a \in A}(a, b)$. Given a formula φ over A , we first construct $\hat{\varphi}$, where $\hat{\varphi}$ is obtained from φ by replacing every occurrence of a letter a by the formula $\bigvee_{b \in B}(a, b)$. Then φ is satisfiable iff $\hat{\varphi}$ is satisfiable iff $\hat{\varphi} \wedge \alpha'_O$ is satisfiable. Moreover, $\hat{\varphi} \wedge \alpha'_O$ cannot be expressed in the fragment in question, provided $\hat{\varphi}$ is satisfiable. Therefore, $\hat{\varphi} \wedge \alpha'_O$ is equivalent to a formula in the fragment iff φ is unsatisfiable. \square

Our upper bounds are as follows:

Theorem 6. *The $\{X, F\}$ -fragment is in E , the other fragments listed in Table 1 are in PSPACE.*

Observe that the result for the $\{U\}$ -fragment is not new, but was already obtained in [15].

Proof (sketch). Observe that each property expressed as forbidden pattern can not only be checked in polynomial time (which is folklore), it can also be checked non-deterministically in logarithmic space, even if we are given a GCMA and need to check it on its left quotient. So if we compose the construction of \mathbf{A}_φ , which has an exponential number of states, with the non-deterministic logarithmic-space tests for the existence of forbidden patterns, we obtain a polynomial-space procedure for testing the conditions on $T(\mathbf{A}_\varphi/\equiv_{\mathbf{A}_\varphi})$.

The situation is more complicated for the conditions on the loop languages. We first deal with 1-local testability and stutter invariance. Observe that from the automaton \mathbf{A}_φ we can get reverse DFA's of polynomial size in the size of \mathbf{A}_φ such that every loop language is the union of the languages recognized by these reverse DFA's. Moreover, 1. and 2. from the proof of Theorem 5 can be transferred to this context as follows. There are two states p and q in \mathbf{A}_φ that are not equivalent with respect to $\equiv_{\mathbf{A}}$ and such that one of the following conditions is true:

1. There are words $u, v \in A^*$ and there is a letter $a \in A$ such that $uav \in \text{LL}(p)$ and $uaav \in \text{LL}(q)$.
2. There are words $u, v \in A^*$ and letters $a, b \in A$ such that $uabv \in \text{LL}(p)$ and $ubav \in \text{LL}(q)$.

From this, it follows that we can bound the length of u and v polynomially in the size of \mathbf{A}_φ , which again yields polynomial-space procedures for both, 1-local testability and stutter invariance.

For (general) local testability, we apply [11] to the product of the reverse DFA's mentioned above, which yields an exponential-time algorithm all together. \square

7 Conclusion

We would like to state some questions:

1. Our lower and upper bounds for the complexity of the $\{X, F\}$ -fragment don't match. What is the exact complexity of this fragment?
2. Clearly, from our proofs it can be deduced that if a formula φ is equivalent to a formula in a fragment, an equivalent formula can be constructed effectively. What is the complexity of this construction task?
3. It is not difficult to come up with examples where every equivalent formula has exponential size (even exponential circuit size). What is the worst-case blow-up?— Observe that, in terms of circuit size, there is a polynomial upper bound for the $\{U\}$ -fragment, see [12].

References

1. Brzozowski, J.A., Simon, I.: Characterizations of locally testable events. *Discrete Math.* 4(3), 243–271 (1973)
2. Carton, O., Michel, M.: Unambiguous Büchi Automata. In: Gonnet, G.H., Panario, D., Viola, A. (eds.) *LATIN 2000*. LNCS, vol. 1776, pp. 407–416. Springer, Heidelberg (2000)

3. Carton, O., Michel, M.: Unambiguous Büchi automata. *Theor. Comput. Sci.* 297, 37–81 (2003)
4. Cohen, J., Perrin, D., Pin, J.-É.: On the expressive power of temporal logic. *J. Comput. System Sci.* 46(3), 271–294 (1993)
5. Diekert, V., Kufleitner, M.: Fragments of first-order logic over infinite words. *Theory Comput. Syst.* 48(3), 486–516 (2011)
6. Allen Emerson, E.: Temporal and modal logic. In: *Handbook of Theoretical Computer Science*, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
7. Etesami, K., Wilke, T.: An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Inf. Comput.* 160(1-2), 88–108 (2000)
8. Gabbay, D.M., Hodkinson, I., Reynolds, M.: *Temporal logic: Mathematical Foundations and Computational Aspects*, vol. 1. Clarendon Press, New York (1994)
9. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Dembinski, P., Sredniawa, M. (eds.) *Protocol Specification, Testing and Verification. IFIP Conference Proceedings*, vol. 38, pp. 3–18. Chapman & Hall (1995)
10. Kamp, H.: *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles (1968)
11. Kim, S.M., McNaughton, R., McCloskey, R.: A polynomial time algorithm for the local testability problem of deterministic finite automata. *IEEE Trans. Comput.* 40, 1087–1093 (1991)
12. Etesami, K.: A note on a question of Peled and Wilke regarding stutter-invariant LTL. *Inform. Process. Lett.* 75(6), 261–263 (2000)
13. McNaughton, R., Papert, S.A.: *Counter-free automata*. MIT Press, Boston (1971)
14. Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. *Inform. Process. Lett.* 63(5), 243–246 (1997)
15. Peled, D., Wilke, T., Wolper, P.: An algorithmic approach for checking closure properties of temporal logic specifications and ω -regular languages. *Theor. Comput. Sci.* 195(2), 183–203 (1998)
16. Perrin, D.: Recent Results on Automata and Infinite Words. In: Chytil, M., Koubek, V. (eds.) *MFCS 1984. LNCS*, vol. 176, pp. 134–148. Springer, Heidelberg (1984)
17. Perrin, D., Pin, J.-É.: *Infinite Words: Automata, Semigroups, Logic and Games. Pure and Applied Mathematics*, vol. 141. Elsevier, Amsterdam (2004)
18. Pnueli, A.: The temporal logic of programs. In: *FOCS*, pp. 46–57. IEEE (1977)
19. Schützenberger, M.P.: On finite monoids having only trivial subgroups. *Inform. and Control* 8(2), 190–194 (1965)
20. Prasad Sistla, A., Clarke, E.M.: The complexity of propositional linear temporal logics. *J. ACM* 32, 733–749 (1985)
21. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. *Inform. and Comput.* 115(1), 1–37 (1994)
22. Wilke, T.: *Classifying discrete temporal properties*. Post-doctoral thesis, Christian-Albrechts-Universität zu Kiel (1998)
23. Wolfgang, T.: Star-free regular sets of ω -sequences. *Inform. and Control* 42(2), 148–156 (1979)
24. Wolper, P., Vardi, M.Y., Prasad Sistla, A.: Reasoning about infinite computation paths (extended abstract). In: *FOCS*, pp. 185–194. IEEE (1983)

Improved Ramsey-Based Büchi Complementation

Stefan Breuers, Christof Löding, and Jörg Olschewski

RWTH Aachen University

Abstract. We consider complementing Büchi automata by applying the Ramsey-based approach, which is the original approach already used by Büchi and later improved by Sistla et al. We present several heuristics to reduce the state space of the resulting complement automaton and provide experimental data that shows that our improved construction can compete (in terms of finished complementation tasks) also in practice with alternative constructions like rank-based complementation. Furthermore, we show how our techniques can be used to improve the Ramsey-based complementation such that the asymptotic upper bound for the resulting complement automaton is $2^{\mathcal{O}(n \log n)}$ instead of $2^{\mathcal{O}(n^2)}$.

1 Introduction

The aim of this paper is to present several techniques to improve the Ramsey-based approach to complementation of Büchi automata, which was originally used by Büchi when he introduced this model of automata on infinite words to show the decidability of monadic second order logic over the successor structure of the natural numbers [2]. The method is called Ramsey-based because its correctness relies on a combinatorial result by Ramsey [10] to obtain a periodic decomposition of the possible behaviors of a Büchi automaton on an infinite word. Starting from a Büchi automaton with n states the construction yields a complement automaton with $2^{\mathcal{O}(n^2)}$ states [13]. A non-trivial lower bound of $n!$ for the complementation of Büchi automata was shown by Michel in [8]. The gap between the lower and the upper bound was made tighter by a determinization construction presented by Safra [11] from which a complementation construction with upper bound $2^{\mathcal{O}(n \log n)}$ could be derived. An elegant and simple complementation construction achieving this upper bound was presented by Klarlund in [7] using progress measures, now also referred to as ranking functions. Based on Klarlund's construction the gap between upper and lower bound has been tightened by Friedgut, Kupferman and Vardi [5] and later even further by Schewe [12], leaving only a polynomial gap compared to the improved lower bound that was obtained by Yan [16]. Besides the optimizations of the rank-based approach, a different construction has been developed by Kähler and Wilke in [6], usually called slice-based complementation, and the determinization construction of Safra has been optimized by Piterman [9].

Given all these different constructions, there has recently been an increased interest in experimental evaluations of the different complementation methods.

In experimental studies conducted by Tsai et al. [15], it turned out that the Ramsey-based approach was not only inferior to the more modern approaches (determinization-based, rank-based, and slice-based) in terms of the size of the resulting complement automata, but that in most cases its implementation did not even terminate within the imposed time and memory bounds.

Though performing inadequately for complementation, the Ramsey-based approach is used in two other fields, namely universality checking and inclusion checking [40]. For these two purposes specific optimization techniques have been developed.

Since the Ramsey-based approach to complementation has got an appealingly simple structure and is nice to teach, our aim is to improve it on both the practical level by using heuristics to reduce the size of the complement automata, and the theoretical level by using the ideas underlying our heuristics to obtain a general optimization of the method that meets the upper bound of $2^{\mathcal{O}(n \log n)}$.

We have implemented these heuristics [17] and conduct experiments on a large set of example automata, showing that the improved construction can compete with other methods for complementation.

The remainder of the paper is structured as follows. In Section 2 we start with some basic definitions and in Section 3 we repeat the original Ramsey-based complementation method. In Section 4 we present our heuristics that are used in the implementation, and we discuss our experimental results. Finally, in Section 5 we show how the ideas from Section 4 can be used to obtain an improvement of the Ramsey-based construction also on a theoretical level.

2 Preliminaries

The set of infinite words over an alphabet Σ is denoted by Σ^ω . For an infinite word $\alpha = a_0 a_1 \dots$, with $\alpha[i]$ we denote the letter a_i at position i .

An *automaton* is a tuple $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$ where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\delta: Q \times \Sigma \rightarrow 2^Q$ is the transition function and $F \subseteq Q$ is the set of accepting states. The transition function δ can be extended to a function $\delta^*: 2^Q \times \Sigma^* \rightarrow 2^Q$ on subsets of the state set and on words in the natural way. By RSC denote the set of all subsets of states that are reachable by the subset construction, i.e. $\text{RSC} := \{P \subseteq Q \mid \exists w \in \Sigma^*: \delta^*({q_0}, w) = P\}$. \mathcal{A} is called *deterministic* if $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$.

For automata we consider two different semantics: the usual NFA semantics, in which the automaton accepts a language of finite words, and the Büchi semantics, in which the automaton accepts a language of infinite words. This is made precise in the following definitions.

A *run* of \mathcal{A} on a word $\alpha \in \Sigma^\omega$ is an infinite sequence of states $\rho = p_0, p_1, \dots$ such that $p_0 = q_0$, and $p_{i+1} \in \delta(p_i, \alpha[i])$ for all $i \geq 0$. A run ρ is *Büchi-accepting* if $\rho(i) \in F$ for infinitely many $i \geq 0$. A *path* from p to q in \mathcal{A} of a word $u = a_1 \dots a_n \in \Sigma^*$ is a finite sequence of states p_0, \dots, p_n such that $p = p_0$, $q = p_n$, and $p_i \in \delta(p_{i-1}, a_i)$ for all $1 \leq i \leq n$. The path is *NFA-accepting* if $p_0 = q_0$ and $p_n \in F$.

The languages recognized by an automaton \mathcal{A} are

$$\begin{aligned} L_*(\mathcal{A}) &:= \{w \in \Sigma^* \mid \text{there is an NFA-accepting path of } w \text{ in } \mathcal{A}\} \text{ and} \\ L_\omega(\mathcal{A}) &:= \{\alpha \in \Sigma^\omega \mid \text{there is a Büchi-accepting run of } \mathcal{A} \text{ on } \alpha\}. \end{aligned}$$

If there is a path from p to q of u in \mathcal{A} , then we denote this fact by $p \xrightarrow{u} q$. If furthermore there is a path from p to q of u in \mathcal{A} that visits a final state, then we additionally denote this by $p \xrightarrow[F]{u} q$ (in particular, this is the case if p or q is final).

3 The Ramsey-Based Approach

Throughout this paper, $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$ is a fixed automaton. Our goal is to complement this automaton regarding the Büchi acceptance condition, i.e. we want to obtain an automaton \mathcal{A}' with $L_\omega(\mathcal{A}') = \Sigma^\omega \setminus L_\omega(\mathcal{A})$. In the Ramsey-based approach the complement automaton basically guesses a decomposition of the input word into finite segments such that the automaton \mathcal{A} has a specific behavior on these segments. These behaviors are captured by transition profiles. Presentations of this complementation proof can be found in [13] and in [14]. In this section we repeat the essential parts of the method that are required to describe our improvements.

A transition profile of a word u essentially describes the behavior of all states of the automaton when reading u with respect to states that are reachable and states that are reachable via a final state.

Definition 1. A pair $t = \langle \rightarrow_t, \Leftrightarrow_t \rangle$ with $\rightarrow_t \subseteq Q \times Q$ and $\Leftrightarrow_t \subseteq \rightarrow_t$ is called a transition profile over \mathcal{A} . The transition profile $\tau(u)$ of the word $u \in \Sigma^*$ over \mathcal{A} is the pair $\langle \rightarrow, \Leftrightarrow \rangle$ with $p \rightarrow q$ iff $p \xrightarrow{u} q$ and $p \Leftrightarrow q$ iff $p \xrightarrow[F]{u} q$.

One can visualize a transition profile as a directed graph. The nodes of the graph represent the states of the automaton, and there is an edge (\rightarrow) between two nodes, if the word u permits a transition from one state to the other. Additionally this edge is marked (\Leftrightarrow) if it permits a transition via a final state. A transition profile $\tau(u)$ contains information about the behavior of \mathcal{A} on u , relevant to a Büchi automaton.

We define TP to be the set of all transition profiles over \mathcal{A} . There is a natural concatenation operation on TP. For $s_1, s_2 \in \text{TP}$, the transition profile $t = s_1 \cdot s_2$ is defined by

- $p \rightarrow_t q$ iff $\exists r \in Q$ such that $p \rightarrow_{s_1} r \wedge r \rightarrow_{s_2} q$, and
- $p \Leftrightarrow_t q$ iff $\exists r \in Q$ such that $(p \Leftrightarrow_{s_1} r \wedge r \rightarrow_{s_2} q) \vee (p \rightarrow_{s_1} r \wedge r \Leftrightarrow_{s_2} q)$.

It is easy to see that the concatenation of transition profiles is associative, so $\langle \text{TP}, \cdot \rangle$, the set of all transition profiles together with concatenation, forms a semigroup. With $\tau(\epsilon)$ as the neutral element, it even is a monoid — the *transition monoid* of \mathcal{A} . Furthermore, by induction on the length of the words one can show $\tau(u \cdot v) = \tau(u) \cdot \tau(v)$, so τ is a monoid homomorphism.

For an automaton with $|Q| = n$ states, there are n^2 distinct pairs of states, and for each pair, there are only three possibilities (either $p \not\rightarrow q$, or $p \rightarrow q$ but $p \not\rightarrow q$, or $p \leftrightarrow q$). So we have exactly 3^{n^2} distinct transition profiles in the transition monoid. However, not for all of them there is a word which induces this profile. By RTP we denote the set of all *reachable* transition profiles, i.e., those $t \in \text{TP}$ for which there is a word $u \in \Sigma^*$ with $\tau(u) = t$.

For a transition profile t , define the language

$$L(t) := \tau^{-1}(t) = \{u \in \Sigma^* \mid \tau(u) = t\}.$$

The nonempty languages $L(t)$ form a partition of the set Σ^* of all words into finitely many distinct classes, for there are only finitely many transition profiles.

It is not difficult to see that for each t the language $L(t)$ is a regular language of finite words. In fact, the transition monoid of \mathcal{A} can be represented by a deterministic finite automaton that we call the *transition monoid automaton*. Each state of this automaton corresponds to a reachable transition profile, the initial state is $\tau(\epsilon)$, and the a -successor of a state t is computed by $t \cdot \tau(a)$. Constructing this automaton can be done by starting with the initial state and spawning new states doing a breadth-first search, until the automaton is complete.

We define $\text{TMA} = \langle \tilde{Q}, \Sigma, \tilde{q}_0, \tilde{\delta} \rangle$ with $\tilde{Q} = \text{RTP}$, and $\tilde{q}_0 = \tau(\epsilon)$, and $\tilde{\delta}(\tilde{q}_0, a) = \tau(a)$ and $\tilde{\delta}(t, a) = t \cdot \tau(a)$ for every $a \in \Sigma$ and $t \in \text{RTP}$. Note that TMA does not have final states. The reason is that we instantiate TMA with different sets of final states, depending on the language we want to accept: By induction on the length of words, one can show that $\tilde{\delta}^*(\tilde{q}_0, w) = \tau(w)$ and from that it follows that the language recognized by TMA with final state set $F = \{t\}$ is exactly $L(t)$.

The key observation in Büchi's proof is the following lemma. It states that each infinite word can be decomposed into finite segments such that the induced sequence of transition profiles is of the form st^ω . This is an almost direct consequence of Ramsey's theorem which states that for each coloring of (unordered) pairs over an infinite set with finitely many colors there is an infinite monochromatic subset. One can even restrict the transition profiles to those which satisfy the equations $t \cdot t = t$ (so t is idempotent) and $s \cdot t = s$.

Lemma 2 (Sequential Lemma). *For every $\alpha \in \Sigma^\omega$ there is a decomposition of α as $\alpha = uv_1v_2 \dots$ with reachable transition profiles s, t such that $\tau(u) = s$, $\tau(v_i) = t$ for every $i \geq 1$, $t \cdot t = t$, and $s \cdot t = s$.*

Define $L(\langle s, t \rangle) := L(s) \cdot L(t)^\omega$ and let

$$\text{s-t-Pairs} := \{\langle s, t \rangle \in \text{RTP}^2 \mid s \cdot t = s, t \cdot t = t\}.$$

The Sequential Lemma states that the languages $L(\langle s, t \rangle)$ cover the set of all infinite words:

$$\Sigma^\omega = \bigcup \{L(\langle s, t \rangle) \mid \langle s, t \rangle \in \text{s-t-Pairs}\}.$$

Given an arbitrary decomposition of an infinite word α into finite segments, the corresponding sequence of transition profiles contains all the relevant information

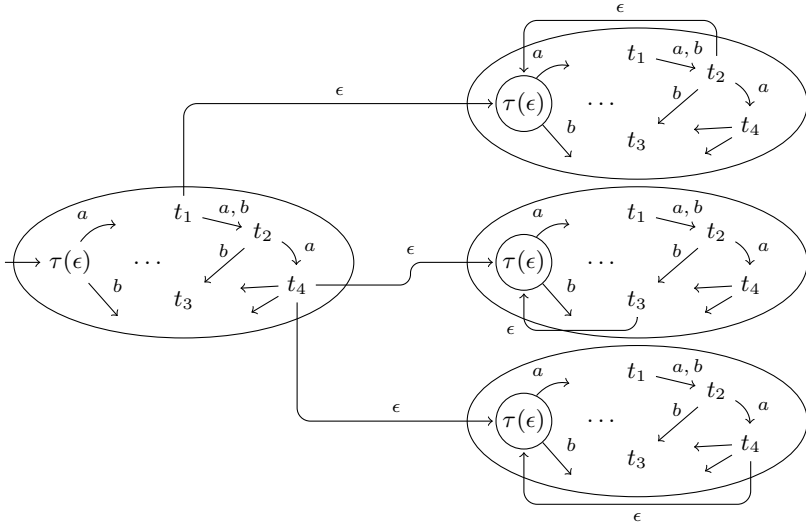


Fig. 1. A layout of a complement automaton with $\text{Reject}_{s,t} = \{\langle t_1, t_2 \rangle, \langle t_4, t_3 \rangle, \langle t_4, t_4 \rangle\}$

on possible runs of \mathcal{A} on α . This means that if two infinite words are both in $L(\langle s, t \rangle)$, they both are accepted or both are rejected.

Lemma 3. *Let s and t be transition profiles. Then either $L(\langle s, t \rangle) \cap L_\omega(\mathcal{A}) = \emptyset$, or $L(\langle s, t \rangle) \subseteq L_\omega(\mathcal{A})$.*

This allows us to separate accepting and rejecting s-t-pairs into disjoint sets:

$$\begin{aligned} \text{Accept}_{s,t} &:= \{\langle s, t \rangle \in \text{s-t-Pairs} \mid L(\langle s, t \rangle) \subseteq L_\omega(\mathcal{A})\} \text{ and} \\ \text{Reject}_{s,t} &:= \{\langle s, t \rangle \in \text{s-t-Pairs} \mid L(\langle s, t \rangle) \cap L_\omega(\mathcal{A}) = \emptyset\}. \end{aligned}$$

As a consequence we obtain the following characterization of the complement of $L_\omega(\mathcal{A})$:

$$\Sigma^\omega \setminus L = \bigcup \{L(\langle s, t \rangle) \mid \langle s, t \rangle \in \text{Reject}_{s,t}\}.$$

Using the transition monoid automaton as a basic building block one can construct a Büchi automaton for the complement. There are automata for each of the languages $L(s)$ and $L(t)$ for every rejecting s-t-pair. These can be combined to obtain an automaton for $L(\langle s, t \rangle) = L(s) \cdot L(t)^\omega$ by connecting the final states of the first automaton (for $L(s)$) to the initial state of the second one (for $L(t)$), and by allowing the second automaton to loop back to its initial state from its final state with an ϵ -transition, and making the initial state the only final state. This construction assumes that the initial state of the automaton for $L(t)$ does not have incoming transitions. Otherwise one can easily obtain this property by adding a new copy of the initial state.

In the construction it is even possible to reuse the same automaton for each of the different languages $L(s)$, as it is done in [13]. This construction is depicted in Figure 1. The ϵ -transitions used in the illustration can be eliminated by standard techniques.

We refer to the first part of the automaton that guesses the initial segment of the decomposition as the *initial part* (on the left-hand side of Figure 1), and to the remaining part of the automaton that guesses the periodic part of the decomposition as the *looping part*.

The complement automaton consists of at most 3^{n^2} Büchi automata in the looping part, each with at most $3^{n^2} + 1$ states (since we might need to add a new copy of the initial state), plus an additional initial automaton component of 3^{n^2} states. This results in a complement automaton with at most $2 \cdot 3^{n^2} + 9^{n^2}$ states.

4 Reducing State Space

In this section, we discuss several ideas to reduce the state space of the complement automaton.

4.1 Subset Construction

The first observation is that for a decomposition st^ω of an infinite word, the precise transition profile s is not required to decide whether the pair s, t is in $\text{Reject}_{s,t}$. The only information that is required on s is, which states are reachable from the initial state in s . Hence, we will replace the copy of TMA that takes care of the initial segment $L(s)$ of the decomposition by a standard subset automaton, as detailed in the following.

For $P \subseteq Q$ define $L(P) := \{u \in \Sigma^* \mid \delta^*(\{q_0\}, u) = P\}$ to be the set of all words with which from q_0 one can reach exactly the states in P . For a transition profile t , define $t(P) := \{q \in Q \mid \exists p \in P: p \rightarrow_t q\}$ to be the set of all states which are reachable from P via t . It is clear that $t(s(P)) = (s \cdot t)(P)$ for all $s, t \in \text{TP}$ and $P \subseteq Q$.

The Sequential Lemma can easily be adapted to the new setting.

Lemma 4. *For every $\alpha \in \Sigma^\omega$ there is a decomposition of α as $\alpha = uv_1v_2 \dots$ with a set $P \subseteq Q$ and a reachable transition profile t such that $u \in L(P)$, $\tau(v_i) = t$ for every $i \geq 1$, $t \cdot t = t$, and $t(P) = P$.*

Proof. We pick a decomposition according to Lemma 2 for transition profiles s and t . Setting $P = s(\{q_0\})$ one easily verifies the claimed properties. \square

So instead of considering s - t -pairs, from now on we work with P - t -pairs

$$\text{P-t-Pairs} := \{\langle P, t \rangle \in \text{RSC} \times \text{RTP} \mid t(P) = P, t \cdot t = t\},$$

and we define $L(\langle P, t \rangle) := L(P) \cdot L(t)^\omega$. Then Theorem 4 shows that the P - t -pairs again cover the set of all infinite words:

$$\Sigma^\omega = \bigcup \{L(\langle P, t \rangle) \mid \langle P, t \rangle \in \text{P-t-Pairs}\};$$

and we can divide the set of P - t -pairs into accepting and rejecting ones.

Lemma 5. *Let $P \subseteq Q$ and let t be a transition profile. Then either $L(\langle P, t \rangle) \cap L_\omega(\mathcal{A}) = \emptyset$, or $L(\langle P, t \rangle) \subseteq L_\omega(\mathcal{A})$.*

Proof. If $L(P) \cdot L(t)^\omega \cap L_\omega(\mathcal{A})$ is not empty, then there is a word α which lies in both sets. Then α can be decomposed as $\alpha = uv_1v_2 \dots$ with $u \in L(P)$ and all v_i being in $L(t)$. Because the word α is in L , there must be an accepting run of \mathcal{A} on α . Consider the form of this run to be $q_0 \xrightarrow{u} q_1 \xrightarrow{v_1} q_2 \xrightarrow{v_2} q_3 \dots$. Note that $q_1 \in P$. This run visits infinitely often an accepting state. So for infinitely many i it holds $q_i \xrightarrow[F]{v_i} q_{i+1}$.

Now let β be any word in $L(P) \cdot L(t)^\omega$. Then β can also be decomposed as $\beta = u'v'_1v'_2 \dots$ with $u' \in L(P)$ and all v'_i being in $L(t)$. We have $u' \in L(P)$ and $\tau(v_i) = \tau(v'_i)$ for all $i \geq 1$. Then there is a run ρ' of \mathcal{A} on β of the form $q_0 \xrightarrow{u'} q_1 \xrightarrow{v'_1} q_2 \xrightarrow{v'_2} q_3 \dots$. It holds $q_i \xrightarrow[F]{v'_i} q_{i+1}$ for the very same i as above. So ρ' is an accepting run and $\beta \in L_\omega(\mathcal{A})$. □

We adapt the definition of the set of accepting and rejecting pairs:

$$\begin{aligned} \text{Accept}_{P,t} &:= \{ \langle P, t \rangle \in \text{P-t-Pairs} \mid L(\langle P, t \rangle) \subseteq L_\omega(\mathcal{A}) \} \\ \text{Reject}_{P,t} &:= \{ \langle P, t \rangle \in \text{P-t-Pairs} \mid L(\langle P, t \rangle) \cap L_\omega(\mathcal{A}) = \emptyset \} \end{aligned}$$

Summarizing the above observations, we can improve the construction from Section 3 by replacing the initial copy TMA (on the left-hand side in Figure 1) by a copy of a simple automaton keeping track of the set of reachable states, denoted by PA. A set P is connected to the copy of the TMA for the transition profile t if $\langle P, t \rangle \in \text{Reject}_{P,t}$.

Note that according to the above description all pairs of the form $\langle \emptyset, t \rangle$ and $\langle P, t_\emptyset \rangle$ with the empty transition profile $t_\emptyset = \langle \emptyset, \emptyset \rangle$ are in $\text{Reject}_{P,t}$. However, all these pairs correspond to words on which no run exists at all and thus have a prefix leading from $\{q_0\}$ to \emptyset in the initial part PA of the complement automaton. In our implementation we hence make \emptyset an accepting state and do not further consider the pairs of the above form in the construction of the automaton.

4.2 Merging Transition Profiles

In this section, we show how to merge different copies of TMA in the looping part of the complement automaton. Our hope is to obtain, for the looping part, a smaller number of automata and/or automata that are smaller in terms of their state space.

As we have seen in Section 3, the automata for the right-hand part are generated by setting the acceptance component of the transition monoid automaton TMA to a singleton set. Let us denote the resulting automata by *singleton automata*. A natural way to extend this practice is to allow arbitrary subsets of the state space to be set as the acceptance component. Then the resulting automaton (considered as a *-automaton) accepts the union of the languages formerly accepted by the singleton automata, and (in the best case) the singleton automata are not needed anymore and can be removed from the right-hand part.

Obviously, this merging cannot be done in an arbitrary fashion. Below we state criteria that are sufficient for merging several of the singleton automata in the looping part.

In the following, a set of P-t-pairs is called a *bucket*. For a bucket $B = \{\langle P_1, t_1 \rangle, \dots, \langle P_n, t_n \rangle\}$, we define the language $L(B) := L^P(B) \cdot (L^t(B))^\omega$, where $L^P(B) := \bigcup_{i=1}^n L(P_i)$ and $L^t(B) := \bigcup_{i=1}^n L(t_i)$. Obviously, for a bucket B , it holds $L(B) \supseteq L(\langle P, t \rangle)$ for each $\langle P, t \rangle \in B$ (simply by definition of $L(B)$).

Now we are interested in a condition that allows to merge several rejecting pairs into a bucket B such that $L(B)$ has an empty intersection with $L(\mathcal{A})$.

Definition 6. For a bucket $B = \{\langle P_1, t_1 \rangle, \dots, \langle P_n, t_n \rangle\}$, we define its join as the pair $\langle P, t \rangle$ with $P = \bigcup_i P_i$ and $p \rightarrow_t q$ iff $\exists i: p \rightarrow_{t_i} q$, and $p \not\rightarrow_t q$ iff $\exists i: p \not\rightarrow_{t_i} q$. We say that such a pair $\langle P, t \rangle$ has a lasso if there is a sequence of states $p_1, \dots, p_k, \dots, p_n$, $1 \leq k < n$ such that

- $p_1 \in P$,
- $p_i \rightarrow_t p_{i+1}$ for all $1 \leq i < n$,
- $p_k \not\rightarrow_t p_{k+1}$, and
- $p_n \rightarrow_t p_k$.

We say that a bucket is mergeable if its join does not have a lasso.

For the join $\langle P, t \rangle$ of a bucket, note that P is not necessarily a reachable subset and that t is not necessarily a reachable transition profile.

The following lemma expresses, that by aggregating several mergeable P-t-pairs into one bucket, the language accepted by the resulting automaton is still inside the complement of $L(\mathcal{A})$.

Lemma 7. Let B be a mergeable bucket. Then $L(B) \cap L_\omega(\mathcal{A}) = \emptyset$.

Proof. Let $B = \{\langle P_1, t_1 \rangle, \dots, \langle P_n, t_n \rangle\}$ and let $\langle P, t \rangle$ be the join of B and let α be a word in $L(B)$. Then $\alpha \in \bigcup_{i=1}^n L(P_i) \cdot (\bigcup_{i=1}^n L(t_i))^\omega$ and there is an infinite sequence i_0, i_1, i_2, \dots with $1 \leq i_j \leq n$ such that α can be decomposed as $\alpha = uv_1v_2 \dots$ with $u \in L(P_{i_0})$ and $v_j \in L(t_{i_j})$ for each $j \geq 1$.

Assume that $\alpha \in L(\mathcal{A})$ and consider an accepting run ρ of \mathcal{A} on α . Consider the form of this run to be $q_0 \xrightarrow{u} q_1 \xrightarrow{v_1} q_2 \xrightarrow{v_2} q_3 \dots$, and let R be the set of those states which occur infinitely often in this form. Since $\alpha \in L(\mathcal{A})$, there is a state $q \in F$ such that q is visited infinitely often in ρ . Now we argue that $\langle P, t \rangle$ has a lasso. We have $q_1 \in P_{i_0}$ and therefore $q_1 \in P$. We have $q_i \rightarrow_t q_{i+1}$ for all $i \geq 1$. Since there are infinitely many accepting states visited in ρ , there must be a state $q_k \in R$ with $q_k \not\rightarrow_{t_{i_k}} q_{k+1}$ and therefore $q_k \not\rightarrow_t q_{k+1}$. Finally since q_k occurs infinitely often in the above form, there must be a state q_n with $n > k$ and $q_n \rightarrow_{t_{i_n}} q_k$, and therefore $q_n \rightarrow_t q_k$. So $\langle P, t \rangle$ has a lasso and this is a contradiction to the premise that B is mergeable. \square

Proposition 8. Let $\{B_1, \dots, B_n\}$ be a set of buckets such that each bucket B_i is mergeable, and for each $\langle P, t \rangle \in \text{Reject}_{P,t}$ there is a bucket B_i with $\langle P, t \rangle \in B_i$. Then $L(B_1) \cup \dots \cup L(B_n) = \Sigma^\omega \setminus L(\mathcal{A})$.

Proof. Let $\alpha \in L(B_i)$ for some $1 \leq i \leq n$. Since B_i is mergeable, by Theorem 7 it follows that $\alpha \notin L(\mathcal{A})$.

For the other direction, let $\alpha \in \Sigma^\omega \setminus L(\mathcal{A})$. By Theorem 4 and the definition of $\text{Reject}_{P,t}$, we know that there is a $\langle P, t \rangle \in \text{Reject}_{P,t}$ such that $\alpha \in L(\langle P, t \rangle)$. Then there is a bucket B_i with $\langle P, t \rangle \in B_i$ and since $L(B_i) \supseteq L(\langle P, t \rangle)$ it follows that $\alpha \in L(B_i)$. \square

Given a set $\{B_1, \dots, B_n\}$ of buckets as in Proposition 8, and automata $\mathcal{A}_i = \langle Q_i, \Sigma, q_0^i, \delta_i, F_i \rangle$ for the languages $L^t(B_i)$, we can now further modify the construction from Section 3 by connecting for each pair $\langle P, t \rangle \in B_i$ the set P from the initial PA to the automaton \mathcal{A}_i for $L^t(B_i)$, and applying the looping construction on the \mathcal{A}_i . This is done formally in the following definition, where we do not use ϵ -transitions for the connections and loops as in Figure 1 but directly eliminate them.

Definition 9. Let $\{B_1, \dots, B_n\}$ be a set of buckets as in Proposition 8 and for each $1 \leq i \leq n$ let $\mathcal{A}_i = \langle Q_i, \Sigma, q_0^i, \delta_i, F_i \rangle$ be an automaton such that $L_*(\mathcal{A}_i) = L^t(B_i)$. Furthermore assume that the initial states of the \mathcal{A}_i do not have incoming transitions. Then define the automaton $\mathcal{A}' := \langle Q', \Sigma, q_0', \delta', F' \rangle$ with

- $Q' = \text{RSC} \cup \bigcup_i Q_i$,
- $q_0' = \{q_0\}$,
- $F' = \{\emptyset\} \cup \{q_0^1, \dots, q_0^n\}$, and
- $\delta'(P, a) = \{\delta^*(P, a)\} \cup \{q \in Q' \mid \exists i(\exists \langle P, t \rangle \in B_i \text{ for some } t \wedge q \in \delta_i(q_0^i, a))\}$ for $P \subseteq Q$ and $a \in \Sigma$, and
- $\delta'(q, a) = \begin{cases} \delta_i(q, a) & \text{if } \delta_i(q, a) \cap F_i = \emptyset \\ \{q_0^i\} \cup \delta_i(q, a) & \text{otherwise} \end{cases}$ for $q \in Q_i$ and $a \in \Sigma$.

It is easy to see that $L_\omega(\mathcal{A}') = L(B_1) \cup \dots \cup L(B_n)$ and by Proposition 8 we obtain that \mathcal{A}' is an automaton for the complement of $L(\mathcal{A})$.

In the above definition the automata for $L^t(B_i)$ are parameters and we did not specify how to construct them. Since for a bucket B , it holds $L^t(B) = \bigcup_{\langle P, t \rangle \in B} L^t(t)$, one can use the transition monoid automaton TMA, setting all the states t to be final for which some pair $\langle P, t \rangle$ is in B . This is how we proceed in our implementation. In Section 5 we work with specific buckets and for those provide an alternative construction that allows us to give a better upper bound on the size of the resulting automata.

Minimizing t-Automata. The automata for the languages $L^t(B)$ that are obtained from TMA, as described above, are deterministic. As these automata are very large and have only few accepting states, they likely have many redundant states, too. So a natural approach here is to minimize these automata. After generating a bucket automaton $\mathcal{A}^t(B)$, our algorithm immediately computes the minimal equivalent deterministic automaton, which in our experiments often results in much smaller automata for the looping part of the complement automaton.

4.3 Experimental Results

In order to compare our complementation method with existing ones, we tried to reflect the experimental setting of Tsai et al. [15]. They compared four different implementations, named **Ramsey**, **Safra-Piterman**, **Rank**, and **Slice**. Each of these implement one of the four approaches, Ramsey-based, determinization-based, rank-based, and slice-based, respectively. They randomly generated 11,000 input automata, each with 15 states, an alphabet of size 2, and combinations of 11 transition densities and 10 acceptance densities. Then they fed these complementation tasks to a cluster of computers. For each task, they allocated a 2.83 GHz CPU with 1 GB of memory and 10 minutes computation time. Since **Ramsey** performed very poorly in their experiments, they excluded this implementation from the overall comparison. The other three implementations were then improved by various heuristics. In the end, the improved versions of **Safra-Piterman**, **Rank**, and **Slice** had 4 tasks, 3,383 tasks, and 216 tasks that aborted unsuccessfully, respectively.

We have implemented our ideas in a Java program. It can be obtained from [17], including its Java source code. Because there is no unique distribution of P-t-pairs to buckets, we had to agree on a concrete way to fill the buckets. We have chosen the following greedy algorithm. Maintain a list (B_1, \dots, B_n) of buckets, which can grow if needed. For each pair $\langle P, t \rangle \in \text{Reject}_{P,t}$, add $\langle P, t \rangle$ to the first bucket B_i such that $B_i \cup \{\langle P, t \rangle\}$ is mergeable. If no such bucket exists, then create a new empty bucket B_{n+1} and start over again. The algorithm that uses all of the above heuristics, the subset construction for the initial part, the merging of transition profiles for the looping part, and the minimization of the bucket automata, together with the greedy bucket filling algorithm, is called **improved-Ramsey**.

Our complementation jobs were conducted in sequence on a single machine with a 2.83 GHz CPU, a timeout of 10 minutes, and 4 GB of memory from which only 1 GB was used for the Java heap space. We used the same 11,000 complementation tasks as Tsai et al. Out of these, 10,839 finished successfully, 152 ran out of memory, and 9 violated the time limit. In terms of successfully finished tasks, this puts **improved-Ramsey** second place, between **Safra-Piterman** and **Slice**.

The size of the complement automata computed by **improved-Ramsey** range from 0 to 337,464 states with an average size of 361.09 states (328.97 after removing dead states). We were not able to adequately compare these sizes with the results of [15] for the following reason. Tsai et al. provide average sizes only for 7,593 of the initial 11,000 automata, namely for those tasks that finished successfully by all of their implementations. Our numbers, on the other hand, base upon all 10,839 finished tasks of our implementation.

5 Preorder-Based Optimization

The aim of this section is to improve the Ramsey-based construction in such a way that the resulting complement automaton is of size $2^{\mathcal{O}(n \log n)}$. There are

two things we have to take care of: The complement automaton is composed of the initial subset automaton and the part for the ω -iteration, in which for each transition profile there is one copy of the transition monoid automaton. To reduce the number of states we have to reduce (1) the number of copies, and (2) the size of these automata.

In Section 4.2 we have seen that we can merge transition profiles in the looping part of the complement automaton, as long as the combination of the merged transition profiles does not introduce an accepting cycle. To obtain an automaton for the complement it is sufficient to cover $\text{Reject}_{P,t}$ by mergeable buckets of transition profiles.

In this section we show how to systematically do this merging such that the number of buckets, and the respective size of the automata for the buckets are of order $2^{\mathcal{O}(n \log n)}$. The key idea is that we can merge all transition profiles that can be embedded into the same preorder in such a way that \Rightarrow edges are strictly increasing in the order and \rightarrow edges are not decreasing, thus avoiding accepting cycles. This is made precise in the following definitions.

A *total preorder* (or *weak order*) on $P \subseteq Q$ is a binary relation \lesssim on P that is total (for all $p, q \in P$ it holds $p \lesssim q$ or $q \lesssim p$), and transitive (for all $p, q, r \in P$ with $p \lesssim q$ and $q \lesssim r$ it holds $p \lesssim r$). With Pre denote the set of all pairs $\langle P, \lesssim \rangle$ such that $P \subseteq Q$ and \lesssim is a total preorder on P .

For any total preorder \lesssim , one can define its corresponding equivalence relation by $p \simeq q \Leftrightarrow p \lesssim q \wedge q \lesssim p$. The resulting equivalence classes are linearly ordered by $[p] \leq [q] \Leftrightarrow p \lesssim q$. As usual one defines $[p] < [q]$ if $[p] \leq [q] \wedge [q] \not\leq [p]$.

Note that the number of preorders on a set with n elements is bounded by n^n because each preorder can be characterized by a mapping that assigns to each element of the set a number from 1 to n corresponding to its position in the order (equivalent elements are mapped to the same number). So the number of pairs $\langle P, \lesssim \rangle$ is bounded by $2^n n^n$.

Definition 10. Let $\langle P, \lesssim \rangle \in \text{Pre}$. We say that a transition profile $t = \langle \rightarrow, \Rightarrow \rangle$ is compatible with $\langle P, \lesssim \rangle$ if

- $t(P) \subseteq P$, and
- for all $p, q \in P$ with $p \rightarrow q$ it holds $[p] \leq [q]$, and
- for all $p, q \in P$ with $p \Rightarrow q$ it holds $[p] < [q]$.

For any preordered set $\langle P, \lesssim \rangle$, let us define the bucket

$$B_{\langle P, \lesssim \rangle} := \{ \langle P, t \rangle \in \text{RSC} \times \text{RTP} \mid t \text{ is compatible with } \langle P, \lesssim \rangle \}.$$

It is not difficult to see that these buckets are mergeable in the sense of Section 4.2 and that each pair in $\text{Reject}_{P,t}$ is compatible with a suitable preorder.

Lemma 11. $B_{\langle P, \lesssim \rangle}$ is mergeable for each $\langle P, \lesssim \rangle \in \text{Pre}$ and for each $\langle P, t \rangle \in \text{Reject}_{P,t}$, there is a total preorder \lesssim on P with $\langle P, t \rangle \in B_{\langle P, \lesssim \rangle}$.

Proof. We start with the first claim. Assume the join of $B_{\langle P, \lesssim \rangle}$ has a lasso. Then there is a sequence of states $p_1, \dots, p_k, \dots, p_n \in Q$, $1 \leq k < n$ with

$p_1 \in P$, $p_i \rightarrow p_{i+1}$ for all $i < n$, $p_k \Leftrightarrow p_{k+1}$, and $p_n \rightarrow p_k$. Since $t(P) \subseteq P$ for all t compatible with $\langle P, \lesssim \rangle$, it holds $p_i \in P$ for all $1 \leq i \leq n$ by induction on i . Then we have $[p_{k+1}] \leq [p_{k+2}] \leq \dots \leq [p_n] \leq [p_k]$, and $[p_k] < [p_{k+1}]$, which is a contradiction.

To prove the second claim, note that by definition of $\text{Reject}_{P,t}$ it holds $t(P) \subseteq P$. Consider the directed graph G with vertex set P and edge relation \rightarrow_t . The SCCs of G are preordered by $C_1 R C_2$ iff there is a path from a state $p \in C_1$ to a state $q \in C_2$ in G . We make this preorder total by ordering incomparable SCCs of G in an arbitrary way. We obtain a total preorder R' on the set of SCCs. This induces a total preorder \lesssim on P by $p \lesssim q$ iff $C_p R' C_q$ for $p \in C_p$ and $q \in C_q$. Clearly $p \rightarrow_t q$ implies $[p] \leq [q]$. Let $C \subseteq P$ be an SCC of G . Then for states $p_1, p_2 \in C$, it cannot hold $p_1 \Leftrightarrow_t p_2$, as otherwise we can construct an accepting run $q_0 \xrightarrow{u} p_1 \xrightarrow{v_1/F} p_2 \xrightarrow{v_2} p_3 \rightarrow \dots p_n \xrightarrow{v_n} p_1 \dots$ of \mathcal{A} on a word $u(v_1 \dots v_n)^\omega$ with $u \in L(P)$ and $v_i \in L(t)$. So $p_1 \Leftrightarrow_t p_2$ implies $[p_1] < [p_2]$. Altogether, t is compatible with $\langle P, \lesssim \rangle$, and thus $\langle P, t \rangle \in B_{\langle P, \lesssim \rangle}$. \square

According to the above lemma we have already found a covering of $\text{Reject}_{P,t}$ by a number of buckets that is bounded by $2^n n^n$. It remains to show that for a given preorder $\langle P, \lesssim \rangle$ the language $L^t(B_{\langle P, \lesssim \rangle})$, that is, those words whose transition profile is compatible with $\langle P, \lesssim \rangle$, can be recognized by a “small” automaton.

We realize this as follows. When reading a word v , the automaton keeps track for each $q \in Q$ which are the maximal equivalence classes $[p']$ and $[p'']$ of $\langle P, \lesssim \rangle$ such that in $\tau(v)$ there is an \rightarrow edge from an element of $[p']$ to q and there is an \Leftrightarrow edge from an element of $[p'']$ to q . This information can easily be updated when appending a new letter to v , and Lemma 13 shows that it suffices to deduce whether $\tau(v)$ is compatible with $\langle P, \lesssim \rangle$.

To formalize this idea let $\mathcal{M}_{\langle P, \lesssim \rangle}$ be the set of all functions $f: Q \rightarrow P_\perp$, where $P_\perp = \{\perp\} \cup (P/\simeq)$ and the linear order \leq on P/\simeq is extended to P_\perp by setting $\perp < [p]$ for all $[p] \in P/\simeq$.

The states of the automaton are pairs of such mappings. The initial state is the pair $\langle \phi_\epsilon, \psi_\epsilon \rangle \in \mathcal{M}_{\langle P, \lesssim \rangle} \times \mathcal{M}_{\langle P, \lesssim \rangle}$ with

$$\phi_\epsilon(q) := \begin{cases} [q] & \text{if } q \in P, \\ \perp & \text{else;} \end{cases} \quad \text{and} \quad \psi_\epsilon(q) := \begin{cases} [q] & \text{if } q \in P \cap F, \\ \perp & \text{else.} \end{cases}$$

For two functions $\phi, \psi \in \mathcal{M}_{\langle P, \lesssim \rangle}$ and for a letter $a \in \Sigma$, we define the update of ϕ and ψ by letter a to be $\langle \phi', \psi' \rangle \cdot a := \langle \phi', \psi' \rangle$ with

$$\begin{aligned}
 \phi'(q) &= \max\{\phi(r) \mid r \in Q, r \xrightarrow{a} q\}, \text{ and} \\
 \psi'(q) &= \begin{cases} \max\{\phi(r) \mid r \in Q, r \xrightarrow{a} q\} & \text{if } q \in F, \\ \max\{\psi(r) \mid r \in Q, r \xrightarrow{a} q\} & \text{else.} \end{cases}
 \end{aligned}$$

We use the convention $\max \emptyset = \perp$. Note that the definition depends on the context $\langle P, \lesssim \rangle$ in which it is used.

We write $\langle \phi_a, \psi_a \rangle$ for $\langle \phi_\epsilon, \psi_\epsilon \rangle \cdot a$, and inductively we write $\langle \phi_{va}, \psi_{va} \rangle$ for $\langle \phi_v, \psi_v \rangle \cdot a$. The function ϕ_v maps every state q to the maximal class in

P/\simeq from which one can reach q by reading v ; it maps to $\max \emptyset = \perp$ if no such class exists. Accordingly, the function ψ_v maps every state q to the maximal class from which one can reach q passing an accepting state by reading v ; it maps to \perp if no such class exists. We formalize this in the following lemma.

Lemma 12. *Let $\langle P, \lesssim \rangle \in \text{Pre}$ and $v \in \Sigma^*$. Then $\phi_v(q) = \max\{[p] \mid p \in P, p \xrightarrow{v} q\}$ and $\psi_v(q) = \max\{[p] \mid p \in P, p \xrightarrow{F} q\}$ for each $q \in Q$.*

To define the set of final states of the automaton we have to identify those pairs of functions that indicate whether the transition profile of the current word is compatible with $\langle P, \lesssim \rangle$.

- Let $\phi, \psi \in \mathcal{M}_{\langle P, \lesssim \rangle}$. We say that the pair $\langle \phi, \psi \rangle$ is *consistent with $\langle P, \lesssim \rangle$* if
- for all $q \in Q \setminus P$ it holds $\phi(q) = \perp$, and
 - for all $q \in P$ it holds $\phi(q) \leq [q]$, and
 - for all $q \in P$ it holds $\psi(q) < [q]$.

Lemma 13. *Let $\langle P, \lesssim \rangle \in \text{Pre}$ and $v \in \Sigma^*$. Then the transition profile $\tau(v)$ is compatible with $\langle P, \lesssim \rangle$ iff $\langle \phi_v, \psi_v \rangle$ is consistent with $\langle P, \lesssim \rangle$.*

Proof. Let $t = \tau(v)$. With Theorem 12 we obtain

$$\begin{aligned} t(P) \subseteq P &\iff \forall q \in Q \setminus P \neg \exists p \in P : p \xrightarrow{v} q \\ &\iff \forall q \in Q \setminus P : \phi_v(q) = \perp, \text{ as well as} \end{aligned}$$

$$\begin{aligned} \forall p, q \in P : (p \xrightarrow{v} q \Rightarrow [p] \leq [q]) &\iff \forall q \in P : \max\{[p] \mid p \in P, p \xrightarrow{v} q\} \leq [q] \\ &\iff \forall q \in P : \phi_v(q) \leq [q], \text{ and} \end{aligned}$$

$$\begin{aligned} \forall p, q \in P : (p \xrightarrow{F} q \Rightarrow [p] < [q]) &\iff \forall q \in P : \max\{[p] \mid p \in P, p \xrightarrow{F} q\} < [q] \\ &\iff \forall q \in P : \psi_v(q) < [q]. \end{aligned}$$

□

Combining the above observations we can define the deterministic automaton $\mathcal{A}_{\langle P, \lesssim \rangle} := \langle Q'', \Sigma, q_0'', \delta'', F'' \rangle$ as follows:

- $Q'' = \{\langle P, \lesssim, \phi, \psi \rangle \mid \phi, \psi \in \mathcal{M}(P)\}$
- $q_0'' = \langle P, \lesssim, \phi_\epsilon, \psi_\epsilon \rangle$
- $\delta''(\langle P, \lesssim, \phi, \psi \rangle, a) = \{\langle P, \lesssim, \phi', \psi' \rangle\}$ where $\langle \phi', \psi' \rangle = \langle \phi, \psi \rangle \cdot a$
- $F'' = \{\langle P, \lesssim, \phi, \psi \rangle \mid \langle \phi, \psi \rangle \text{ is consistent with } \langle P, \lesssim \rangle\}$

As a consequence of Lemma 13 we obtain the following result.

Lemma 14. *For every $\langle P, \lesssim \rangle$, the automaton $\mathcal{A}_{\langle P, \lesssim \rangle}$ accepts those words for which the transition profile is compatible with $\langle P, \lesssim \rangle$, that is, $L_*(\mathcal{A}_{\langle P, \lesssim \rangle}) = L^t(B_{\langle P, \lesssim \rangle})$.*

Now we can plug the automata $\mathcal{A}_{\langle P, \lesssim \rangle}$ into the construction from Definition 9. The results from this section in combination with Proposition 8 imply that the resulting automaton indeed recognizes the complement language of the original automaton.

Theorem 15. *Applied to a Büchi automaton with n states, the Ramsey-based method in combination with preorder merging of transition profiles yields a complement automaton with at most $2^n + 2^n n^n ((n + 1)^{2^n} + 1)$ states.*

Proof. As mentioned above, the number of pairs $\langle P, \lesssim \rangle$ is bounded by $2^n n^n$.

The number of states in $\mathcal{A}_{\langle P, \lesssim \rangle}$ is not greater than $(n + 1)^{2^n}$. For applying the ω -iteration to these automata a new initial state has to be introduced.

The initial part of the complement automaton consists of a subset automaton. Altogether this gives the claimed bound. \square

6 Conclusion

We proposed several heuristics to improve the Ramsey-based Büchi complementation method. We implemented these heuristics and showed that, in practice, our implementation can compete with implementations of other complementation methods. We introduced a novel optimization of the Ramsey-based method using preorders, with a $2^{\mathcal{O}(n \log n)}$ upper bound. From this and from the fact that the improved Ramsey-based approach yields good experimental results, we conclude that the Ramsey-based approach still has its place, in contrast to the results in [15].

Although the preorder-based optimization results in a complement automaton for which we can prove a better upper bound, our implementation uses a different strategy (the greedy strategy described in Section 4.3) to construct the buckets. The reason is that the greedy strategy easily allows to restrict to reachable transition profiles, while the preorder based approach does not allow this (at least not directly). Therefore, the improved Ramsey-based method still has to compute the entire reachable part of the transition monoid, and we are not aware of any upper bound on its size better than 3^{n^2} . However, the experiments suggest that in many cases the reachable part of the monoid is much smaller than the worst case.

We see our paper also in the broader context of comparing and unifying different complementation methods. Only recently, Fogarty et al. [3] compared the rank-based with the slice-based method. They combined both approaches and obtained, utilizing a total preorder on the nodes in a run DAG, a unified complementation method. We would not be surprised if the improved Ramsey-based method could also be unified with one of the other methods. This is one way, of extending our work. A second direction of future work could investigate whether the heuristics of our work can be used for universality and inclusion checking of Büchi automata.

Acknowledgments. We thank the authors of [15] for providing the 11,000 example automata, and the anonymous referees for their helpful comments.

References

1. Abdulla, P.A., Chen, Y.-F., Clemente, L., Holík, L., Hong, C.-D., Mayr, R., Vojnar, T.: Advanced Ramsey-based Büchi Automata Inclusion Testing. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011 – Concurrency Theory. LNCS, vol. 6901, pp. 187–202. Springer, Heidelberg (2011)
2. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1962)
3. Fogarty, S., Kupferman, O., Vardi, M.Y., Wilke, T.: Unifying Büchi complementation constructions. In: CSL (2011)
4. Fogarty, S., Vardi, M.Y.: Efficient Büchi Universality Checking. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 205–220. Springer, Heidelberg (2010)
5. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. International Journal of Foundations of Computer Science 17(4), 851–868 (2006)
6. Kähler, D., Wilke, T.: Complementation, Disambiguation, and Determinization of Büchi Automata Unified. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 724–735. Springer, Heidelberg (2008)
7. Klarlund, N.: Progress measures for complementation of ω -automata with applications to temporal logic. In: FOCS, pp. 358–367. IEEE Computer Society (1991)
8. Michel, M.: Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris (1988)
9. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science 3(3) (2007)
10. Ramsey, F.P.: On a problem of formal logic. Proceedings of the London Mathematical Society 2(1), 264 (1930)
11. Safra, S.: On the complexity of ω -automata. In: FOCS, pp. 319–327. IEEE (1988)
12. Schewe, S.: Büchi complementation made tight. In: STACS. LIPIcs, vol. 3, pp. 661–672. Schloss Dagstuhl (2009)
13. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science 49, 217–237 (1987)
14. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 133–192. Elsevier Science Publishers, Amsterdam (1990)
15. Tsai, M.-H., Fogarty, S., Vardi, M.Y., Tsay, Y.-K.: State of Büchi Complementation. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 261–271. Springer, Heidelberg (2011)
16. Yan, Q.: Lower bounds for complementation of ω -automata via the full automata technique. Logical Methods in Computer Science 4(1) (2008)
17. <http://www.automata.rwth-aachen.de/research/Alekto/>

Extending \mathcal{H}_1 -Clauses with Path Disequalities

Helmut Seidl and Andreas Reuß*

Technische Universität München, Institut für Informatik I2, Boltzmannstraße 3,
D-85748 Garching, Germany

Abstract. We extend \mathcal{H}_1 -clauses with disequalities between paths. This extension allows conveniently to reason about freshness of keys or nonces, as well as about more intricate properties such as that a voter may deliver at most one vote. We show that the extended clauses can be normalized into an equivalent tree automaton with path disequalities and therefore conclude that satisfiability of conjunctive queries to predicates defined by such clauses is decidable.

1 Introduction

In general, satisfiability of a finite conjunction of Horn clauses is undecidable. In [12,8], a class \mathcal{H}_1 of Horn clauses has been identified for which satisfiability is decidable in exponential time. The class \mathcal{H}_1 differs from general Horn clauses in that it requires the heads of clauses to contain at most one constructor and to be linear, i.e., no variable may occur twice in the head. Since any finite conjunction of Horn clauses can be approximated by means of \mathcal{H}_1 -clauses, the decision procedure for \mathcal{H}_1 can be used for automatic program analysis based on abstract interpretation, given that the analysis problem can conveniently be described by means of Horn clauses. This approach has been successfully applied to the automatic analysis of secrecy in cryptographic protocols and their implementation [7].

In [11], the class \mathcal{H}_1 of Horn clauses has been extended with *disequality* constraints on terms. Such kinds of constraints allow to express that a key is *fresh*, i.e., different from all keys in a given list.

Example 1. The \mathcal{H}_1 -clauses

$$\begin{aligned} \mathit{fresh_key}(X) &\leftarrow q(X, []) \\ q(X, Z) &\leftarrow q(X, :: (Y, Z)), X \neq Y \\ q(X, Y) &\leftarrow \mathit{key}(X), \mathit{old_keys}(Y) \end{aligned}$$

define a predicate *fresh_key* which expresses that a key is not contained in the list *old_keys*. Here, the upper case letters represent variables, [] denotes an empty list, and :: is the list constructor. The definition of the predicate *q* ensures that (within the least model) $q(t, [])$ only holds if $q(t, l)$ holds for a list *l* where *t* is not contained in *l*. \square

In [11], we extended the normalization procedure for \mathcal{H}_1 from [8,6] to clauses with term disequality constraints. This procedure transforms every finite set of \mathcal{H}_1 -clauses

* The author was supported by the DFG Graduiertenkolleg 1480 (PUMA).

with term disequality constraints into an equivalent finite set of *automata clauses* with disequality constraints [9] and thus allows to decide whether or not a given query is satisfiable.

Disequalities between terms, however, are not expressive enough for expressing more involved properties of cryptographic protocols than freshness of keys or nonces. Consider, e.g., a voting protocol where for a submitted vote not freshness of the overall expression must be checked but that the voter has not submitted any vote so far [4,1].

Example 2. Assume that we are given a list l of votes where each element of l is of the form $vote(p, v)$ for a constructor symbol $vote$, a person p who has voted, and his or her vote v . The next vote $vote(p', v')$ then should not only be not contained in the list l but should differ from all elements of l in the first argument. This can be expressed by the predicate *valid* as defined by:

$$\begin{aligned} \text{valid}(X) &\leftarrow q(X, []) \\ q(X, Z) &\leftarrow q(X, :: (Y, Z)), X.1 \neq Y.1 \\ q(X, Y) &\leftarrow \text{is_vote}(X), \text{votes}(Y) \end{aligned}$$

The second clause of this definition involves a comparison between the person trying to vote, identified by $X.1$, and the leftmost subterm of the first list entry, identified by $Y.1$, which denotes one of the persons who have already voted. \square

Disequalities between subterms identified by paths, though, turn out to be provably more expressive than disequalities between terms. Finite tree automata with disequality constraints between paths have extensively been studied by Comon and Jacquemard [2] who showed that emptiness for these automata is decidable. Moreover, they applied this emptiness test to provide a DEXPTIME algorithm for deciding inductive reducibility [3]. While for tree automata with term disequalities, universality is decidable [9] — this problem is undecidable for tree automata with disequalities between paths [5].

In this paper, we consider \mathcal{H}_1 -clauses with disequalities between paths and try to provide a normalization procedure in the spirit of [8,11] to construct for every finite set of clauses an equivalent automaton with disequalities between paths. The construction, turns out to be more intricate than the construction for \mathcal{H}_1 -clauses with disequalities between terms. The reason is that now extra precautions must be taken that only finitely many different clauses are encountered during normalization. The key issue is to avoid that the lengths of paths occurring in constraints grow. In order to deal with that, we extend the language of constraints by additionally allowing disequalities between arbitrary terms which, instead of plain variables, contain *path queries* to variables. Another problem arises when a clause is to be split in order to remove variables which do not occur in the head. In this particular case of quantifier elimination, pigeon-hole like arguments as applied in [11] do no longer suffice. Instead, we develop a novel technique which is based on *blind exploration*.

The rest of the paper is organized as follows. Section 2 contains basic notions of paths and constraints, while Section 3 introduces subclasses of Horn clauses extended with disequality constraints. Section 4 then compares classes of automata extended with disequalities with respect to their expressiveness. In Section 5 we provide the normalization procedure for \mathcal{H}_1 -clauses with path disequalities. Finally Section 6 concludes.

2 Basics

In the following, we consider a fixed finite ranked alphabet Σ of atoms (with arity 0) and constructor symbols (with arities ≥ 1). In order to avoid trivial cases, we assume that there is at least one atom and at least one constructor. Let \mathcal{T}_Σ denote the set of all *ground* (i.e., variable-free) terms over Σ . Note that, according to our assumption on Σ , \mathcal{T}_Σ is infinite. A *labeled path* π is a finite sequence $(f_1, i_1).(f_2, i_2).\dots.(f_n, i_n)$ where for every j , $f_j \in \Sigma$ and $1 \leq i_j \leq m_j$ if m_j is the arity of f_j . As usual, the empty sequence is denoted ϵ . An expression $X.\pi$ for a variable X and a path π is called *path expression*. In case, $\pi = \epsilon$, we also write X for $X.\epsilon$.

A *general term* is built up from path expressions and nullary constructors by means of constructor application. For a general term t , the sub-term at path π , denoted by $t.\pi$, is recursively defined by:

- $t.\epsilon = t$;
- $(Y.\pi_1).\pi_2 = Y.(\pi_1.\pi_2)$;
- $t.(f, i).\pi' = t_i.\pi'$ if $t = f(t_1, \dots, t_m)$ and $1 \leq i \leq m$. In particular, for $g \neq f$, $t.(g, i).\pi'$ is *undefined*.

Example 3. For an atom a and a binary constructor b , the following expressions t_i with

$$t_1 = a \quad t_2 = b(b(a, a), X) \quad t_3 = b(a, X.(b, 1).(b, 2))$$

all are general terms. We have, e.g.:

$$\begin{aligned} t_2.(b, 1).(b, 1) &= a & t_3.(b, 2) &= X.(b, 1).(b, 2) \\ t_2.(b, 2).(b, 2) &= X.(b, 2) & t_3.(b, 2).(b, 1) &= X.(b, 1).(b, 2).(b, 1) \end{aligned}$$

□

A *general disequality constraint* is a finite conjunction of general disequalities, each of which is of the form $t_1 \neq t_2$ where t_1, t_2 are general terms. This notion subsumes *term* disequality constraints as considered in [911] which allow variables X only, i.e., rule out path expressions $X.\pi$ with $\pi \neq \epsilon$. This notion also subsumes *labeled-path* disequality constraints where each disequality is of the form $X.\pi \neq t$ where $X.\pi$ is a path expression and t is either another path expression $Y.\pi'$ or a *ground* term, i.e., a term not containing any variables or path expressions. In the following, we refer to general disequality constraints when we mention disequality constraints without further qualification.

Consider a general term t and a ground substitution θ which provides ground terms for all variables occurring in t . If $\theta(X).\pi$ is defined for each path expression $X.\pi$ occurring in t , then $t\theta$ is obtained from t by replacing each occurrence of $X.\pi$ with the ground term $\theta(X).\pi$. Otherwise, $t\theta$ is undefined. The ground substitution θ satisfies a disequality $t_1 \neq t_2$ between general terms t_1, t_2 , if either $t_1\theta$ or $t_2\theta$ is undefined, or both terms are defined but different. In this case, we write $\theta \models (t_1 \neq t_2)$. Likewise, we extend satisfiability to arbitrary monotone Boolean combinations of disequalities by:

$$\begin{aligned} \theta \models (\phi_1 \wedge \phi_2) &\text{ iff } (\theta \models \phi_1) \wedge (\theta \models \phi_2) \\ \theta \models (\phi_1 \vee \phi_2) &\text{ iff } (\theta \models \phi_1) \vee (\theta \models \phi_2) \end{aligned}$$

For convenience, we also consider *equality* constraints $t_1 = t_2$ between general terms. A ground substitution θ satisfies the equality $t_1 = t_2$ if and only if it does not satisfy the corresponding disequality $t_1 \neq t_2$, i.e., if both $t_1\theta$ and $t_2\theta$ are defined *and equal*.

Lemma 1. *For every disequality $t_1 \neq t_2$ between general terms t_1, t_2 , a finite disjunction ϕ of path disequalities can be constructed such that $t_1 \neq t_2$ is equivalent to ϕ , i.e., for every substitution θ , it holds that $\theta \models (t_1 \neq t_2)$ iff $\theta \models \phi$.*

Proof. In the first step, we observe that the disequality $t_1 \neq t_2$ is equivalent to a finite disjunction of disequalities $X.\pi \neq t$ for suitable path expressions $X.\pi$ and subterms t occurring in t_1 or t_2 . Now let Π denote the set of all labeled paths π' such that $t.\pi'$ either is a path expression or a ground term. Let Π_0 denote the minimal elements in Π , i.e., the set of all paths $\pi' \in \Pi$ where Π does not contain a proper prefix of π' . The elements in Π_0 denote labeled paths in t reaching maximal ground subterms or path expressions contained in t . Therefore, the subset Π_0 is finite. Then the disequality $X.\pi \neq t$ is equivalent to the disjunction

$$\bigvee_{\pi' \in \Pi_0} X.\pi.\pi' \neq t.\pi' \quad \square$$

3 Horn Clauses

Let us briefly introduce the notions of Horn clauses and subclasses of Horn clauses which we consider here. Essentially, these classes are obtained from the classes considered in [11] by replacing constraints consisting of disequalities between terms with constraints consisting of disequalities between terms which now may also refer to path expressions. Thus, a Horn clause with (now general) disequality constraints is of the form:

$$q(t) \Leftarrow p_1(t_1), \dots, p_k(t_k), \phi$$

where ϕ is a finite conjunction of disequalities between general terms, q, p_1, \dots, p_k are unary predicates, t, t_1, \dots, t_k are (ordinary) terms. Non-unary predicates can be integrated in our framework by equipping their arguments with an *implicit* constructor. As usual, $q(t)$ is called the *head*, while $p_1(t_1), \dots, p_k(t_k), \phi$ is the *body* or *precondition* of the clause. The Horn clause is from the class \mathcal{H}_1 , if the term t in the head contains at most one constructor and is linear, i.e., no variable occurs twice in t . For convenience, we adopt the convention that the (distinct) variables in the head are enumerated X_1, \dots, X_k . This means that t either equals X_1 or is of the form $f(X_1, \dots, X_k)$ for some constructor of arity k where the case of atoms is subsumed by choosing $k = 0$ (writing f instead of $f()$ in this case). Moreover for a distinction, variables *not* occurring in the head will be denoted by Y, Y_1, \dots .

The Horn clause is a *normal clause* if it is of one of the forms:

$$q(X_1) \Leftarrow \phi \quad \text{or} \\ q(f(X_1, \dots, X_k)) \Leftarrow p_1(X_{i_1}), \dots, p_r(X_{i_r}), \phi$$

where all variables occurring in the body of the clause also occur in the head. Moreover, the Horn clause is an *automata clause* if additionally each variable X_i occurring in the head occurs exactly once in the literals occurring in the body and the head contains exactly one constructor, i.e., the clause is of the form:

$$q(f(X_1, \dots, X_k)) \Leftarrow p_1(X_1), \dots, p_k(X_k), \phi .$$

In particular, each normal clause as well as each automata clause is \mathcal{H}_1 .

Let \mathcal{C} denote a (possibly infinite) set of Horn clauses. Then the *least model* of \mathcal{C} is the least set M such that $q(t\theta) \in M$ for a ground substitution θ whenever there is a Horn clause $q(t) \Leftarrow p_1(t_1), \dots, p_k(t_k), \phi$ such that $p_i(t_i\theta) \in M$ for all $i = 1, \dots, k$, and $\theta \models \phi$. The set of all all terms s with $q(s) \in M$, for a given predicate q , then is denoted by $\llbracket q \rrbracket_{\mathcal{C}}$. Similar to the case of ordinary Horn clauses or Horn clauses with term constraints, we have:

Lemma 2 ([11]). *For every finite set \mathcal{N} of normal clauses, a finite set \mathcal{A} of automata clauses can be effectively constructed such that for every predicate p occurring in \mathcal{N} , $\llbracket p \rrbracket_{\mathcal{N}} = \llbracket p \rrbracket_{\mathcal{A}}$. \square*

Example 4. For terms t , let $s^0(t) = t$, $s^{i+1}(t) = s(s^i(t))$. Consider the set \mathcal{N} of normal clauses:

$$\begin{aligned} adult(pers(X_1, X_2)) &\Leftarrow age(X_1), name(X_2), old(X_1) \\ old(X_1) &\Leftarrow \bigwedge_{i \leq 18} X_1 \neq s^i(0) \\ age(0) &\Leftarrow \\ age(s(X_1)) &\Leftarrow age(X_1) \end{aligned}$$

In order to construct a corresponding finite set of automata clauses, the variables X_1 occurring in the heads of normal clauses $p(X_1) \Leftarrow \phi$ are instantiated with all terms $c(X_1, \dots, X_k)$, c a constructor of arity $k \geq 0$. Additionally, we introduce auxiliary predicates for all conjunctions of predicates, possibly occurring in preconditions. For the set \mathcal{N} of normal clauses, we obtain the set \mathcal{A} of clauses:

$$\begin{aligned} adult(pers(X_1, X_2)) &\Leftarrow age_old(X_1), name(X_2) & \top(0) &\Leftarrow \\ old(s(X_1)) &\Leftarrow \top(X_1), \bigwedge_{i \leq 17} X_1 \neq s^i(0) & \top(s(X_1)) &\Leftarrow \top(X_1) \\ age_old(s(X_1)) &\Leftarrow age(X_1), \bigwedge_{i \leq 17} X_1 \neq s^i(0) & age(0) &\Leftarrow \\ \top(pers(X_1, X_2)) &\Leftarrow \top(X_1), \top(X_2) & age(s(X_1)) &\Leftarrow age(X_1) \\ old(pers(X_1, X_2)) &\Leftarrow \top(X_1), \top(X_2) \end{aligned}$$

From the three possible new clauses defining the predicate *old*, we only kept the clauses for the constructors s and *pers*, since the precondition of the clause

$$old(0) \Leftarrow \bigwedge_{i \leq 18} 0 \neq s^i(0)$$

contains the disequality $0 \neq 0$ which is unsatisfiable. Concerning the required conjunctions, the new predicate \top denotes the *empty* conjunction of predicates, i.e., accepts all terms in \mathcal{T}_{Σ} where $\Sigma = \{0, s, pers\}$ is given by the atoms and constructors occurring in \mathcal{N} . And the new predicate *age_old*, on the other hand represents the conjunction of the predicates *age* and *old*. \square

4 Automata Classes and Expressiveness

A finite set of automata clauses together with a finite set of accepting predicates can be considered as a non-deterministic tree automaton with disequality constraints. Different classes of disequality constraints thus correspond to different classes of tree automata. Automata clauses with term disequality constraints have been considered in [11], while tree automata with disequalities of path expressions have been investigated by Comon and Jacquemard [2]. In fact, they have only considered disequalities between *unlabeled* paths. Unlabeled paths are obtained from labeled paths $(f_1, i_1). \dots (f_n, i_n)$ by omitting the labels f_1, \dots, f_n . The resulting automata, though, are equally expressive since the constructors occurring in paths can be recorded by means of specialized predicates. E.g., a clause

$$p(f(X_1, X_2)) \Leftarrow q(X_1), r(X_2), X_1.(f, 1) \neq X_2$$

is replaced by the clauses

$$\begin{aligned} p_{f(-, _)}(f(X_1, X_2)) &\Leftarrow q_{f(-, _)}(X_1), r_t(X_2), X_1.1 \neq X_2 \\ p_{f(-, _)}(f(X_1, X_2)) &\Leftarrow q_{g(-, \dots, _)}(X_1), r_t(X_2) \end{aligned}$$

for arbitrary patterns t and all $g \neq f$. In this construction, only those patterns are enumerated which are made up from suffixes of paths occurring in the given set of automata clauses. For the reverse direction, we observe that every unlabeled-path disequality is equivalent to a finite conjunction of labeled-path disequalities (since Σ is finite). Recalling Lemma 1, the following simulations therefore can be proven.

Theorem 1. *Assume that \mathcal{C} is a finite subset of automata clauses with general term disequality constraints. Then the following holds:*

1. *A finite subset \mathcal{C}' of automata clauses with labeled-path disequality constraints can be effectively constructed such that for every predicate p , $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\mathcal{C}'}$.*
2. *A finite subset \mathcal{C}'' of automata clauses with unlabeled-path disequality constraints can be effectively constructed such that for all predicates p , $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\mathcal{C}''}$.*

□

In [2], it has also been proven that emptiness for finite tree automata with unlabeled-path disequality constraints is decidable. From that, we deduce that emptiness is also decidable for automata clauses with general term disequalities. We have:

Corollary 1. *Given a finite set \mathcal{C} of automata clauses with general term disequality constraints and a predicate p , it is decidable whether or not $\llbracket p \rrbracket_{\mathcal{C}} = \emptyset$. Moreover in case that $\llbracket p \rrbracket_{\mathcal{C}} \neq \emptyset$, a witness $t \in \llbracket p \rrbracket_{\mathcal{C}}$ can effectively be computed.*

□

Since unlabeled-path disequality constraints can be expressed by means of labeled-path disequality constraints, and labeled-path disequality constraints are a special case of general term disequality constraints, all three classes of automata clauses compared in Theorem 1, are equally expressive.

Term disequality constraints are special cases of general term disequality constraints. Therefore by Theorem 1, automata clauses with term disequality constraints can be simulated by means of automata clauses with labeled-path or unlabeled-path disequality

constraints. The reverse simulation, though, is not possible. One indication is that universality is decidable [9] for tree automata with term disequality constraints — while it is undecidable for tree automata with path disequality constraints, since emptiness for automata with path *equalities* only [5] is undecidable.

Here, we additionally present a specific language T which can be expressed as $\llbracket p \rrbracket_{\mathcal{C}}$ for a finite set \mathcal{C} of automata clauses with labeled-path disequality constraints, but which cannot be characterized by means of finite sets of automata clauses with term disequality constraints only. Let $T = \llbracket p \rrbracket_{\mathcal{C}}$ for the following set \mathcal{C} of automata clauses:

$$\begin{aligned} p(f(X_1, X_2)) &\Leftarrow \top(X_1), \top(X_2), X_1 \neq X_2.(f, 1) \\ \top(f(X_1, X_2)) &\Leftarrow \top(X_1), \top(X_2) \\ \top(a) &\Leftarrow \end{aligned}$$

Lemma 3. *There is no finite set \mathcal{C}' of automata clauses with term disequality constraints which define a predicate p such that $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\mathcal{C}'}$.*

Proof. Assume for a contradiction that there is such a finite set \mathcal{C}' which defines such a predicate p . Then we construct a finite set \mathcal{N} of normal clauses for \mathcal{C}' using equality term constraints only such that for every predicate q of \mathcal{C}' , \mathcal{N} has a predicate \bar{q} with $\llbracket \bar{q} \rrbracket_{\mathcal{N}}$ is the complement of $\llbracket q \rrbracket_{\mathcal{C}'}$. This set is constructed as follows. Assume that $q(f(X_1, \dots, X_k)) \Leftarrow l_{i_1}, \dots, l_{i_r}$ for $i = 1, \dots, m$ are the clauses for q and constructor f where each l_{ij} either is a literal of the form $p'(X_s)$ or a single disequality. Then the predicate \bar{q} for constructor f is defined by the set of all clauses

$$\bar{q}(f(X_1, \dots, X_k)) \Leftarrow \bigwedge_{1 \leq i \leq m} \bar{l}_{ij_i}$$

where for each i , $1 \leq j_i \leq r_i$. For a literal l_{ij_i} of the form $p'(X_s)$, \bar{l}_{ij_i} is given by $\bar{p}'(X_s)$, and if l_{ij_i} equals a disequality $t_1 \neq t_2$, \bar{l}_{ij_i} is given by $t_1 = t_2$. By this construction the resulting clauses contain equality constraints only. As in Lemma 2, a finite set \mathcal{A} of automata clauses with term equality constraints can be computed such that for all predicates \bar{q} of \mathcal{N} , $\llbracket \bar{q} \rrbracket_{\mathcal{N}} = \llbracket \bar{q} \rrbracket_{\mathcal{A}}$.

A similar construction for automata without constraints has been described in [10]. Another variant recently has been presented in [5].

Let \bar{T} denote the complement of T , i.e., the set $\mathcal{T}_{\Sigma} \setminus T$. \bar{T} consists of all elements of the form

$$f(t, f(t, s))$$

for arbitrary terms s, t . By construction, $\llbracket \bar{p} \rrbracket_{\mathcal{A}} = \bar{T}$. Then \mathcal{A} must contain a clause

$$\bar{p}(f(X_1, X_2)) \Leftarrow q_1(X_1), q_2(X_2), \phi$$

where ϕ is a finite conjunction of term equalities, such that there are distinct ground terms $t_1, t_2 \in \llbracket q_1 \rrbracket_{\mathcal{A}}$ and for $i = 1, 2$ there are two distinct terms t_{i1}, t_{i2} such that $f(t_i, t_{ij}) \in \llbracket q_2 \rrbracket_{\mathcal{A}}$ and $f(t_i, f(t_i, t_{ij})) \in \llbracket \bar{p} \rrbracket_{\mathcal{A}}$ by application of this clause. This means for $\theta_{ij}(X_1) = t_i$ and $\theta_{ij}(X_2) = f(t_i, t_{ij})$, that $\theta_{ij} \models \phi$. In order to see this, we first convince ourselves that for every term t there must be some clause c_t by which for

two distinct terms s_1, s_2 , facts $\overline{p}(f(t, f(t, s_1)))$ and $\overline{p}(f(t, f(t, s_2)))$ can be derived. Assume for a contradiction, this were not the case. Then for some t and every clause c for predicate \overline{p} , there is at most one term t_c such that a fact $\overline{p}(f(t, f(t, t_c)))$ is derived by means of c . Accordingly, for this t , the set $\{s \mid f(t, f(t, s)) \in \overline{T}\}$ were finite — which is not the case. Consequently, for every t we can find a clause c_t by which for two distinct terms s_1, s_2 , facts $\overline{p}(f(t, f(t, s_1)))$ and $\overline{p}(f(t, f(t, s_2)))$ can be derived. Since the number of terms is infinite while the number of clauses is finite, we conclude that there must be two distinct terms t_1, t_2 for which the clauses c_{t_1} and c_{t_2} coincide.

Now recall that each finite conjunction of term equalities $s_i = s_j$ between arbitrary terms s_i, s_j can be expressed as a finite conjunction of term equalities of the form $Z = s$ for variables Z . W.l.o.g. let ϕ be of this form. Furthermore, recall that a term equality $Z = s$ where s contains Z either is trivially true or trivially false. In addition to equalities $X_1 = X_1$ and $X_2 = X_2$, the satisfiable constraint ϕ therefore can only contain equalities of one of the following forms:

- (1) $X_1 = g$ or $X_2 = g$, where g denotes a ground term;
- (2) $X_1 = s$, where s contains variable X_2 ;
- (3) $X_2 = s$, where s contains variable X_1

In the following, we show that an equality of any of these forms leads to a contradiction.

Case 1. Assume that there is an equality $X_r = g$ for some ground term g . If $r = 1$, then either $t_1 \neq g$ or $t_2 \neq g$ implying that $\theta_{ij} \not\models (X_r = g)$ for some i, j . If on the other hand, $r = 2$, then either $f(t_1, t_{11}) \neq g$ or $f(t_1, t_{12}) \neq g$, and hence also $\theta_{ij} \not\models (X_r = g)$ for some i, j .

Case 2. Assume that there is an equality $X_1 = s$ where s contains X_2 . If $\theta_{11} \models (X_1 = s)$, then t_1 would contain the term $f(t_1, t_{11})$ — which is impossible.

Case 3. Finally, assume that there is an equality $X_2 = s$ where s contains X_1 . Then consider the two substitutions θ_{11} and θ_{12} . If $\theta_{11} \models (X_2 = s)$, we conclude that $f(t_1, t_{11}) = \theta_{11}(X_2) = s[t_1/X_1]$. Then $\theta_{12} \models (X_2 = s)$ implies that $s[t_1/X_1]$ also equals $f(t_1, t_{12})$, and therefore, $f(t_1, t_{11}) = f(t_1, t_{12})$. This however, implies that $t_{11} = t_{12}$ — in contradiction to our assumption.

We conclude that the conjunction ϕ is equivalent to true. But then $\llbracket \overline{p} \rrbracket_{\mathcal{A}}$ must also contain the term $f(t_1, f(t_2, t_{21}))$ — which is not contained in \overline{T} . This completes the proof. \square

5 \mathcal{H}_1 -Normalization

This section describes a normalization procedure which constructs for each finite set of \mathcal{H}_1 -clauses with general term disequalities a finite set of normal clauses with general term disequalities which is equivalent. The normalization procedure repeatedly applies three rules. Each rule adds finitely many simpler clauses which are implied by the current set of clauses. The three rules are *resolution*, *splitting* and *propagation*. Thus, this general procedure is quite in-line with the normalization procedures for unconstrained \mathcal{H}_1 -clauses [8,6] or \mathcal{H}_1 -clauses with (ordinary) term disequalities [11]. In order to make this idea also work in presence of disequalities involving path expressions, a completely new construction for splitting is required (see subsection 5.2). Also the argument for

termination must be appropriately generalized. The following subsections provide the normalization rules. We refer to the current set of all implied clauses (whether originally present or added during normalization) as \mathcal{C} , while $\mathcal{N} \subseteq \mathcal{C}$ denotes the subset of normal clauses in \mathcal{C} .

5.1 Resolution

The first type of rules simplifies a complicated clause from \mathcal{C} by applying a resolution step with a normal clause. Assume that \mathcal{C} contains a clause $h \Leftarrow \alpha_1, p(t), \alpha_2, \psi$.

If \mathcal{N} contains a clause $p(X_1) \Leftarrow \phi$, then we add the clause $h \Leftarrow \alpha_1, \alpha_2, \psi, \psi'$ where $\psi' = \phi[t/X_1]$.

If \mathcal{N} has a clause $p(f(X_1, \dots, X_k)) \Leftarrow \beta, \phi$, and $t = f(t_1, \dots, t_k)$, then we add the clause:

$$h \Leftarrow \alpha_1, \alpha', \alpha_2, \psi, \psi'$$

where $\alpha' = \beta[t_1/X_1, \dots, t_k/X_k]$ and likewise, $\psi' = \phi[t_1/X_1, \dots, t_k/X_k]$. These resolution steps may introduce new disequalities. The new constraints are obtained from already available constraints by substitution of terms for variables. We remark, though, that after simplification of queries $t.\pi$ with t not a variable, the new constraints only contain path expressions for paths which are *suffixes* of paths already occurring in constraints of \mathcal{C} .

Example 5. Consider again the voting protocol Example 2:

$$\begin{aligned} \text{valid}(X_1) &\Leftarrow q(X_1, []) \\ q(X_1, X_2) &\Leftarrow q(X_1, :: (Y, X_2)), X_1.(vote, 1) \neq Y.(vote, 1) \\ q(X_1, X_2) &\Leftarrow \text{is_vote}(X_1), \text{votes}(X_2) \end{aligned}$$

enhanced with the normal clauses:

$$\begin{aligned} \text{empty}([]) &\Leftarrow h_b(\text{pers}(X_1, X_2)) \Leftarrow n_{bob}(X_1), \text{age}_{25}(X_2) \\ \text{age}_{15}(15) &\Leftarrow h_a(\text{pers}(X_1, X_2)) \Leftarrow n_{alice}(X_1), \text{age}_{15}(X_2) \\ \text{age}_{25}(25) &\Leftarrow p_{bob}(\text{vote}(X_1, X_2)) \Leftarrow h_b(X_1) \\ n_{alice}(\text{alice}) &\Leftarrow p_{alice}(\text{vote}(X_1, X_2)) \Leftarrow h_a(X_1) \\ n_{bob}(\text{bob}) &\Leftarrow \text{votes}(:: (X_1, X_2)) \Leftarrow p_{bob}(X_1), v'(X_2) \\ &\quad v'(:: (X_1, X_2)) \Leftarrow p_{alice}(X_1), \text{empty}(X_2) \end{aligned}$$

to fill the list of already submitted votes with the two entries $\text{vote}(\text{pers}(\text{alice}, 15), _)$ and $\text{vote}(\text{pers}(\text{bob}, 25), _)$ where $_$ is intended to represent one of the atoms *yes* or *no* ($\text{yes}, \text{no} \in \Sigma$). Resolving the first two clauses of this example with the third one for the substitution $X_2 \mapsto []$ and $X_2 \mapsto :: (Y, X_2)$, respectively, yields the clauses:

$$\begin{aligned} \text{valid}(X_1) &\Leftarrow \text{is_vote}(X_1), \text{votes}([]) \\ q(X_1, X_2) &\Leftarrow \text{is_vote}(X_1), \text{votes}(:: (Y, X_2)), X_1.(vote, 1) \neq Y.(vote, 1) \end{aligned}$$

With the clause $\text{votes}(:: (X_1, X_2)) \Leftarrow p_{bob}(X_1), v'(X_2)$, the second new clause can further be resolved to obtain

$$q(X_1, X_2) \Leftarrow \text{is_vote}(X_1), p_{bob}(Y), v'(X_2), X_1.(vote, 1) \neq Y.(vote, 1)$$

5.2 Splitting

The next type of saturation rules is concerned with the removal of variables not contained in the head of a clause. Assume that \mathcal{C} contains a clause $h \Leftarrow \alpha, \psi$ and Y is a variable occurring in α, ψ but not in h . We rearrange the precondition α into a sequence $\alpha', q_1(Y), \dots, q_r(Y)$ where α' does not contain the variable Y . Then we construct a finite sequence t_1, \dots, t_l of ground terms such that, w.r.t. \mathcal{N} ,

$$\psi[t_1/Y] \vee \dots \vee \psi[t_l/Y]$$

is equivalent to $\exists Y. q_1(Y), \dots, q_r(Y), \psi$, and add the clauses

$$h \Leftarrow \alpha', \psi[t_j/Y], \quad j = 1, \dots, l$$

to the set \mathcal{C} . This operation is referred to as *splitting*.

According to this construction, splitting may introduce new disequalities. As in the case of resolution, new constraints are obtained from already available constraints by substitution of (ground) terms. This means that, after simplification of queries $t.\pi$ with t not a variable, the new constraints only contain path expressions for paths which are *suffixes* of paths already occurring in constraints of \mathcal{C} . The remainder of this section provides a proof that the finite sequence t_1, \dots, t_l of ground terms exists together with an effective construction of the t_j . For notational convenience, let us assume that we are not given a finite sequence q_1, \dots, q_r of predicates but just a single predicate p which is defined by means of a finite set of automata clauses \mathcal{A} .

Theorem 2. *Let \mathcal{A} be a finite set of automata clauses, p a predicate, and Y a variable. For every conjunction of labeled-path disequalities ψ , a finite sequence of ground terms t_1, \dots, t_l can be constructed such that with respect to \mathcal{A} , the disjunction $\phi = \psi[t_1/Y] \vee \dots \vee \psi[t_l/Y]$ is equivalent to the expression $\exists Y. p(Y), \psi$.*

Proof. W.l.o.g. we assume that the variable Y does not occur on both sides within the same disequality in ψ . Otherwise, we modify the set \mathcal{A} of clauses in such a way that only those terms of the (original) set $\llbracket p \rrbracket_{\mathcal{A}}$ are accepted by the predicate p which satisfy those disequalities.

Now let Π denote the set of path expressions $Y.\pi$ occurring in ψ , and m the total number of occurrences of such expressions in ψ . We construct a finite sequence of terms t_1, \dots, t_l of $\llbracket p \rrbracket_{\mathcal{A}}$ such that for each ground substitution θ not mentioning Y , the following holds: if $\theta \oplus \{Y \mapsto t\} \models \psi$ for some $t \in \llbracket p \rrbracket_{\mathcal{A}}$, then also $\theta \oplus \{Y \mapsto t_i\} \models \psi$ for some i . Each of the terms t_i is generated during one possible run of the following nondeterministic algorithm. The algorithm starts with one term $s_0 \in \llbracket p \rrbracket_{\mathcal{A}}$. If no such term exists, the empty sequence is returned. Otherwise, the algorithm adds s_0 to the output sequence and proceeds according to one permutation of the occurrences of path expressions $Y.\pi$ occurring in ψ . Then it iterates of the path expressions in the permutation. In the round i for the path expression $Y.\pi$, the current set \mathcal{A} of automata clauses is modified in such a way that all terms t with $t.\pi = s_{i-1}.\pi$ are excluded from $\llbracket p \rrbracket_{\mathcal{A}}$. Let \mathcal{A}' be the resulting set of automata clauses. If $\llbracket p \rrbracket_{\mathcal{A}'}$ is empty the algorithm terminates. Otherwise, a term $s_i \in \llbracket p \rrbracket_{\mathcal{A}'}$ is selected and added to the output sequence.

For the correctness of the approach, consider an arbitrary ground substitution θ defined for all variables occurring in ψ with the exception of Y . First, assume that $\exists Y. p(Y), \psi\theta$ is not satisfiable. Then for no $s \in \llbracket p \rrbracket_{\mathcal{A}}$, $\psi\theta[s/Y]$ is true. Hence, also the finite disjunction provided by our construction cannot be satisfiable for such a θ , and therefore is equivalent to $\exists Y. p(Y), \psi\theta$. Now assume that $\theta \models \exists Y. p(Y), \psi$, i.e., some $s \in \llbracket p \rrbracket_{\mathcal{A}}$ exists with $\theta \models \psi[s/Y]$. Then we claim that there exists some s' occurring during one run of the nondeterministic algorithm with $\theta \models \psi[s'/Y]$. We construct this run as follows. Let $s_0 \in \llbracket p \rrbracket_{\mathcal{A}}$ denote the start term of the algorithm. If s_0 satisfies all disequalities, we are done. Otherwise, we choose one disequality $Y.\pi \neq t$ in $\psi\theta$ which is not satisfied. This means that $s_0.\pi = t$. Accordingly, we choose $Y.\pi$ as the first occurrence of a path expression selected by the algorithm. In particular, this means that all further terms s_i output by the algorithm will satisfy the disequality $Y.\pi \neq t$. After each round, one further disequality is guaranteed to be satisfied – while still s is guaranteed to be accepted at p by the resulting set \mathcal{A} of automata clauses. \square

Corollary 2. *Let \mathcal{N} be a finite set of normal clauses, q_1, \dots, q_r a finite sequence of predicates, and Y a variable. For every conjunction of arbitrary constraints ψ , a finite sequence of ground terms s_1, \dots, s_l can be constructed such that with respect to \mathcal{N} , the disjunction $\phi = \psi[s_1/Y] \vee \dots \vee \psi[s_l/Y]$ is equivalent to the expression $\exists Y. q_1(Y), \dots, q_r(Y), \psi$.*

Proof. First, recall that we can construct a finite set \mathcal{A} of automata clauses together with a predicate p such that $\llbracket p \rrbracket_{\mathcal{A}} = \llbracket q_1 \rrbracket_{\mathcal{N}} \cap \dots \cap \llbracket q_r \rrbracket_{\mathcal{N}}$. Clearly, $\exists Y. p(Y), \psi$ is implied by every constraint $\psi[s/Y]$ with $s \in \llbracket p \rrbracket_{\mathcal{A}}$.

By Lemma 1 every disequality $t_1 \neq t_2$ is equivalent to a disjunction of disequalities of the form $X.\pi \neq Z.\pi'$ or $X.\pi \neq t$ for variables X, Z and ground terms t . Accordingly, ψ is equivalent to a disjunction $\psi_1 \vee \dots \vee \psi_k$ for suitable labeled-path constraints ψ_i . By Theorem 2 each conjunction $p(Y), \psi_i$ is equivalent to a disjunction $\psi_i[s_{i1}/Y] \vee \dots \vee \psi_i[s_{il_i}/Y]$. Since $p(Y), \psi$ is equivalent to the disjunction

$$p(Y), \psi_1 \vee \dots \vee p(Y), \psi_k$$

we conclude that it is equivalent to the disjunction:

$$\psi_1[s_{11}/Y] \vee \dots \vee \psi_k[s_{kl_k}/Y]$$

The latter, on the other hand implies the disjunction

$$\psi[s_{11}/Y] \vee \dots \vee \psi[s_{kl_k}/Y]$$

Therefore, the sequence s_{11}, \dots, s_{kl_k} satisfies the requirements of the corollary. \square

Example 6. The resolution steps of Example 5 produced the clause

$$q(X_1, X_2) \Leftarrow is_vote(X_1), p_{bob}(Y), v'(X_2), X_1.(vote, 1) \neq Y.(vote, 1)$$

In order to decide satisfiability of $\exists Y. p_{bob}(Y), X_1.(vote, 1) \neq Y.(vote, 1)$, the algorithm of Theorem 2 starts with the term $vote(pers(bob, 25), yes) \in \llbracket p_{bob} \rrbracket_{\mathcal{N}}$ for the subset \mathcal{N} of normal clauses. Then it enforces the disequality $s.(vote, 1) \neq pers(bob, 25)$

for terms $s \in \llbracket p_{bob} \rrbracket_{\mathcal{N}}$ and finds out by a successful emptiness-test that no such term exists. Therefore only the *normal* clause:

$$q(X_1, X_2) \Leftarrow is_vote(X_1), v'(X_2), X_1.(vote, 1) \neq pers(bob, 25)$$

is added. This new clause in turn enables two more resolution steps for the first two clauses of this voting protocol example, yielding

$$\begin{aligned} valid(X_1) &\Leftarrow is_vote(X_1), v'([], X_1.(vote, 1) \neq pers(bob, 25)) \\ q(X_1, X_2) &\Leftarrow is_vote(X_1), v'(:, (Y, X_2)), X_1.(vote, 1) \neq pers(bob, 25), \\ &X_1.(vote, 1) \neq Y.(vote, 1) \end{aligned}$$

for the substitution $X_2 \mapsto []$ and $X_2 \mapsto :: (Y, X_2)$, respectively. Another resolution step with the clause $v'(:, (X_1, X_2)) \Leftarrow p_{alice}(X_1), empty(X_2)$ now yields:

$$\begin{aligned} q(X_1, X_2) &\Leftarrow is_vote(X_1), p_{alice}(Y), empty(X_2), X_1.(vote, 1) \neq pers(bob, 25), \\ &X_1.(vote, 1) \neq Y.(vote, 1) \end{aligned}$$

with substitution $X_1 \mapsto Y$. To the last clause, again splitting can be applied in order to replace the precondition $p_{alice}(Y), X_1.(vote, 1) \neq Y.(vote, 1)$ with a disjunction of constraints not containing Y . As is the case with p_{bob} , the algorithm of Theorem 2 only finds one term for predicate p_{alice} , say $vote(pers(alice, 15), no)$. Then for the path $(vote, 1)$ the term $pers(alice, 15)$ is excluded and p_{alice} becomes empty. Thus, the new normal clause

$$\begin{aligned} q(X_1, X_2) &\Leftarrow is_vote(X_1), empty(X_2), X_1.(vote, 1) \neq pers(bob, 25), \\ &X_1.(vote, 1) \neq pers(alice, 15) \end{aligned}$$

is added. Again the obtained normal clause enables two resolution steps for the first two clauses of the voting protocol, yielding

$$\begin{aligned} valid(X_1) &\Leftarrow is_vote(X_1), empty([], \phi) \\ q(X_1, X_2) &\Leftarrow is_vote(X_1), empty(:, (Y, X_2)), X_1.(vote, 1) \neq Y.(vote, 1), \phi \end{aligned}$$

where ϕ abbreviates the conjunction $X_1.(vote, 1) \neq pers(bob, 25), X_1.(vote, 1) \neq pers(alice, 15)$. Finally a last resolution step with the clause $empty([], \phi) \Leftarrow$ for the first of these two clauses achieves the result

$$valid(X_1) \Leftarrow is_vote(X_1), \phi$$

where ϕ again equals $X_1.(vote, 1) \neq pers(bob, 25), X_1.(vote, 1) \neq pers(alice, 15)$, stating that a vote is valid if it is submitted by a person who is different from the two persons as stored in the given list. \square

5.3 Propagation

The last type of rules considers clauses $p(X_1) \Leftarrow q_1(X_1), \dots, q_r(X_1), \psi$ where ψ only contains the variable X_1 (or none). Assume that $r > 0$, and \mathcal{N} contains normal clauses $q_j(f(X_1, \dots, X_k)) \Leftarrow \alpha_j, \psi_j$ for $j = 1, \dots, r$. Then we add the normal clause:

$$p(f(X_1, \dots, X_k)) \Leftarrow \alpha_1, \dots, \alpha_r, \psi_1, \dots, \psi_r, \psi'$$

where $\psi' = \psi[f(X_1, \dots, X_k)/X_1]$.

Also this rule may create new disequalities. Again, however, after simplification of queries $t.\pi$ where t is not a variable, the new constraints only contain path expressions for paths which are suffixes of paths already occurring in disequalities of the original set \mathcal{C} .

Example 7. Consider the following variant of the voting protocol example

$$\begin{aligned} \text{person}(\text{pers}(X_1, X_2)) &\Leftarrow \text{name}(X_1), \text{age}(X_2) \\ \text{adult}(X_1) &\Leftarrow \text{person}(X_1), \bigwedge_{0 \leq i \leq 17} X_1.(\text{pers}, 2) \neq i \\ \text{valid}(\text{vote}(X_1, Y)) &\Leftarrow q(X_1, []) \\ q(X_1, X_2) &\Leftarrow q(X_1, :: (Y, X_2)), X_1 \neq Y.(\text{vote}, 1) \\ q(X_1, X_2) &\Leftarrow \text{adult}(X_1), \text{votes}(X_2) \end{aligned}$$

intending to only allow adults to submit a vote. Here, the first argument of q is a person instead of a vote. Then the second clause is instantiated for constructor pers with substitution $X_1 \mapsto \text{pers}(X_1, X_2)$. Together with the first clause, we obtain:

$$\text{adult}(\text{pers}(X_1, X_2)) \Leftarrow \text{name}(X_1), \text{age}(X_2), \bigwedge_{0 \leq i \leq 17} \text{pers}(X_1, X_2).(\text{pers}, 2) \neq i$$

or simplified: $\text{adult}(\text{pers}(X_1, X_2)) \Leftarrow \text{name}(X_1), \text{age}(X_2), \bigwedge_{0 \leq i \leq 17} X_2 \neq i \quad \square$

Theorem 3. *Let \mathcal{C} denote a finite set of \mathcal{H}_1 -clauses. Let $\bar{\mathcal{C}}$ denote the set of all clauses obtained from \mathcal{C} by adding all clauses according to the resolution, splitting and propagation rules. Then the subset \mathcal{N} of all normal clauses in $\bar{\mathcal{C}}$ is equivalent to \mathcal{C} , i.e., $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\mathcal{N}}$ for every predicate p occurring in \mathcal{C} .*

Proof. The proof follows the same lines as the proof of the corresponding statement in [11]: every clause added to $\bar{\mathcal{C}}$ by means of resolution, splitting or propagation is implied by the set of \mathcal{H}_1 -clauses in \mathcal{C} . Therefore for every predicate p , $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\bar{\mathcal{C}}}$. In the second step one verifies that every fact $p(t)$ which can be deduced by means of the clauses in $\bar{\mathcal{C}}$ can also be deduced in at most as many steps with clauses from \mathcal{N} alone. This completes the proof. \square

For the proof of termination of \mathcal{H}_1 -normalization we consider *families* of clauses which (semantically) only differ in their constraints, and conceptually replace them by single clauses whose constraints are disjunctions of (conjunctions of) disequalities. Two clauses $h \Leftarrow \alpha_1, \phi_1$ and $h \Leftarrow \alpha_2, \phi_2$ belong to the same family if α_1 and α_2 contain the same set of literals.

To enforce termination of \mathcal{H}_1 -normalization it suffices not to add clauses that are already *subsumed* by the current set of clauses. A clause $h \Leftarrow \alpha_0, \phi_0$ is subsumed by a set of clauses $h \Leftarrow \alpha_i, \phi_i, i \geq 1$, if all clauses belong to the same family and ϕ_0 implies the disjunction $\bigvee_{i \geq 1} \phi_i$.

Theorem 4. *Let \mathcal{C} denote a finite set of \mathcal{H}_1 -clauses. Let $\bar{\mathcal{C}}$ denote the set of clauses obtained from \mathcal{C} by adding all clauses according to the resolution, splitting and propagation rules that are not subsumed by the current set of clauses. Then $\bar{\mathcal{C}}$ is finite.*

Proof. Since the number of predicates as well as the number of constructors is finite, there are only finitely many distinct heads of clauses. Also, the number of literals occurring in preconditions is bounded since new literals $p(t)$ are only added for *subterms* t of terms already present in the original set \mathcal{C} of clauses. Therefore, the number of occurring families of clauses is finite. For each family f let $\psi_{\mathcal{C},f}$ denote the disjunction of constraints of clauses of \mathcal{C} which belong to f . Each clause that is added to \mathcal{C} extends one of the finitely many constraints $\psi_{\mathcal{C},f}$ to $\psi_{\mathcal{C},f} \vee \phi$ for a conjunction of disequalities ϕ . The number of variables in each constraint $\psi_{\mathcal{C},f}$ is bounded. Resolution with normal clauses does not introduce new variables, and propagation steps always produce normal clauses, which contain at most as many variables as the maximum arity in Σ .

In order to show that the resulting disjunctions eventually are implied, we recall that in every normalization step, the terms in constraints may grow — but the lengths of paths in path expressions remain bounded. Therefore, the number of all possibly occurring path expressions is finite.

In [11] we have shown that for each sequence ψ_i of conjunctions of (ordinary) *term* disequalities over finitely many plain variables, the disjunction $\bigvee_{i=1}^m \psi_i$, $m \geq 1$, eventually becomes *stable*, i.e., there exists some M such that $\bigvee_{i=1}^m \psi_i = \bigvee_{i=1}^M \psi_i$ for all $m \geq M$. This also holds true if we use finitely many path expressions instead of variables. Therefore a disjunction $\bigvee_{i=1}^m \psi_i$, $m \geq 1$, also eventually becomes stable if each ψ_i is a conjunction of general term disequalities if the set of occurring path expressions is finite.

We conclude that eventually, every newly added clause is subsumed — implying that the modified normalization procedure terminates with a finite set of clauses $\bar{\mathcal{C}}$. \square

By Theorem 3 and Theorem 4, \mathcal{H}_1 -normalization constitutes a sound and complete procedure which constructs for every finite set \mathcal{C} of \mathcal{H}_1 -clauses with path disequalities an equivalent finite set \mathcal{N} of normal clauses within finitely many steps. By Lemma 2, \mathcal{N} can in turn be transformed into an equivalent finite set \mathcal{A} of automata clauses, for which by Corollary 1 emptiness can be decided for every predicate. Altogether this proves our main result:

Theorem 5. *Assume that \mathcal{C} is a finite set of \mathcal{H}_1 -clauses with general term disequality constraints. Then a finite set \mathcal{A} of automata clauses can be effectively constructed such that for every predicate p of \mathcal{C} , $\llbracket p \rrbracket_{\mathcal{C}} = \llbracket p \rrbracket_{\mathcal{A}}$. In particular, it is decidable whether or not $\llbracket p \rrbracket_{\mathcal{C}}$ is empty.* \square

6 Conclusion

We showed that finite sets of \mathcal{H}_1 -clauses with path disequalities can be effectively transformed into finite tree automata with path disequalities. By that, we have provided a procedure to decide arbitrary queries to predicates defined by \mathcal{H}_1 -clauses. From the proof of termination, however, little information could be extracted about the behavior of the method on realistic examples. It is a challenging open problem to provide explicit upper or nontrivial lower bounds to our algorithms.

\mathcal{H}_1 -clauses can be used to approximate general Horn clauses and are therefore suitable for the analysis of term-manipulating programs such as cryptographic protocols.

Beyond unconstrained \mathcal{H}_1 -clauses, \mathcal{H}_1 -clauses with path disequalities additionally allow to approximate notions such as freshness of nonces or keys directly and also to some extent, *negative* information extracted from failing checks. It remains for future work to provide a practical implementation of our methods and to evaluate it on realistic protocols.

Acknowledgements. We thank Florent Jacquemard for a lively discussion, many helpful remarks and in particular, for pointing out reference [5].

References

1. Baskar, A., Ramanujam, R., Suresh, S.P.: Knowledge-based modelling of voting protocols. In: Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2007, pp. 62–71. ACM, New York (2007)
2. Comon, H., Jacquemard, F.: Ground Reducibility and Automata with Disequality Constraints. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 1994. LNCS, vol. 775, pp. 151–162. Springer, Heidelberg (1994)
3. Comon, H., Jacquemard, F.: Ground reducibility is exptime-complete. In: LICS 1997: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, pp. 26–34. IEEE Computer Society, Washington, DC, USA (1997)
4. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
5. Godoy, G., Giménez, O., Ramos, L., Álvarez, C.: The hom problem is decidable. In: STOC, pp. 485–494 (2010)
6. Goubault-Larrecq, J.: Deciding $H1$ by resolution. *Information Processing Letters* 95(3), 401–408 (2005)
7. Goubault-Larrecq, J., Parrennes, F.: Cryptographic Protocol Analysis on Real C Code. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 363–379. Springer, Heidelberg (2005)
8. Nielson, F., Nielson, H.R., Seidl, H.: Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 20–35. Springer, Heidelberg (2002)
9. Reuß, A., Seidl, H.: Bottom-Up Tree Automata with Term Constraints. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 581–593. Springer, Heidelberg (2010)
10. Seidl, H., Neumann, A.: On Guarding Nested Fixpoints. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 484–498. Springer, Heidelberg (1999)
11. Seidl, H., Reuß, A.: Extending $H1$ -clauses with disequalities. *Information Processing Letters* 111(20), 1007–1013 (2011)
12. Weidenbach, C.: Towards an Automatic Analysis of Security Protocols in First-Order Logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 314–328. Springer, Heidelberg (1999)

Brookes Is Relaxed, Almost!*

Radha Jagadeesan, Gustavo Petri, and James Riely

School of Computing, DePaul University

Abstract. We revisit the Brookes [1996] semantics for a shared variable parallel programming language in the context of the Total Store Ordering (TSO) relaxed memory model. We describe a denotational semantics that is fully abstract for Brookes' language and also sound for the new commands that are specific to TSO. Our description supports the folklore sentiment about the simplicity of the TSO memory model.

1 Introduction

Sequential Consistency (SC), defined by Lamport [1979], enforces total order on memory operations — reads and writes to the memory — respecting the program order of each individual thread in the program. Operationally, SC is realized by traditional interleaving semantics, where shared memory is represented as a map from locations to values. For such an operational semantics, Brookes [1996] describes a fully abstract denotational view that identifies a process with its transition traces. This technique supports several approaches to program logics for shared memory concurrent programs based on separation logic (see Reynolds [2002] for an early survey). For example, O'Hearn [2007] and Brookes [2007] develop the semantics of Concurrent Separation Logic (CSL), an adaptation of separation logic to reason about concurrent threads operating on shared memory. CSL has been used to prove correctness of several concurrent data structures; for example, Parkinson et al. [2007] and Vafeiadis and Parkinson [2007]. Similarly, Brookes [1996] gives the foundation for refinement approaches to prove the correctness of concurrent data structures such as in Turon and Wand [2011].

There are at least two motivations to consider memory models that are *weaker*, or more *relaxed*, than SC: First, modern multicore architectures permit executions that are not sequentially consistent. Second, SC disables some common compiler optimizations for sequential programs, such as the reordering of independent statements. This has led to a large body of work on relaxed memory models; Adve and Gharachorloo [1996] and Adve and Boehm [2010] provide a tutorial introduction with detailed bibliography on architectures and their impact on language design.

The operational semantics of programming languages in the presence of such relaxed memory models has now been explored. For example, Boudol and Petri [2009] explore the operational semantics of a process language with write buffers; Sevcík et al. [2011] explore the operational semantics of CLight executing with the TSO memory model; and Jagadeesan et al. [2010] describe the operational semantics of an object language under the Java Memory Model (JMM) of Manson et al. [2005].

* Research supported by NSF 0916741.

However, what has not been investigated in the literature is the denotational semantics of a language with a relaxed memory execution model. We solve this open problem in this paper.

1.1 Overview of the Paper

Our investigations are carried out in the context of the TSO memory model described in [SPARC \[1994\]](#), recently proposed as the model of x86 architectures by [Sewell et al. \[2010\]](#). In TSO, each sequential thread carries its own write buffer that serves as the initial target of the writes executed by the thread. Thus, TSO permits executions that are not possible with SC.

To illustrate this relaxed behavior let us consider the canonical example depicted in [1](#) below. We have two sequential threads running in parallel. The left thread, with code $(x := 1; y)$, sets x to 1 and then reads y returning the value read. The thread on the right, with code $(y := 1; x)$, sets y to 1 and then reads and returns x . We consider that the initial state has $x = y = 0$. In the SC model, the execution where both threads read 0 is impermissible. It is however achieved by the following TSO execution with write buffers. Below we depict the initial configuration, where both threads have empty buffers (indicated by \emptyset) and the memory state is denoted by $\{x := 0, y := 0\}$.

$$(\{x := 0, y := 0\}, \langle \emptyset, x := 1; y \rangle \parallel \langle \emptyset, y := 1; x \rangle) \quad (1)$$

The writes performed by a thread go into its write buffer (rather than the shared memory). Thus, the above process configuration can evolve to

$$(\{x := 0, y := 0\}, \langle [x := 1], y \rangle \parallel \langle [y := 1], x \rangle)$$

where $\langle [x := 1], y \rangle$ stands for the thread with local buffer containing the assignment of 1 to x , which is not visible to the other thread, and similarly for $\langle [y := 1], x \rangle$. Now, both reads can return the value from the shared store, which is 0.

Of course, the usual SC executions are also available in a TSO model, which we demonstrate an example execution where both reads yield 1 starting from the initial process configuration. From the intermediate configuration above, both buffer updates can nondeterministically move into memory before the reads execute. Then, we get:

$$(\{x := 1, y := 1\}, \langle \emptyset, y \rangle \parallel \langle \emptyset, x \rangle)$$

leading to an execution where both reads yield 1.

We provide a precise formalization of the denotational semantics for the language of [Brookes \[1996\]](#) in the context of the TSO memory model. Our model includes the characteristic mfence instructions of TSO, which terminates only when the local buffer of the thread executing the instruction is empty.

Our formalization satisfies the Data Race Free (DRF) property [Adve and Hill \[1990\]](#). Informally, a program is DRF if no SC execution of the program leads to a state in which a write happens concurrently with another operation on the same location. A DRF *model* requires that the programmer view of computation coincides with SC for programs that satisfy the DRF property.

Let us review [Brookes, 1996] before adapting it to a TSO setting. We use the metavariable s to stand for a shared memory, that is a partial map of variables to values, and C for commands (possibly partially executed). [Brookes 1996] views the denotation of a command, $\mathcal{T}[[C]]$, as a set of completed transition traces, ranged by the metavariable α , and with the form $\alpha = (s_0, s'_0) \cdot (s_1, s'_1) \dots (s_n, s'_n)$. These traces describe the interaction between a *system* and its *environment*, where the following conditions hold.

- The execution starts with the command under consideration, so $C_0 = C$.
- Transitions from s_k to s'_k model a *system step*, i.e. $\forall k \in [0, n] . s_k, C_k \rightarrow s'_k, C_{k+1}$.
- Transitions from s'_k to s_{k+1} model an *environment step*.
- The transition trace represents a terminated execution, so $C_n = \text{skip}$.

As in any sensible semantics, skip must be a unit for sequential composition.

$$\text{skip}; C; \text{skip} \equiv C \quad (2)$$

This equation motivates the *stuttering* and *mumbling* closure properties. Closure by stuttering accommodates the case when the system does not move at all but just observes the current state, i.e. $s'_i = s_i$. Closure by mumbling permits the combination of system steps that have no intervening environment step.

We can now describe the model for TSO. The type of command denotations, $\mathcal{T}[[C]]$, changes to a function that takes an input buffer b and yields a set of pairs of the form $\langle \alpha, b' \rangle$ where α is a transition trace as before, and b' is the resulting buffer. The pair $\langle \alpha, b' \rangle$ is to be understood as follows, where we use P 's as metavariables for threads, and letting $\alpha = (s_0, s'_0) \cdot (s_1, s'_1) \dots (s_n, s'_n)$.

- The execution of the command starts with the input buffer b , so $P_0 = \langle b, C \rangle$.
- The state pairs still represent system steps, i.e. $\forall k \in [0, n] . s_k, P_k \rightarrow s'_k, P_{k+1}$.
- The change from s'_k to s_{k+1} still represents an environment step.
- The transition trace represents a terminated execution leaving b' as the resulting buffer, so $P_n = \langle b', \text{skip} \rangle$. Thus, the pending updates in the resulting buffer b' are yet to reach the shared memory even though there is no command left to be executed.

Our TSO semantics has analogues of the stuttering and mumbling properties for the same reasons as discussed above. In addition, it has two buffer closure properties.

Buffer update closure. Consider the program skip. Executions in $\mathcal{T}[[\text{skip}]](b)$ can result in a smaller buffer b' , because buffer updates can propagate into the shared memory. Furthermore, the change from b to b' can be done piecemeal, one buffer update at a time. Thus, skip should permit any executions of $\text{upd}(b)$ defined as the stuttering and mumbling closure of the set

$$\{ \langle (s_0, s'_0) \dots (s_n, s'_n), b' \rangle \mid b = [x_0 := v_0, \dots, x_n := v_n] \uparrow b' \ \& \ \forall i \in [0, n] . s'_i = s_i[x_i := v_i] \}$$

Each step in the above trace corresponds to the addition of one buffer update into memory. Mumbling closure introduces the possibility of multiple buffer updates in one atomic step.

To validate Equation 2, *buffer-update closure* permits data to potentially move from the buffers to shared state before and after any command executes:

$$\frac{\langle \alpha_1, b_1 \rangle \in \text{upd}(b), \langle \alpha_2, b_2 \rangle \in \mathcal{T}[[C]](b_1), \langle \alpha_3, b' \rangle \in \text{upd}(b_2)}{\langle \alpha_1 \cdot \alpha_2 \cdot \alpha_3, b' \rangle \in \mathcal{T}[[C]](b)}$$

Buffer reduction closure. The program $(x := 1; x := 1)$ simulates the program $(x := 1)$ (taking the two steps uninterruptedly), whereas the converse is not true. In buffer terms, this motivates the idea that two identical contiguous writes can be replaced by one copy of the write without leading to any new behaviors. We formalize this notion of buffer simulation as a binary relation $b_1 \triangleright b'$ and demand:

$$\frac{\langle \alpha, b_1 \rangle \in \mathcal{F}[\![C]\!](b), b_1 \triangleright b'}{\langle \alpha, b' \rangle \in \mathcal{F}[\![C]\!](b)}$$

Results. We present the following results.

- We describe operational and denotational semantics for the language that accommodate the extra executions permitted by TSO.
- We prove that our denotational semantics is fully abstract when we observe termination of programs.
- We use the model to identify some equational principles that hold for parallel programs under the TSO memory model.

Our results provide some formal validation for the “folklore” sentiment about the simplicity of the TSO memory model.

Organization of paper. We eschew a separate related works section since we cite the related work in context. In [Section 2](#) we discuss the transition system for the programming language. We develop the model theory in [Section 3](#), and prove the correspondence between operational and denotational semantics in [Section 4](#). In [Section 5](#), we illustrate the differences from [Brookes \[1996\]](#) by describing some laws that hold for programs. More detailed proof sketches are found in a fuller version of the paper.¹

2 Operational Semantics

We assume disjoint sets of *variables*, x, y and *values*, v . The only values we consider are natural numbers. In conditionals, we interpret non-zero (resp. zero) integers as true (resp. false). As usual we denote by $FV(C)$ the set of *free variables* of command C .

$$E ::= x \mid v \mid E_1 + E_2 \mid \neg E \mid \dots \quad (\text{Expression})$$

$$C, D ::= \text{skip} \mid x := E \mid C; D \mid C \parallel D \mid \text{if } E \text{ then } C \text{ else } D \quad (\text{Command})$$

$$\mid \text{while } E \text{ do } C \mid \text{local } x \text{ in } C \mid \text{await } E \text{ then } C \mid \text{mfence}$$

$$P, Q ::= \langle b, C \rangle \mid P; D \mid P \parallel Q \mid \text{new } x := v \text{ in } P \quad (\text{Process})$$

$$\mathbb{P}, \mathbb{Q} ::= [-] \mid \mathbb{P}; D \mid \mathbb{P} \parallel \mathbb{Q} \mid P \parallel Q \mid \text{new } x := v \text{ in } \mathbb{P} \quad (\text{Process context})$$

A *buffer*, $b \in \text{Buff}$, is a list of variable/value pairs, with Buff the domain of all buffers. If $b = [x_1 := v_1, \dots, x_n := v_n]$, then $\text{dom}(b) \triangleq \{x_1, \dots, x_n\}$. We write $++$ for concatenation, \emptyset for the empty buffer and $b|_x$ for the buffer that results from removing x from b . We consider buffer rewrites ($\triangleright : \text{Buff} \times \text{Buff}$) that can merge contiguous identical writes, e.g. $[x_1 := v_1, \dots, x_n := v_n, x_n := v_n] \triangleright [x_1 := v_1, \dots, x_n := v_n]$.

¹ <http://fpl.cs.depaul.edu/jriely/papers/2011brookes.pdf>

Definition 1. The relation $\triangleright : \text{Buff} \times \text{Buff}$ is defined inductively as follows.

$$\frac{}{\forall x, v. [x := v, x := v] \triangleright [x := v]} \quad \frac{}{b \triangleright b} \quad \frac{b \triangleright b_1, b_1 \triangleright b'}{b \triangleright b'} \quad \frac{b_1 \triangleright b'_1, b_2 \triangleright b'_2}{b_1 ++ b_2 \triangleright b'_1 ++ b'_2}$$

A memory, $s \in \Sigma$, is a partial map from variables to values, where Σ is the domain of all memories. We adopt several notation conventions for partial maps: if $s = \{x_1 := v_1, \dots, x_n := v_n\}$, then $\text{dom}(s) \triangleq \{x_1, \dots, x_n\}$. We write $s[x := v]$ for the memory s with the value of reference x substituted for v , and $s[b]$ to denote the memory which results from applying the updates contained in b from left to right.

As usual, we suppose a semantic function which maps expressions to functions from memories to values (in notation $\llbracket E \rrbracket s = v$). In the forthcoming transition rules, the memory passed to this function is already updated with (any) relevant buffer. The function is defined by induction on e as

$$\frac{s(x) = v}{\llbracket x \rrbracket (s) = v} \quad \frac{\llbracket E_1 \rrbracket (s) = v_1, \llbracket E_2 \rrbracket (s) = v_2}{\llbracket E_1 + E_2 \rrbracket (s) = v_1 + v_2} \quad \dots$$

In this paper, we consider that expressions evaluate atomically, following the first language considered in [Brookes \[1996\]](#). There are two standard approaches to formalizing finer grain semantics; either 1. a compilation of complex expressions to a sequence of simpler commands that only perform a single read or add local variables, or 2. a direct formalization in terms of a transition system as done in the later sections of [Brookes \[1996\]](#). Our presentation can accommodate either of these changes. We elide details in the interest of space.

Each sequential thread has its own buffer. Process are parallel compositions of commands. A *configuration* is a pair of a memory and a process. In [Figure 1](#) we define the evaluation relation $s, P \rightarrow s', P'$, where \rightarrow^* is the reflexive and transitive closure of the relation \rightarrow , and $C\{y/x\}$ denotes the command derived from C by replacing every occurrence of x with y .

The buffers grow larger in ASSIGN that adds a new update to the buffer, and become smaller in COMMIT that moves thread local buffer updates into the shared memory. CTXT-BUF allows contiguous and identical updates in the buffer to be collapsed.

The command skip captures our notion of termination. For example, in SKIP-SEQ, the succeeding command moves into the evaluation context when the preceding process evaluates to skip. When a process terminates, its associated buffer is not necessarily empty; e.g. when $x := E$ terminates, the update to x might still be in the buffer and not yet reflected in the shared memory.

The rule FENCE implements `mfence` as an assertion that can terminate only when the threads buffer is empty; e.g. $x := E; mfence$ terminates only when the update to x has been moved to the shared memory, thus making it visible to every other parallel thread.

The rule PAR-CMD enables the initiation of a parallel composition only when the buffer is empty. This restriction is in conformance with Appendix J of [SPARC \[1994\]](#) to ensure that the newly created threads can be scheduled on different processors. For similar reasons, SKIP-PAR ensures that a parallel composition terminates only when the buffers of both parallel processes are empty.

$$\begin{array}{c}
\frac{}{s, \langle b, \text{while } E \text{ do } C \rangle \rightarrow s, \langle b, \text{if } E \text{ then } (C; \text{while } E \text{ do } C) \text{ else skip} \rangle} \text{(WHILE)} \\
\frac{\llbracket E \rrbracket(s[b]) \neq 0}{s, \langle b, \text{if } E \text{ then } C \text{ else } D \rangle \rightarrow s, \langle b, C \rangle} \text{(THEN)} \quad \frac{\llbracket E \rrbracket(s[b]) = 0}{s, \langle b, \text{if } E \text{ then } C \text{ else } D \rangle \rightarrow s, \langle b, D \rangle} \text{(ELSE)} \\
\frac{y \notin \text{dom}(b) \cup \text{FV}(C)}{s, \langle b, \text{local } x \text{ in } C \rangle \rightarrow s, \text{new } y := 0 \text{ in } \langle b, C \{y/x\} \rangle} \text{(LOCAL)} \\
\frac{\llbracket E \rrbracket s \neq 0 \quad s, \langle \emptyset, C \rangle \rightarrow^* s', \langle \emptyset, \text{skip} \rangle}{s, \langle \emptyset, \text{await } E \text{ then } C \rangle \rightarrow s', \langle \emptyset, \text{skip} \rangle} \text{(AWAIT)} \quad \frac{\llbracket E \rrbracket(s[b]) = v}{s, \langle b, x := E \rangle \rightarrow s, \langle b \uparrow [x := v], \text{skip} \rangle} \text{(ASSIGN)} \\
\frac{}{s, \langle [x := v] \uparrow b, C \rangle \rightarrow s[x := v], \langle b, C \rangle} \text{(COMMIT)} \quad \frac{}{s, \langle \emptyset, \text{mfence} \rangle \rightarrow s, \langle \emptyset, \text{skip} \rangle} \text{(FENCE)} \\
\frac{}{s, \langle \emptyset, (C \parallel D) \rangle \rightarrow s, \langle \emptyset, C \rangle \parallel \langle \emptyset, D \rangle} \text{(PAR-CMD)} \quad \frac{}{s, \langle \emptyset, \text{skip} \rangle \parallel \langle \emptyset, \text{skip} \rangle \rightarrow s, \langle \emptyset, \text{skip} \rangle} \text{(SKIP-PAR)} \\
\frac{s, P \rightarrow s', P'}{s, P \parallel Q \rightarrow s', P' \parallel Q} \text{(CTXT-LEFT)} \quad \frac{s, Q \rightarrow s', Q'}{s, P \parallel Q \rightarrow s', P \parallel Q'} \text{(CTXT-RIGHT)} \\
\frac{}{s, \text{new } y := v \text{ in } \langle b, \text{skip} \rangle \rightarrow s, \langle b|_y, \text{skip} \rangle} \text{(SKIP-NEW)} \quad \frac{}{s, \langle \emptyset, \text{skip} \rangle; D \rightarrow s, \langle \emptyset, D \rangle} \text{(SKIP-SEQ)} \\
\frac{b \triangleright b'}{s, \langle b, C \rangle \rightarrow s', \langle b', C \rangle} \text{(CTXT-BUF)} \quad \frac{s, \langle b, C \rangle \rightarrow s, \langle b', C' \rangle}{s, \langle b, C; D \rangle \rightarrow s', \langle b', C'; D \rangle} \text{(CTXT-CMD)} \\
\frac{s, P \rightarrow s', P'}{s, P; D \rightarrow s', P'; D} \text{(CTXT-SEQ)} \\
\frac{s[y := v], P \rightarrow s', P' \quad s'(y) = v'}{s, \text{new } y := v \text{ in } P \rightarrow s'[y := s(y)], \text{new } y := v' \text{ in } P'} \text{(CTXT-NEW)}
\end{array}$$

Fig. 1. Evaluation: $s, P \rightarrow s', P'$

Our sole use of the local construct is to provide a model of thread-local registers in the special case when C is a sequential thread. However, our more general formalization permits the description of state that is shared among parallel processes. The process context $\text{new } y := v \text{ in } \mathbb{P}$ carries the shared state of this variable. The hypothesis on the initial buffer in LOCAL ensures that any mfence in C does not affect the global x . The renaming ensures that the updates of CTXT-NEW do not affect the global x . SKIP-NEW discards any remaining updates to the local y . The commands IF and WHILE are standard. The AWAIT construct from Brookes [1996] is a conditional critical region. It provides atomic protection to the entire command C which in our use will be generally be a series of assignments. The compare-and-set instruction of TSO architectures is programmable as follows:

$$\text{cas}(x, v, w) = \text{await } 1 \text{ then if } x = v \text{ then } x := w \text{ else } x := v$$

And similarly for the other atomic instruction of TSO. Following the semantics of cas in x86-TSO given by Owens et al. [2009], AWAIT ensures that the buffers are empty

$$\begin{array}{cc}
\left[\begin{array}{l} \text{flag}_0 := 1; \\ \text{if } \text{flag}_1 = 0 \text{ then} \\ \quad CS_0 \end{array} \right] \parallel \left[\begin{array}{l} \text{flag}_1 := 1; \\ \text{if } \text{flag}_0 = 0 \text{ then} \\ \quad CS_1 \end{array} \right] & \left[\begin{array}{l} \text{data} := 1; \\ \text{flag} := 1 \end{array} \right] \parallel \left[\begin{array}{l} \text{local } r \text{ in} \\ \text{if } \text{flag} = 0 \text{ then} \\ \quad r := \text{data} \end{array} \right] \\
\text{(a) Dekker Mutual Exclusion} & \text{(b) Safe Publication}
\end{array}$$

Fig. 2. Examples of TSO Programs

before and after the command executes and prevents buffer updates from other threads cf. the LOKD modifier of Owens et al. [2009]. While TSO does not directly support such multi-instruction atomic conditional critical regions, our semantics continues to be sound for a traditional TSO programming model, only providing the simpler `cas` and the single-word atomics alluded to above. We use this construct to permit a direct comparison with Brookes [1996] and use it (as in that work) to construct discriminating contexts in the proof of full abstraction.

Let us revise some examples of TSO in Figure 2. Dekker’s mutual exclusion algorithm 2a fails under TSO. In initial memories that contain 0 for flag_0 and flag_1 , the initial write of both threads can be put in their internal buffers, remaining inaccessible to the other thread while the reads can proceed before the updates are performed. Thus, both threads can get values 0 for their respective reads and execute their critical sections concurrently. On the other hand, the standard safe publication idiom of Figure 2b is safe under TSO, since the updates of flag and data will proceed in order. Thus, if flag is seen to have value 1 in the thread to the right, the update of 1 on data has also propagated to the memory.

We end this section by remarking that our programming language satisfies the standard DRF guarantee, following traditional proofs, e.g. see Adve and Gharachorloo [1996]; Boudol and Petri [2009]; Owens et al. [2009].

3 Denotational Semantics

We use α, β etc. for elements of $(\Sigma \times \Sigma)^*$, the sequences of state pairs, and ε for the empty trace. We will consider $\mathcal{P}((\Sigma \times \Sigma)^*)$, the powerset of sequences of state pairs, with the subset ordering. Similar assumptions are made for $\mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$, ranged by the metavariable \mathcal{U} . Commands yield functions in $\text{Buff} \rightarrow \mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$.

Definition 2. For any $b \in \text{Buff}$, define $\mathcal{T}[C](b) \in \mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$ as follows.

$$\begin{aligned}
\mathcal{T}[C](b) = \{ \langle (s_0, s'_0) \cdot \dots \cdot (s_n, s'_n), b' \rangle \mid \forall k \in [0, n-1]. s_k, P_k \xrightarrow{*} s'_k, P_{k+1} \ \& \\
P_0 = \langle b, C \rangle \ \& \ P_n = \langle b', \text{skip} \rangle \}
\end{aligned}$$

Thus, we only consider transition traces where the residual left of the command is skip, albeit with potentially unfinished buffer updates.

As in [Brookes, 1996], the transition traces are closed under stuttering and mumbling, to capture the reflexivity and transitivity of the operational transition relation.

$$\frac{\langle \alpha \cdot \beta, b \rangle \in \mathcal{U}}{\langle \alpha \cdot (s, s) \cdot \beta, b \rangle \in \mathcal{U}} \text{ STUTTERING} \qquad \frac{\langle \alpha \cdot (s, s') \cdot (s', s'') \cdot \beta, b \rangle \in \mathcal{U}}{\langle \alpha \cdot (s, s'') \cdot \beta, b \rangle \in \mathcal{U}} \text{ MUMBLING}$$

Let $\mathcal{U} \in \mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$, we define \mathcal{U}^\ddagger to be the smallest set containing \mathcal{U} such that is stuttering and mumbling closed.

Definition 3. Define $\text{upd}(b)$ to be the stuttering and mumbling closure of

$$\{ \langle (s_0, s'_0) \cdots (s_n, s'_n), b' \rangle \mid b = [x_0 := v_0, \dots, x_n := v_n] \# b' \ \& \ \forall k \in [0, n]. s'_k = s_k[x_k := v_k] \}$$

And then we can deduce the inclusion: $\forall b \in \text{Buff}. \text{upd}(b) \subseteq \mathcal{T}[\llbracket \text{skip} \rrbracket](b)$.

We now let $f : \text{Buff} \rightarrow \mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$, and consider the following closure properties.

$$\frac{\langle \alpha_1, b_1 \rangle \in \text{upd}(b), \langle \alpha_2, b_2 \rangle \in f(b_1), \langle \alpha_3, b' \rangle \in \text{upd}(b_2)}{\langle \alpha_1 \cdot \alpha_2 \cdot \alpha_3, b' \rangle \in f(b)} \text{BUFF-UPD}$$

$$\frac{\langle \alpha, b_1 \rangle \in f(b), b_1 \triangleright b'}{\langle \alpha, b' \rangle \in f(b)} \text{BUFF-RED}$$

Definition 4. Let $f : \text{Buff} \rightarrow \mathcal{P}((\Sigma \times \Sigma)^* \times \text{Buff})$. Then f^\ddagger is the smallest function (in the pointwise order) such that:

1. For all b , $f(b)$ is stuttering and mumbling closed.
2. f is buffer-update and buffer-reduction closed.

If $f = f^\ddagger$, we say f is closed. Any command yields a closed function.

Lemma 5. For every command C , $(\mathcal{T}[\llbracket C \rrbracket])^\ddagger = \mathcal{T}[\llbracket C \rrbracket]$.

The following auxiliary definitions enable us to describe the equations satisfied by the transition traces semantics. Let h be a partial function from buffers to sets of transition traces such that $\forall b \in \text{Buff}. (\exists b_1 \in \text{dom}(h). (\exists b' \in \text{Buff}. b = b' \# b_1))$; then, there is a unique closed function that contains h . Formally, we overload the closure notation and write:

$$h^\ddagger = \lambda b. \{ \langle \alpha \cdot \beta, b' \rangle \mid \langle \alpha, b_1 \rangle \in \text{upd}(b), \langle \beta, b' \rangle \in h(b_1) \}^\ddagger$$

We define the operator $\parallel : (\Sigma \times \Sigma)^* \times (\Sigma \times \Sigma)^* \rightarrow \mathcal{P}^+((\Sigma \times \Sigma)^*)$ that yields the set of all interleavings of its arguments. We write it infix and define it inductively.

$$\alpha \parallel \varepsilon = \{ \alpha \}$$

$$\frac{\beta \in \alpha_1 \parallel \alpha_2}{\beta \in \alpha_2 \parallel \alpha_1}$$

$$\frac{\beta \in \alpha_1 \parallel \alpha_2}{(s_0, s'_0) \cdot \beta \in ((s_0, s'_0) \cdot \alpha_1) \parallel \alpha_2}$$

We say that the system does not alter x in $(s_0, s'_0) \cdots (s_n, s'_n)$ if $\forall k \in [1, n]. s_k(x) = s'_k(x)$ and we use $(\Sigma \times \Sigma)_{x^+}^*$ for the set of such transition sequences. We say that the environment does not alter x in $(s_0, s'_0) \cdots (s_n, s'_n)$, if $\forall k \in [1, n-1]. s'_k(x) = s_{k+1}(x)$ and we use $(\Sigma \times \Sigma)_{x^-}^*$ for the set of such transition sequences. We write $\alpha|_x = \beta|_x$ if traces α and β are identical except for the values of reference x . We write $\text{Buff}|_x$ for the set of buffers that do not have x in their domain. We let $\llbracket E=0 \rrbracket = \lambda b. \{ \langle (s, s), b \rangle \mid \llbracket E \rrbracket(s[b]) = 0 \}^\ddagger$ and similarly for $\llbracket E \neq 0 \rrbracket$.

The transition traces semantics from [Theorem 2](#) satisfies the equations of [Figure 3](#).

Lemma 6. For every command C , $\llbracket C \rrbracket = \mathcal{T}[\llbracket C \rrbracket]$

The proof is a straightforward structural induction on the command, and we elide it in the interests of space. In this light, we are able to freely interchange $\llbracket C \rrbracket$ and $\mathcal{T}[\llbracket C \rrbracket]$ in the rest of the paper.

$$\begin{aligned}
\llbracket \text{skip} \rrbracket &= \lambda b . \{ \langle \varepsilon, b \rangle \}^\dagger \\
\llbracket C; D \rrbracket &= \lambda b . \{ \langle \alpha \cdot \beta, b' \rangle \mid \exists b_1 \in \text{Buff} . \langle \alpha, b_1 \rangle \in \llbracket C \rrbracket(b), \langle \beta, b' \rangle \in \llbracket D \rrbracket(b_1) \}^\dagger \\
\llbracket \text{mfence} \rrbracket &= \lambda b . \{ \langle \alpha, \emptyset \rangle \in \llbracket \text{skip} \rrbracket(b) \} \\
\llbracket x := E \rrbracket &= \lambda b . \{ \langle (s, s), b \uparrow [x := v] \rangle \mid \llbracket E \rrbracket(s[b] = v) \}^\dagger \\
\llbracket \text{if } E \text{ then } C \text{ else } D \rrbracket &= \llbracket E=0 \rrbracket; \llbracket D \rrbracket \cup \llbracket E \neq 0 \rrbracket; \llbracket C \rrbracket \\
\llbracket \text{while } E \text{ do } C \rrbracket &= (\llbracket E \neq 0 \rrbracket; \llbracket C \rrbracket)^*; \llbracket E=0 \rrbracket \\
\llbracket \text{await } E \text{ then } C \rrbracket &= \lambda b \in \{ \emptyset \} . \{ \langle (s, s'), \emptyset \rangle \mid \llbracket E \rrbracket(s) \neq 0, \langle (s, s'), \emptyset \rangle \in \llbracket C \rrbracket(\emptyset) \}^\dagger \\
\llbracket C_1 \parallel C_2 \rrbracket &= \lambda b \in \{ \emptyset \} . \{ \langle \beta, \emptyset \rangle \mid \beta \in \beta_1 \parallel \beta_2, \forall i \in [1, 2] . \langle \beta_i, \emptyset \rangle \in \llbracket C_i \rrbracket(\emptyset) \}^\dagger \\
\llbracket \text{local } x \text{ in } C \rrbracket &= \lambda b \in \text{Buff}|_x . \{ \langle \beta, b' |_x \rangle \mid \beta \in (\Sigma \times \Sigma)_{x^+}^*, \\
&\quad \exists \langle \beta_1, b' \rangle \in \llbracket C \rrbracket(b) . \beta_1 \in (\Sigma \times \Sigma)_{x^-}^* \ \& \ \beta |_x = \beta_1 |_x \}^\dagger
\end{aligned}$$

Fig. 3. Denotational semantics of TSO + await

4 Full Abstraction

In this section we follow [Brookes \[1996\]](#) as closely as possible in order to highlight the differences caused by TSO.

The input-output relation of a program is defined using only the shared memory, i.e. the program is started with an empty buffer and the output state is observed when the buffer is empty.

Definition 7 (IO). For every command C , $\text{IO}\llbracket C \rrbracket = \{ (s, s') \mid \langle (s, s'), \emptyset \rangle \in \mathcal{T}\llbracket C \rrbracket(\emptyset) \}$

Definition 8. The trace $\alpha = (s_0, s'_0) \cdots (s_n, s'_n)$ is *Interference Free (IF)* if and only if for all $i \in [0, n-1]$ we have $s'_i = s_{i+1}$.

Notice that every $(s, s') \in \text{IO}\llbracket C \rrbracket$ arises from the mumbling closure of IF traces.

We add the following notations for technical convenience:

$$\begin{aligned}
\text{IO}\llbracket C \rrbracket /_s &= \{ (s, s') \mid (s, s') \in \text{IO}\llbracket C \rrbracket \} \\
\llbracket C \rrbracket(b) /_s &= \{ \langle \alpha, b' \rangle \mid \langle \alpha, b' \rangle \in \llbracket C \rrbracket(b) \ \& \ \alpha = (s, s') \cdot \alpha' \}
\end{aligned}$$

Definition 9. The operational ordering compares the IO relation of commands in all possible command contexts $\mathbb{C}[-]$,

$$C \leq_{IO} D \iff \forall \mathbb{C}[-], s . \text{FV}(\mathbb{C}[C]) \cup \text{FV}(\mathbb{C}[D]) \subseteq \text{dom}(s) \Rightarrow \text{IO}\llbracket \mathbb{C}[C] \rrbracket /_s \subseteq \text{IO}\llbracket \mathbb{C}[D] \rrbracket /_s$$

Definition 10. There is a natural ordering induced by the denotational semantics,

$$C \sqsubseteq D \iff \forall b, s . \text{FV}(C) \cup \text{FV}(D) \cup \text{dom}(b) \subseteq \text{dom}(s) \Rightarrow \llbracket C \rrbracket(b) /_s \subseteq \llbracket D \rrbracket(b) /_s$$

In the rest of this section, we prove that \sqsubseteq and \leq_{IO} coincide.

From [Figure 3](#), it is evident that all the program combinators are monotone with respect to set inclusion. Thus, we deduce the following lemma.

Lemma 11 (Compositional Monotonicity). For all commands C and D ,

$$C \sqsubseteq D \Rightarrow \forall \mathbb{C}[-] . \mathbb{C}[C] \sqsubseteq \mathbb{C}[D]$$

Since $\llbracket \cdot \rrbracket$ and $\mathcal{T}\llbracket \cdot \rrbracket$ coincide by [Theorem 6](#) we obtain:

Corollary 12 (Adequacy). *For all commands C and D , we have $C \sqsubseteq D \Rightarrow C \leq_{IO} D$.*

We now introduce some macros that we will use for the following developments. For all memories s, s' and buffer b , there is evidently an expression IS_s such that

$$\llbracket IS_s \rrbracket (s'[b]) \neq 0 \Leftrightarrow \text{dom}(s) = \text{dom}(s') \ \& \ (\forall x \in \text{dom}(s) . s(x) = s'[b](x))$$

Moreover, for all memories s, s' and buffer b , there is evidently a program consisting of a sequence of assignments $MAKE_s$ such that

$$s', \langle b, MAKE_s \rangle \rightarrow^* s, \langle \emptyset, \text{skip} \rangle$$

Finally, for each buffer b , there is evidently a program consisting of a sequence of assignments $MAKE_b$ such that for any s, b'

$$s, \langle b', MAKE_b \rangle \rightarrow^* s[b'], \langle b, \text{skip} \rangle$$

The program $MAKE_b$ can be used to encode input buffers as a command context.

Lemma 13. *For any command C and buffers b and b' , $\llbracket MAKE_{b'}; C \rrbracket (b) = \llbracket C \rrbracket (b ++ b')$.*

Proof (SKETCH). By induction on the length of b' . The base case is immediate and the inductive case follows from the definition of sequential composition.

Corollary 14. *For all C_1 and C_2 , $C_1 \not\sqsubseteq C_2 \Rightarrow \exists C, s. \llbracket C; C_1 \rrbracket (\emptyset) / s \not\sqsubseteq \llbracket C; C_2 \rrbracket (\emptyset) / s$.*

Proof. If $\llbracket C_1 \rrbracket (b) / s \not\sqsubseteq \llbracket C_2 \rrbracket (b) / s$, choose $C = MAKE_b$.

For the proof of our main result we will need to encode a context that simulates the environment of an arbitrary trace α . To that end we define the following program.

Definition 15. *Given $\alpha = (s_0, s'_0) \cdots (s_n, s'_n)$, define the command $SIMULATE_\alpha$ as*

$$\begin{aligned} SIMULATE_\alpha = & \text{await } IS_{s_0} \text{ then skip;} \\ & \text{await } IS_{s'_0} \text{ then } MAKE_{s_1}; \\ & \text{await } IS_{s'_1} \text{ then } MAKE_{s_2}; \\ & \dots \\ & \text{await } IS_{s'_{n-1}} \text{ then } MAKE_{s_n} \end{aligned}$$

Intuitively, $\llbracket SIMULATE_\alpha \rrbracket$ is given by the closure of the single trace that is ‘‘complementary’’ to α . Formally,

$$\llbracket SIMULATE_\alpha \rrbracket = \lambda b \in \{\emptyset\} . \{ \langle (s'_0, s_1) \cdot (s'_1, s_2) \cdots (s'_{n-1}, s_n), \emptyset \rangle \}^\dagger$$

Lemma 16. *Given α as in [Theorem 15](#) letting $\{flag, finish\}$ be disjoint from $FV(C) \cup \text{dom}(b) \cup \bigcup_i (\text{dom}(s_i) \cup \text{dom}(s'_i))$, and considering the command context,*

$$\mathbb{C}[-] = \text{flag} := 0; \text{finish} := 0; \left(\begin{array}{c} MAKE_{b_0}; \\ [-] \end{array} \parallel SIMULATE_\alpha \right)$$

we obtain $\langle \alpha, b \rangle \in \llbracket MAKE_{b_0}; C \rrbracket (\emptyset) \iff \langle \alpha_0 \cdot (s_0, s'_n) \cdot \alpha_1, \emptyset \rangle \in \llbracket C[C] \rrbracket (\emptyset)$, where $\langle \alpha', \emptyset \rangle \in \llbracket SIMULATE_\alpha \rrbracket (\emptyset)$, $(s_0, s'_n) \in (\alpha \parallel \alpha')^\ddagger$, $\langle \alpha_0, \emptyset \rangle \in \text{upd}([flag := 0, finish := 0])$, and $\langle \alpha_1, \emptyset \rangle \in \text{upd}(b)$.

This lemma characterizes the IF traces where the final state before flushing the final buffer b is s'_n , the first state is s_0 and the trace terminates by flushing the buffer b . The variables $flag$ and $finish$ play essentially no role in this lemma and are included only to accommodate the use-case later.

The proof follows [Brookes \[1996\]](#). For the forward direction, if $\langle \alpha, b \rangle \in \llbracket C \rrbracket(\emptyset)$, the IF trace $\langle (s_0, s'_0) \cdot (s'_0, s_1) \cdot (s_1, s'_1) \cdot (s'_1, s_2) \cdots (s'_{n-1}, s_n) \cdot (s_n, s'_n), b \rangle$ is in $\llbracket C[C] \rrbracket(\emptyset)$ by interleaving. Thus, by mumbling closure, $\langle (s_0, s'_n), b \rangle \in \llbracket C[C] \rrbracket(\emptyset)$. Conversely, $\langle (s_0, s'_n), b \rangle \in \llbracket C[C] \rrbracket(\emptyset)$ for some b only if there is some β that can be interleaved with $(s'_0, s_1) \cdot (s'_1, s_2) \cdots (s'_{n-1}, s_n)$ to fill up the gaps between s_i and s'_i for all i . Such a trace yields α by stuttering and mumbling.

A significant difference from [Brookes \[1996\]](#) is that we need to check that the final buffers – since they are part of the trace semantics – coincide. To that end, we define the following program $CHECK_b$ that “observes” all the updates of buffer b as they are performed one by one into the memory.

Definition 17. For any buffer $b = [x_1 := v_1, \dots, x_n := v_n]$ and memories s and \bar{s} , define:

$$\begin{aligned} CHECK_{b,s,\bar{s}} = & \text{await } IS_s \text{ then } MAKE_{\bar{s}}; \\ & \text{await } IS_{\bar{s}[x_1 := v_1]} \text{ then } MAKE_{\bar{s}}; \\ & \dots \\ & \text{await } IS_{\bar{s}[x_n := v_n]} \text{ then } MAKE_{\bar{s}} \end{aligned}$$

Informally, the program $CHECK_b$ starts by replacing the state s for a state \bar{s} . In our use case, \bar{s} maps every variable to values that do not appear in the trace generating the state s . The await commands are intended to observe each update from the buffer b of another thread. Upon observing each update in state \bar{s} that state is reinitialized to observe the following buffer update.

Lemma 18. Let $\{flag, finish\}$ be disjoint from $FV(C) \cup \text{dom}(b) \cup_i (\text{dom}(s_i) \cup \text{dom}(s'_i))$. Let \bar{s} be any memory such that the range of \bar{s} is disjoint from the range of s and b . Considering the command context

$$\begin{aligned} C[-] = & flag := 0; finish := 0; \\ & \left(\begin{array}{l|l} \begin{array}{l} MAKE_{b_0}; \\ [-]; \\ \text{if } flag \text{ then } finish := 1 \end{array} & \begin{array}{l} D; \\ \text{await } 1 \text{ then } flag := 1; \\ \text{await } 1 \text{ then } flag := 0; \\ CHECK_{b,s,\bar{s}}; \\ \text{await } IS_{\bar{s}[finish:=1]} \text{ then skip} \end{array} \end{array} \right) \end{aligned}$$

there exist α_0 and α_1 such that $\langle \alpha_0, b \rangle \in \llbracket C \rrbracket(b_0)$ and $\langle \alpha_1, \varepsilon \rangle \in \llbracket D \rrbracket(\varepsilon)$ with $(s_0, s) \in (\alpha_0 \parallel \alpha_1)^\ddagger$ if and only if $\text{IO}\llbracket C[C] \rrbracket(\emptyset) \neq \emptyset$.

Proof (SKETCH). Let $\langle \alpha_0, b \rangle \in \llbracket C \rrbracket(b_0)$ and $\langle \alpha_1, \emptyset \rangle \in \llbracket D \rrbracket$ such that $(s_0, s) \in (\alpha_0 \parallel \alpha_1)^\ddagger$. Consider the execution given by the following interleaving:

- obviously we start by executing the initial assignments of $flag$ and $finish$, which are updated before spawning the new threads,
- C, D execute with an appropriate interleaving to yield the shared memory s and a buffer b' for the thread on the left of the parallel component and an empty buffer for the thread on the right, where $b' \triangleright b$,

- we then execute the first await on the right hand of the parallel composition to set $flag$ in shared memory,
- the left thread of the parallel composition observes the update on $flag$ and sets $finish$ and this update is added to the buffer of the left hand thread,
- the await on the right hand thread executes unsetting $flag$ in shared memory,
- $CHECK_{b,s,\bar{s}}$ terminates successfully since the individual awaits can be interleaved with the propagation of buffer updates from b into the shared memory,
- the update to $finish$ moves into shared memory from the buffer of left thread. Since b was exhausted in the previous step, there is no change in shared memory on $dom(\bar{s})$,
- the final await in the right thread terminates successfully because $finish$ is set and the state remains at $\bar{s}[finish := 1]$.

□

Lemma 19. $C_1 \not\sqsubseteq C_2 \Rightarrow C_1 \not\leq_{IO} C_2$

Proof (SKETCH). We have to construct a command context to distinguish the IO behavior of C_1, C_2 . By [Theorem 14](#), we can assume that $\llbracket MAKE_{b_0}; C_1 \rrbracket(\emptyset) \not\sqsubseteq \llbracket MAKE_{b_0}; C_2 \rrbracket(\emptyset)$. Now let $\langle \alpha, b \rangle \in \llbracket C_1 \rrbracket(\emptyset) \setminus \llbracket C_2 \rrbracket(\emptyset)$. Consider the program context

$$\mathbb{C}[-] = flag := 0; finish := 0; \left(\begin{array}{l} MAKE_{b_0}; \\ [-]; \\ \text{if } flag \text{ then } finish := 1 \end{array} \parallel \begin{array}{l} SIMULATE_{\alpha}; \\ \text{await } 1 \text{ then } flag := 1; \\ \text{await } 1 \text{ then } flag := 0; \\ CHECK_{b,s,\bar{s}}; \\ \text{await } IS_{\bar{s}[finish:=1]} \text{ then skip} \end{array} \right)$$

where $flag, finish, \bar{s}, s$ and b satisfy the naming constraints of Lemmas [18](#) and [16](#). Since $\langle \alpha, b \rangle \in \llbracket C_1 \rrbracket(b_0)$, we use the forward direction of Lemmas [16](#) and [18](#) to deduce that $IO\llbracket C[C_1] \rrbracket(\emptyset) \neq \emptyset$. Let $IO\llbracket C[C_2] \rrbracket(\emptyset) \neq \emptyset$. Then there are α_0 and α' with $\langle \alpha_0, b \rangle \in \llbracket C_2 \rrbracket(b_0)$ and $\langle \alpha', \emptyset \rangle \in \llbracket SIMULATE_{\alpha} \rrbracket$ such that $(s_0, s) \in \{\alpha_0 \parallel \alpha'\}^{\ddagger}$. So by [Theorem 18](#) $\langle \alpha, b \rangle \in \llbracket C_2 \rrbracket(b_0)$, which is a contradiction. □

Combining [Theorem 12](#) and [Theorem 19](#), we deduce that the denotational semantics $\llbracket \cdot \rrbracket$ is inequationally fully abstract.

Theorem 20 (Full Abstraction). *For any commands C and D we have*

$$C \sqsubseteq D \iff C \leq_{IO} D$$

A simple corollary of the proof of [Theorem 19](#) is that it suffices to consider simple sequential contexts to prove inter-substitutivity of programs. For a given sequential command D and a given b , consider:

$$C_D^b[-] = flag := 0; finish := 0; \left(\begin{array}{l} MAKE_b; \\ [-]; \\ \text{if } flag \text{ then } finish := 1 \end{array} \parallel D \right)$$

where *flag* and *finish* satisfy the naming constraints of Lemmas [18](#) and [16](#). Then:

$$C_1 \sqsubseteq C_2 \iff \forall D, b. (\text{IO}[\llbracket C_D^b[C_1] \rrbracket] \neq \emptyset \Rightarrow \text{IO}[\llbracket C_D^b[C_2] \rrbracket] \neq \emptyset)$$

This validates the folklore analysis of TSO programs using only sequential testers in parallel.

5 Examples and Laws

We examine some laws of parallel programming under a TSO memory model, and consider some standard TSO examples from the perspective of the denotational semantics introduced in [Section 3](#).

Laws of parallel programming. Most of the laws inherited from [Brookes \[1996\]](#) hold in our setting.

$$\begin{aligned} \text{skip}; C &\equiv C \equiv C; \text{skip} & (1) \\ (C_1; C_2); C_3 &\equiv C_1; (C_2; C_3) & (2) \\ C_1 \parallel C_2 &\equiv C_2 \parallel C_1 & (3) \\ (C_1 \parallel C_2) \parallel C_3 &\equiv C_1 \parallel (C_2 \parallel C_3) & (4) \\ (\text{if } E \text{ then } C_0 \text{ else } C_1); C &\equiv \text{if } E \text{ then } C_0; C \text{ else } C_1; C & (5) \\ \text{while } E \text{ do } C &\equiv \text{if } E \text{ then } (C; \text{while } E \text{ do } C) \text{ else skip} & (6) \end{aligned}$$

In (1) and (2) we see that sequential composition is associative with unit skip. Laws (3) and (4) say that parallel composition is commutative and associative. However, skip is not a unit for parallel composition in general, since parallel composition requires flushing the buffers before spawning the threads and when synchronizing them at the end. Instead what holds is:

$$\text{skip} \parallel C \equiv (\text{mfence}; C; \text{mfence})$$

Law (5) implies that sequential composition distributes into conditionals, and finally law (6) is the usual unrolling law for while loops. Also, The usual laws for local variables hold. If x is not free in C then:

$$\begin{aligned} \text{local } x \text{ in } C &\equiv C \\ \text{local } x \text{ in } C; D &\equiv C; \text{local } x \text{ in } D \\ \text{local } x \text{ in } (C \parallel D) &\equiv C \parallel \text{local } x \text{ in } D \end{aligned}$$

Thread inlining. Thread inlining is always sound in [Brookes \[1996\]](#), where for example the following rule holds

$$x := y; C \sqsubseteq x := y; \parallel C$$

In our setting however, this equation holds only if C does not read reference x . In the case where C reads x , C in the left hand side can potentially access newer local updates that are not available globally. In this case, a mfence is needed to validate the equation:

$$x := y; \text{mfence}; C \sqsubseteq x := y \parallel C$$

$$\begin{array}{c}
 \left[\begin{array}{l} \text{local } r_0, r_1 \text{ in} \\ x := 1; \\ r_0 := x; \\ r_1 := y \end{array} \right] \parallel \left[\begin{array}{l} \text{local } r_2, r_3 \text{ in} \\ y := 1; \\ r_2 := y; \\ r_3 := x \end{array} \right] \\
 \text{Possible: } r_0 = r_2 = 1 \ \& \ r_1 = r_3 = 0 \\
 \text{(a) Buffer Forwarding}
 \end{array}
 \quad
 \begin{array}{c}
 [x := 1] \parallel [y := 1] \parallel \left[\begin{array}{l} \text{local } r_0, r_1 \text{ in} \\ r_0 := x; \\ r_1 := y \end{array} \right] \parallel \left[\begin{array}{l} \text{local } r_2, r_3 \text{ in} \\ r_2 := y; \\ r_3 := x \end{array} \right] \\
 \text{Impossible: } r_0 = r_2 = 0 \ \& \ r_1 = r_3 = 1 \\
 \text{(b) IRIW}
 \end{array}$$

Fig. 4. TSO Examples

Commutation of independent statements. The TSO memory model permits reads to move ahead of previous writes on independent references. This is generally seen with the example below. Using the denotational semantics, we are able to prove the inequality, and moreover the denotations imply the existence of counterexamples to show that the inequality cannot be strengthened to an equality. Thus we get:

$$\left[\begin{array}{l} \text{local } r \text{ in} \\ r := y; \\ x := 1; \\ z := r; \end{array} \right] \sqsubseteq \left[\begin{array}{l} \text{local } r \text{ in} \\ x := 1; \\ r := y; \\ z := r; \end{array} \right] \quad \& \quad \left[\begin{array}{l} \text{local } r \text{ in} \\ x := 1; \\ r := y; \\ z := r; \end{array} \right] \not\sqsubseteq \left[\begin{array}{l} \text{local } r \text{ in} \\ r := y; \\ x := 1; \\ z := r; \end{array} \right]$$

In general, TSO does not permit writes of independent references or reads of independent reference to commute. However, a special case of this latter class of transformation can be modeled by the capability of reading one threads own writes (as shown in the example of Figure 4a). Notice in particular that the example in Figure 4a is a case of inlining of the standard IRIW example (shown in Figure 4b), which provides evidence of our previous claim that inlining is not a legal TSO transformation in general. Our denotational semantics is able to explain this relaxed behavior by means of the inequalities below. In particular, the one on the right can be proved using the inequality discussed above and the one on the left.

$$\left[\begin{array}{l} \text{local } r \text{ in} \\ x := 1; \\ r := 1; \\ z := r \end{array} \right] \sqsubseteq \left[\begin{array}{l} \text{local } r \text{ in} \\ x := 1; \\ r := x; \\ z := r \end{array} \right] \quad \left[\begin{array}{l} \text{local } r_1, r_2 \text{ in} \\ r_2 := y; \\ x := 1; \\ r_1 := 1; \\ z_0 := r_1; z_1 := r_2 \end{array} \right] \sqsubseteq \left[\begin{array}{l} \text{local } r_1, r_2 \text{ in} \\ x := 1; \\ r_1 := x; \\ r_2 := y; \\ z_0 := r_1; z_1 := r_2 \end{array} \right]$$

6 Conclusion

We describe how to modify the Brookes semantics for a shared variable parallel programming language Brookes [1996] to address the TSO relaxed memory model. We view our results as the foundations towards two developments: (a) separation logics for relaxed memory models, and (b) refinement theory for relaxed memory models.

References

- Adve, S.V., Boehm, H.-J.: Memory models: a case for rethinking parallel languages and hardware. *Commun. ACM* 53, 90–101 (2010) ISSN 0001-0782
- Adve, S.V., Gharachorloo, K.: Shared memory consistency models: A tutorial. *Computer* 29(12), 66–76 (1996)
- Adve, S.V., Hill, M.D.: Weak ordering - a new definition. In: ISCA, pp. 2–14 (1990)
- Boudol, G., Petri, G.: Relaxed memory models: an operational approach. In: POPL, pp. 392–403 (2009)
- Brookes, S.: A semantics for concurrent separation logic. *Theor. Comput. Sci.* 375(1-3), 227–270 (2007)
- Brookes, S.: Full abstraction for a shared-variable parallel language. *Inf. Comput.* 127(2), 145–163 (1996)
- Jagadeesan, R., Pitcher, C., Riely, J.: Generative Operational Semantics for Relaxed Memory Models. In: Gordon, A.D. (ed.) ESOP 2010. LNCS, vol. 6012, pp. 307–326. Springer, Heidelberg (2010)
- Lampert, L.: How to make a multiprocessor computer that correctly executes multiprocess program. *IEEE Trans. Comput.* 28(9), 690–691 (1979)
- Manson, J., Pugh, W., Adve, S.V.: The java memory model. In: POPL, pp. 378–391 (2005)
- O’Hearn, P.W.: Resources, concurrency, and local reasoning. *Theor. Comput. Sci.* 375(1-3), 271–307 (2007)
- Owens, S., Sarkar, S., Sewell, P.: A Better x86 Memory Model: x86-TSO. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 391–407. Springer, Heidelberg (2009)
- Parkinson, M.J., Bornat, R., O’Hearn, P.W.: Modular verification of a non-blocking stack. In: POPL, pp. 297–302 (2007)
- Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74 (2002)
- Sevcík, J., Vafeiadis, V., Nardelli, F.Z., Jagannathan, S., Sewell, P.: Relaxed-memory concurrency and verified compilation. In: POPL, pp. 43–54 (2011)
- Sewell, P., Sarkar, S., Owens, S., Nardelli, F.Z., Myreen, M.O.: x86-TSO: a rigorous and usable programmer’s model for x86 multiprocessors. *Commun. ACM* 53(7), 89–97 (2010)
- Inc. CORPORATE. SPARC. The SPARC Architecture Manual (version 9). Prentice-Hall, Inc., Upper Saddle River (1994)
- Turon, A.J., Wand, M.: A separation logic for refining concurrent objects. *SIGPLAN Not.* 46, 247–258 (2011) ISSN 0362-1340
- Vafeiadis, V., Parkinson, M.J.: A Marriage of Rely/Guarantee and Separation Logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 256–271. Springer, Heidelberg (2007)

Revisiting Trace and Testing Equivalences for Nondeterministic and Probabilistic Processes^{*}

Marco Bernardo¹, Rocco De Nicola², and Michele Loreti³

¹ Dipartimento di Scienze di Base e Fondamenti, Università di Urbino, Italy

² IMT, Istituto Alti Studi Lucca, Italy

³ Dipartimento di Sistemi e Informatica, Università di Firenze, Italy

Abstract. One of the most studied extensions of testing theory to nondeterministic and probabilistic processes yields unrealistic probabilities estimations that give rise to two anomalies. First, probabilistic testing equivalence does not imply probabilistic trace equivalence. Second, probabilistic testing equivalence differentiates processes that perform the same sequence of actions with the same probability but make internal choices in different moments and thus, when applied to processes without probabilities, does not coincide with classical testing equivalence. In this paper, new versions of probabilistic trace and testing equivalences are presented for nondeterministic and probabilistic processes that resolve the two anomalies. Instead of focussing only on suprema and infima of the set of success probabilities of resolutions of interaction systems, our testing equivalence matches all the resolutions on the basis of the success probabilities of their identically labeled computations. A simple spectrum is provided to relate the new relations with existing ones. It is also shown that, with our approach, the standard probabilistic testing equivalences for generative and reactive probabilistic processes can be retrieved.

1 Introduction

The testing theory for concurrent processes [5] is based on the idea that two processes are equivalent if and only if they cannot be told apart when interacting with their environment, which is represented by arbitrary processes with distinguished successful actions or states, often called observers. In case purely nondeterministic processes are considered, this approach has been very successful and the induced relations enjoy a number of interesting properties. Testing equivalence has been used in many contexts and it is a good compromise between abstraction and inspection capabilities; it distinguishes processes that have different behaviors with respect to deadlock, but abstracts from the exact moment in which a process performs internal (unobservable) choices.

When probabilities enter the game, the possible choices in defining the set of observers, in deciding how to resolve nondeterminism, or in assembling the results of the observations are many more and can give rise to significantly different behavioral relations.

^{*} Work partially supported by the FP7-IST-FET Project ASCENS, grant n. 257414.

One of the most studied variants of testing equivalence for nondeterministic and probabilistic processes [17,10,14,6] considers the probability of performing computations along which the same tests are passed. Due to the possible presence of equally labeled transitions departing from the same state, there is not necessarily a single probability value with which a nondeterministic and probabilistic process passes a test. Given two states s_1 and s_2 and the initial state o of an observer, the above mentioned probabilistic testing equivalence computes the probability of performing a successful computation from (s_1, o) and (s_2, o) in every resolution of the interaction systems, then it compares the suprema (\sqcup) and the infima (\sqcap) of these values over all possible resolutions of the interaction systems. This equivalence, which we denote by $\sim_{\text{PTe},\sqcup\sqcap}$, enjoys nice properties and possesses logical and equational characterizations but, if contrasted with classical testing for purely nondeterministic processes [5], suffers from two anomalies.

The first anomaly is that $\sim_{\text{PTe},\sqcup\sqcap}$ is not always included in one of the most well-studied probabilistic versions of trace equivalence, namely $\sim_{\text{PTr},\text{dis}}$ of [13]. Actually, the inclusion depends on the class of schedulers used for deriving resolutions of interaction systems. It holds if randomized schedulers are admitted [14], while it does not hold when only deterministic schedulers are considered like in [17,10,6]. This anomaly could be solved by (i) considering a coarser probabilistic trace equivalence $\sim_{\text{PTr},\text{new}}$ that compares the execution probabilities of single traces rather than trace distributions and (ii) replacing $\sim_{\text{PTe},\sqcup\sqcap}$ with a finer probabilistic testing equivalence $\sim_{\text{PTe},\text{new}}$, which does not focus only on the highest and the lowest probability of passing a test but matches the maximal resolutions of the interaction systems according to their success probability. Unfortunately, $\sim_{\text{PTe},\text{new}}$ does not overcome the other anomaly.

The second anomaly of $\sim_{\text{PTe},\sqcup\sqcap}$ (which also affects the testing equivalence of [14]) is that, when used to test purely nondeterministic processes, it does not preserve classical testing equivalence. In fact, given two fully nondeterministic processes that are testing equivalent according to [5], they may be told apart by $\sim_{\text{PTe},\sqcup\sqcap}$ because observers with probabilistic choices make the latter equivalence sensitive to the moment of occurrence of internal choices, thus yielding unrealistic probability estimations. This problem has been recently tackled in [8] for a significantly different probabilistic model by relying on a label massaging that avoids over-/under-estimations of success probabilities in a parallel context.

In this paper, by using $\sim_{\text{PTe},\text{new}}$ as a stepping stone, we propose a new probabilistic testing equivalence, $\sim_{\text{PTe},\text{tbt}}$, that solves both anomalies by matching success probabilities in a trace-by-trace fashion rather than on entire resolutions. With respect to [8], the interesting feature of our definition is that it does not require any model modification. We also show that the standard notions of testing equivalences for generative probabilistic processes and reactive probabilistic processes can be redefined, by following the same approach taken for the general model, without altering their discriminating power. Finally, we relate $\sim_{\text{PTe},\text{tbt}}$ with some of the other probabilistic equivalences by showing that it is comprised between $\sim_{\text{PTr},\text{new}}$ and a novel probabilistic failure equivalence $\sim_{\text{PF},\text{new}}$, which in turn is comprised between $\sim_{\text{PTe},\text{tbt}}$ and $\sim_{\text{PTe},\text{new}}$.

2 Nondeterministic and Probabilistic Processes

Processes combining nondeterminism and probability are typically represented by means of extensions of labeled transitions systems (LTS), in which every action-labeled transition goes from a source state to a probability distribution over target states rather than to a single target state. The resulting processes are essentially Markov decision processes [7] and correspond to a number of slightly different probabilistic computational models including real nondeterminism, among which we mention concurrent Markov chains [16], alternating probabilistic models [9,17], and probabilistic automata in the sense of [12].

Definition 1. *A nondeterministic and probabilistic labeled transition system, NPLTS for short, is a triple (S, A, \longrightarrow) where S is an at most countable set of states, A is a countable set of transition-labeling actions, and $\longrightarrow \subseteq S \times A \times \text{Distr}(S)$ is a transition relation where $\text{Distr}(S)$ is the set of discrete probability distributions over S . ■*

Given a transition $s \xrightarrow{a} \mathcal{D}$, we say that $s' \in S$ is not reachable from s via that transition if $\mathcal{D}(s') = 0$, otherwise we say that it is reachable with probability $p = \mathcal{D}(s')$. The choice among all the transitions departing from s is external and nondeterministic, while the choice of the target state for a specific transition is internal and probabilistic. A NPLTS represents (i) a fully nondeterministic process when every transition leads to a distribution that concentrates all the probability mass into a single target state, (ii) a fully probabilistic process when every state has at most one outgoing transition, or (iii) a reactive probabilistic process [15] when no state has several transitions labeled with the same action.

A NPLTS can be depicted as a directed graph-like structure in which vertices represent states and action-labeled edges represent action-labeled transitions. Given a transition $s \xrightarrow{a} \mathcal{D}$, the corresponding a -labeled edge goes from the vertex representing state s to a set of vertices linked by a dashed line, each of which represents a state s' such that $\mathcal{D}(s') > 0$ and is labeled with $\mathcal{D}(s')$ – label omitted if $\mathcal{D}(s') = 1$. Figure 1 shows six NPLTS models, with the first two mixing nondeterminism and probability and the last four being fully probabilistic.

In this setting, a computation is a sequence of state-to-state steps (denoted by \longrightarrow_s) derived from the state-to-distribution transitions of the NPLTS.

Definition 2. *Let $\mathcal{L} = (S, A, \longrightarrow)$ be a NPLTS, $n \in \mathbb{N}$, $s_i \in S$ for all $i = 0, \dots, n$, and $a_i \in A$ for all $i = 1, \dots, n$. We say that:*

$$c \equiv s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \dots s_{n-1} \xrightarrow{a_n} s_n$$

is a computation of \mathcal{L} of length n going from s_0 to s_n iff for all $i = 1, \dots, n$ there exists a transition $s_{i-1} \xrightarrow{a_i} \mathcal{D}_i$ such that $\mathcal{D}_i(s_i) > 0$, with $\mathcal{D}_i(s_i)$ being the execution probability of step $s_{i-1} \xrightarrow{a_i} s_i$ of c conditioned on the selection of transition $s_{i-1} \xrightarrow{a_i} \mathcal{D}_i$ of \mathcal{L} at state s_{i-1} . ■

In the following, given $s \in S$ we denote by $\mathcal{C}_{\text{fin}}(s)$ the set of finite-length computations from s . Given $c \in \mathcal{C}_{\text{fin}}(s)$, we say that c is maximal iff it cannot be further extended, i.e., it is not a proper prefix of any other element of $\mathcal{C}_{\text{fin}}(s)$.

In order to define testing equivalence, we also need to introduce the notion of parallel composition of two NPLTS models, borrowed from [10].

Definition 3. Let $\mathcal{L}_i = (S_i, A, \longrightarrow_i)$ be a NPLTS for $i = 1, 2$. The parallel composition of \mathcal{L}_1 and \mathcal{L}_2 is the NPLTS $\mathcal{L}_1 \parallel \mathcal{L}_2 = (S_1 \times S_2, A, \longrightarrow)$ where $\longrightarrow \subseteq (S_1 \times S_2) \times A \times \text{Distr}(S_1 \times S_2)$ is such that $(s_1, s_2) \xrightarrow{a} \mathcal{D}$ iff $s_1 \xrightarrow{a} \mathcal{D}_1$ and $s_2 \xrightarrow{a} \mathcal{D}_2$ with $\mathcal{D}(s'_1, s'_2) = \mathcal{D}_1(s'_1) \cdot \mathcal{D}_2(s'_2)$ for each $(s'_1, s'_2) \in S_1 \times S_2$. ■

3 Trace Equivalences for NPLTS Models

Trace equivalence for NPLTS models [13] examines the probability of performing computations labeled with the same action sequences for each possible way of solving nondeterminism. To formalize this for a NPLTS \mathcal{L} , given a state s of \mathcal{L} we take the set of resolutions of s . Each of them is a tree-like structure whose branching points represent probabilistic choices. This is obtained by unfolding from s the graph structure underlying \mathcal{L} and by selecting at each state a single transition of \mathcal{L} – deterministic scheduler – or a convex combination of equally labeled transitions of \mathcal{L} – randomized scheduler – among all the transitions possible from that state. Below, we introduce the notion of resolution arising from a deterministic scheduler as a fully probabilistic NPLTS. In this case, resolutions coincide with computations if \mathcal{L} is fully nondeterministic.

Definition 4. Let $\mathcal{L} = (S, A, \longrightarrow)$ be a NPLTS and $s \in S$. We say that a NPLTS $\mathcal{Z} = (Z, A, \longrightarrow_{\mathcal{Z}})$ is a resolution of s obtained via a deterministic scheduler iff there exists a state correspondence function $\text{corr} : Z \rightarrow S$ such that $s = \text{corr}(z_s)$, for some $z_s \in Z$, and for all $z \in Z$:

- If $z \xrightarrow{a} \mathcal{Z} \mathcal{D}$, then $\text{corr}(z) \xrightarrow{a} \mathcal{D}'$ with $\mathcal{D}(z') = \mathcal{D}'(\text{corr}(z'))$ for all $z' \in Z$.
- If $z \xrightarrow{a_1} \mathcal{Z} \mathcal{D}_1$ and $z \xrightarrow{a_2} \mathcal{Z} \mathcal{D}_2$, then $a_1 = a_2$ and $\mathcal{D}_1 = \mathcal{D}_2$. ■

Given a state s of a NPLTS \mathcal{L} , we denote by $\text{Res}(s)$ the set of resolutions of s obtained via deterministic schedulers. Since $\mathcal{Z} \in \text{Res}(s)$ is fully probabilistic, the probability $\text{prob}(c)$ of executing $c \in \mathcal{C}_{\text{fin}}(z_s)$ can be defined as the product of the (no longer conditional) execution probabilities of the individual steps of c , with $\text{prob}(c)$ being always equal to 1 if \mathcal{L} is fully nondeterministic. This notion is lifted to $C \subseteq \mathcal{C}_{\text{fin}}(z_s)$ by letting $\text{prob}(C) = \sum_{c \in C} \text{prob}(c)$ whenever C is finite and none of its computations is a proper prefix of one of the others.

Given $\alpha \in A^*$, we then say that c is compatible with α iff the sequence of actions labeling the steps of c is equal to α . We denote by $\mathcal{CC}(z_s, \alpha)$ the set of computations in $\mathcal{C}_{\text{fin}}(z_s)$ that are compatible with α . Below we introduce a variant of the probabilistic trace distribution equivalence of [13] in which only deterministic schedulers are admitted.

Definition 5. Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are probabilistic trace distribution equivalent, written $s_1 \sim_{\text{PTr,dis}} s_2$, iff:

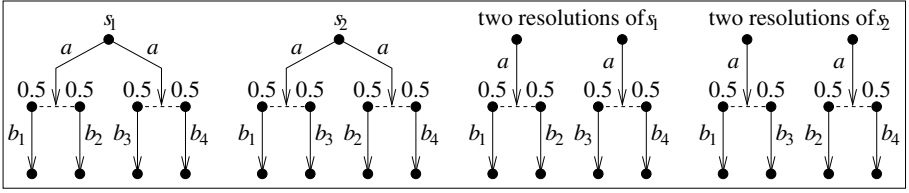


Fig. 1. Counterexample for probabilistic trace equivalences

- For each $\mathcal{Z}_1 \in \text{Res}(s_1)$ there exists $\mathcal{Z}_2 \in \text{Res}(s_2)$ such that for all $\alpha \in A^*$:
 $\text{prob}(\text{CC}(z_{s_1}, \alpha)) = \text{prob}(\text{CC}(z_{s_2}, \alpha))$
- For each $\mathcal{Z}_2 \in \text{Res}(s_2)$ there exists $\mathcal{Z}_1 \in \text{Res}(s_1)$ such that for all $\alpha \in A^*$:
 $\text{prob}(\text{CC}(z_{s_2}, \alpha)) = \text{prob}(\text{CC}(z_{s_1}, \alpha))$ ■

The relation $\sim_{\text{PTr,dis}}$ is quite discriminating, because it compares trace distributions and hence imposes a constraint on the execution probability of *all* the traces of any pair of matching resolutions. This constraint can be relaxed by considering a *single* trace at a time, i.e., by anticipating the quantification over traces. In this way, differently labeled computations of a resolution are allowed to be matched by computations of different resolutions, which leads to the following coarser probabilistic trace equivalence that we will use later on.

Definition 6. Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are probabilistic trace equivalent, written $s_1 \sim_{\text{PTr,new}} s_2$, iff for all traces $\alpha \in A^*$:

- For each $\mathcal{Z}_1 \in \text{Res}(s_1)$ there exists $\mathcal{Z}_2 \in \text{Res}(s_2)$ such that:
 $\text{prob}(\text{CC}(z_{s_1}, \alpha)) = \text{prob}(\text{CC}(z_{s_2}, \alpha))$
- For each $\mathcal{Z}_2 \in \text{Res}(s_2)$ there exists $\mathcal{Z}_1 \in \text{Res}(s_1)$ such that:
 $\text{prob}(\text{CC}(z_{s_2}, \alpha)) = \text{prob}(\text{CC}(z_{s_1}, \alpha))$ ■

Theorem 1. Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:

$$s_1 \sim_{\text{PTr,dis}} s_2 \implies s_1 \sim_{\text{PTr,new}} s_2 \quad \blacksquare$$

The inclusion of $\sim_{\text{PTr,dis}}$ in $\sim_{\text{PTr,new}}$ is strict. Indeed, if we consider the two NPLTS models on the left-hand side of Fig. 1, when $b_i \neq b_j$ for $i \neq j$ we have that $s_1 \sim_{\text{PTr,new}} s_2$ while $s_1 \not\sim_{\text{PTr,dis}} s_2$. In fact, the sets of traces of the two resolutions of s_1 depicted in the figure are $\{\varepsilon, a, a b_1, a b_2\}$ and $\{\varepsilon, a, a b_3, a b_4\}$, respectively, while the sets of traces of the two resolutions of s_2 depicted in the figure are $\{\varepsilon, a, a b_1, a b_3\}$ and $\{\varepsilon, a, a b_2, a b_4\}$, respectively. As a consequence, neither of the two considered resolutions of s_1 (resp. s_2) can have the same trace distribution as one of the two considered resolutions of s_2 (resp. s_1).

Both probabilistic trace equivalences are totally compatible with classical trace equivalence \sim_{Tr} [2], i.e., two fully nondeterministic NPLTS models are related by \sim_{Tr} iff $\sim_{\text{PTr,dis}}$ and $\sim_{\text{PTr,new}}$ relate them.

Theorem 2. Let (S, A, \longrightarrow) be a fully nondeterministic NPLTS and $s_1, s_2 \in S$. Then:

$$s_1 \sim_{\text{Tr}} s_2 \iff s_1 \sim_{\text{PTr,dis}} s_2 \iff s_1 \sim_{\text{PTr,new}} s_2 \quad \blacksquare$$

4 Testing Equivalences for NPLTS Models

Testing equivalence for NPLTS models [17,10,14,6] considers the probability of performing computations along which the same tests are passed, where tests specify which actions of a process are permitted at each state and are formalized as NPLTS models equipped with a success state. For the sake of simplicity, we restrict ourselves to tests whose underlying graph structure is acyclic – i.e., only finite-length computations are considered – and finitely branching – i.e., only a choice among finitely many alternative actions is made available at each state.

Definition 7. *A nondeterministic and probabilistic test is an acyclic and finitely-branching NPLTS $\mathcal{T} = (O, A, \longrightarrow)$ where O contains a distinguished success state denoted by ω that has no outgoing transitions. We say that a computation of \mathcal{T} is successful iff its last state is ω . ■*

Definition 8. *Let $\mathcal{L} = (S, A, \longrightarrow)$ be a NPLTS and $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ be a nondeterministic and probabilistic test. The interaction system of \mathcal{L} and \mathcal{T} is the acyclic and finitely-branching NPLTS $\mathcal{I}(\mathcal{L}, \mathcal{T}) = \mathcal{L} \parallel \mathcal{T}$ where:*

- Every element $(s, o) \in S \times O$ is called a configuration and is said to be successful iff $o = \omega$.
- A computation of $\mathcal{I}(\mathcal{L}, \mathcal{T})$ is said to be successful iff its last configuration is successful. Given $s \in S$, $o \in O$, and $\mathcal{Z} \in \text{Res}(s, o)$, we denote by $\text{SC}(z_{s,o})$ the set of successful computations from the state of \mathcal{Z} corresponding to (s, o) . ■

Due to the possible presence of equally labeled transitions departing from the same state, there is not necessarily a single probability value with which a NPLTS passes a test. Thus, given two states s_1 and s_2 of the NPLTS and the initial state o of the test, we need to compute the probability of performing a successful computation from (s_1, o) and (s_2, o) in every resolution of the interaction system. Then, one option is comparing, for the two configurations, the suprema (\bigsqcup) and the infima (\bigsqcap) of these values over all resolutions of the interaction system.

Given a state s of a NPLTS \mathcal{L} and the initial state o of a nondeterministic and probabilistic test \mathcal{T} , we denote by $\text{Res}_{\max}(s, o)$ the set of resolutions in $\text{Res}(s, o)$ that are maximal, i.e., that cannot be further extended in accordance with the graph structure of $\mathcal{I}(\mathcal{L}, \mathcal{T})$ and the constraints of Def. 4. In the following, we will consider only maximal resolutions because the non-maximal ones would lead to obtain always 0 as infimum being them unsuccessful.

Definition 9. *Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are probabilistic testing equivalent according to [17,10,6], written $s_1 \sim_{\text{PTe}, \bigsqcup, \bigsqcap} s_2$, iff for all nondeterministic and probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ with initial state $o \in O$:*

$$\begin{aligned} \bigsqcup_{\mathcal{Z}_1 \in \text{Res}_{\max}(s_1, o)} \text{prob}(\text{SC}(z_{s_1, o})) &= \bigsqcup_{\mathcal{Z}_2 \in \text{Res}_{\max}(s_2, o)} \text{prob}(\text{SC}(z_{s_2, o})) \\ \bigsqcap_{\mathcal{Z}_1 \in \text{Res}_{\max}(s_1, o)} \text{prob}(\text{SC}(z_{s_1, o})) &= \bigsqcap_{\mathcal{Z}_2 \in \text{Res}_{\max}(s_2, o)} \text{prob}(\text{SC}(z_{s_2, o})) \end{aligned} \quad \blacksquare$$

Following the structure of classical testing equivalence \sim_{Te} for fully nondeterministic processes [5], the constraint on suprema is the may-part of $\sim_{PTe, \sqcup}$ while the constraint on infima is the must-part of $\sim_{PTe, \sqcup}$. The probabilistic testing equivalence of [14] is defined in a similar way, but resolves nondeterminism through randomized schedulers and makes use of countably many success actions. The relation $\sim_{PTe, \sqcup}$ possesses several properties investigated in [17, 10, 6, 14], but suffers from two anomalies when contrasting it with \sim_{Te} .

The first anomaly of $\sim_{PTe, \sqcup}$ is that it is not always included in $\sim_{PTr, dis}$ and $\sim_{PTr, new}$. The inclusion depends on the class of schedulers that are considered for deriving resolutions of interaction systems. If randomized schedulers are admitted, then inclusion holds as shown in [14]. However, this is no longer the case when only deterministic schedulers are taken into account like in [17, 10, 6].

For instance, if we take the two NPLTS models on the left-hand side of Fig. 2(i), when $b \neq c$ it turns out that $s_1 \sim_{PTe, \sqcup} s_2$ while $s_1 \not\sim_{PTr, dis} s_2$ and $s_1 \not\sim_{PTr, new} s_2$. States s_1 and s_2 are not related by the two probabilistic trace equivalences because the maximal resolution of s_1 starting with the central a -transition is not matched by any of the two maximal resolutions of s_2 (note that we would have $s_1 \sim_{PTr, dis} s_2$ if randomized schedulers were admitted). It holds that $s_1 \sim_{PTe, \sqcup} s_2$ because, for any test, the same maximal resolution of s_1 cannot give rise to a success probability not comprised between the success probabilities of the other two maximal resolutions of s_1 , which basically coincide with the two maximal resolutions of s_2 .

The inclusion problem can be overcome by considering $\sim_{PTr, new}$ instead of $\sim_{PTr, dis}$ and by replacing $\sim_{PTe, \sqcup}$ with the finer probabilistic testing equivalence below. This equivalence does not only focus on the highest and the lowest probability of passing a test but requires matching all maximal resolutions of the interaction system according to their success probabilities.

Definition 10. *Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are probabilistic testing equivalent, written $s_1 \sim_{PTe, new} s_2$, iff for all nondeterministic and probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ with initial state $o \in O$:*

- For each $Z_1 \in Res_{\max}(s_1, o)$ there exists $Z_2 \in Res_{\max}(s_2, o)$ such that:
 $prob(SC(z_{s_1, o})) = prob(SC(z_{s_2, o}))$
- For each $Z_2 \in Res_{\max}(s_2, o)$ there exists $Z_1 \in Res_{\max}(s_1, o)$ such that:
 $prob(SC(z_{s_2, o})) = prob(SC(z_{s_1, o}))$ ■

Theorem 3. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{PTe, new} s_2 \implies s_1 \sim_{PTe, \sqcup} s_2 \quad \blacksquare$$

The inclusion of $\sim_{PTe, new}$ in $\sim_{PTe, \sqcup}$ is strict. Indeed, if we consider again the two NPLTS models on the left-hand side of Fig. 2(i) together with the test next to them, it turns out that $s_1 \sim_{PTe, \sqcup} s_2$ while $s_1 \not\sim_{PTe, new} s_2$. The considered test distinguishes s_1 from s_2 with respect to $\sim_{PTe, new}$ because – looking at the interaction system on the right-hand side of Fig. 2(i) – the maximal resolution of (s_1, o) starting with the central a -transition gives rise to a success probability

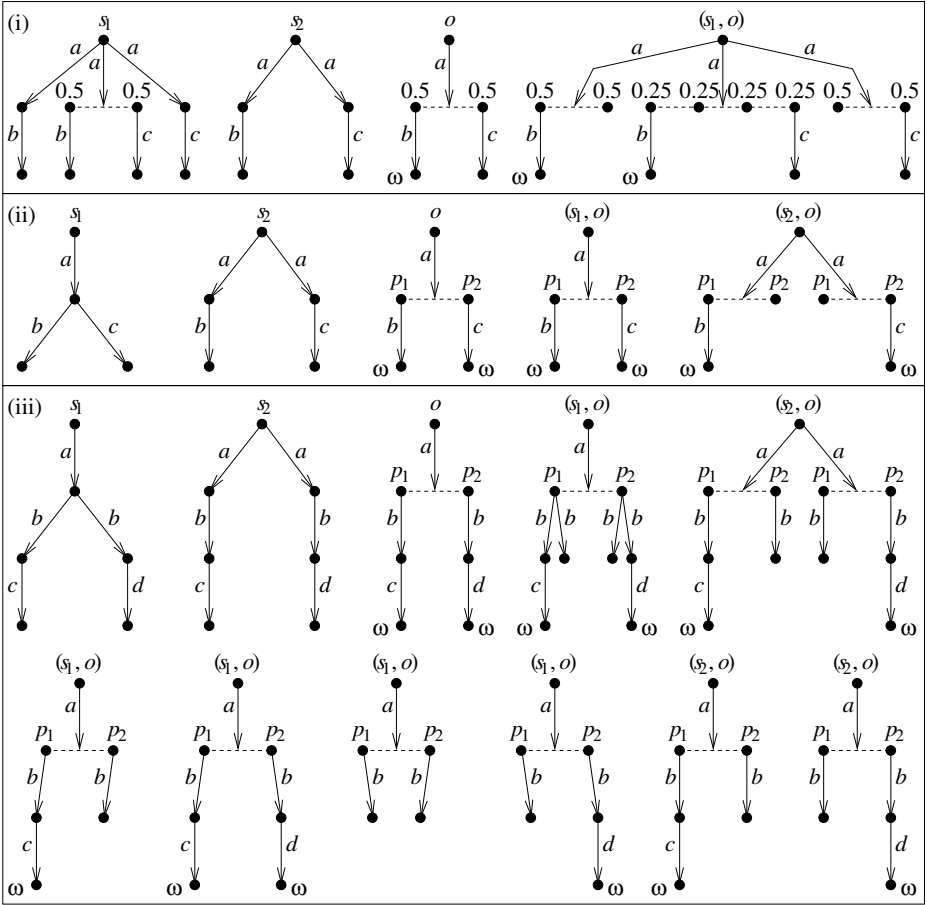


Fig. 2. Counterexamples for probabilistic testing and trace equivalences

equal to 0.25 that is not matched by any of the two maximal resolutions of (s_2, o) . These resolutions – not shown in the figure – basically coincide with the maximal resolutions of (s_1, o) starting with the two outermost a -transitions, hence their success probabilities are respectively 0.5 and 0.

We now show that $\sim_{\text{PTe,new}}$ does not suffer from the first anomaly because it is included in $\sim_{\text{PTr,new}}$. As expected, the inclusion is strict. For instance, if we consider the two NPLTS models on the left-hand side of Fig. 2(ii) together with the test next to them, when $b \neq c$ it turns out that $(s_1 \sim_{\text{PTr,dis}} s_2)$ and $s_1 \sim_{\text{PTr,new}} s_2$ while $s_1 \not\sim_{\text{PTe,new}} s_2$. In fact, the considered test distinguishes s_1 from s_2 with respect to $\sim_{\text{PTe,new}}$ because – looking at the two interaction systems on the right-hand side of Fig. 2(ii) – the only maximal resolution of (s_1, o) gives rise to a success probability equal to 1 that is not matched by any of the two maximal resolutions of (s_2, o) , whose success probabilities are p_1 and p_2 .

Theorem 4. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{PTe,new}} s_2 \implies s_1 \sim_{\text{PTr,new}} s_2 \quad \blacksquare$$

Unfortunately, $\sim_{\text{PTe,new}}$ still does not avoid the second anomaly of $\sim_{\text{PTe},\square\cap}$ (which affects [14] too) because it does not preserve \sim_{Te} . In fact, there exist two fully nondeterministic processes that are testing equivalent according to [5], but are differentiated by $\sim_{\text{PTe,new}}$ when probabilistic choices are present within tests. The reason is that such probabilistic choices make it possible to take copies of intermediate states of the processes under test, and thus to enhance the discriminating power of observers [1].

As an example, if we consider the two NPLTS models on the left-hand side of the upper part of Fig. 2(iii) together with the test next to them, when $c \neq d$ it turns out that $s_1 \sim_{\text{Te}} s_2$ while $s_1 \not\sim_{\text{PTe},\square\cap} s_2$ and $s_1 \not\sim_{\text{PTe,new}} s_2$. Let us look at the two interaction systems on the right-hand side of the upper part of Fig. 2(iii), whose maximal resolutions are shown in the lower part of the same figure. The considered test distinguishes s_1 from s_2 with respect to $\sim_{\text{PTe},\square\cap}$ because the supremum of the success probabilities of the four maximal resolutions of (s_1, o) is 1 – see the second maximal resolution of (s_1, o) – whereas the supremum of the success probabilities of the two maximal resolutions of (s_2, o) is equal to the maximum between p_1 and p_2 . The considered test distinguishes s_1 from s_2 with respect to $\sim_{\text{PTe,new}}$ because the third maximal resolution of (s_1, o) gives rise to a success probability equal to 0 that is not matched by any of the two maximal resolutions of (s_2, o) , whose success probabilities are p_1 and p_2 .

The second anomaly is essentially originated from an unrealistic estimation of success probabilities. For instance, if we consider again the four maximal resolutions of (s_1, o) in the lower part of Fig. 2(iii), we have that their success probabilities are $p_1, 1, 0,$ and p_2 , respectively. However, value 1 is clearly an overestimation of the success probability, in the same way as value 0 is an underestimation. These two values come from the fact that in each of the two corresponding maximal resolutions of (s_1, o) the deterministic scheduler selects a different b -transition in the two states of the probabilistic choice. The selection is instead consistent in the other two maximal resolutions of (s_1, o) , which thus yield realistic estimations of the success probability.

The issue of realistic probability estimation has been recently addressed in [8]. Their models are significantly different from ours, with three kinds of transition (visible, invisible, and probabilistic) and each state having only one kind of outgoing transition. Equally labeled transitions departing from the same state are tagged to be kept distinct. Moreover, in presence of cycles, models are unfolded and the tagged transitions are further decorated with the unfolding stage. Since schedulers, while testing, might encounter several instances of a given state with tagged transitions, they must resolve nondeterminism consistently in all the instances at the same stage; choices at different stages are instead independent. Therefore, in Fig. 2(iii) the two pairs of b -transitions in the interaction system with initial configuration (s_1, o) would be identically tagged with b_l and b_r and the only allowed maximal resolutions of that interaction system would be the first one (choice of b_l) and the fourth one (choice of b_r).

5 Trace-by-Trace Redefinition of Testing Equivalence

In this section, we propose a solution to the problem of estimating success probabilities – and hence to the second anomaly – which is alternative to the solution in [8]. Our solution is not invasive at all, in the sense that it does not require any transition relabeling. In order to counterbalance the strong discriminating power deriving from the presence of probabilistic choices within tests, our basic idea is changing the definition of $\sim_{\text{PTe,new}}$ by considering success probabilities in a *trace-by-trace fashion* rather than on entire resolutions.

In the following, given a state s of a NPLTS, a state o of a nondeterministic and probabilistic test, and $\alpha \in A^*$, we denote by $\text{Res}_{\max,\alpha}(s, o)$ the set of resolutions $Z \in \text{Res}_{\max}(s, o)$ such that $\text{CC}_{\max}(z_{s,o}, \alpha) \neq \emptyset$, where $\text{CC}_{\max}(z_{s,o}, \alpha)$ is the set of computations in $\text{CC}(z_{s,o}, \alpha)$ that are maximal. Moreover, for each such resolution Z we denote by $\text{SCC}(z_{s,o}, \alpha)$ the set of computations in $\text{SC}(z_{s,o})$ that are compatible with α .

Definition 11. *Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are trace-by-trace probabilistic testing equivalent, written $s_1 \sim_{\text{PTe,tbt}} s_2$, iff for all non-deterministic and probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ with initial state $o \in O$ and for all traces $\alpha \in A^*$:*

- For each $Z_1 \in \text{Res}_{\max,\alpha}(s_1, o)$ there exists $Z_2 \in \text{Res}_{\max,\alpha}(s_2, o)$ such that:

$$\text{prob}(\text{SCC}(z_{s_1,o}, \alpha)) = \text{prob}(\text{SCC}(z_{s_2,o}, \alpha))$$
- For each $Z_2 \in \text{Res}_{\max,\alpha}(s_2, o)$ there exists $Z_1 \in \text{Res}_{\max,\alpha}(s_1, o)$ such that:

$$\text{prob}(\text{SCC}(z_{s_2,o}, \alpha)) = \text{prob}(\text{SCC}(z_{s_1,o}, \alpha))$$
 ■

If we consider again the two NPLTS models on the left-hand side of the upper part of Fig. 2(iii), it turns out that $s_1 \sim_{\text{PTe,tbt}} s_2$. As an example, let us examine the interaction with the test in the same figure, which originates maximal computations from (s_1, o) or (s_2, o) that are all labeled with traces abc , or abd . It is easy to see that, for each of these traces, say α , the probability of performing a successful computation compatible with it in any of the four maximal resolutions of (s_1, o) having a maximal computation labeled with α is matched by the probability of performing a successful computation compatible with α in one of the two maximal resolutions of (s_2, o) , and vice versa. For instance, the probability p_1 (resp. p_2) of performing a successful computation compatible with abc (resp. abd) in the second maximal resolution of (s_1, o) is matched by the probability of performing a successful computation compatible with that trace in the first (resp. second) maximal resolution of (s_2, o) . As another example, the probability 0 of performing a successful computation compatible with ab in the third maximal resolution of (s_1, o) is matched by the probability of performing a successful computation compatible with that trace in any of the two maximal resolutions of (s_2, o) .

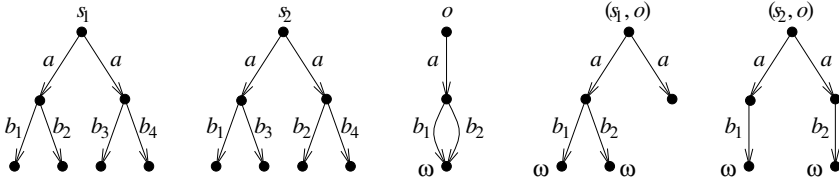
The previous example shows that $\sim_{\text{PTe,tbt}}$ is included neither in $\sim_{\text{PTe,\sqcup}}$ nor in $\sim_{\text{PTe,new}}$. On the other hand, $\sim_{\text{PTe,\sqcup}}$ is not included in $\sim_{\text{PTe,tbt}}$ as witnessed by the two NPLTS models in Fig. 2(i), where the considered test distinguishes s_1 from s_2 with respect to $\sim_{\text{PTe,tbt}}$. In fact, the probability 0.25 of performing a

successful computation compatible with $a b$ in the maximal resolution of (s_1, o) beginning with the central a -transition is not matched by the probability 0.5 of performing a successful computation compatible with $a b$ in the only maximal resolution of (s_2, o) that has a maximal computation labeled with $a b$. In contrast, $\sim_{\text{PTe,new}}$ is (strictly) included in $\sim_{\text{PTe,tbt}}$.

Theorem 5. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{PTe,new}} s_2 \implies s_1 \sim_{\text{PTe,tbt}} s_2 \quad \blacksquare$$

Apart from the use of $\text{prob}(\text{SCC}(z_{s,o}, \alpha))$ values instead of $\text{prob}(\text{SC}(z_{s,o}))$ values, another major difference between $\sim_{\text{PTe,tbt}}$ and $\sim_{\text{PTe,new}}$ is the consideration of resolutions in $\text{Res}_{\max, \alpha}(s, o)$ rather than in $\text{Res}_{\max}(s, o)$. The reason is that it is not appropriate to match the (zero) success probability of unsuccessful maximal computations labeled with α with the (zero) success probability of computations labeled with α that are not maximal, as it may happen when considering $\text{Res}_{\max}(s, o)$. For example, let us take the two NPLTS models on the left-hand side of the following figure:



where $b_i \neq b_j$ for $i \neq j$. If we employed maximal resolutions not necessarily having maximal computations labeled with a , then the test in the figure would not be able to distinguish s_1 from s_2 with respect to $\sim_{\text{PTe,tbt}}$. In fact, the success probability of the maximal resolution of (s_1, o) formed by the rightmost a -transition departing from (s_1, o) – which is 0 – would be inappropriately matched by the success probability of the a -prefix of the only maximal computation of both maximal resolutions of (s_2, o) .

We now show that $\sim_{\text{PTe,tbt}}$ does *not* suffer from the two anomalies discussed in Sect. 4. We start with the inclusion in $\sim_{\text{PTr,new}}$, which is easily met. As expected, the inclusion is strict. For instance, if we consider again the two NPLTS models on the left-hand side of Fig. 2(ii) together with the test next to them, it turns out that $(s_1 \sim_{\text{PTr,dis}} s_2)$ and $s_1 \sim_{\text{PTr,new}} s_2$ while $s_1 \not\sim_{\text{PTe,tbt}} s_2$. In fact, the considered test distinguishes s_1 from s_2 with respect to $\sim_{\text{PTe,tbt}}$ because – looking at the two interaction systems on the right-hand side of Fig. 2(ii) – each of the two maximal resolutions of (s_2, o) has a maximal computation labeled with a while the only maximal resolution of (s_1, o) has not.

Theorem 6. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{PTe,tbt}} s_2 \implies s_1 \sim_{\text{PTr,new}} s_2 \quad \blacksquare$$

With regard to the second anomaly, we show that $\sim_{\text{PTe,tbt}}$ is totally compatible with \sim_{Te} , in the sense that two fully nondeterministic NPLTS models are related by \sim_{Te} iff they are related by $\sim_{\text{PTe,tbt}}$ regardless of the class of tests. Due to the anomaly, only partial compatibility results could be provided in [17] for $\sim_{\text{PTe}, \sqcup, \sqcap}$,

because only tests without probabilistic choices (i.e., only fully nondeterministic tests) could be considered.

Theorem 7. *Let (S, A, \longrightarrow) be a fully nondeterministic NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{Te}} s_2 \iff s_1 \sim_{\text{PTe,tbt}} s_2 \quad \blacksquare$$

We conclude by showing that $\sim_{\text{PTe,tbt}}$ is a congruence with respect to parallel composition of NPLTS models (see Def. 3).

Theorem 8. *Let $\mathcal{L}_i = (S_i, A, \longrightarrow_i)$ be a NPLTS and $s_i \in S_i$ for $i = 0, 1, 2$ and consider $\mathcal{L}_1 \parallel \mathcal{L}_0$ and $\mathcal{L}_2 \parallel \mathcal{L}_0$. Then:*

$$s_1 \sim_{\text{PTe,tbt}} s_2 \implies (s_1, s_0) \sim_{\text{PTe,tbt}} (s_2, s_0) \quad \blacksquare$$

6 Placing Trace-by-Trace Testing in the Spectrum

In this section, we show that $\sim_{\text{PTe,tbt}}$ is strictly comprised between $\sim_{\text{PTR,new}}$ (see Thm. 6) and a novel probabilistic failure equivalence $\sim_{\text{PF,new}}$ that, in turn, is strictly comprised between $\sim_{\text{PTe,tbt}}$ and $\sim_{\text{PTe,new}}$.

In the following, we denote by 2_{fin}^A the set of finite subsets of A and we call failure pair any element β of $A^* \times 2_{\text{fin}}^A$, which is formed by a trace α and a finite action set F . Given a state s of a NPLTS \mathcal{L} , $\mathcal{Z} \in \text{Res}(s)$, and $c \in \mathcal{C}_{\text{fin}}(z_s)$, we say that c is compatible with β iff $c \in \mathcal{CC}(z_s, \alpha)$ and the last state reached by c has no outgoing transitions in \mathcal{L} labeled with an action in F . We denote by $\text{FCC}(z_s, \beta)$ the set of computations in $\mathcal{C}_{\text{fin}}(z_s)$ that are compatible with β .

Definition 12. *Let (S, A, \longrightarrow) be a NPLTS. We say that $s_1, s_2 \in S$ are probabilistic failure equivalent, written $s_1 \sim_{\text{PF,new}} s_2$, iff for all $\beta \in A^* \times 2_{\text{fin}}^A$:*

- For each $\mathcal{Z}_1 \in \text{Res}(s_1)$ there exists $\mathcal{Z}_2 \in \text{Res}(s_2)$ such that:

$$\text{prob}(\text{FCC}(z_{s_1}, \beta)) = \text{prob}(\text{FCC}(z_{s_2}, \beta))$$
- For each $\mathcal{Z}_2 \in \text{Res}(s_2)$ there exists $\mathcal{Z}_1 \in \text{Res}(s_1)$ such that:

$$\text{prob}(\text{FCC}(z_{s_2}, \beta)) = \text{prob}(\text{FCC}(z_{s_1}, \beta))$$

Theorem 9. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{PF,new}} s_2 \implies s_1 \sim_{\text{PTe,tbt}} s_2 \quad \blacksquare$$

The inclusion of $\sim_{\text{PF,new}}$ in $\sim_{\text{PTe,tbt}}$ is strict. For instance, if we consider again the two NPLTS models on the left-hand side of Fig. 1, when $b_i \neq b_j$ for $i \neq j$ it turns out that $s_1 \sim_{\text{PTe,tbt}} s_2$ while $s_1 \not\sim_{\text{PF,new}} s_2$. In fact, given the failure pair $\beta = (a, \{b_1, b_2\})$, the second maximal resolution of s_1 has probability 1 of performing a computation compatible with β , whilst each of the two maximal resolutions of s_2 has probability 0.5.

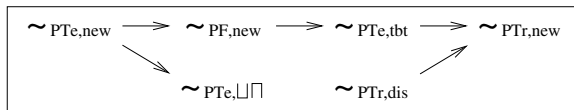


Fig. 3. Spectrum of the considered probabilistic equivalences for NPLTS models

Theorem 10. *Let (S, A, \longrightarrow) be a NPLTS and $s_1, s_2 \in S$. Then:*

$$s_1 \sim_{\text{PTe,new}} s_2 \implies s_1 \sim_{\text{PF,new}} s_2 \quad \blacksquare$$

The inclusion of $\sim_{\text{PTe,new}}$ in $\sim_{\text{PF,new}}$ is strict as $\sim_{\text{PTe,new}}$ suffers from the second anomaly. Indeed, if we consider again the two NPLTS models on the left-hand side of Fig. 2(iii), it holds that $s_1 \sim_{\text{PF,new}} s_2$ while $s_1 \not\sim_{\text{PTe,new}} s_2$.

We also note that $\sim_{\text{PTe},\sqcup\cap}$ is incomparable not only with $\sim_{\text{PTr,dis}}$, $\sim_{\text{PTr,new}}$, and $\sim_{\text{PTe,tbt}}$, but with $\sim_{\text{PF,new}}$ too. The NPLTS models on the left-hand side of Fig. 2(i) show that $\sim_{\text{PTe},\sqcup\cap}$ is not included in $\sim_{\text{PF,new}}$ and the NPLTS models on the left-hand side of Fig. 2(iii) show that $\sim_{\text{PF,new}}$ is not included in $\sim_{\text{PTe},\sqcup\cap}$. Similarly, $\sim_{\text{PTr,dis}}$ is incomparable with $\sim_{\text{PTe,tbt}}$, $\sim_{\text{PF,new}}$, and $\sim_{\text{PTe,new}}$. The NPLTS models on the left-hand side of Fig. 1 show that $\sim_{\text{PTe,tbt}}$ is not included in $\sim_{\text{PTr,dis}}$ and the NPLTS models on the left-hand side of Fig. 2(ii) show that $\sim_{\text{PTr,dis}}$ is not included in $\sim_{\text{PTe,tbt}}$. Moreover, the NPLTS models on the left-hand side of Fig. 2(ii) show that $\sim_{\text{PTr,dis}}$ is included in neither $\sim_{\text{PF,new}}$ nor $\sim_{\text{PTe,new}}$. On the other hand, neither of $\sim_{\text{PF,new}}$ and $\sim_{\text{PTe,new}}$ is included in $\sim_{\text{PTr,dis}}$ as can be seen by considering two NPLTS models both having four states $s_{i,a}$, $s_{i,b}$, $s_{i,c}$, and $s_{i,-}$ for $i = 1, 2$ such that: $s_{i,a} \xrightarrow{a} \mathcal{D}_{i,j}$ for $j = 1, 2, 3$ with $\mathcal{D}_{1,1}(s_{1,b}) = 0.6 = 1 - \mathcal{D}_{1,1}(s_{1,-})$, $\mathcal{D}_{1,2}(s_{1,b}) = 0.4 = 1 - \mathcal{D}_{1,2}(s_{1,c})$, $\mathcal{D}_{1,3}(s_{1,-}) = 0.6 = 1 - \mathcal{D}_{1,3}(s_{1,c})$, and $\mathcal{D}_{2,j}(s_{2,*}) = 1 - \mathcal{D}_{1,j}(s_{1,*})$ for $* = b, c, -$; $s_{i,b} \xrightarrow{b} \mathcal{D}_{i,b}$ with $\mathcal{D}_{i,b}(s_{i,-}) = 1$; and $s_{i,c} \xrightarrow{c} \mathcal{D}_{i,c}$ with $\mathcal{D}_{i,b}(s_{i,-}) = 1$.

The relationships among the examined equivalences are summarized in Fig. 3, where arrows mean more-discriminating-than. For the sake of completeness, we remark that $\sim_{\text{PTe},\sqcup\cap}$ has been characterized in [10,6] through probabilistic simulation equivalence for the may-part and probabilistic failure simulation equivalence for the must-part. With regard to the variant of [14] based on randomized schedulers, the may-part coincides with the coarsest congruence contained in $\sim_{\text{PTr,dis}}$ (again based on randomized schedulers) and the must-part coincides with the coarsest congruence contained in probabilistic failure distribution equivalence. The probabilistic testing equivalence of [8] has instead been characterized via probabilistic ready-trace equivalence. Its connection with $\sim_{\text{PTe,tbt}}$ is hindered by the different underlying models and is still an open problem.

7 Trace-by-Trace Testing for GPLTS and RPLTS Models

Our variant of testing equivalence naturally fits to generative and reactive probabilistic processes [15]. In this section, we show that the two testing equivalences for those two classes of processes can be redefined in a uniform trace-by-trace fashion without altering their discriminating power.

Definition 13. *A probabilistic labeled transition system, PLTS for short, is a triple (S, A, \longrightarrow) where S is an at most countable set of states, A is a countable set of transition-labeling actions, and $\longrightarrow \subseteq S \times A \times \mathbb{R}_{(0,1]} \times S$ is a transition relation satisfying one of the following two conditions:*

- $\sum\{p \in \mathbb{R}_{(0,1]} \mid \exists a \in A. \exists s' \in S. s \xrightarrow{a,p} s'\} \in \{0, 1\}$ for all $s \in S$ (generative PLTS, or GPLTS for short).
- $\sum\{p \in \mathbb{R}_{(0,1]} \mid \exists s' \in S. s \xrightarrow{a,p} s'\} \in \{0, 1\}$ for all $s \in S$ and $a \in A$ (reactive PLTS, or RPLTS for short). ■

A test consistent with a PLTS $\mathcal{L} = (S, A, \longrightarrow_{\mathcal{L}})$ is an acyclic and finitely-branching PLTS $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ equipped with a success state, which is generative (resp. reactive) if so is \mathcal{L} . Their interaction system is the acyclic and finitely-branching PLTS $\mathcal{I}(\mathcal{L}, \mathcal{T}) = (S \times O, A, \longrightarrow)$ whose transition relation $\longrightarrow \subseteq (S \times O) \times A \times \mathbb{R}_{(0,1]} \times (S \times O)$ is such that $(s, o) \xrightarrow{a,p} (s', o')$ iff $s \xrightarrow{a,p_1}_{\mathcal{L}} s'$ and $o \xrightarrow{a,p_2}_{\mathcal{T}} o'$ with p being equal to:

$$p_1 \cdot p_2 / \sum\{q_1 \cdot q_2 \mid \exists b \in A, s'' \in S, o'' \in O. s \xrightarrow{b,q_1}_{\mathcal{L}} s'' \wedge o \xrightarrow{b,q_2}_{\mathcal{T}} o''\} \text{ if GPLTS}$$

$$p_1 \cdot p_2 \text{ if RPLTS}$$

Given $s \in S$ and $o \in O$, we denote by $\mathcal{SC}(s, o)$ the set of successful computations of $\mathcal{I}(\mathcal{L}, \mathcal{T})$ with initial configuration (s, o) and by $\mathcal{SCC}(s, o, \alpha)$ the set of computations in $\mathcal{SC}(s, o)$ that are compatible with $\alpha \in A^*$. Moreover, we denote by $Tr_{\max}(s, o)$ the set of traces labeling the maximal computations from (s, o) .

Definition 14. Let (S, A, \longrightarrow) be a GPLTS. We say that $s_1, s_2 \in S$ are probabilistic testing equivalent according to [34], written $s_1 \sim_{\text{PTe,G}} s_2$, iff for all generative probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ with initial state $o \in O$:

$$\text{prob}(\mathcal{SC}(s_1, o)) = \text{prob}(\mathcal{SC}(s_2, o)) \quad \blacksquare$$

Definition 15. Let (S, A, \longrightarrow) be a RPLTS. We say that $s_1, s_2 \in S$ are probabilistic testing equivalent according to [11], written $s_1 \sim_{\text{PTe,R}} s_2$, iff for all reactive probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ with initial state $o \in O$:

$$\bigsqcup_{\alpha \in Tr_{\max}(s_1, o)} \text{prob}(\mathcal{SCC}(s_1, o, \alpha)) = \bigsqcup_{\alpha \in Tr_{\max}(s_2, o)} \text{prob}(\mathcal{SCC}(s_2, o, \alpha))$$

$$\prod_{\alpha \in Tr_{\max}(s_1, o)} \text{prob}(\mathcal{SCC}(s_1, o, \alpha)) = \prod_{\alpha \in Tr_{\max}(s_2, o)} \text{prob}(\mathcal{SCC}(s_2, o, \alpha)) \quad \blacksquare$$

Definition 16. Let $\mathcal{L} = (S, A, \longrightarrow)$ be a PLTS. We say that $s_1, s_2 \in S$ are trace-by-trace probabilistic testing equivalent, written $s_1 \sim_{\text{PTe,tbt}} s_2$, iff for all probabilistic tests $\mathcal{T} = (O, A, \longrightarrow_{\mathcal{T}})$ consistent with \mathcal{L} with initial state $o \in O$ and for all traces $\alpha \in A^*$:

$$\text{prob}(\mathcal{SCC}(s_1, o, \alpha)) = \text{prob}(\mathcal{SCC}(s_2, o, \alpha)) \quad \blacksquare$$

Theorem 11. Let (S, A, \longrightarrow) be a GPLTS and $s_1, s_2 \in S$. Then:

$$s_1 \sim_{\text{PTe,G}} s_2 \iff s_1 \sim_{\text{PTe,tbt}} s_2 \quad \blacksquare$$

Theorem 12. Let (S, A, \longrightarrow) be a RPLTS and $s_1, s_2 \in S$. Then:

$$s_1 \sim_{\text{PTe,R}} s_2 \iff s_1 \sim_{\text{PTe,tbt}} s_2 \quad \blacksquare$$

8 Conclusion

In this paper, we have proposed solutions for avoiding two anomalies of probabilistic testing equivalence for NPLTS models by (i) matching all resolutions

on the basis of their success probabilities rather than taking only maximal and minimal success probabilities and (ii) considering success probabilities in a trace-by-trace fashion rather than on entire resolutions. The trace-by-trace approach – which fits also testing equivalences for nondeterministic processes (Thm. 7), generative probabilistic processes (Thm. 11), and reactive probabilistic processes (Thm. 12) – thus annihilates the impact of the copying capability introduced by probabilistic observers. In the future, we would like to find equational and logical characterizations of $\sim_{\text{PTe,tbt}}$. Moreover, we plan to investigate the whole spectrum of probabilistic behavioral equivalences for NPLTS models.

References

1. Abramsky, S.: Observational equivalence as a testing equivalence. *Theoretical Computer Science* 53, 225–241 (1987)
2. Brookes, S., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. *Journal of the ACM* 31, 560–599 (1984)
3. Christoff, I.: Testing Equivalences and Fully Abstract Models for Probabilistic Processes. In: Baeten, J.C.M., Klop, J.W. (eds.) *CONCUR 1990*. LNCS, vol. 458, pp. 126–140. Springer, Heidelberg (1990)
4. Cleaveland, R., Dayar, Z., Smolka, S.A., Yuen, S.: Testing preorders for probabilistic processes. *Information and Computation* 154, 93–148 (1999)
5. De Nicola, R., Hennessy, M.: Testing equivalences for processes. *Theoretical Computer Science* 34, 83–133 (1984)
6. Deng, Y., van Glabbeek, R.J., Hennessy, M., Morgan, C.: Characterising testing preorders for finite probabilistic processes. *LMCS* 4(4), 1–33 (2008)
7. Derman, C.: *Finite State Markovian Decision Processes*. Academic Press (1970)
8. Georgievska, S., Andova, S.: Retaining the Probabilities in Probabilistic Testing Theory. In: Ong, L. (ed.) *FOSSACS 2010*. LNCS, vol. 6014, pp. 79–93. Springer, Heidelberg (2010)
9. Hansson, H., Jonsson, B.: A calculus for communicating systems with time and probabilities. In: *Proc. of RTSS 1990*, pp. 278–287. IEEE-CS Press (1990)
10. Jonsson, B., Yi, W.: Compositional testing preorders for probabilistic processes. In: *Proc. of LICS 1995*, pp. 431–441. IEEE-CS Press (1995)
11. Kwiatkowska, M.Z., Norman, G.: A testing equivalence for reactive probabilistic processes. In: *Proc. of EXPRESS 1998*. ENTCS, vol. 16(2), pp. 114–132 (1998)
12. Segala, R.: *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD Thesis (1995)
13. Segala, R.: A Compositional Trace-Based Semantics for Probabilistic Automata. In: Lee, I., Smolka, S.A. (eds.) *CONCUR 1995*. LNCS, vol. 962, pp. 234–248. Springer, Heidelberg (1995)
14. Segala, R.: Testing Probabilistic Automata. In: Sassone, V., Montanari, U. (eds.) *CONCUR 1996*. LNCS, vol. 1119, pp. 299–314. Springer, Heidelberg (1996)
15. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. *Information and Computation* 121, 59–80 (1995)
16. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: *Proc. of FOCS 1985*, pp. 327–338. IEEE-CS Press (1985)
17. Yi, W., Larsen, K.G.: Testing probabilistic and nondeterministic processes. In: *Proc. of PSTV 1992*, pp. 47–61. North-Holland (1992)

Is It a “Good” Encoding of Mixed Choice?*

Kirstin Peters and Uwe Nestmann

Technische Universität Berlin, Germany

Abstract. Mixed choice is a widely-used primitive in process calculi. It is interesting, as it allows to break symmetries in distributed process networks. We present an encoding of mixed choice in the context of the π -calculus and investigate to what extent it can be considered “good”. As a crucial novelty, we introduce a suitable criterion to measure whether the degree of distribution in process networks is preserved.

1 Introduction

It is well-known [Pa03, Gor10, PN10] that there is no good encoding from the full π -calculus—the synchronous π -calculus including mixed choice—into its asynchronous variant if the encoding translates the parallel operator rigidly (a criterion included in *uniformity* in [Pa03]). Palamidessi was the first to point out that mixed choice strictly raises the absolute expressive power of the synchronous π -calculus compared to its asynchronous variant. Analysing this result [PN10], we observed that it boils down to the fact that the full π -calculus can break merely syntactic symmetries, where its asynchronous variant can not. However, the condition of rigid translation of the parallel operator is rather strict. Therefore, Gorla proposed the weaker criterion of compositional translation of the source language operators (see Definition 4 at page 214). We show that this weakening of the structural condition on the encoding of the parallel operator turns the separation result into an encodability result, by presenting a good encoding of mixed choice 1. So, merely considering the (abstract) behaviour of terms, the full π -calculus and its asynchronous variant have the same expressive power.

The situation changes again if we additionally take into account the *degree of distribution*. In the area of distributed communicating systems it is natural to consider distributed algorithms, that perform at least some of their tasks concurrently. Thus, an answer to the question whether an encoding preserves the degree of distribution of the original algorithm, becomes important. In order to measure the preservation of the degree of distribution we introduce a novel but intuitive (semantic) criterion, which is strictly weaker than the (syntactic) requirement of rigid translation of the parallel operator. Using this criterion in addition to the criteria presented in [Gor10], we again obtain, as expected, a separation result by showing that there is no good encoding of mixed choice that preserves the degree of distribution of source terms.

* Supported by the DFG (German Research Foundation), grant NE-1505/2-1.

¹ Note that this encoding is neither prompt nor is the assumed equivalence \approx strict, so the similar separation results of [Gor08] and [Gor10] do not apply here.

Overview of the Paper. In Section 2, we introduce the considered variants of the π -calculus, some abbreviations to simplify the presentation of encodings, and the criteria of [Gor10] to measure the correctness of encodings. In Section 3, we revisit the encoding given in [Nes00]; based on it, we present a novel encoding, denoted by $\llbracket \cdot \rrbracket_a^m$, of mixed choice. Section 4 discusses in how far an encoding like $\llbracket \cdot \rrbracket_a^m$ can preserve the degree of distribution. We conclude in Section 5.

2 Technical Preliminaries

2.1 The π -Calculus

Our source language is the monadic π -calculus as described for instance in [SW01]. As already demonstrated in [Pa03] the most interesting operator for a comparison of the expressive power between the full π -calculus and its asynchronous variant is mixed choice, i.e., choice between input and output capabilities. Thus we denote the full π -calculus also by π_m . Let \mathcal{N} denote a countably infinite set of names with $\tau \notin \mathcal{N}$ and $\overline{\mathcal{N}}$ the set of co-names, i.e., $\overline{\mathcal{N}} = \{\overline{n} \mid n \in \mathcal{N}\}$. We use lower case letters $a, a', a_1, \dots, x, y, \dots$ to range over names.

Definition 1 (π_m). *The set of process terms of the synchronous π -calculus (with mixed choice), denoted by \mathcal{P}_m , is given by*

$$P ::= (\nu n)P \quad | \quad P_1 \mid P_2 \quad | \quad [a = b]P \quad | \quad y^*(x).P \quad | \quad \sum_{i \in I} \pi_i.P_i$$

$$\pi ::= y(x) \quad | \quad \overline{y}\langle z \rangle \quad | \quad \tau$$

where $n, a, b, x, y, z \in \mathcal{N}$ range over names and I ranges over finite index sets.

The interpretation of process terms is as usual. We consider two subcalculi of π_m . The process terms \mathcal{P}_s of π_s —the *π -calculus with separate choice*—are obtained by restricting the choice primitive such that in each choice either no input guarded or no output guarded alternatives appear. The process terms \mathcal{P}_a of the *asynchronous π -calculus* π_a [Bou92, HT91] are obtained by limiting each sum to at most one branch and requiring that outputs can only guard the empty sum.

Note that we augment all three variants of the π -Calculus with matching, because we need it at least in π_a to encode mixed choice. Of course, the presence of match influences the expressive power of π_a . However, we do not know, whether the use of match in the encoding of mixed choice can be circumvented, although there are reasons indicating that this is indeed not possible. We leave the consideration of this problem to further research.

We use capital letters $P, P', P_1, \dots, Q, R, \dots$ to range over processes. Let $\text{fn}(P)$, $\text{bn}(P)$, and $\text{n}(P)$ denotes the sets of free names, bound names and all names occurring in P , respectively. Their definitions are completely standard. Given an input prefix $y(x)$ or an output prefix $\overline{y}\langle x \rangle$ we call y the subject and x

$$\begin{array}{c}
\text{TAU}_m \quad \dots + \tau.P + \dots \mapsto P \\
\text{COM}_m \quad (\dots + y(x).P + \dots) \mid (\dots + \bar{y}(z).Q + \dots) \mapsto \{z/x\}P \mid Q \\
\text{PAR} \quad \frac{P \mapsto P'}{P \mid Q \mapsto P' \mid Q} \quad \text{RES} \quad \frac{P \mapsto P'}{(\nu x)P \mapsto (\nu x)P'} \\
\text{CONG} \quad \frac{P \equiv P' \quad P' \mapsto Q' \quad Q' \equiv Q}{P \mapsto Q}
\end{array}$$

Fig. 1. Reduction Semantics of π_m

the object of the action. We denote the subject of an action also as link or channel name, while we denote the object as value or parameter. Note that in case the object does not matter we omit it, i.e., we abbreviate an input guarded term $y(x).P$ or an output guarded term $\bar{y}(z).P$ such that $x \notin \text{fn}(P)$ by $y.P$ or $\bar{y}.P$, respectively. We denote the empty sum with $\mathbf{0}$ and often omit it in continuations. As usual, we sometimes write a sum $\sum_{i \in \{i_1, \dots, i_n\}} \pi_i.P_i$ as $\pi_{i_1}.P_{i_1} + \dots + \pi_{i_n}.P_{i_n}$.

We use $\sigma, \sigma', \sigma_1, \dots$ to range over substitutions. A substitution is a mapping $\{x_1/y_1, \dots, x_n/y_n\}$ from names to names. The application of a substitution on a term $\{x_1/y_1, \dots, x_n/y_n\}(P)$ is defined as the result of simultaneously replacing all free occurrences of y_i by x_i for $i \in \{1, \dots, n\}$, possibly applying alpha-conversion to avoid capture or name clashes. For all names in $\mathcal{N} \setminus \{y_1, \dots, y_n\}$, the substitution behaves as the identity mapping. Let id denote identity, i.e. id is the empty substitution. We naturally extend substitutions to co-names, i.e. $\forall \bar{n} \in \bar{\mathcal{N}}. \sigma(\bar{n}) = \overline{\sigma(n)}$ for all substitutions σ .

The *reduction semantics* of π_m is given by the transition rules in Figure 1, where *structural congruence*, denoted by \equiv , is defined as usual². The reduction semantics of π_s is the same, and it is even simpler for π_a because of the restrictions on its syntax. As usual, we use \equiv_α if we refer to alpha-conversion only.

Let $P \mapsto (P \not\mapsto)$ denote existence (non-existence) of a step from P , i.e. there is (no) $P' \in \mathcal{P}$ such that $P \mapsto P'$. Moreover, let \mapsto^* be the reflexive and transitive closure of \mapsto and let \mapsto^∞ define an infinite sequence of steps.

In Section 2.3, we present several criteria to measure the quality of an encoding. The first of these criteria relies on the notion of a context. A *context* $\mathcal{C}([\cdot]_1, \dots, [\cdot]_n)$ is a π -term, i.e., a π_a -term in case of Definition 4, with n so-called *holes*. Plugging the π_a -terms P_1, \dots, P_n into the respective holes $[\cdot]_1, \dots, [\cdot]_n$ of the context, yields a term denoted $\mathcal{C}(P_1, \dots, P_n)$. A context $\mathcal{C}([\cdot]_1, \dots, [\cdot]_n)$ can be seen as a function of type $\mathcal{P}_a \times \dots \times \mathcal{P}_a \rightarrow \mathcal{P}_a$ of *arity* n , applicable to *parameters* P_1, \dots, P_n . Note that a context may bind some free names of P_1, \dots, P_n .

² Note that, since we do not use “!” but replicated input, the common rule $!P \equiv P \mid !P$ becomes $y^*(x).P \equiv y(x).P \mid y^*(x).P$.

2.2 Abbreviations

To shorten the presentation and ease the readability of the rather lengthy encoding function in Section 3, we use some abbreviations on π_a -terms. First note that we defined only monadic versions of the calculi π_m , π_s , and π_a , where across any link exactly one value is transmitted. However, within the presented encoding function in Section 3, we treat the target language π_a as if it allows for polyadic communication. More precisely, we allow asynchronous links to carry any number of values from zero to five, of course under the requirement that within each π_a -term no link name is used twice with different multiplicities. Let \tilde{x} denote a sequence of names. Note that these polyadic actions can be simply translated into monadic actions by a standard encoding as given in [SW01]. Thus, we silently use the polyadic version of π_a in the following. Second, as already done in [Nes00], we use the following abbreviations to define boolean values and a conditional construct.

Definition 2 (Tests on Booleans). Let $\mathbb{B} \triangleq \{\top, \perp\}$ be the set of boolean values, where \top denotes true and \perp denotes false.

Let $l, t, f \in \mathcal{N}$ and $P, Q \in \mathcal{P}_a$. Then a boolean instantiation of l , i.e., the allocation of a boolean value to a link l , and a test-statement on a boolean instantiation are defined by

$$\begin{aligned} \bar{l}\langle\top\rangle &\triangleq l(t, f) \cdot \bar{t} \\ \bar{l}\langle\perp\rangle &\triangleq l(t, f) \cdot \bar{f} \\ \text{test } l \text{ then } P \text{ else } Q &\triangleq (\nu t, f) (\bar{l}\langle t, f \rangle \mid t.P \mid f.Q) \end{aligned}$$

for some $t, f \notin \text{fn}(P) \cup \text{fn}(Q)$.

Finally, we define forwarders, i.e., a simple process to forward each received message along some specified set of links.

Definition 3 (Forwarder). Let I be a finite index set and for all $i \in I$ let y and y_i be channel names with multiplicity $n \in \mathbb{N}$, then a forwarder is given by:

$$y \rightarrow \{y_i \mid i \in I\} \triangleq y^*(x_1, \dots, x_n) \cdot \left(\prod_{i \in I} \bar{y}_i \langle x_1, \dots, x_n \rangle \right)$$

In case of a singleton set we omit the brackets, i.e., $y \rightarrow y' \triangleq y \rightarrow \{y'\}$.

2.3 Quality Criteria for Encodings

Within this paper we consider two encodings, (1) an encoding from π_s into π_a presented in [Nes00], denoted by $\llbracket \cdot \rrbracket_a^s$, and (2) a new encoding from π_m into π_a , denoted by $\llbracket \cdot \rrbracket_a^m$. To measure the quality of such an encoding, Gorla [Gor10] suggested five criteria well suited for language comparison. Accordingly, we consider an encoding to be “good”, if it satisfies Gorla’s five criteria.

As in [Gor10], an encoding is a mapping from a source into a target language; in our case, π_m and π_s are source languages and π_a is the target language. To distinguish terms on these languages or definitions for the respective encodings, we use m , s , and a as super- and subscripts. Thereby, the superscript usually refers to the source and the subscript to the target language. Moreover, we use S, S', S_1, \dots to range over terms of the source languages and T, T', T_1, \dots to range over terms of the target language.

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *name invariance* and (2) *compositionality*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection* and (5) *success sensitiveness*. We do not repeat them formally, here, except for compositionality. Note that for name invariance and operational correspondence a behavioural equivalence \approx on the target language is assumed. Its purpose is to describe the “abstract” behaviour of a target process, where abstract basically means “with respect to the behaviour of the source term”.

(1) The first structural criterion *name invariance* states that the encoding should not depend on specific names used in the source term. This is important, as sometimes it is necessary to translate a source term name into a sequence of names or reserve some names for the encoding function. To ensure that there are no conflicts between these reserved names and the source term names, the encoding is equipped with a *renaming policy*, more precisely, a substitution ϕ from names into sequences of names. Note that in the case of $\llbracket \cdot \rrbracket_a^s$ and $\llbracket \cdot \rrbracket_a^m$ the renaming policies are injective substitutions from names into single names. Based on such a renaming policy, an encoding is independent of specific names if it preserves all substitutions σ on source terms by a substitution σ' on target terms such that σ' respects the changes made by the renaming policy.

(2) The second structural criterion *compositionality* aims at relaxing rigidity. We call the translation of an operator **op** *rigid*, if $\llbracket \mathbf{op} \rrbracket$ is mapped onto \mathbf{op}^ϕ , i.e., essentially the same operator, but possibly adapted to comply with the renaming policy ϕ . For example, $\llbracket (\nu x) P \rrbracket = (\nu \phi(x)) \llbracket P \rrbracket$ translates the restriction on a single name into the restriction on the sequence of associated names. Now, we call the translation of an operator **op** “merely” *compositional* if $\llbracket \mathbf{op} \rrbracket$ is defined quite more liberally as a context $\mathcal{C}_{\mathbf{op}}^\phi$ (of the same arity as **op**) that mediates between the translations of **op**’s parameters, while those parameters are still translated independently of **op**. In order to realize this mediation, the context $\mathcal{C}_{\mathbf{op}}$ must at least be allowed to know some of the parameters’ free names.

Definition 4 (Compositionality). *The encoding $\llbracket \cdot \rrbracket$ is compositional if, for every k -ary operator **op** of the source language and for every subset of names N , there exists a k -ary context $\mathcal{C}_{\mathbf{op}}^N([\cdot]_1, \dots, [\cdot]_k)$ such that, for all S_1, \dots, S_k with $\text{fn}(S_1) \cup \dots \cup \text{fn}(S_k) = N$, it holds that*

$$\llbracket \mathbf{op}(S_1, \dots, S_k) \rrbracket = \mathcal{C}_{\mathbf{op}}^N(\llbracket S_1 \rrbracket, \dots, \llbracket S_k \rrbracket).$$

Note that Gorla requires the parallel composition operator “ \parallel ” to be binary and unique in both the source and the target language. Thus, compositionality prevents us from introducing a global coordinator or to use global knowledge, i.e.,

$$\begin{aligned}
 \left[\sum_{i \in I} \pi_i . P_i \right]_{\mathbf{a}}^{\mathbf{s}} &\triangleq (\nu l) \left(\bar{l} \langle \top \rangle \mid \prod_{i \in I} \llbracket \pi_i . P_i \rrbracket_{\mathbf{a}}^{\mathbf{s}} \right) \\
 \llbracket \tau . P \rrbracket_{\mathbf{a}}^{\mathbf{s}} &\triangleq \text{test } l \text{ then } (\bar{l} \langle \perp \rangle \mid \llbracket P \rrbracket_{\mathbf{a}}^{\mathbf{s}}) \text{ else } \bar{l} \langle \perp \rangle \\
 \llbracket \bar{y} \langle z \rangle . P \rrbracket_{\mathbf{a}}^{\mathbf{s}} &\triangleq (\nu s) (\bar{y} \langle l, s, z \rangle \mid s . \llbracket P \rrbracket_{\mathbf{a}}^{\mathbf{s}}) \\
 \llbracket y(x) . P \rrbracket_{\mathbf{a}}^{\mathbf{s}} &\triangleq (\nu r) (\bar{r} \mid r^* . y(l', s, x) . \\
 &\quad \text{test } l \text{ then test } l' \text{ then } \bar{l} \langle \perp \rangle \mid \bar{l}' \langle \perp \rangle \mid \bar{s} \mid \llbracket P \rrbracket_{\mathbf{a}}^{\mathbf{s}} \\
 &\quad \quad \text{else } \bar{l} \langle \top \rangle \mid \bar{l}' \langle \perp \rangle \mid \bar{r} \\
 &\quad \quad \text{else } \bar{l} \langle \perp \rangle \mid \bar{y} \langle l', s, x \rangle) \\
 \llbracket y^*(x) . P \rrbracket_{\mathbf{a}}^{\mathbf{s}} &\triangleq y^*(l, s, x) . \text{test } l \text{ then } \bar{l} \langle \perp \rangle \mid \bar{s} \mid \llbracket P \rrbracket_{\mathbf{a}}^{\mathbf{s}} \text{ else } \bar{l} \langle \perp \rangle
 \end{aligned}$$

Fig. 2. Encode $\pi_{\mathbf{s}}$ into $\pi_{\mathbf{a}}$ [Nes00](#)

knowledge about surrounding source terms or the structure of the parameters. We discuss this point in Section [4](#).

(3) The first semantic criterion and usually the most elaborate one to prove is *operational correspondence*, which consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be emulated by its translation, i.e., the translation does not shrink the set of computations of the source term. Note that encodings often translate single source term steps into sequences of target term steps. We call such sequences *emulations* of the corresponding source term step. *Soundness* requires that every computation of a target term corresponds to some computation of the corresponding source term, i.e., the translation does not introduce new computations.

(4) With *divergence reflection* we require, that any infinite execution of a target term corresponds to an infinite execution of the respective source.

(5) The last criterion *success sensitiveness* links the behaviour of source terms to the behaviour of their encodings. With Gorla [Gor10](#), we assume a *success* operator \checkmark as part of the syntax of both the source and the target language, i.e., of $\pi_{\mathbf{m}}$, $\pi_{\mathbf{s}}$, and $\pi_{\mathbf{a}}$. Since \checkmark can not be further reduced, the operational semantics is left unchanged in all three cases. Moreover, note that $\mathbf{n}(\checkmark) = \mathbf{fn}(\checkmark) = \mathbf{bn}(\checkmark) = \emptyset$, so also interplay of \checkmark with the rules of structural congruence is smooth and does not require explicit treatment. The test for reachability of success is standard. Finally, an encoding preserves the abstract behaviour of the source term if it and its encoding answer the tests for success in exactly the same way.

For a more exhaustive and formal description we refer to [Gor10](#).

3 Encoding Mixed Choice

Nestmann [Nes00](#) presents an encoding from $\pi_{\mathbf{s}}$ into $\pi_{\mathbf{a}}$, in the following denoted by $\llbracket \cdot \rrbracket_{\mathbf{a}}^{\mathbf{s}}$, that encodes the parallel operator rigidly: $\llbracket P \mid Q \rrbracket_{\mathbf{a}}^{\mathbf{s}} \triangleq \llbracket P \rrbracket_{\mathbf{a}}^{\mathbf{s}} \mid \llbracket Q \rrbracket_{\mathbf{a}}^{\mathbf{s}}$. In the following, for simplicity, we omit the indication of the renaming policy.

The full details are given in [PN12]. The encodings of sum, guarded terms, and replicated input are given in Figure 2 where, in the last four clauses, we assume that the name l that is used on the right-hand sides is an implicit parameter of the encoding function, as supplied in the first of the above clauses. The remaining operators are translated rigidly (compare to their translations in $[\cdot]_a^m$). The main idea of this encoding is to introduce a so-called sum lock l carrying a boolean value for each sum. In order to emulate a step on a source term summand the respective sum lock is checked. In case its boolean value is \top the respective source term step is emulated, else the emulation is aborted and the terms corresponding to this emulation attempt remain as junk (compare to [Nes00] for a more detailed discussion of this encoding). [Nes00] argues for the correctness of this encoding by proving its deadlock- and divergence-freedom; he also discussed the possibilities to state full abstraction results. In [PN12], we present a proof of its correctness with respect to the criteria presented in Section 2.3.

As already proved in [Pal03] and later on rephrased in [Gor10, PN10], it is not possible to encode π_m into π_a , while translating the parallel operator rigidly. However, by weakening this requirement, the separation result no longer holds—instead, an encodability result is possible. To prove this, we give an encoding from π_m into π_a , denoted as $[\cdot]_a^m$, that is correct with respect to the criteria established by [Gor08, Gor10].

As stated in [Nes00], the encoding presented above introduces deadlock when applied in the case of mixed choice, due to the nested test-statements in the encoding of an input-guarded source term. However, [Nes00] also states that all potential deadlocks can be avoided by using a total ordering on the sum locks. Of course, compositionality forbids to simply augment the encoding with an arbitrary previously created ordering, because this would require some form of global knowledge on the source terms. So, the main idea behind the design of $[\cdot]_a^m$ is to augment $[\cdot]_a^s$ with an algorithm to dynamically compute an order on the sum locks—at runtime. Unfortunately, this algorithm fairly blows up the translation of the parallel operator and replicated input.

For sums, the translation via $[\cdot]_a^m$ follows exactly the scheme of $[\cdot]_a^s$.

$$\left[\sum_{i \in I} \pi_i.P_i \right]_a^m \triangleq (\nu l) \left(\bar{l} \langle \top \rangle \mid \prod_{i \in I} [\pi_i.P_i]_a^m \right)$$

This translation splits up the encoded summands in parallel and introduces the sum locks, which are initialised by \top . To *order* these sum locks, we first have to transport them to a surrounding parallel operator encoding: for example, in $P \mid Q$, with P and Q being sequential processes, the sums occurring in either P or Q will have their locks ordered by means of the translation $[P \mid Q]_a^m$. Therefore, in the translation, we let input- and output-guarded source terms not communicate directly, but instead require that they first register their send/receive abilities to a surrounding parallel operator encoding, by sending an output request $\bar{p}_o \langle y, l, s, z \rangle$ or an input request $\bar{p}_i \langle y, l, r \rangle$. A request carries all necessary information to resolve a nested test-statement, i.e., the translated link name, the corresponding sum lock, a sender lock or a receiver lock, and, in case of

an output request, the translation of the transmitted value. Note that a sender lock, i.e., the s in $\llbracket \cdot \rrbracket_a^m$, is used to guard the encoded continuation of the sender, while over the receiver lock, i.e., the r in $\llbracket \cdot \rrbracket_a^m$, the ordered sum locks are transmitted back to the receiver. For convenience, the mapping $\llbracket \cdot \rrbracket_a^m$ is implicitly parametrised by the names p_o and p_i ; in the clause for parallel composition, some of their occurrences will be bound, while others will be free.

$$\begin{aligned}
 \llbracket \tau.P \rrbracket_a^m &\triangleq \text{test } l \text{ then } \bar{l} \langle \perp \rangle \mid \llbracket P \rrbracket_a^m \text{ else } \bar{l} \langle \perp \rangle \\
 \llbracket \bar{y} \langle z \rangle . P \rrbracket_a^m &\triangleq (\nu s) (\bar{p}_o \langle y, l, s, z \rangle \mid s. \llbracket P \rrbracket_a^m) \\
 \llbracket y(x) . P \rrbracket_a^m &\triangleq (\nu r) (\bar{p}_i \langle y, l, r \rangle \mid r^* \langle l_1, l_2, -, s, x \rangle . \\
 &\quad \text{test } l_1 \text{ then test } l_2 \text{ then } \bar{l}_1 \langle \perp \rangle \mid \bar{l}_2 \langle \perp \rangle \mid \bar{s} \mid \llbracket P \rrbracket_a^m \\
 &\quad \text{else } \bar{l}_1 \langle \top \rangle \mid \bar{l}_2 \langle \perp \rangle \\
 &\quad \text{else } \bar{l}_1 \langle \perp \rangle)
 \end{aligned}$$

Apart from requests, the encoding of guarded terms is very similar to $\llbracket \cdot \rrbracket_a^s$. The requests push the task of finding matching source term communication partners to the surrounding parallel operator encodings. There, a strict policy controls the redirection of requests. First, it restricts the request channels p_o and p_i for both of its parameters to be able to distinguish requests from the left from those from the right side.

$$\begin{aligned}
 \llbracket P \mid Q \rrbracket_a^m &\triangleq (\nu m_o, m_i, p_o, up, p_i, up, c_o, c_i, m_o, up, m_i, up) (\\
 &\quad (\nu p_o, p_i) (\llbracket P \rrbracket_a^m \mid \text{procLeftOutReq} \mid \text{procLeftInReq}) \\
 &\quad \mid (\nu p_o, p_i) (\llbracket Q \rrbracket_a^m \mid \text{procRightOutReq} \mid \text{procRightInReq}) \\
 &\quad \mid \text{pushReq})
 \end{aligned}$$

Note that, since “ \mid ” is binary, a source term is structured as a binary tree—its *parallel structure*—with a sum or a replicated input in its leaves and a parallel operator in each remaining node. At each such node, a matching pair of communication partners can meet. More precisely, for every pair of matching communication partners, there is exactly one node such that one partner is at its left and the other at its right side. Therefore, each parallel operator encoding pushes all received (left or right) requests further upwards to a surrounding parallel operator encoding by means of the forwarders in $\text{pushReq} \triangleq p_o, up \rightarrow p_o \mid p_i, up \rightarrow p_i$.

Requests from the left are forwarded to the links p_o, up or p_i, up , to be pushed further upwards with pushReq . Moreover, in order to combine requests from the left with requests from the right side, all left requests are forwarded to the right side over m_o and m_i . Thus left requests are processed by two simple forwarders, $\text{procLeftOutReq} \triangleq p_o \rightarrow \{m_o, p_o, up\}$ and $\text{procLeftInReq} \triangleq p_i \rightarrow \{m_i, p_i, up\}$.

The processing of requests from the right is more difficult. Intuitively, the encoding ensures that any request of the left side is combined exactly once with each opposite request of the right side. Then, the respective first parameters of each pair of requests are matched, to reveal a pair that results from the translation of matching communication partners. If such a pair is found, then the

information necessary to resolve the respective test-statement are retransmitted over the receiver lock back to the receiver, where the test-statement completes the emulation in case of positive instantiated sum locks. To avoid deadlock, the sum lock of the left request is always checked first. Since the encoding relies on the parallel structure of the source term, which is a binary tree, to prefer always the left lock indeed results in a total ordering of the sum locks.

$$\begin{aligned}
\text{procRightOutReq} &\triangleq \overline{c_o} \langle m_i \rangle \mid c_o^* (m_i) . p_o (y, l_s, s, z) . (\\
&(\nu m_{i,up}) (m_i^* (y', l_r, r) . ([y' = y] \overline{r} \langle l_r, l_s, l_s, s, z \rangle \mid \overline{m_{i,up}} \langle y', l_r, r \rangle) \\
&\quad \mid (\nu m_i) (m_{i,up} \rightarrow m_i \mid \overline{c_o} \langle m_i \rangle)) \\
&\mid \overline{p_{o,up}} \langle y, l_s, s, z \rangle) \\
\text{procRightInReq} &\triangleq \overline{c_i} \langle m_o \rangle \mid c_i^* (m_o) . p_i (y, l_r, r) . (\\
&(\nu m_{o,up}) (m_o^* (y', l_s, s, z) . ([y' = y] \overline{r} \langle l_s, l_r, l_s, s, z \rangle \mid \overline{m_{o,up}} \langle y', l_s, s, z \rangle) \\
&\quad \mid (\nu m_o) (m_{o,up} \rightarrow m_o \mid \overline{c_i} \langle m_o \rangle)) \\
&\mid \overline{p_{i,up}} \langle y, l_r, r \rangle)
\end{aligned}$$

In order to emulate arbitrary source term steps, all pairs of left and right requests have to be checked at least once. On the other side, a careless checking of the same pairs infinitely often introduces divergence. Thus, only a single copy of each left request is transmitted to the right side and, there, each pair of left and right requests is combined exactly once. To do so, the right requests are linked together within two chains; one for right output requests and one for right input requests. The first member of the chain receives all left requests via m_o or m_i , combines them with its own information, and sends a copy of each left request to the next member over $m_{o,up}$ or $m_{i,up}$, respectively. Subsequent members of a chain are linked by m_o or m_i , i.e., each member creates a new version of the corresponding name and sends this new version over c_o or c_i to enable the addition of a new member. Moreover, it transmits all received left requests along this new version. A new member is then added to the chain by the consumption of its request, also triggering to transmit a copy to **pushReq** via $p_{o,up}$ or $p_{i,up}$.

Finally restriction, match, and success are translated rigidly:

$$\begin{aligned}
[[(\nu x) P]_a^m] &\triangleq (\nu \varphi_a^m(x)) [[P]_a^m] \\
[[[a = b] P]_a^m] &\triangleq [\varphi_a^m(a) = \varphi_a^m(b)] [[P]_a^m] \\
[[\checkmark]_a^m] &\triangleq \checkmark
\end{aligned}$$

In the discussion so far, we omitted the encoding of replicated input, because it is slightly tricky. The crux is that each replicated input implicitly represents an unbounded number of copies of the respective input in parallel. Each such copy changes the parallel structure of the source term, on which our encoding function relies. Obviously, a compositional encoding can not first compute the number of required copies. By the reduction semantics, the copies of a replicated input are generated as soon they are needed. Likewise, the encoding of replicated

input adds a branch to the constructed parallel structure, for each emulated communication with a replicated input. To do so, it adapts the parallel operator translation for each unguarded continuation in `encodedContinuations`.

$$\begin{aligned} \llbracket y^*(x).P \rrbracket_a^m &\triangleq (\nu l, r, c_{r1}, c_{r2}, r_o, r_i) (\overline{p_i} \langle y, l, r \rangle \\ &\quad | r^* (-, -, l_s, s, x) . \text{test } l_s \text{ then } \overline{l_s} \langle \perp \rangle | \overline{s} | \overline{c_{r1}} \langle x \rangle \text{ else } \overline{l_s} \langle \perp \rangle \\ &\quad | \overline{r_i} \langle y, l, r \rangle | \overline{l} \langle \top \rangle | \text{encodedContinuations}) \end{aligned}$$

To direct the flow of requests among the additional branches, they are again ordered into a chain.

$$\begin{aligned} \text{encodedContinuations} &\triangleq \overline{c_{r2}} \langle r_o, r_i \rangle | c_{r1}^* (x) . c_{r2} (r_o, r_i) . \\ &\quad (\nu m_o, m_i, p_o, up, p_i, up, r_o, up, r_i, up, c_o, c_i, m_o, up, m_i, up) (\text{pushReqIn} \\ &\quad | (\nu p_o, p_i) (\llbracket P \rrbracket_a^m | \text{procRightOutReq} | \text{procRightInReq}) \\ &\quad | (\nu r_o, r_i) (\overline{c_{r2}} \langle r_o, r_i \rangle | \text{pushReqOut})) \end{aligned}$$

For each successful emulation on a replicated input, a new branch with the encoded continuation is unguarded by transmitting the received source term value over c_{r1} . As in the chains of right requests, each branch in `encodedContinuations` restricts its own versions of r_o and r_i to receive all requests from its successor. These links are transmitted over c_{r2} to the respective next member. The translation of the replicated input serves itself as first member of the chain by providing its own request over r_i . Note that the third line of `encodedContinuations` is exactly the same as the right side of a parallel operator encoding. There, all received requests are combined with the requests of the respective continuation to enable the emulation of a communication with the replicated input or another of its unguarded continuations. Moreover, to enable an emulation of a communication with the rest of the term, its requests are pushed upwards. The remaining terms `pushReqIn` and `pushReqOut` direct the flow of requests.

$$\begin{aligned} \text{pushReqIn} &\triangleq r_o \rightarrow \{m_o, r_o, up\} | r_i \rightarrow \{m_i, r_i, up\} \\ \text{pushReqOut} &\triangleq p_o, up \rightarrow \{p_o, r_o\} | r_o, up \rightarrow r_o | p_i, up \rightarrow \{p_i, r_i\} | r_i, up \rightarrow r_i \end{aligned}$$

`pushReqIn` receives all requests from a predecessor in the chain and forwards one copy to the encoded continuation over m_o and m_i and one copy to `pushReqOut`. There, all requests of the encoded continuation are pushed upwards to a surrounding parallel operator encoding over p_o or p_i , and for all such requests and all requests received from a previous member, a copy is forwarded to the successor over r_o or r_i .

For a more exhaustive description of the algorithm implemented by this encoding and how it emulates source term steps, we refer to the proof of its correctness in [\[PN12\]](#).

Theorem 1. *The encodings $\llbracket \cdot \rrbracket_a^s$ and $\llbracket \cdot \rrbracket_a^m$ are good.*

Observations

The existence of a good encoding from π_m into π_a shows that π_a is as expressive as π_m with respect to the abstract behaviour of terms. This looks surprising. From [Pal03, Gor10, PN10], we know that it is not possible to implement mixed choice without introducing some additional amount of coordination. The existence of the good encoding $\llbracket \cdot \rrbracket_a^m$ proves that, to do so, no global coordination is necessary. Instead the little amount of local coordination, which is allowed in compositional encodings, suffices to completely implement the full power of mixed (guarded) choice within an asynchronous and thus choice-free setting.

However, the encoding presented above comes with some drawbacks. The most crucial of these drawbacks—at least with respect to efficiency measures—is the impact of the encoding function on the degree of distribution of source terms. We consider this problem in the next section. Another drawback is the necessity of the match operator in the target language. Examining the proof of the main result in [PSN11], we observe that it already indicates how to solve the problem of the deadlocks in the test-statements of $\llbracket \cdot \rrbracket_a^s$. Moreover, it reveals a second solution to that problem. Instead of implementing an algorithm to order the sum locks, we could always test the sum lock of the receiver first, if we restrict the number of emulations that can be performed simultaneously. To do so, we augment $\llbracket \cdot \rrbracket_a^m$ with an additional coordinator lock—an output \bar{c} on a new channel c —for each encoding of a parallel operator and require that this lock must be available in order to send an output over the receiver lock r . Then, each completion of a test-statement in the encoding of input or replicate input—regardless of its outcome—restores the coordinator lock of the respective parallel operator encoding. Due to the restriction of simultaneous emulations, the impact of such an encoding on the degree of distribution of source terms is even worse than it is the case for $\llbracket \cdot \rrbracket_a^m$. However, for both solutions, it is necessary to send some kind of input and output requests and to combine requests of communication partners in order to emulate a communication step. Due to scope extrusion in the source and the necessity in the target to restrict the request channels, we cannot ensure that the requests of different source term steps can be distinguished by their channel names. Thus, to examine which pairs of requests refer to matching communication partners, we need the matching primitive.

A problem that already occurs in $\llbracket \cdot \rrbracket_a^s$ is the introduction of observable junk, i.e., of observable remainders left over by further emulations. In π_m , if we perform a step on a summand of a sum, immediately any other summand of that sum disappears. In the implementation, we have to split up the encoded summands in parallel, such that it is not possible to immediately withdraw the encoded summands as soon as one summand is used within an emulation. In $\llbracket \cdot \rrbracket_a^s$ and $\llbracket \cdot \rrbracket_a^m$ such observable junk is marked by a false instantiation of its sum locks. As a consequence, the encodings are not good w.r.t. a standard equivalence \approx , as asynchronous barbed congruence. However, for both encodings, we can prove correctness with respect to a non trivial variant of barbed equivalence, by redefining the notion of barbs to the result of translating source term barbs. The result is a congruence w.r.t. contexts that respect the protocol of the encoding.

As mentioned above, our encoding $\llbracket \cdot \rrbracket_a^m$ augments the parallel structure of source terms to order sum locks. Accordingly, another parallel structure of the source—e.g. as a result of applying the rule $P \mid Q \equiv Q \mid P$ to it—results in a different order of the respective sum locks. Hence, it is possible that, for some source terms S_1 and S_2 , the target terms $\llbracket S_1 \mid S_2 \rrbracket_a^m$ and $\llbracket S_2 \mid S_1 \rrbracket_a^m$ differ in the number of necessary pre- or postprocessing steps within an emulation, but also in the reachability of “intermediate”, i.e.: “partially committed”, states. Although these states exhibit different observables, their differences do not introduce deadlock or influence the possibility to emulate source term steps, i.e. $\llbracket S_1 \mid S_2 \rrbracket_a^m$ and $\llbracket S_2 \mid S_1 \rrbracket_a^m$ still have the same abstract behaviour. Interestingly, the alternative solution on coordinator locks reveals similar problems with the rule $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. To overcome this problem, the equivalence \asymp has either to abstract from the reachability of intermediate states or we have to avoid the rule CONG in our reduction semantics.

For a more formal and exhaustive discussion of these drawbacks and a definition of the used equivalences, we refer to [PN12]. We believe that none of the described drawbacks can be circumvented. In this sense, we think that the encoding $\llbracket \cdot \rrbracket_a^m$ given above is the best encoding from π_m into π_a we can achieve.

4 Distributability

The first result comparing the expressive power of π_m and π_a is given by the separation result in [Pa03]. The main difference to our encodability result in the last section is due to the requirement on the rigid translation of “|”. This requirement ensures that the encoding “preserves the degree of distribution” of the source term, which—thinking of distributed computing systems—is a crucial measure for the efficiency of such an encoding. A distributed system is a network of parallel processes. Accordingly, a distributed algorithm is an algorithm that may perform at least some of its tasks in parallel. Therefore, a main issue when considering an encoding between distributed algorithms is to ensure that it does not sequentialise all tasks by introducing a *global coordinator*. On the other side, the rigid translation of the parallel operator is a rather hard requirement. Therefore, Gorla instead requires the compositional translation of all source term operators. Note that also this requirement already prevents the use of global coordinators. In that view, compositionality can be seen as a minimal criterion to ensure the preservation of the degree of distribution.

However, sometimes—as in the current case—compositionality alone is too weak to consider the preservation of the degree of distribution, because it still allows for *local coordinators*: a compositional encoding may still sequentialise some of the parallel tasks of a distributed algorithm. If we are not only interested in the expressive power in terms of the abstract behaviour but additionally in how far problems can be solved exploiting at least the same degree of distribution, we must consider an additional criterion.

Up to now, there have been various approaches to explicitly consider the concurrent execution of independent steps directly within an operational semantics, often called *step semantics* (e.g., [Lan07] for the case of process calculi), and also

in the form of dedicated behavioral equivalences. In our case, we do not want to explicitly quantify the degree of distribution in the source and the target term, but only to measure whether it is preserved by an encoding. To this aim, we choose a simple and intuitive formulation—in the style of Gorla’s criteria—of our additional requirement based on the notion of parallel component.

Note that it does *not* suffice to consider the *initial* degree of distribution, i.e., to require that each source term and its encoding are distributed in the same way. We also have to require, that whenever a part of a source term can solve a task independently of the rest—i.e., it can reduce on its own—then the respective part of its encoding must also be able to emulate this reduction independent of the rest of the encoded term. Accordingly, we require that not only the source term and its encoding are distributed in the same way, but also their derivatives. In the following, \equiv_1 is the usual structural congruence naturally of the source language and \equiv_2 is the usual structural congruence of the target language.

Definition 5. *An encoding $\llbracket \cdot \rrbracket$ preserves the degree of distribution if, for every S such that $S \equiv_1 S_1 \mid \dots \mid S_n$ and $S_i \Longrightarrow S'_i$ for all i with $1 \leq i \leq n$, there exists T_1, \dots, T_n and a context \mathcal{C} with n holes such that $\llbracket S \rrbracket \equiv_2 \mathcal{C}(T_1, \dots, T_n)$ and $T_i \Longrightarrow \llbracket S'_i \rrbracket$ for all i with $1 \leq i \leq n$.*

Here, the context \mathcal{C} is introduced to allow, e.g., for some global restrictions or parts of the encoded term that may be necessary to emulate communications between S_i and S_j for $i \neq j$. This only makes sense because compositionality already rules out global coordinators. Since the parallel operator is considered to be binary, context \mathcal{C} can be the result of assembling parts of the contexts introduced by several parallel operator encodings. In essence, the requirement in Definition 5 is a concurrency-enhanced adaptation of operational completeness: whenever a source term can perform n steps in parallel, then its encoding must be able to emulate all n steps in parallel; note that the T_i must be able to move independent of the context \mathcal{C} . So, Definition 5 describes a *semantic* criterion.

Definition 5 is not the only way to measure the preservation of the degree of distribution. However, when considering the degree of distribution, we find it natural and appealing to require that parallel source term steps can be emulated truly in parallel, i.e., that for each pair of independent source term steps there is at least the possibility to emulate them independently. Moreover, by the following consideration, we observe that this requirement indeed suffices to reveal a fundamental difference in the expressive power of π_m compared to π_s or π_a considering the degree of distribution. Since $\llbracket \cdot \rrbracket_a^s$ translates the parallel operator rigidly, it naturally preserves the degree of distribution.

Lemma 1. *The encoding $\llbracket \cdot \rrbracket_a^s$ preserves the degree of distribution.*

Notably, in the proof of the lemma above we do not use any features of the encoding $\llbracket \cdot \rrbracket_a^s$ except that it satisfies operational completeness, i.e., it is a good encoding, translates the parallel operator rigidly, and preserves structural congruence. So, any such encoding preserves the degree of distribution. Not surprisingly, the most crucial requirement here is the rigid translation of “ \parallel ”.

Lemma 2. *Any good encoding, that translates the parallel operator rigidly and preserves enough of the structural congruence on source terms to ensure that $S \equiv_1 S_1 \mid \dots \mid S_n$ implies $\llbracket S \rrbracket \equiv_2 \llbracket S_1 \mid \dots \mid S_n \rrbracket$, preserves the degree of distribution.*

Thus, the (semantic) criterion formalised in Definition 5 can be considered to be at most as hard as the (syntactic) criterion to rigidly translate the parallel operator. To see that it is not an equivalent requirement, but indeed strictly weaker, we may consider an encoding (spelled out in [PN12]) from π_m (without replicated input) into π_a^2 , the asynchronous π -calculus augmented with a two-level polyadic synchronisation by Carbone and Maffei [CM03]. It is a simplified version of the encoding $\llbracket \cdot \rrbracket_a^m$, based on the same way to order sum locks but without the necessity to link right requests in chains. To prove that it is good, an argumentation similar as for $\llbracket \cdot \rrbracket_a^m$ can be applied. Moreover, [CM03] prove that there is no good encoding from π_m into π_a^2 that translates “ \mid ” rigidly; this separation result does not rely on replication, i.e., it also implies that there is no such encoding from π_m without replicated input into π_a^2 . On the other side, since all parts of the context introduced by the parallel operator encoding are replicated inputs, it preserves the degree of distribution.

The encoding $\llbracket \cdot \rrbracket_a^m$ does not preserve the degree of distribution, because we can not distribute the linking of right requests within a chain at the right side of a parallel operator encoding. Because of that, all steps on communication partners that meet at the same parallel operator in the source term, can never be emulated independently even if the source term steps are.

Lemma 3. *The encoding $\llbracket \cdot \rrbracket_a^m$ does not preserve the degree of distribution.*

This lemma is not due to an awkward design of the encoding function $\llbracket \cdot \rrbracket_a^m$, but is a general restriction on the encodability of mixed choice, i.e., it is not possible to design a good encoding from π_m into π_a that preserves the degree of distribution. This fact is a direct consequence of the theorem proved in [PSN11].

Theorem 2. *There is no good encoding from π_m into π_a that preserves the degree of distribution.*

5 Conclusion

We present a novel and for some readers perhaps surprising encodability result, showing that the asynchronous π -calculus is “as expressive as” the synchronous π -calculus with mixed choice, if the non-rigid translation of parallel composition is allowed. Furthermore, we present a fundamental limitation of each good encoding between these two languages concerning a novel criterion that measures the preservation of the degree of distribution. In contrast to the three semantic criteria of operational correspondence, divergence reflection, and success sensitivity, our new criterion does not primarily consider the (abstract) behaviour of terms but an additional dimension: the potential for concurrent execution. We conclude that considering the behaviour of terms, the full π -calculus and its asynchronous variant have the same expressive power. Our result complements

Palamidessi's result [Pal03], as her rigidity criterion includes more than just abstract behavior. Likewise, as expected, then translating a π_m -algorithm into a π_a -algorithm by such an encoding, one must tolerate losses in the efficiency of the respective algorithm—which Palamidessi's rigidity requirement would forbid.

Note that the separation of criteria considering the behaviour from additional requirements as the degree of distribution offers additional advantages, because we can more easily analyse the reasons of separation results and in how far they limit the degree of distribution of the encoded algorithm. There is no way to overcome the theoretical border stated by our separation result. However, the proof that an encoding does not preserve the degree of distribution can point out ways to nevertheless optimise a translation of algorithms, because it exactly states which parts can not be distributed.

Of course, only a study of other process calculi and corresponding encoding functions can reveal whether the proposed criterion is suited to measure the degree of distribution in general.

Acknowledgements. We thank Daniele Gorla for his very constructive comments and some fruitful discussions on preliminary versions of this work.

References

- [Bou92] Boudol, G.: Asynchrony and the π -calculus (note). Note, INRIA (May 1992)
- [CM03] Carbone, M., Maffei, S.: On the Expressive Power of Polyadic Synchronisation in π -Calculus. *Nordic Journal of Computing* 10, 1–29 (2003)
- [Gor08] Gorla, D.: Towards a Unified Approach to Encodability and Separation Results for Process Calculi. In: van Breugel, F., Chechik, M. (eds.) *CONCUR 2008*. LNCS, vol. 5201, pp. 492–507. Springer, Heidelberg (2008)
- [Gor10] Gorla, D.: Towards a Unified Approach to Encodability and Separation Results for Process Calculi. *Information and Computation* 208(9), 1031–1053 (2010)
- [HT91] Honda, K., Tokoro, M.: An Object Calculus for Asynchronous Communication. In: America, P. (ed.) *ECOOP 1991*. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)
- [Lan07] Lanese, I.: Concurrent and Located Synchronizations in π -Calculus. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) *SOFSEM 2007*. LNCS, vol. 4362, pp. 388–399. Springer, Heidelberg (2007)
- [Nes00] Nestmann, U.: What is a "Good" Encoding of Guarded Choice? *Information and Computation* 156(1-2), 287–319 (2000)
- [Pal03] Palamidessi, C.: Comparing the Expressive Power of the Synchronous and the Asynchronous π -calculi. *Mathematical Structures in Computer Science* 13(5), 685–719 (2003)
- [PN10] Peters, K., Nestmann, U.: Breaking Symmetries. In: Frösche, S.B., Valencia, F.D. (eds.) *EXPRESS. EPTCS*, vol. 41, pp. 136–150 (2010)
- [PN12] Peters, K., Nestmann, U.: Is it a "Good" Encoding of Mixed Choice? (Technical Report). Technical Report, TU Berlin, Germany (January 2012), <http://arxiv.org/corr/home>
- [PSN11] Peters, K., Schicke-Uffmann, J.-W., Nestmann, U.: Synchrony vs Causality in the Asynchronous Pi-Calculus. In: Luttkik, B., Valencia, F.D. (eds.) *EXPRESS. EPTCS*, vol. 64, pp. 89–103 (2011)
- [SW01] Sangiorgi, D., Walker, D.: *The π -calculus: A Theory of Mobile Processes*. Cambridge University Press, New York (2001)

Event Structure Semantics of Parallel Extrusion in the Pi-Calculus

Silvia Crafa¹, Daniele Varacca², and Nobuko Yoshida³

¹ Dip. di Matematica - Univ. Padova

² PPS - CNRS & Univ. Paris Diderot

³ Dept. of Computing - Imperial College London

Abstract. We give a compositional event structure semantics of the π -calculus. The main issues to deal with are the communication of free names and the extrusion of bound names. These are the source of the expressiveness of the π -calculus, but they also allow subtle forms of causal dependencies. We show that free name communications can be modeled in terms of “incomplete/potential synchronization” events. On the other hand, we argue that it is not possible to satisfactorily model parallel extrusion within the framework of stable event structures. We propose to model a process as a pair (\mathcal{E}, X) where \mathcal{E} is a prime event structure and X is a set of (bound) names. Intuitively, \mathcal{E} encodes the structural causality of the process, while the set X affects the computation on \mathcal{E} so as to capture the causal dependencies introduced by scope extrusion. The correctness of our true concurrent semantics is shown by an operational adequacy theorem with respect to the standard late semantics of the π -calculus.

1 Introduction

In the study of concurrent and distributed systems, the true-concurrent semantics approach takes concurrency as a primitive concept rather than reducing it to nondeterministic interleaving. One of the by-products of this approach is that the causal links between the process actions are more faithfully represented in true-concurrent models.

Prime event structures [14] are a causal model for concurrency which is particularly suited for the traditional process calculi such as CCS and CSP since they directly represent causality and concurrency simply as a partial order and an irreflexive binary relation. Winskel [18] proposed a compositional event structure semantics of CCS, that has been proved to be operationally adequate with respect to the standard labelled transition semantics, hence sound with respect to bisimilarity [20]. Similar results have been proved for variants of the π -calculus, namely for a restricted typed subcalculus [17] and for the internal π I-calculus [5], which are however less expressive than the full calculus. In this paper we extend this result to the full π -calculus.

The main issues when dealing with the full π -calculus are name passing and the extrusion of bound names. These two ingredients are the source of the expressiveness of the calculus, but they are problematic in that they allow complex forms of causal dependencies, as detailed below.

1.1 Free Name Passing

Compared to pure CCS, (either free or bound) name passing adds the ability to dynamically acquire new synchronization capabilities. For instance consider the π -calculus process $P = n(z).(\bar{z}\langle a \rangle \mid m(x))$, that reads from the channel n and uses the received name to output the name a in parallel with a read action on m . Hence a synchronization along the channel m is possible if a previous communication along the channel n substitutes the variable z exactly with the name m . Then, in order to be compositional, the semantics of P must also account for “potential” synchronizations that might be activated by parallel compositions, like the one on channel m .

To account for this phenomenon, we define the parallel composition of event structures so that synchronization events that involve input and output on different channels, at least one of which is a variable, are not deleted straight away. Moreover, the events produced by the parallel composition are relabelled by taking into account their causal history. For instance, the event corresponding to the synchronization pair $(\bar{z}\langle a \rangle, m(x))$ is relabelled into a τ action if, as in the process P above, its causal history contains a synchronization that substitutes the variable z with the name m .

1.2 The Causal Links Created by Name Extrusion

Causal dependencies in π -calculus processes arise in two ways [18]: by nesting prefixes (called *structural* or *prefixing* or *subject* causality) and by using a name that has been bound by a previous action (called *link* or *name* or *object* causality). While subject causality is already present in CCS, object causality is distinctive of the π -calculus. The interactions between the two forms of causal dependencies are quite complex. We illustrate them by means of examples.

Parallel Scope Extrusion. Consider the two processes $P = (vn)(\bar{a}\langle n \rangle.n(x))$ and $Q = (vn)(\bar{a}\langle n \rangle \mid n(x))$. The causal dependence of the action $n(x)$ on the output $\bar{a}\langle n \rangle$ is clear in the process P (i.e. there is a structural causal link), however, a similar dependence appears also in Q since a process cannot synchronize on the fresh name n before receiving it along the channel a (i.e. there is an objective causal link). Now consider the process $P_1 = (vn)(\bar{a}\langle n \rangle \mid \bar{b}\langle n \rangle)$: in the standard interleaving semantics of π -calculus only one output extrudes, either $\bar{a}\langle n \rangle$ or $\bar{b}\langle n \rangle$, and the other one does not. As a consequence, the second (free) output *depends* on the previous extruding output. However, in a true concurrent model we can hardly say that there is a dependence between the two parallel outputs, which in principle could be concurrently executed resulting in the parallel/simultaneous extrusion of the same name n to two different threads reading respectively on channel a and on channel b .

Dynamic Addition of New Extruders. We have seen that a bound name may have multiple extruders. In addition, the coexistence of free and bound outputs allows the set of extruders to dynamically change during the computation. Consider the process $P_2 = (vn)(\bar{a}\langle n \rangle \mid n(z)) \mid a(x).(\bar{x}\langle b \rangle \mid \bar{c}\langle x \rangle)$. It can either open the scope of n by extruding it along the channel a , or it can evolve to the process $(vn)(n(z) \mid \bar{n}\langle b \rangle \mid \bar{c}\langle n \rangle)$ where the output of the variable x has become a new extruder for both the actions with subject n . Hence after the first synchronization there is still the possibility of opening the scope of n by extruding it along the channel c .

The Lesson we Learned. The examples above show that the causal dependencies introduced by the scope extrusion mechanisms distinctive of the π -calculus can be understood in terms of the two ingredients of extrusion: name restriction and communication.

1. The restriction $(\nu n)P$ adds to the semantics of P a causal dependence between *every* action with subject n and *one of* the outputs with object n .
2. The communication of a restricted name adds *new* causal dependencies since both new extruders and new actions that need an extrusion may be generated by variable substitution.

A causal semantics for the π -calculus should account for such a dynamic additional objective causality introduced by scope extrusion. In particular, the first item above hints at the fact that we have to deal with a form of disjunctive (objective) causality. Prime event structures are stable models that represent disjunctive causality by duplicating events and so that different copies causally depend on different (alternative) events. In our case this amounts to represent different copies of any action with a bound subject, each one causally depending on different (alternative) extrusions. However, the fact that the set of extruders dynamically changes complicates the picture since new copies of any action with a bound subject should be dynamically spawned for each new extruder. In this way the technical details quickly become intractable, as discussed in Section 6.

In this paper we follow a different approach, that leads to an extremely simple technical development. The idea is to represent the disjunctive objective causality in a so-called inclusive way: in order to trace the causality introduced by scope extrusion it is sufficient to ensure that whenever an action with a bound subject is executed, at least one extrusion of that bound name must have been already executed, but it is not necessary to record which output was the real extruder. Clearly, such an inclusive-disjunctive causality is no longer representable with stable structures like prime event structures. However, we show that an operational adequate true concurrent semantics of the π -calculus can be given by encoding a π -process simply as a pair (\mathcal{E}, X) where \mathcal{E} is a prime event structure and X is a set of (bound) names. Intuitively, the causal relation of \mathcal{E} encodes the structural causality of a process. Instead, the set X affects the computation on \mathcal{E} : we define a notion of *permitted configurations*, ensuring that any computation that contains an action whose subject is a bound name in X , also contains a previous extrusion of that name. Hence a further benefit of this semantics is that it clearly accounts for both forms of causality: subjective causality is captured by the causal relation of event structures, while objective causality is implicitly captured by permitted configurations.

2 The π -Calculus

In this section we illustrate the synchronous, monadic π -calculus that we consider. We assume a countably-infinite set of names and a countably-infinite set of variables ranged over by m, \dots, q and by x, \dots, z , respectively. Let a, b, c range over both names and variables.

$$\begin{array}{ll}
 \text{Prefixes} & \pi ::= a(x) \mid \bar{a}\langle b \rangle \\
 \text{Processes} & P, Q ::= \sum_{i \in I} \pi_i.P_i \mid P \mid Q \mid (\nu n)P \mid A\langle \tilde{x}, \tilde{p} \mid \mathbf{z}, \mathbf{n} \rangle
 \end{array}
 \quad
 \begin{array}{l}
 \text{Definitions} \\
 A\langle \tilde{x}, \tilde{p} \mid \mathbf{z}, \mathbf{n} \rangle = P_A
 \end{array}$$

The syntax consists of the parallel composition, name restriction, finite summation of guarded processes and recursive definition. In $\sum_{i \in I} \pi_i.P_i$, I is a finite indexing set; when

<p>(IN LATE)</p> $\frac{}{a(x).P \xrightarrow{a(x)} P}$	<p>(OUT)</p> $\frac{}{\bar{a}\langle b \rangle.P \xrightarrow{\bar{a}\langle b \rangle} P}$	<p>(COMM)</p> $\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}\langle b \rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \{b/x\} \mid Q'}$
<p>(PAR)</p> $\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	<p>(OPEN)</p> $\frac{P \xrightarrow{\bar{a}\langle n \rangle} P' \quad n \neq a}{(vn)P \xrightarrow{\bar{a}\langle n \rangle} P'}$	<p>(CLOSE)</p> $\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}\langle n \rangle} Q'}{P \mid Q \xrightarrow{\tau} (vn)(P' \{n/x\} \mid Q')}$
<p>(RES)</p> $\frac{P \xrightarrow{\alpha} P'}{(vn)P \xrightarrow{\alpha} (vn)P'}$	<p>(SUM)</p> $\frac{P_i \xrightarrow{\alpha} P'_i \quad i \in I}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_i}$	<p>(REC)</p> $\frac{P_A \{\bar{y}, \bar{q} / \bar{x}, \bar{p}\} \{\mathbf{w}, \mathbf{m} / \mathbf{z}, \mathbf{n}\} \xrightarrow{\alpha} P' \quad A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n}) = P_A}{A(\bar{y}, \bar{q} \mid \mathbf{w}, \mathbf{m}) \xrightarrow{\alpha} P'}$

Fig. 1. Labelled Transition System of the π -calculus

If I is empty we write $\mathbf{0}$, or we simply omit it; we denote by $+$ the binary sum. A process $a(x).P$ can perform an input at a and the variable x is the placeholder for the name so received. The output case is symmetric: a process $\bar{a}\langle b \rangle.P$ can perform an output along the channel a . Notice that an output can send a name (either free or restricted) or a variable.

We assume that every constant A has a unique defining equation $A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n}) = P_A$. The symbol \bar{p} , resp. \bar{x} , denotes a tuple of distinct names, resp. variables, that correspond to the free names, resp. variables, of P_A . \mathbf{n} , resp. \mathbf{z} , represents an infinite sequence of distinct names $\mathbb{N} \rightarrow \text{Names}$, resp. distinct variables $\mathbb{N} \rightarrow \text{Variables}$, that is intended to enumerate the (possibly infinite) bound names, resp. bound variables, of P_A . The parameters \mathbf{n} and \mathbf{z} do not usually appear in recursive definitions in the literature. The reason we add them is that we want to maintain the following Basic Assumption: *Every bound name/variable is different from any other name/variable, either bound or free.* In the π -calculus, this policy is usually implicit and maintained along the computation by dynamic α -conversion: every time the definition A is unfolded, a copy of the process P_A is created whose bound names and variables must be fresh. This dynamic choice is difficult to interpret in the event structures. Hence, in order to obtain a precise semantic correspondence, our recursive definitions prescribe all the names and variables that will be possibly used in the recursive process (see [5] for some examples).

The sets of free and bound names and free and bound variables of P , denoted by $\text{fn}(P)$, $\text{bn}(P)$, $\text{fv}(P)$, $\text{bv}(P)$, are defined as usual but for constant processes, whose definitions are as follows: $\text{fn}(A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n})) = \{\bar{p}\}$, $\text{bn}(A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n})) = \mathbf{n}(\mathbb{N})$, $\text{fv}(A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n})) = \{\bar{x}\}$ and $\text{bv}(A(\bar{x}, \bar{p} \mid \mathbf{z}, \mathbf{n})) = \mathbf{z}(\mathbb{N})$. The operational semantics is given in Figure 1 in terms of an LTS (in late style [1]) where we let α, β range over the set of labels

¹ We could as well choose the early style semantics. However, in that case the event structure corresponding to a simple input process would be the sum of all possible (infinite) variable instantiations. Instead, the use of late semantics allows a cleaner and more intuitive approach.

$\{\tau, a(x), \bar{a}(b), \bar{a}(n)\}$. The syntax of labels shows that the object of an input is always a variable, whereas the object of a free output is either a variable (e.g. $b(x)$ or $\bar{a}(x)$) or a name. On the other hand, the object of a bound output is always a name, since it must occur under a restriction. Moreover, thanks to the Basic Assumption, the side conditions in rules (PAR) and (RES) are not needed anymore.

3 Event Structures

This section reviews basic definitions of prime event structures [7][14][19].

Definition 1 (Labelled Event Structure). *Let L be a set of labels. A labelled event structure is a tuple $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ s.t.*

- E is a countable set of events;
- $\langle E, \leq \rangle$ is a partial order, called the causal order;
- for every $e \in E$, the set $[e] := \{e' \mid e' < e\}$, called the enabling set of e , is finite;
- \smile is an irreflexive and symmetric relation, called the conflict relation, satisfying the following: for every $e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.
- $\lambda: E \rightarrow L$ is a labelling function that associates a label to each event in E .

Intuitively, labels represent *actions*, and events should be thought of as *occurrences of actions*. Labels allow us to identify events which represent different occurrences of the same action. In addition, labels are essential when composing two event structures in a parallel composition, as they identify which events correctly synchronise.

We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$, when $e_1 < e_2$. If a conflict is not inherited we say that it is *immediate*. If two events are not causally related nor in conflict they are said to be *concurrent*.

The notion of computation is usually captured in event structures in terms of configurations. A *configuration* C of an event structure \mathcal{E} is a conflict free downward closed subset of E , i.e. a subset C of E satisfying: (1) if $e \in C$ then $[e] \subseteq C$ and (2) for every $e, e' \in C$, it is not the case that $e \smile e'$, that is e and e' are either causally dependent or concurrent. In other words, a configuration represents a run of an event structure, where events are partially ordered. The set of configurations of \mathcal{E} , partially ordered by inclusion, is denoted as $\mathcal{L}(\mathcal{E})$. An alternative notion of computation can be defined in terms of labelled transition systems of event structures. Such a definition allows to more directly state (and prove) that the computational steps of a π -calculus process are reflected into its event structure semantics.

Definition 2 (LTS of event structures). *Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let e be one of its minimal events with $\lambda(e) = \beta$. Then we write $\mathcal{E} \xrightarrow{\beta} \mathcal{E} \lfloor e$, where $\mathcal{E} \lfloor e$ is the event structure $\langle E', \leq_{|E'}, \smile_{|E'}, \lambda_{E'} \rangle$ with $E' = \{e' \in E \mid e' \neq e \text{ and } e' \not\smile e\}$.*

Roughly speaking, $\mathcal{E} \lfloor e$ is \mathcal{E} minus the event e , and minus all events that are in conflict with e . The reachable LTS with initial state \mathcal{E} corresponds to the computations over \mathcal{E} .

Event structures have been shown to be the class of objects of a category [20]. Moreover, it is easily shown that an isomorphism in this category is a label-preserving bijective function that preserves and reflects causality and conflict. We denote by $\mathcal{E}_1 \cong \mathcal{E}_2$ the fact that there is an isomorphism between \mathcal{E}_1 and \mathcal{E}_2 .

We review here an informal description of several operations on labelled event structures, that we are going to use in the next section. See [19] for more details.

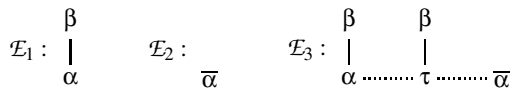
- *Prefixing* $a.\mathcal{E}$. This operation adds to the event structure a new minimal element, labelled by a , below every other event in \mathcal{E} .
- *Prefixed sum* $\sum_{i \in I} a_i.\mathcal{E}_i$. This is obtained as the disjoint union of copies of the event structures $a_i.\mathcal{E}_i$. The conflict relation is extended by putting in conflict every pair of events belonging to two different copies of $a_i.\mathcal{E}_i$.
- *Restriction* (or *Hiding*) $\mathcal{E} \setminus X$ where $X \subseteq L$ is a set of labels. This is obtained by removing from \mathcal{E} all events with label in X and all events that are above (i.e., causally depend on) one of those.
- *Relabelling* $\mathcal{E}[f]$ where L and L' are two sets of labels and $f : L \rightarrow L'$. This operation just consists in composing the labelling function λ of \mathcal{E} with the function f . The new event structure is labelled over L' and its labelling function is $f \circ \lambda$.

The parallel composition of two event structures \mathcal{E}_1 and \mathcal{E}_2 gives a new event structure \mathcal{E}' whose events model the parallel occurrence of pairs of events $e_1 \in E_1$ and $e_2 \in E_2$. In particular, when the labels of e_1 and e_2 match according to an underlying synchronisation model, \mathcal{E}' records (with an event $e' \in E'$) that a synchronisation between e_1 and e_2 is possible, and deals with the causal effects of such a synchronisation.

Technically, the parallel composition is defined as the categorical product followed by restriction and relabelling [20]. The categorical product represents all conceivable synchronisations, relabelling implements a synchronisation model by expressing which events are allowed to synchronise, and hiding removes synchronisations that are not permitted. The synchronisation model underlying the relabelling operation is formalised by the notion of *synchronisation algebra*, that is a partial binary operation \bullet_S defined on $L_* := L \uplus \{*\}$ where $*$ is a distinguished label. If α_i are the labels of events $e_i \in E_i$, then the event $e' \in E'$ representing the synchronisation of e_1 and e_2 is labelled by $\alpha_1 \bullet_S \alpha_2$. When $\alpha_1 \bullet_S \alpha_2$ is undefined, the synchronisation event e' is given a distinguished label *bad*, indicating that this event is not allowed and should be deleted.

Definition 3 (Parallel Composition of Event Structures). *Let $\mathcal{E}_1, \mathcal{E}_2$ two event structures labelled over L , let \bullet_S be a synchronisation algebra, and let $f_S : L_* \rightarrow L' = L_* \cup \{\text{bad}\}$ be a function defined as $f_S(\alpha_1, \alpha_2) = \alpha_1 \bullet_S \alpha_2$, if S is defined on (α_1, α_2) , and $f_S(\alpha_1, \alpha_2) = \text{bad}$ otherwise. The parallel composition $\mathcal{E}_1 \parallel_S \mathcal{E}_2$ is defined as the categorical product followed by relabelling and restriction: $\mathcal{E}_1 \parallel_S \mathcal{E}_2 = (\mathcal{E}_1 \times \mathcal{E}_2)[f_S] \setminus \{\text{bad}\}$. The subscripts S are omitted when the synchronisation algebra is clear from the context.*

Example 4. We show a simple example of parallel composition. Consider the set of labels $L = \{\alpha, \beta, \bar{\alpha}, \tau\}$ and the synchronisation algebra obtained as the symmetric closure of the following rules: $\alpha \bullet \bar{\alpha} = \tau$, $\alpha \bullet * = \alpha$, $\bar{\alpha} \bullet * = \bar{\alpha}$, $\beta \bullet * = \beta$ and undefined otherwise. Consider the two event structures $\mathcal{E}_1, \mathcal{E}_2$, where $E_1 = \{a, b\}, E_2 = \{a'\}$, with $a \leq b$ and $\lambda_1(a) = \alpha, \lambda_1(b) = \beta, \lambda_2(a') = \bar{\alpha}$. The event structures are represented as follows:



where dotted lines represent immediate conflict, while the causal order proceeds upwards along the straight lines. Then $\mathcal{E}_3 := \mathcal{E}_1 \parallel \mathcal{E}_2$ is the event structure $\langle E_3, \leq, \sim, \lambda \rangle$ where $E_3 = \{e := (\emptyset, a, *), e' := (\emptyset, *, a'), e'' := (\emptyset, a, a'), d := (\{e\}, a', *), d'' := (\{e''\}, a', *)\}$, and the ordering, immediate conflict and the labelling are as in the picture above.

We say that an event structure \mathcal{E} is a *prefix* of an event structure \mathcal{E}' , denoted $\mathcal{E} \leq \mathcal{E}'$ if there exists $\mathcal{E}'' \cong \mathcal{E}'$ such that $E \subseteq E''$, no event in $E'' \setminus E$ is below any event of E , and conflict and order in \mathcal{E} are the restriction of those in \mathcal{E}'' . Winskel [18] has shown that the class of event structures with the prefix order is a large CPO, and thus the limits of countable increasing chains exist. Moreover all operators on event structures are continuous. We will use this fact to define the semantics of the recursive definitions.

4 Free Name Passing

We present the event structure semantics of the full π -calculus in two phases, dealing separately with the two main issues of the calculus. We start in this section discussing free name passing, and we postpone to the next section the treatment of scope extrusion.

The core of a compositional semantics of a process calculus is parallel composition. When a process P is put in parallel with another process, new synchronizations can be triggered. Hence the semantics of P must also account for “potential” synchronizations that might be activated by parallel compositions. In Winskel’s event structure semantics of CCS [18], the parallel composition is defined as a product in a suitable category followed by relabelling and hiding, as we have presented in Section 3. For the semantics of the π -calculus, when the parallel composition of two event structures is computed, synchronisation events that involve input and output on different channels cannot be hidden straight away. If at least one of the two channels is a variable, then it is possible that, after prefixing and parallel composition, the two channels will be made equal.

We then resort to a technique similar to the one used in [5]: we consider a generalized notion of relabelling that takes into account the history of a (synchronization) event. Such a relabelling is defined according to the following ideas:

- each pair $(a(x), \bar{a}\langle b \rangle)$ made of two equal names or two equal variables is relabelled $\tau_{x \rightarrow b}$, to indicate that it represents a legal synchronization where b is substituted for x . Moreover, such a substitution must be propagated in all the events that causally depend on this synchronization. However, after all substitutions have taken place, there is no need to remember the extra information carried by the τ action, than the subscripts of the τ events are erased.
- Synchronisations pairs, like $(a(x), \bar{b}\langle c \rangle)$, involving different channels (at least one of which is a variable), are relabelled $(a(x), \bar{b}\langle c \rangle)_{x \rightarrow c}$, postponing the decision whether they represent a correct synchronization or not.
- Each pair $(n(x), \bar{m}\langle b \rangle)$ made of two different names is relabelled bad to denote a synchronization that is not allowed.

Definition 5 (Generalised Relabelling). *Let L and L' be two sets of labels, and let $\text{Pom}(L')$ be the set of pomsets (i.e., partially ordered multisets) over L' . Given an event structure $\mathcal{E} = \langle E, \leq, \dashv, \lambda \rangle$ with labels in L , and a function $f : \text{Pom}(L') \times L \rightarrow L'$, we define the relabelling operation $\mathcal{E}[f]$ as the event structure $\mathcal{E}' = \langle E, \leq, \dashv, \lambda' \rangle$ with labels in L' , where $\lambda' : E \rightarrow L'$ is defined by induction on the height of an element of E : if $h(e) = 0$ then $\lambda'(e) = f(\emptyset, \lambda(e))$, if $h(e) = n + 1$ then $\lambda'(e) = f(\lambda'([e]), \lambda(e))$.*

In words, an event e is relabelled with a label $\lambda'(e)$ that depends on the (pomset of) labels of the events belonging to its causal history $[e]$.

In the case of π -calculus with free names, let $L = \{a(x), \bar{a}(b), \tau \mid a, b \in \text{Names} \cup \text{Variables}, x \in \text{Variables}\}$ be the set of labels used in the LTS of π -calculus without restriction. We define the relabelling function needed by the parallel composition operation around the extended set of labels $L' = L \cup \{(\alpha, \beta)_{x \rightarrow b} \mid \alpha, \beta \in L\} \cup \{\tau_{x \rightarrow b}, \text{bad}\}$, where bad is a distinguished label. The relabelling function $f_\pi : \text{Pom}(L') \times (L' \uplus \{*\}) \times (L' \uplus \{*\}) \rightarrow L'$ is defined as follows (we omit the symmetric clauses):

$$\begin{aligned}
 f_\pi(X, \langle a(y), \bar{a}(b) \rangle) &= \tau_{y \rightarrow b} & f_\pi(X, \langle a(x), \bar{y}(c) \rangle) &= \begin{cases} \tau_{x \rightarrow c} & \text{if } \alpha_{y \rightarrow a} \in X \\ (a(x), \bar{y}(c))_{x \rightarrow c} & \text{otherwise} \end{cases} \\
 f_\pi(X, \langle n(y), \bar{m}(b) \rangle) &= \text{bad} & f_\pi(X, \langle y(x), \bar{a}(n) \rangle) &= \begin{cases} \tau_{x \rightarrow n} & \text{if } \alpha_{y \rightarrow a} \in X \\ (y(x), \bar{a}(n))_{x \rightarrow n} & \text{otherwise} \end{cases} \\
 f_\pi(X, \langle y(x), * \rangle) &= \begin{cases} a(x) & \text{if } \alpha_{y \rightarrow a} \in X \\ y(x) & \text{otherwise} \end{cases} & f_\pi(X, \langle \bar{y}(b), * \rangle) &= \begin{cases} \bar{a}(b) & \text{if } \alpha_{y \rightarrow a} \in X \\ \bar{y}(b) & \text{otherwise} \end{cases} \\
 f_\pi(X, \langle \alpha, * \rangle) &= \alpha & f_\pi(X, \langle \alpha, \beta \rangle) &= \text{bad} \quad \text{otherwise}
 \end{aligned}$$

The extra information carried by the τ -actions, differently from that of “incomplete synchronization” events, is only necessary in order to *define* the relabelling, but there is no need to keep it after the synchronization has been completed. Hence we apply a second relabelling *er* that simply erases the subscript of τ actions.

The semantics of the π -calculus is then defined as follows by induction on processes, where the parallel composition of event structure is defined by

$$\mathcal{E}_1 \parallel_\pi \mathcal{E}_2 = ((\mathcal{E}_1 \times \mathcal{E}_2) [f_\pi][er]) \setminus \{\text{bad}\}$$

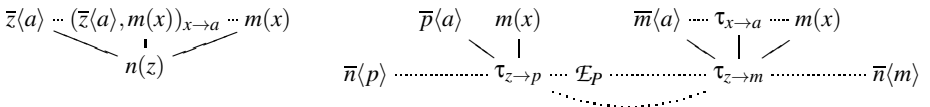
To deal with recursive definitions, we use an index k to denote the level of unfolding.

$$\begin{aligned}
 \{\mathbf{0}\}_k &= \mathbf{0} & \{\sum_{i \in I} \pi_i.P_i\}_k &= \sum_{i \in I} \pi_i.\{P_i\}_k & \{P \mid Q\}_k &= \{P\}_k \parallel_\pi \{Q\}_k \\
 \{A(\bar{y}, \tilde{q} \mid \mathbf{w}, \mathbf{m})\}_0 &= \mathbf{0} & \{A(\bar{y}, \tilde{q} \mid \mathbf{w}, \mathbf{m})\}_{k+1} &= \{P_A\{\bar{y}, \tilde{q} / \bar{x}, \tilde{p}\}\{\mathbf{w}, \mathbf{m} / \mathbf{z}, \mathbf{n}\}\}_k
 \end{aligned}$$

Recall that all operators on event structures are continuous with respect to the prefix order. It is thus easy to show that, for any k , $\{P\}_k \leq \{P\}_{k+1}$. We define $\{P\}$ to be the limit of the increasing chain $\dots \{P\}_k \leq \{P\}_{k+1} \leq \{P\}_{k+2} \dots$, that is $\{P\} = \sup_{k \in \mathcal{N}} \{P\}_k$. Since all operators are continuous w.r.t. the prefix order we have the following result:

Theorem 6 (Compositionality). *The semantics $\{P\}$ is compositional, i.e. $\{P \mid Q\} = \{P\} \parallel_\pi \{Q\}$, $\{\sum_{i \in I} \pi_i.P_i\} = \sum_{i \in I} \pi_i.\{P_i\}$.*

Example 7. As an example, consider the process $P = n(z).(\bar{z}(a) \mid m(x))$. The synchronization along the channel m can be only performed if the previous synchronization along n substitutes the variable z with the name m . Accordingly, the semantics of the process P is the leftmost event structure depicted below, denoted by \mathcal{E}_P . Moreover, the rightmost structure corresponds to the semantics of the process $P \mid \bar{n}(m) \mid \bar{n}(p)$.



The following theorem shows that the event structure semantics is operationally correct. Indeed, given a process P , the computational steps of P in the LTS of Section 2 are reflected by the semantics $\llbracket P \rrbracket$.

Theorem 8 (Operational Adequacy). *Let $\beta \in \{a(x), \bar{a}(b), \tau\}$. Suppose $P \xrightarrow{\beta} P'$ in the π -calculus. Then there exists \mathcal{E} such that $\llbracket P \rrbracket \xrightarrow{\beta} \mathcal{E}$ and $\mathcal{E} \cong \llbracket P' \rrbracket$. Conversely, suppose $\llbracket P \rrbracket \xrightarrow{\beta} \mathcal{E}'$. Then there exists P' such that $P \xrightarrow{\beta} P'$ and $\llbracket P' \rrbracket \cong \mathcal{E}'$.*

Note that the correspondence holds for the labels in the LTS of the calculus. Labels that identify “incomplete synchronizations” have been introduced in the event structure semantics for the sake of compositionality, but they are not considered in the theorem above since they do not correspond to any operational step. Moreover, the semantics is clearly not fully abstract in any reasonable sense, since interleaving equivalences are less discriminating than the corresponding causal equivalences on event structures.

5 Scope Extrusion

In this section we show how the causal dependencies introduced by scope extrusion can be captured by event structure-based models. As we discussed in Section 1, the communication of bound names implies that any action with a bound subject causally depends on a dynamic set of possible extruders of that bound subject. Hence dealing with scope extrusion requires modelling some form of disjunctive causality. Prime event structures are stable models that represent an action α that can be caused either by the action β_1 or the action β_2 as two different events e, e' that are both labelled α but e causally depends on the event labeled β_1 while e' is caused by the event labeled β_2 . In order to avoid the proliferation of events representing the same action with different extruders, we follow here a different approach, postponing to the next section a more detailed discussion on the use of prime event structures.

5.1 Event Structure with Bound Names

We define the semantics of the full π -calculus in terms of pairs (\mathcal{E}, X) , where \mathcal{E} is a prime event structure, and X is a set of names. We call such a pair an *event structure with bound names*. Intuitively, the causal relation of \mathcal{E} encodes the structural causality of a process, while the set X records bound names. Given a pair (\mathcal{E}, X) we define a notion of *permitted configurations*: a configuration that contains an action whose subject is a bound name, is permitted if it also contains a previous extrusion of that name. Objective causality is then implicitly captured by permitted configurations.

Definition 9 (Semantics). *The semantics of the full π -calculus is inductively defined as follows, where k denote the level of unfolding of recursive definitions, and we write \mathcal{E}_P^k , resp. X_P^k , for the first, resp. the second, projection of the pair $\llbracket P \rrbracket_k$:*

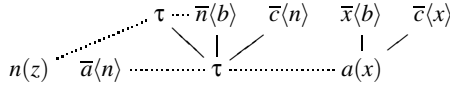
$$\begin{aligned} \llbracket \sum_{i \in I} \pi_i.P_i \rrbracket_k &= (\sum_{i \in I} \pi_i.\mathcal{E}_{P_i}^k, \cup_{i \in I} X_{P_i}^k) & \llbracket P \mid Q \rrbracket_k &= (\mathcal{E}_P^k \parallel_{\pi} \mathcal{E}_Q^k, X_P^k \cup X_Q^k) \\ \llbracket A\langle \bar{y}, \bar{q} \mid \mathbf{w}, \mathbf{m} \rangle \rrbracket_0 &= (\emptyset, \{\mathbf{w}(\mathbb{N})\}) & \llbracket (\nu n)P \rrbracket_k &= (\mathcal{E}_P^k, X_P^k \cup \{n\}) \\ \llbracket A\langle \bar{y}, \bar{q} \mid \mathbf{w}, \mathbf{m} \rangle \rrbracket_{k+1} &= (\mathcal{E}_{P_A}^k \{\bar{y}, \bar{q} / \bar{x}, \bar{p}\}^{\{\mathbf{w}, \mathbf{m} / \mathbf{z}, \mathbf{n}\}}, \{\mathbf{w}(\mathbb{N})\}) & \llbracket \mathbf{0} \rrbracket_k &= (\emptyset, \emptyset) \end{aligned}$$

It is easy to show that, for any k , $\mathcal{E}_P^k \leq \mathcal{E}_P^{k+1}$ and $X_P^k = X_P^{k+1} = X_P$. Then the semantics of a process P is defined as the following limit: $\llbracket P \rrbracket = (\sup_{k \in \mathcal{N}} \mathcal{E}_P^k, X_P)$.

The semantics is surprisingly simple: a restricted process $(\nu n)P$ is represented by a prime event structure that encodes the process P where the scope of n has been opened, and we collect the name n in the set of bound names. As for parallel composition, the semantics $\llbracket P \mid Q \rrbracket$ is a pair (\mathcal{E}, X) where X collects the bound names of both $\llbracket P \rrbracket$ and $\llbracket Q \rrbracket$ (recall that we assumed that bound names are pairwise different), while the event structure \mathcal{E} is obtained exactly as in the previous sections. This is since the event structures that get composed correspond to the processes P and Q where the scope of any bound name has been opened. The following property can be immediately proved.

Proposition 10. *Let P, Q be two processes, then $\llbracket (\nu n)P \mid Q \rrbracket = \llbracket (\nu n)(P \mid Q) \rrbracket$.*

Example 11. Consider the process $P = (\nu n)(\bar{a}\langle n \rangle \mid n(z) \mid a(x).(\bar{x}\langle b \rangle \mid \bar{c}\langle x \rangle))$, whose first synchronization produces a new extruder $\bar{c}\langle n \rangle$ for the bound name n . The semantics of P is the pair $(\mathcal{E}_P, \{n\})$, where \mathcal{E}_P is the following event structure:



In order to study the operational correspondence between the LTS semantics of the π -calculus and the event structure semantics above, we first need to adapt the notion of computational steps of the pairs (\mathcal{E}, X) . The definition of labelled transitions between prime event structures, i.e., Definition 2 is generalized as follows.

Definition 12 (Permitted Transitions). *Let (\mathcal{E}, X) be a labelled event structure with bound names. Let e be a minimal event of \mathcal{E} with $\lambda(e) = \beta$. We define the following permitted labelled transitions:*

- $(\mathcal{E}, X) \xrightarrow{\beta} (\mathcal{E} \setminus e, X)$, if $\beta \in \{\tau, a(x), \bar{a}\langle b \rangle\}$ with $a, b \notin X$.
- $(\mathcal{E}, X) \xrightarrow{\bar{a}\langle n \rangle} (\mathcal{E} \setminus e, X \setminus \{n\})$, if $\beta = \bar{a}\langle n \rangle$ with $a \notin X$ and $n \in X$.

According to this definition, the set of bound names constrains the set of transitions that can be performed. In particular, no transition whose label has a bound subject is allowed. On the other hand, when a minimal event labeled $\bar{a}\langle n \rangle$ is consumed, if the name n is bound, the transition's labels records that this event is indeed a bound output. Moreover, in this case we record that the scope of n is opened by removing n from the set of bound names of the target pair. Finally, observe that the previous definition only allows transitions whose labels are in the set $L = \{\tau, a(x), \bar{a}\langle b \rangle, \bar{a}\langle n \rangle\}$, which is exactly the sets of labels in the LTS of Section 2.

Theorem 13 (Operational Adequacy). *Let $\beta \in \{a(x), \bar{a}\langle b \rangle, \bar{a}\langle n \rangle, \tau\}$. Suppose $P \xrightarrow{\beta} P'$ in the π -calculus. Then there exists \mathcal{E} s.t. $\llbracket P \rrbracket \xrightarrow{\beta} \mathcal{E}$ and $\mathcal{E} \cong \llbracket P' \rrbracket$. Conversely, suppose $\llbracket P \rrbracket \xrightarrow{\beta} \mathcal{E}'$. Then there exists P' s.t. $P \xrightarrow{\beta} P'$ and $\llbracket P' \rrbracket \cong \mathcal{E}'$.*

5.2 Subjective and Objective Causality

Given an event structure with bound names (\mathcal{E}, X) , Definition 12 shows that some configurations of \mathcal{E} are no longer allowed. For instance, if e is minimal but its label has a subject that is a name in X , e.g. $\lambda(e) = n(x)$ with $n \in X$, then the configuration $\{e\}$ is no longer allowed since the event e requires a previous extrusion of the name n .

Definition 14 (Permitted Configuration). *Let (\mathcal{E}, X) be an event structure with bound names. For an event $e \in E$ define $e \uparrow = \{e' \mid e \leq e'\}$. Given a configuration C of \mathcal{E} , we say that C is permitted in (\mathcal{E}, X) whenever, for any $e \in C$ whose label has subject n with $n \in X$,*

- $C \setminus e \uparrow$ is permitted, and
- $C \setminus e \uparrow$ contains an event whose label is an output action with object n .

The first item of the definition above is used to avoid circular definitions that would allow wrong configurations like $\{\bar{n}\langle m \rangle, \bar{m}\langle n \rangle\}$ with $X = \{n, m\}$. Now, the two forms of causality of the π -calculus can be defined using event structures with bound names and permitted configurations.

Definition 15 (Subjective and Objective Causality). *Let P be a process of the π -calculus, and $\llbracket P \rrbracket = (\mathcal{E}_P, X_P)$ be its semantics. Let be $e_1, e_2 \in E_P$, then*

- e_2 has a subjective dependence on e_1 if $e_1 \leq_{\mathcal{E}_P} e_2$;
- e_2 has a objective dependence on e_1 if (i) the label of e_1 is the output of a name in X which is also the subject of the label of e_2 , and if (ii) there exists a configuration C that is permitted in (\mathcal{E}, X) and that contains both e_1 and e_2 .

Example 16. Let consider again the process P in Example 11. The configurations $C_1 = \{\bar{a}\langle n \rangle, n\langle z \rangle\}$ and $C_2 = \{\tau, \bar{c}\langle n \rangle, n\langle z \rangle\}$ are both permitted by $\llbracket P \rrbracket$, and they witness the fact that the action $n\langle z \rangle$ has an objective dependence on $\bar{a}\langle n \rangle$ and on $\bar{c}\langle n \rangle$. We could also say that $n\langle z \rangle$ objectively depends either on $\bar{a}\langle n \rangle$ or on $\bar{c}\langle n \rangle$.

Example 17. Let be $P = (\nu n)(\bar{a}\langle n \rangle \mid \bar{b}\langle n \rangle \mid n(x))$, then $\llbracket P \rrbracket = (\mathcal{E}_P, \{n\})$ where \mathcal{E}_P has three concurrent events. In this process there is no subjective causality, however the action $n(x)$ has an objective dependence on $\bar{a}\langle n \rangle$ and on $\bar{b}\langle n \rangle$ since both $C_1 = \{\bar{a}\langle n \rangle, n(x)\}$ and $C_2 = \{\bar{b}\langle n \rangle, n(x)\}$ are permitted configurations.

Example 18. Let be $P = (\nu n)(\bar{a}\langle n \rangle.\bar{b}\langle n \rangle.n(x))$, then $\llbracket P \rrbracket = (\mathcal{E}_P, \{n\})$ where \mathcal{E}_P is a chain of three events. According to the causal relation of \mathcal{E}_P , the action $n(x)$ has a structural dependence on both the outputs. Moreover, the permitted configuration $C = \{\bar{a}\langle n \rangle, \bar{b}\langle n \rangle, n(x)\}$ shows that $n(x)$ has an objective dependence on $\bar{a}\langle n \rangle$ and on $\bar{b}\langle n \rangle$. In this case we do not know which of the two outputs really extruded the bound name, accordingly to the inclusive disjunctive causality approach we are following.

5.3 The Meaning of Labelled Causality

In this paper we focus on compositional semantics, studying a true concurrent semantics that operationally matches the LTS semantics of the π -calculus. Alternatively, one

could take as primitive the reduction semantics of the π -calculus, taking the perspective that only τ -events are “real” computational steps of a concurrent system. Therefore one could argue that the concept of causal dependency makes only sense between τ events. In this perspective, we propose to interpret the causal relation between non- τ events as an anticipation of the causal relations involving the synchronizations they will take part in. In other terms, non- τ events (from now on simply called labelled events) represent “incomplete” events, that are waiting for a synchronization or a substitution to be completed. Hence we can prove that in our semantics two labelled events e_1 and e_2 are causally dependent *if and only if* the τ -events they can take part in are causally dependent. This property is expressed by the following theorem in terms of permitted configurations. Recall that the parallel composition of two event structures is obtained by first constructing the cartesian product. Therefore there are projection morphisms π_1, π_2 on the two composing structures. Let call τ -configuration a configuration whose events are all τ -events. Note that every τ -configuration is permitted.

Theorem 19. *Let P be a process. A configuration C is permitted in $\{\!\{P\}\!\}$ if and only if there exists a process Q and a τ -configuration C' in $\{\!\{P \mid Q\}\!\}$ such that $\pi_1(C') = C$.*

Let e_1, e_2 be two labelled events of $\{\!\{P\}\!\} = (\mathcal{E}_P, X_P)$. If e_1, e_2 are structurally dependent, i.e., $e_1 \leq_{\mathcal{E}_P} e_2$, then such a structural dependence is preserved and reflected in the τ -actions they are involved in because of the way the parallel composition of event structures is defined. On the other hand, let be e_1, e_2 objectively dependent. Consider the parallel composition $(\mathcal{E}_P \parallel_{\pi} \mathcal{E}_Q, X_P \cup X_Q)$ for some Q such that there is a τ event e'_2 in $\mathcal{E}_P \parallel_{\pi} \mathcal{E}_Q$ with $\pi_1(e'_2) = e_2$ and $[e'_2]$ is a τ -configuration. Then there must be an event $e'_1 \in [e'_2]$ such that $\pi_1(e'_1) = e_1$.

6 Disjunctive Causality

As we discussed in Section 4, objective causality introduced by scope extrusion requires for the π -calculus a semantic model that is able to express some form of disjunctive causality. In the previous section we followed an approach that just ensures that some extruder (causally) precedes any action with a bound subject. However, we could alternatively take the effort of tracing the identity of the actual extruders. We could do it by duplicating the events corresponding to actions with bound subject and letting different copies depend on different, alternative, extruders. Such a duplication allows us to use prime event structures as semantics models. In this section we discuss this alternative approach showing to what extent it can be pursued.

As a first example, the semantics of the process $P = (vn)(\bar{a}\langle n \rangle \mid \bar{b}\langle n \rangle \mid n(x))$, containing two possible extruders for the action $n(x)$, can be represented by left-most prime event structure in Figure 2. When more than a single action use as subject the same bound name, each one of these actions must depend on one of the possible extruders. Then the causality of the process $(vn)(\bar{a}\langle n \rangle \mid \bar{b}\langle n \rangle \mid n(x).n(y))$, represented by the right-most event structure in Figure 2, shows that the two read actions might depend either on the same extruder or on two different extrusions.

Things get more complicate when dealing with the dynamic addition of new extruders by means of communications. In order to guarantee that there are distinct copies of any event with a bound subject that causally depend on different extruders, we have

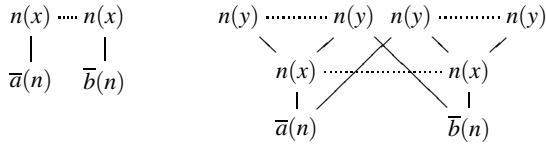
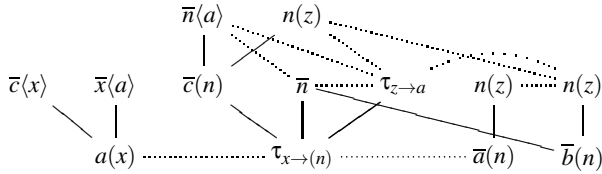


Fig. 2.

to consider the objective causalities generated by a communication. More precisely, when the variable x is substituted with a bound name n by effect of a synchronization, (i) any action with subject x appearing in the reading thread becomes an action that requires a previous scope extrusion, and (ii) the outputs with object x become new extruders for any action with subject n or x . To exemplify, consider the process $P' = (vn)(\bar{a}\langle n \rangle \mid \bar{b}\langle n \rangle \mid n\langle z \rangle \mid a\langle x \rangle.(\bar{x}\langle a \rangle \mid \bar{c}\langle x \rangle))$ that initially contains two extruders for n , and with the synchronization along the channel a evolves to $(vn)(\bar{b}\langle n \rangle \mid n\langle z \rangle \mid \bar{n}\langle a \rangle \mid \bar{c}\langle n \rangle)$. Its causal semantics can be represented with the following prime event structure:



The read action $n\langle z \rangle$ may depend on one of the two initial extruders $\bar{a}\langle n \rangle$ and $\bar{b}\langle n \rangle$, or on the new extruder $\bar{c}\langle n \rangle$ that is generated by the first communication. Accordingly, three different copies of the event $n\langle z \rangle$ appear over each of the three extruders. On the other hand, the output action on the bound name n is generated by the substitution entailed by the communication along the channel a , hence any copy of that action keeps a (structural) dependence on the corresponding τ event. Moreover, since it is an action with bound subject, there must be a copy of it for each of the remaining extruders of n , that is $\bar{b}\langle n \rangle$ and $\bar{c}\langle n \rangle$. To enhance readability, the event structure resulting from the execution of the communication along the channel a is the leftmost e.s. in Figure 3.

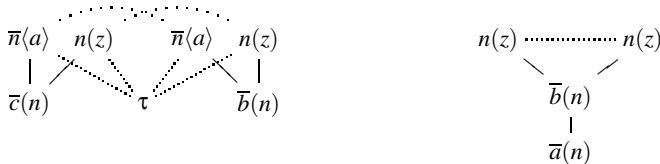


Fig. 3.

So far so good, in particular it seems possible to let the causal relation of prime event structures encode both structural and objective causality of π -processes. However, this is not the case. Consider the process $P = (vn)(\bar{a}\langle n \rangle. \bar{b}\langle n \rangle. n\langle z \rangle)$ of Example 13. If we just duplicate the event $n\langle z \rangle$ to distinguish the fact that it might depend on an extrusion along a or along b , we obtain the rightmost structure in Figure 3, that we

denote \mathcal{E}_p . In particular, even if the two copies intends to represent two different objective causalities, nothing distinguishes them since they both structurally depend on both outputs. This is a problem when we compose the process P in parallel with, e.g., $Q = a(x).\overline{c}(x) \mid b(y).\overline{d}(y)$. After two synchronizations we would like to obtain two copies of the read actions on n that depend on the two different remaining extruders $\overline{c}(n)$ and $\overline{d}(n)$. However, in order to obtain such an event structure as the parallel composition of the semantics of P and Q we must be able to record somehow the different objective causality that distinguishes the two copies of $n(z)$ in the semantics of P .

The technical solution would be to enrich the event labels so that the label of an event e also records the identity of the extruder events that e (objectively) causally depends on. A precise account of this approach is technically involved and intractable, so that the intuition on the semantics gets lost. Moreover, we think that this final example sheds light on the fact that structural and objective causality of π -processes cannot be expressed by the sole causal relation of event structures. To conclude, at the price of losing the information about which extruder an event depends on, the approach we developed in the previous section brings a number of benefits: it is technically much simpler, it is operationally adequate, and it gives a clearer account of the two forms of causality distinctive of π -processes.

7 Related Work

There are several causal models for the π -calculus, that use different techniques. There exist noninterleaving semantics in terms of labelled transition systems, where the causal relations between transitions are represented by “proofs” which allow to distinguish different occurrences of the same transition [16][18]. In [4], a more abstract approach is followed, which involves indexed transition systems. In [11], a semantics of the π -calculus in terms of pomsets is given, following ideas from dataflow theory. The two papers [3][9] present Petri nets semantics of the π -calculus. However, none of these approaches accounts for parallel extrusion. We finally recall [13] that introduces a graph rewriting-based semantics of the π -calculus that allows parallel extrusions.

Previous work on an event structure semantics of the π -calculus are [17][5] that study fragments of the calculus, while [2] gives an unlabelled event structure semantics of the full calculus which only corresponds to the *reduction* semantics, hence which is not compositional.

We plan for future work the application of the present semantics to the study of a labelled reversible semantics of the π -calculus that would extend the work of Danos and Krivine [6]. Phillips and Ulidowski [15] noted the strict correspondence between reversible transition systems and event structures. A first step in this direction is [12], which proposes a reversible semantics of the π -calculus that only considers reductions. It would also be interesting to study which kind of configuration structures [10] can naturally include our definition of permitted configuration.

Acknowledgment. We thank Ilaria Castellani, whose curiosity and skepticism pushed us to dig deeper. We also thank Giotto, for his wonderful frescos, and seaside greek iced coffes, for their mind-refreshing power. The third author is partially supported by EPSRC EP/F003757/01 and G015635/01.

References

1. Boreale, M., Sangiorgi, D.: A fully abstract semantics for causality in the π -calculus. *Acta Inf.* 35(5), 353–400 (1998)
2. Bruni, R., Melgratti, H., Montanari, U.: Event Structure Semantics for Nominal Calculi. In: Baier, C., Hermanns, H. (eds.) *CONCUR 2006*. LNCS, vol. 4137, pp. 295–309. Springer, Heidelberg (2006)
3. Busi, N., Gorrieri, R.: A Petri Net Semantics for Pi-Calculus. In: Lee, I., Smolka, S.A. (eds.) *CONCUR 1995*. LNCS, vol. 962, pp. 145–159. Springer, Heidelberg (1995)
4. Cattani, G.L., Sewell, P.: Models for name-passing processes: Interleaving and causal. In: *Proceedings of 15th LICS*, pp. 322–332. IEEE (2000)
5. Crafa, S., Varacca, D., Yoshida, N.: Compositional Event Structure Semantics for the Internal Pi-Calculus. In: Caires, L., Vasconcelos, V.T. (eds.) *CONCUR 2007*. LNCS, vol. 4703, pp. 317–332. Springer, Heidelberg (2007)
6. Danos, V., Krivine, J.: Reversible Communicating Systems. In: Gardner, P., Yoshida, N. (eds.) *CONCUR 2004*. LNCS, vol. 3170, pp. 292–307. Springer, Heidelberg (2004)
7. Degano, P., De Nicola, R., Montanari, U.: On the consistency of truly concurrent operational and denotational semantics. In: *LICS*, pp. 133–141. IEEE (1988)
8. Degano, P., Priami, C.: Non-interleaving semantics for mobile processes. *Theor. Comp. Sci.* 216(1-2), 237–270 (1999)
9. Engelfriet, J.: A multiset semantics for the pi-calculus with replication. *Theor. Comp. Sci.* 153(1&2), 65–94 (1996)
10. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures, event structures and petri nets. *Theor. Comput. Sci.* 410(41), 4111–4159 (2009)
11. Jagadeesan, L.J., Jagadeesan, R.: Causality and True Concurrency: A Data-Flow Analysis of the Pi-Calculus. In: Alagar, V.S., Nivat, M. (eds.) *AMAST 1995*. LNCS, vol. 936, pp. 277–291. Springer, Heidelberg (1995)
12. Lanese, I., Mezzina, C.A., Stefani, J.: Reversing Higher-Order Pi. In: Gustin, P., Laroussinie, F. (eds.) *CONCUR 2010*. LNCS, vol. 6269, pp. 478–493. Springer, Heidelberg (2010)
13. Montanari, U., Pistore, M.: Concurrent semantics for the pi-calculus. *Electr. Notes Theor. Comput. Sci.* 1, 411–429 (1995), *Proceedings of MFPS 1995*
14. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. *Theor. Comp. Sci.* 13(1), 85–108 (1981)
15. Phillips, I., Ulidowski, I.: Reversibility and models for concurrency. *Electr. Notes Theor. Comput. Sci.* 192(1), 93–108 (2007), *Proceedings of SOS 2007*
16. Sangiorgi, D.: Locality and True-Concurrency in Calculi for Mobile Processes. In: Hagiya, M., Mitchell, J.C. (eds.) *TACS 1994*. LNCS, vol. 789, pp. 405–424. Springer, Heidelberg (1994)
17. Varacca, D., Yoshida, N.: Typed event structures and the linear pi-calculus. *Theor. Comput. Sci.* 411(19), 1949–1973 (2010)
18. Winskel, G.: Event Structure Semantics for CCS and Related Languages. In: Nielsen, M., Schmidt, E.M. (eds.) *ICALP 1982*. LNCS, vol. 140, pp. 561–576. Springer, Heidelberg (1982)
19. Winskel, G.: Event Structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) *APN 1986*. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)
20. Winskel, G., Nielsen, M.: Models for concurrency. In: *Handbook of Logic in Computer Science*, vol. 4. Clarendon Press (1995)

Narcissists Are Easy, Stepmothers Are Hard

Daniel Gorín¹ and Lutz Schröder^{1,2}

¹ DFKI GmbH, Bremen, Germany

² Department of Computer Science, Universität Bremen, Germany

firstname.lastname@dfki.de

Abstract. Modal languages are well-known for their robust decidability and relatively low complexity. However, as soon as one adds a self-referencing construct, like hybrid logic’s down-arrow binder, to the basic modal language, decidability is lost, even if one restricts binding to a single variable. Here, we concentrate on the latter case and investigate the logics obtained by restricting the nesting depth of modalities between binding and use. In particular, for distances strictly below 3 we obtain well-behaved logics with a relatively high descriptive power. We investigate the fragment with distance 1 in the framework of coalgebraic modal logic, for which we provide very general decidability and complexity results. For the fragment with distance 2 we focus on the case of Kripke semantics and obtain optimum complexity bounds (no harder than the base logic). We show that this fragment is expressive enough to accommodate the guarded fragment over the correspondence language.

1 Introduction

Modal logics are known for their robust decidability and relatively low complexity. However, they don’t play along well with binding constructs such as the \downarrow binder of hybrid logic, which allows naming the current point of evaluation for later reference. Hybrid logic with \downarrow and the satisfaction operator $@$ is a conservative reduction class for first-order logic and, therefore, has undecidable satisfiability and finite satisfiability problems (see, e.g., [3]). Undecidability in the presence of \downarrow is rather robust. E.g. it persists without $@$, without nominals, and even if only one variable is allowed to be bound by \downarrow (or, semantically, if only one state can be remembered at any given time) [14]. Also, satisfiability with respect to classes of models that are typically computationally well-behaved (e.g., linear, transitive or equivalence relation frames) is undecidable (except in the uni-modal case) [18]. Weakened versions of \downarrow were investigated and also turned out to be undecidable [2].

Syntactic fragments of the hybrid language with \downarrow were investigated in [22]. There, it is observed that undecidability of the logic with \downarrow can be established by reduction from the tiling problem using a formula that contains the so-called $\Box\downarrow\Box$ pattern, i.e. a \Box -modality occurs under the scope of a \downarrow that occurs under the scope of a \Box . It is then shown that interdicting this pattern (in negation normal forms) ensures decidability.

Here, we investigate a different form of restriction, coming from the same observation. The reduction of the tiling problem given in [22] uses three modalities: \Box is made a master modality using a spypoint; \Diamond_1 and \Diamond_2 are forced to be total functions; and the crucial grid is defined by the formula

$$\Box \downarrow x. \Box (s \rightarrow \Box (\Diamond_1 \Diamond_2 x \rightarrow \Diamond_2 \Diamond_1 x)).$$

Observe that there are four modalities between the binding of x and its uses. Informally, we say that x occurs at depth 4 from its binding. This raises the question of the decidability status of fragments of the logic with \downarrow (and only one variable) where every use of the variable occurs at a given maximal depth k from its binding. Known undecidability proofs work for $k = 4$ [14]; as a first step in our investigation, we show that undecidability holds also for $k = 3$.

We then show (Section 3) that for depth at most 1, and not only for the Kripke semantics of modal logic but for any logic in the framework of *coalgebraic logic* (which supports, e.g., probabilities, counting, preferences, and game logic) [16,21], we obtain a decidable logic with the exponential model property, even when the global modality \mathbf{A} is added to the language. Under mild assumptions, it can be shown that satisfiability is in fact in EXPTIME, a tight bound in most cases. These results can be shown to hold also in the presence of nominals.

For Kripke semantics, we can improve these results in several ways (Section 4): we can allow up to depth 2 and retain decidability in EXPTIME, even though the finite model property breaks. Moreover, we establish a quasi-tree model property for the fragment without \mathbf{A} and prove decidability in PSPACE.

The language with \mathbf{A} and a depth bound of 2 for occurrence of the bound variable is quite expressive. In particular, we show (Section 5) that it subsumes the guarded fragment over the modal correspondence language (without constants and on formulas with two free variables). Because the guarded fragment does enjoy the finite model property, the containment is proper.

2 Coalgebraic Logics with Self-reference

For greater generality, we work in the framework of *coalgebraic modal logic* [16], which covers a broad range of modalities beyond the standard relational setup, including probabilistic and game-theoretic phenomena as well as neighbourhood semantics and non-material conditionals [21]. This framework is parametric in syntax and semantics. The syntax is given by a *similarity type* A , i.e. a set of *modal operators* with finite arities ≥ 0 (hence possibly including propositional atoms); to simplify notation, we will pretend that all operators are unary. Adopting the notation of [14], we use the personal pronouns I and me as binder and bindee, respectively. The full language $\mathcal{F}(A, \mathbf{I}, @, \mathbf{A})$ is given by the grammar

$$\mathcal{F}(A, \mathbf{I}, @, \mathbf{A}) \ni \phi, \psi ::= \perp \mid \text{me} \mid \phi \rightarrow \psi \mid \heartsuit \phi \mid \mathbf{A}\phi \mid \mathbf{I}.\phi \mid @_{\text{me}}\phi$$

where $\heartsuit \in A$. We use the standard derived boolean operators \neg, \wedge , etc., while \mathbf{E} denotes the dual of \mathbf{A} . Occasionally, we will assume all formulas to be in

negation normal form (nnf) and in such cases \neg , \wedge and \vee will be taken as primitive. We will also usually restrict our attention to syntactic fragments of $\mathcal{F}(\Lambda, \mathbf{I}, @, \mathbf{A})$, such as $\mathcal{F}(\Lambda, \mathbf{I}, @)$ or $\mathcal{F}(\Lambda, \mathbf{I})$: the fragments without \mathbf{A} and with neither \mathbf{A} nor $@$, respectively. We use $rank(\phi)$ to denote the maximum number of nested occurrences of $\heartsuit \in \Lambda$ in ϕ . The notion of $rank$ is trivially extended to finite sets of formulas.

The semantics of the logic is given by an endofunctor $T : \mathbf{Set} \rightarrow \mathbf{Set}$ and, for each $\heartsuit \in \Lambda$, a *predicate lifting* $\llbracket \heartsuit \rrbracket$, i.e., a natural transformation $\llbracket \heartsuit \rrbracket : \mathcal{Q} \rightarrow \mathcal{Q} \circ T^{op}$, where \mathcal{Q} is the contravariant powerset functor $\mathbf{Set}^{op} \rightarrow \mathbf{Set}$ (i.e. $\mathcal{Q}X = 2^X$ and $\mathcal{Q}f(A) = f^{-1}[A]$). As usual, T is assumed w.l.o.g. to preserve injective maps [5] and to be non-trivial, in the sense that $TX = \emptyset$ implies $X = \emptyset$.

Models of the logic are T -coalgebras $\langle X, \gamma \rangle$, where X is a non-empty set of *states* and $\gamma : X \rightarrow TX$ is the *transition* function. Given two states $x, y \in X$ (the *current* and the *remembered* state, respectively), the truth value of $\mathcal{F}(\Lambda, \mathbf{I}, @, \mathbf{A})$ -formulas is inductively defined by:

$$\begin{aligned} y, x \models_{\gamma} \text{me} &\iff x = y & y, x \models_{\gamma} \mathbf{A}\phi &\iff \forall z \in X. z, z \models_{\gamma} \phi \\ y, x \models_{\gamma} \heartsuit\phi &\iff \gamma(x) \in \llbracket \heartsuit \rrbracket_X \llbracket \phi \rrbracket_{\gamma, y} & y, x \models_{\gamma} @_{\text{me}}\phi &\iff y, y \models_{\gamma} \phi \\ y, x \models_{\gamma} \mathbf{I}.\phi &\iff x, x \models_{\gamma} \phi \end{aligned}$$

where $\llbracket \phi \rrbracket_{\gamma, x} = \{z \in X \mid x, z \models_{\gamma} \phi\}$; Boolean operations were omitted. When clear from context, we shall write simply $y, x \models \phi$ and $\llbracket \phi \rrbracket_x$.

Occurrences of *me* in ϕ that are not under the scope of an \mathbf{I} -binder are said to be *free*. A formula that contains no free occurrences of *me* is called a *sentence*.

Lemma 1. *If ϕ is a sentence, then $y, x \models_{\gamma} \phi$ iff $x, x \models_{\gamma} \phi$, for all x, y .*

Therefore, for a sentence ϕ , we may omit the remembered state and write $x \models_{\gamma} \phi$.

Example 2. 1. *Kripke semantics* is an instance of coalgebraic semantics: For $n < \omega$, the signature K_n has unary modal operators $\square_1, \square_2, \dots, \square_n$ and a countably infinite set \mathbf{P} of propositional atoms (nullary operators). As usual, \diamond_i is the dual of \square_i , i.e. $\diamond_i\phi = \neg\square_i\neg\phi$. The semantics is given by the endofunctor T defined by $TX = (\mathcal{P}X)^n \times \mathcal{P}\mathbf{P}$, equipped with predicate liftings $\llbracket \square_i \rrbracket_X(C) = \{(A_1, \dots, A_n, B) \in TX \mid A_i \subseteq C\}$ and $\llbracket p \rrbracket_X = \{(A_1, \dots, A_n, B) \in TX \mid p \in B\}$ for $p \in \mathbf{P}$. Then T -coalgebras assign to each state an n -tuple of sets of successors and a valuation for \mathbf{P} , and hence are exactly Kripke models with n relations. We shall denote these as \mathcal{P}^n -models $\langle X, \gamma, \pi \rangle$ with $\gamma : X \rightarrow (\mathcal{P}X)^n$ and $\pi : X \rightarrow \mathcal{P}\mathbf{P}$.

2. In *graded logic* [8] one has modal operators \diamond_k for $k \geq 0$, read ‘in more than k successors, it holds that’. We can interpret this logic over the functor \mathcal{B} that takes a set X to the set $\mathcal{B}X$ of multisets over X , i.e. maps $B : X \rightarrow \mathbb{N}\cup\{\infty\}$, using the predicate liftings $\llbracket \diamond_k \rrbracket_X(A) = \{B \in \mathcal{B}(X) \mid \sum_{x \in A} B(x) > k\}$. This captures the *multigraph* semantics of graded modalities [7], which in the absence of \mathbf{I} -me engenders the same notion of satisfiability as the more standard Kripke semantics of graded modalities [19].

3. *Probabilistic logic* [13] has operators L_p ‘in the next step, it holds with probability at least p ’, for $p \in [0, 1] \cap \mathbb{Q}$. Its semantics is based on the functor \mathcal{D} mapping X to the set of discrete probability distributions on X , with $\llbracket L_p \rrbracket_X(A) = \{P \in \mathcal{D}(X) \mid PA \geq p\}$. \mathcal{D} -coalgebras are Markov chains.

Other modal logics that fit in the coalgebraic paradigm include neighbourhood semantics, coalition logic, and non-monotonic conditionals \Rightarrow (‘if – then normally’), to name just a few [21].

Remark 3. One may consider variants $@_{me}^*$ and A^* of the operators $@_{me}$ and A with semantic clauses $y, x \models_\gamma @_{me}^* \phi \iff x, y \models_\gamma \phi$ and $y, x \models_\delta A^* \phi \iff \forall z \in X. y, z \models_\delta \phi$, respectively (so that $@_{me} = @_{me}^* I$ and $A = A^* I$). The operator $@_{me}^*$ is not in general expressible in $\mathcal{F}(A, I, @, A)$: one can show using the notion of bisimulation introduced in Section 4 that the formula $I.\diamond(\neg me \wedge @_{me}^* \diamond \neg me)$ is not expressible in $\mathcal{F}(K_n, I, @, A)$. In the relational case, one can however give a polynomial reduction using inverse relations (which can be defined in $\mathcal{F}(K_n, I, @)$ as shown below). Also A^* is not in general expressible in $\mathcal{F}(A, I, @, A)$: in the relational case, a *sink* can be defined by the formula $I.A^*(\neg me \rightarrow \diamond me)$; again, inexpressibility of this property in $\mathcal{F}(K_n, I, @, A)$ is shown using bisimulations. Technically, A^* has a number of undesirable properties; e.g. unlike A it cannot be reduced to TBox reasoning (Remark 4), and moreover it subverts the semantic idea behind the depth restriction.

Remark 4. In the following, we will work with TBoxes in the style of description logic instead of the A operator in full generality, i.e. we will assume that all formulas are of the form $(A\phi) \wedge \psi$ where ϕ, ψ do not contain A ; satisfiability of such a formula will be referred to as *satisfiability of ψ over (the TBox) ϕ* . Similarly, we refer to validity over ϕ , soundness over ϕ etc. Algorithmically, we can reduce the A -operator to TBox reasoning as follows. We first guess which subformulas $A\phi$ of the target formula are valid (one can do this already in NP; algorithms for A happen in EXPTIME). Given this guess, we can eliminate all occurrences of A , thus ending up with a modified target formula, a TBox, and a conjunction $\neg A\rho_1 \wedge \dots \wedge \neg A\rho_k$. As our logic is invariant under coproducts of models, it now suffices to check (separately) that the target formula and the formulas $\neg\rho_i$ are satisfiable over the TBox.

We are interested in the *modal distance* between every bound occurrence of me and its associated I . We now make this notion precise. We define sets of formulas $\mathcal{F}_i^n(A, I, @, A)$ ($0 \leq i \leq n$) intuitively containing those formulas where i) every free occurrence of me is under at most $i - 1$ modalities; and ii) there are at most n modalities between every bound me and its corresponding I :

$$\begin{aligned} \mathcal{F}_0^n(A, I, @, A) \ni \phi_0^n, \psi_0^n &::= \perp \mid p \mid \phi_0^n \rightarrow \psi_0^n \mid \heartsuit \phi_0^n \mid I.\phi_0^n \mid A\phi_0^n \\ \mathcal{F}_l^n(A, I, @, A) \ni \phi_l^n, \psi_l^n &::= me \mid \perp \mid p \mid \phi_l^n \rightarrow \psi_l^n \mid \heartsuit \phi_{l-1}^n \mid I.\phi_l^n \mid A\phi_0^n \mid @_{me} \phi_l^n \end{aligned}$$

where $l > 0$. Notice that me does not occur in $\mathcal{F}_0^n(A, I, @, A)$. According to the grammar, me cannot occur free under A , which is not a real restriction as any

Table 1. Encoding of the grid of an $\mathbb{N} \times \mathbb{N}$ tiling in $\mathcal{F}_3(K_9, \mathbf{I})$. Here $L = \{u, r, ur, ru\}$.

$$\bigwedge \left\{ \begin{array}{l} \mathbf{I}.\diamond\neg\text{me} \wedge \mathbf{I}.\square\neg\text{me} \wedge \mathbf{I}.\square\diamond\text{me} \wedge \mathbf{I}.\square\square\text{me} \\ \bigwedge_{* \in L} (\mathbf{I}.\square\square_*\diamond\text{me} \wedge \mathbf{I}.\square\blacksquare_*\diamond\text{me}) \wedge \bigwedge_{* \in L} (\square\square_*\mathbf{I}.\square\diamond\text{me} \wedge \square\blacksquare_*\mathbf{I}.\square\diamond\text{me}) \\ \bigwedge_{* \in L} (\square\mathbf{I}.\square_*\diamond_*\text{me} \wedge \square\mathbf{I}.\blacksquare_*\diamond_*\text{me} \wedge \square\mathbf{I}.\square_*\blacksquare_*\text{me} \wedge \square\mathbf{I}.\blacksquare_*\square_*\text{me}) \\ \square\mathbf{I}.\square_u\square_r\diamond_{ur}\text{me} \wedge \square\mathbf{I}.\square_r\square_u\diamond_{ru}\text{me} \\ \square\diamond_u\top \wedge \square\diamond_r\top \wedge \square\mathbf{I}.\square_{ur}\diamond_{ru}\text{me} \end{array} \right.$$

formula $\mathbf{A}\phi$ is equivalent to $\mathbf{AI}.\phi$. We will usually write $\mathcal{F}_n(\dots)$ instead of $\mathcal{F}_n^n(\dots)$ and informally refer to it as the formulas of *depth* n , which form the *depth- n fragment*. We assume w.l.o.g. that \mathbf{I} occurs only in front of modal operators.

Theorem 5. *Satisfiability is undecidable for $\mathcal{F}_4(K_1, \mathbf{I})$ and for $\mathcal{F}_3(K_9, \mathbf{I})$.*

Proof (sketch). Both claims are shown by reduction from the $\mathbb{N} \times \mathbb{N}$ tiling problem (cf. [6]) using the so-called *spypoint technique*. The proof for $\mathcal{F}_4(K_1, \mathbf{I})$ can be found in [14] (a simpler one but for $\mathcal{F}_4(K_3, \mathbf{I})$ is given in [22]). Here we show the construction for $\mathcal{F}_3(K_9, \mathbf{I})$. For the sake of clarity, we assume $K_9 = \{\square, \square_u, \square_r, \square_{ur}, \square_{ru}, \blacksquare_u, \blacksquare_r, \blacksquare_{ur}, \blacksquare_{ru}\}$; \square_u and \square_r represent moving one step, *up* or *right*, in the grid, while \square_{ur} and \square_{ru} correspond to a two-step move *up-right* and *right-up*. Each \blacksquare_* is intended to be the inverse of \square_* . We use \square to go back and forth between the points of the grid and the spypoint. The latter is the initial point of evaluation. The encoding of the grid is shown in Table 1. The first two lines define a spypoint. The next line makes \square_* and \blacksquare_* inverses and, exploiting this fact, injective functions. The last two lines make \square_{ur} and \square_{ru} composites of \square_u and \square_r , and force \square_u and \square_r to commute, respectively. \square

The stepmother example from [14] ($\mathbf{I}.\diamond_{\text{hasSpouse}}\diamond_{\text{hasChild}}\square_{\text{hasParent}}\neg\text{me}$) lives in the depth-3 fragment, while the celebrity needs only depth 2 ($\mathbf{I}.\square_{\text{meets}}\diamond_{\text{knows}}\text{me}$). Although it is possible to reduce the number of relation symbols required in the proof above (see, e.g. [14]), the decidability of $\mathcal{F}_3(K_1, \mathbf{I})$ remains open.

3 The Depth-1 Fragment in Coalgebraic Semantics

As a first step in our program, we investigate the depth-1 fragment, which allows for defining the narcissist ($\mathbf{I}.\diamond_{\text{loves}}\text{me}$) and the egotist ($\mathbf{I}.\square_{\text{loves}}.\text{me}$), but can also be applied in the coalgebraic setting to express phenomena such as usually knowing oneself on Tuesdays (if maybe not on Mondays), $\mathbf{I}.\text{Tuesday} \Rightarrow_{\text{knows}} \text{me}$ (here we index conditionals by role names) or a Markov chain staying in the current state with probability at least $2/3$ ($\mathbf{I}.\text{L}_{2/3}\text{me}$). Specifically, we prove that even at the general coalgebraic level, the depth-1 fragment has the exponential model property over general TBoxes (and hence in the presence of \mathbf{A}), with ensuing generic decidability and complexity results.

As we work in the depth-1 fragment, we can assume that every occurrence of $\heartsuit \in \mathcal{A}$ is prefixed by \mathbf{I} , and we regard $\mathbf{I}.\heartsuit$ as a single operator — consequently,

the *subformulas* of $I.\heartsuit\phi$ are $I.\heartsuit\phi$ and the subformulas of ϕ , but not $\heartsuit\phi$. We next recall some notation and previous results.

Definition 6. We use $\text{Prop}(V)$ to denote the set of Boolean expressions over a set V (of *atoms*) and put $\Lambda(V) = \{I.\heartsuit v \mid \heartsuit \in \Lambda, v \in V\}$. For any set X , an X -*substitution* for V is a function $V \rightarrow X$. Given $\alpha \in \text{Prop}(V)$ and a $\mathcal{P}(X)$ -substitution τ , $\llbracket \alpha \rrbracket_{X,\tau}$ is the extension of α in the boolean algebra $\mathcal{P}(X)$ under the interpretation τ . Similarly, for a functor T and predicate liftings $\llbracket \heartsuit \rrbracket$, we write $\llbracket \alpha \rrbracket_{TX,\tau}$, with $\alpha \in \text{Prop}(\Lambda(V))$ and τ a $\mathcal{P}(X)$ -substitution for V , to denote the extension of α in the boolean algebra $\mathcal{P}(TX)$ under the interpretation function $\lambda\heartsuit v.\llbracket \heartsuit \rrbracket_X(\tau(v))$. We say that that α is *one-step satisfiable* over τ if $\llbracket \alpha \rrbracket_{TX,\tau} \neq \emptyset$.

Definition 7. A clause over V is a disjunction of atoms in $V \cup \neg V$. A *one-step rule* over a set V is a pair ϕ/ψ where $\phi \in \text{Prop}(V)$ and ψ is a clause over $\Lambda(V)$. A one-step rule ϕ/ψ is *one-step sound* if whenever $\llbracket \phi \rrbracket_{X,\tau} = X$, then $\llbracket \psi \rrbracket_{TX,\tau} = TX$. A collection \mathcal{R} of one-step rules is *one-step complete* if whenever $\llbracket \chi \rrbracket_{TX,\tau} = TX$ with χ a clause over $\Lambda(V)$, then there exist $\phi/\psi \in \mathcal{R}$ over V and a V -substitution σ such that $\psi\sigma$ propositionally entails χ and $\llbracket \phi\sigma \rrbracket_{X,\tau} = X$.

In proofs, a one-step rule ϕ/ψ over V is applied in substituted form, i.e. conclude $\psi\sigma$ from $\phi\sigma$, where σ substitutes formulas for the variables in V . Thanks to the following result, we can assume that we have a one-step complete set \mathcal{R} of one-step sound rules.

Theorem 8 ([19]). *The set of all one-step sound rules for a functor T and its corresponding predicate liftings is one-step complete.*

We will first show that whenever ϕ is satisfiable over a TBox χ , then ϕ has an exponentially bounded model (in $|\chi| + |\phi|$) satisfying χ . To this end, we let Σ be the closure of $\{\phi, \chi\}$ under subformulas (in the above sense), negation, and prefixing with $@_{\text{me}}$, where we identify $\neg\neg\phi$ with ϕ , $\neg@_{\text{me}}\phi$ with $@_{\text{me}}\neg\phi$, and $@_{\text{me}}@_{\text{me}}\phi$ with $@_{\text{me}}\phi$. We use Σ' to denote the set of sentences in Σ . We let S_χ be the set of all maximally satisfiable subsets of Σ' containing χ .

We build a syntactic model whose domain $2S_\chi$ contains two copies of each $B \in S_\chi$. Formally, we define $2S_\chi = \{\top, \perp\} \times S_\chi$ and let $l : 2S_\chi \rightarrow S_\chi$ denote the second projection. For $A \in S_\chi$, let $S_{\chi,A}$ denote the set of maximally satisfiable subsets of Σ (which may contain free occurrences of me) extending $\{\chi\} \cup @_{\text{me}}A$, where $@_{\text{me}}A = \{@_{\text{me}}\psi \mid \psi \in A\}$. For each $(a, A) \in 2S_\chi$, we define a function $r_{(a,A)} : S_{\chi,A} \rightarrow 2S_\chi$ as $r_{(a,A)}(B) = (b, B \cap \Sigma')$ where $b = \top$ iff B entails the formula $a \leftrightarrow \text{me}$.

For $x \in 2S_\chi$, we define a valuation $\tau_x : V_\Sigma \rightarrow \mathcal{P}(2S_\chi)$ on $V_\Sigma = \{a_\rho \mid \rho \in \Sigma\}$ by $\tau_x(a_\rho) = [\rho]_x$, with $[\rho]_x$ given by the usual clauses for Boolean operators plus

$$\begin{aligned} [a]_x &= \{y \in 2S_\chi \mid a \in l(y)\} & [\text{me}]_x &= \{x\} \\ [@_{\text{me}}\rho]_x &= \{y \in 2S_\chi \mid x \in [\rho]_x\} & [I.\heartsuit\rho]_x &= \{y \in 2S_\chi \mid I.\heartsuit\rho \in l(y)\}. \end{aligned}$$

Also, define $\eta_A : V_\Sigma \rightarrow \mathcal{P}(S_{\chi,A})$ by $\eta_A(a_\rho) = \{B \in S_{\chi,A} \mid \rho \in B\}$. The following key lemma, whose proof depends crucially on the fact that for every $x \in 2S_\chi$, $r_x : S_{\chi l(x)} \rightarrow 2S_\chi$ is injective, allows us to move from the complex definition of τ_x to the simpler $\eta_{l(x)}$:

Lemma 9. For all $\rho \in \Sigma$ and $x \in 2S_\chi$, $\eta_{l(x)}(a_\rho) = r_x^{-1}[\tau_x(a_\rho)]$.

We let $\sigma_\Sigma : V_\Sigma \rightarrow \Sigma$ by $\sigma_\Sigma(a_\rho) = \rho$ and then have, similar to Lemma 27 in [19],

Lemma 10. For $A \in S_\chi$ and $\theta \in \text{Prop}(V_\Sigma)$, $[\theta]_{S_\chi, A, \eta_A} = S_{\chi, A}$ iff $@_{\text{me}} \wedge A \rightarrow \theta_{\sigma_\Sigma}$ is valid over the TBox χ ($\wedge A$ is the conjunction of all formulas in A).

Lemma 11 (Existence). There is a coherent coalgebra structure ξ on $2S_\chi$, i.e., one such that $\xi(x) \in [\heartsuit][\rho]_x$ iff $x \in [I.\heartsuit\rho]_x$ for every x and every $I.\heartsuit\rho \in \Sigma$.

The key point in the proof of the existence lemma in comparison to the base case [19] is the following lemma.

Lemma 12 (Rule relativization). If θ/ψ is a sound one-step rule, then we can soundly conclude $\rho \rightarrow \psi\sigma$ from $@_{\text{me}}\rho \rightarrow \theta\sigma$, for any substitution σ and any sentence ρ .

Lemma 13 (Truth). Let ξ be a coherent coalgebra structure on $2S_\chi$. Then $x, y \models_\xi \rho$ iff $y \in [\rho]_x$ for $\rho \in \Sigma$ and hence $x, y \models_\xi \rho'$ iff $\rho' \in l(y)$ for $\rho' \in \Sigma'$.

Theorem 14. $\mathcal{F}_1(\wedge, I, @, A)$ has the exponential model property. Moreover, if one-step satisfiability is in NP, satisfiability is NEXPTIME; when the former is in P, the latter is in EXPTIME.

Proof. The exponential model property is established by the above construction, as $2S_\chi$ has exponential size. One-step satisfiability in NP allows guessing and checking an exponential model in exponential time [19]. With one-step satisfiability in P we can use Hintikka set elimination, similarly as in [20].

Remark 15. The EXPTIME bound is tight already in the relational case. Almost all instances of coalgebraic logics, including conditional logics, alternating-time/coalition logics, and probabilistic logics, have one-step satisfiability problems in P [19], so that the EXPTIME result applies in these cases. We have little doubt that using global caching, one can show an EXPTIME bound for so-called EXPTIME-tractable instances [9], which would cover essentially all cases of interest. For graded logics (whose one-step satisfiability problem is NP-complete), we have recent results proving an EXPTIME bound even at depth 2 [10].

Remark 16. Theorem [14] generalizes to a setting with *nominals* [3], i.e. propositional symbols i denoting single states which can appear by themselves or in *satisfaction operators* $@_{i\rho}$ (ρ holds at state i). That is, the result also holds in the framework of *coalgebraic hybrid logic* [15]. Only minor modifications of the construction are required, the most notable one being that elements of S_χ that are *named*, i.e. contain a positive nominal, are not duplicated in the construction of the carrier. This is in sharp contrast to the depth-2 case, studied next, where nominals cause drastic effects even in the relational case (Remark [25]).

4 The Depth-2 Fragment in Kripke Semantics

We will next we focus on the depth-2 fragment under relational semantics, i.e., $\mathcal{F}_2(K_n, I, @, A)$ and some of its fragments. Syntactically, we view this as a logic with two types of modal operators, $I.\Box_i$ and \Box_i ; by the depth-2 restriction, no \Box_i can occur directly under another \Box_j . Our main result is decidability in EXPTIME (even in PSPACE without A). This contrasts with Theorem 5 and, more generally, with the robust undecidability of $\mathcal{F}(K_n, I)$ (Section 1).

A detailed observation of the proof of Theorem 5 shows that $\mathcal{F}_2(K_n, I, @, A)$ is quite expressive: only two formulas are needed outside this fragment. In particular, Table 1 illustrates that in depth-2, one can define inverse and functional relations, and similarly one can define reflexive and symmetric relations. As another example, consider the formula $\gamma = \Diamond \top \wedge I.\Box(-me \wedge \Diamond me) \wedge \Box I.\Box \Diamond \neg me$. It is not hard to see that $x \models \gamma$ implies that x has at least two successors, i.e. satisfies the formula $\Diamond^{>1} \top$, where $\Diamond^{>1}$ is a *graded modality* (cf. Example 2). We can delineate the expressive power of $\mathcal{F}_2(K_n, I, @, A)$ using a notion of bisimulation:

Definition 17. Let $\langle X, \gamma, \pi \rangle$ and $\langle Y, \delta, \tau \rangle$ be \mathcal{P}^n -models. Then $Z_1, Z_2 \subseteq X^2 \times Y^2$ constitute an $I@A_2$ -bisimulation whenever they satisfy all the conditions in Table 2. The weaker notions of $I@_2$ -, IA_2 - and I_2 -bisimulations are obtained by ignoring clauses $\{A_l, A_r\}$, $\{@\}$ and $\{A_l, A_r, @\}$, respectively.

Proposition 18. Let $\langle X, \gamma, \pi \rangle$ and $\langle Y, \delta, \tau \rangle$ be two \mathcal{P}^n -models such that relations $Z_1, Z_2 \subseteq X^2 \times Y^2$ constitute an $I@A_2$ -bisimulation between them. Then:

1. $(x_0, x_1)Z_1(y_0, y_1) \implies (x_0, x_1 \models_\gamma^\pi \phi \iff y_0, y_1 \models_\delta^\tau \phi) \forall \phi \in \mathcal{F}_1(K_n, I, @, A)$
2. $(x_0, x_1)Z_2(y_0, y_1) \implies (x_0, x_1 \models_\gamma^\pi \phi \iff v_0, v_1 \models_\delta^\tau \phi) \forall \phi \in \mathcal{F}_2(K_n, I, @, A)$

Analogous results hold for $I@_2$ -, IA_2 - and I_2 -bisimulations.

Example 19. Formula γ above shows that $\mathcal{F}_2(K_n, I)$ can express some cardinality requirements on successors. We make this expressivity more precise by showing that it cannot distinguish structures with two and three successors. Consider two \mathcal{P} -models $\langle X, \gamma, \pi \rangle$ and $\langle A, \delta, \tau \rangle$ where $X = \{x, y_0, y_1\}$; $A = \{a, b_0, b_1, b_2\}$; $\pi(p) = \tau(p) = \emptyset$ for all proposition p ; and γ and δ are such that

Table 2. Conditions that define a $IA@_2$ -bisimulation. Z_* stands for both Z_1 and Z_2 .

(p)	$(x_0, x_1)Z_*(y_0, y_1) \implies (x_1 \in \pi(p) \iff y_1 \in \tau(p))$, for each proposition p
(me)	$(x_0, x_1)Z_*(y_0, y_1) \implies (x_0 = x_1 \iff y_0 = y_1)$
(I. \Box_l)	$(x_0, x_1)Z_1(y_0, y_1)$ and $x_1 R_i x_2 \implies \exists y_2$ st. $y_1 R_i y_2$ and $(x_1, x_2)Z_*(y_1, y_2)$
(\Box_l)	$(x_0, x_1)Z_2(y_0, y_1)$ and $x_1 R_i x_2 \implies \exists y_2$ st. $y_1 R_i y_2$ and $(x_0, x_2)Z_1(y_0, y_2)$
(I. \Box_r)	$(x_0, x_1)Z_1(y_0, y_1)$ and $y_1 R_i y_2 \implies \exists x_2$ st. $x_1 R_i x_2$ and $(x_1, x_2)Z_*(y_1, y_2)$
(\Box_r)	$(x_0, x_1)Z_2(y_0, y_1)$ and $y_1 R_i y_2 \implies \exists w_2$ st. $x_1 R_i x_2$ and $(x_0, x_2)Z_1(y_0, y_2)$
(@)	$(x_0, x_1)Z_*(y_0, y_0) \implies (x_0, x_0)Z_*(y_0, y_0)$
(A _l)	$\forall x \exists y$ st. $(x, x)Z_*(y, y)$
(A _r)	$\forall y \exists x$ st. $(x, x)Z_*(y, y)$

$R_\gamma = \{(x, y_i) \mid 0 \leq i \leq 1\}$ and $R_\delta = \{(a, b_i) \mid 0 \leq i \leq 2\}$. It is not hard to verify that $Z_1 = Z_2 = \{(s, t) \in X^2 \times A^2 \mid \forall 0 \leq i \leq 1, \pi_i(s) = a \iff \pi_i(t) = x\}$ constitute an I_2 -bisimulation.

4.1 $\mathcal{F}_2(K_n, I)$ Is PSPACE-Complete

Using Proposition 18 we will show that $\mathcal{F}_2(K_n, I)$ possesses a variant of the tree-model property: every satisfiable formula has a model that is contained in the reflexive-symmetric closure of a tree. Moreover these quasi-tree-models can be shown to be shallow, i.e., have a depth polynomial on the size of the formula. As usual, this is key in deriving a decision procedure for satisfiability that runs in polynomial space.

Definition 20. We say that $S \subseteq X^2$ is a *quasi-tree with root r* if for some tree R with root r , $R \subseteq S \subseteq R \cup R^{-1} \cup \text{Id}_X$. A \mathcal{P}^n model $\langle X, \gamma, \pi \rangle$ is a quasi-tree-model with root r when $\bigcup_i R_{\gamma,i}$ is a quasi-tree with root r .

Theorem 21 (Quasi-tree model property). *If $\phi \in \mathcal{F}_2(K_n, I)$ is a satisfiable sentence then there exists a quasi-tree model that satisfies ϕ at its root.*

Proof (sketch). By Lemma 11 let \mathcal{X} be a model with domain X such that $r, r \models_\gamma^\pi \phi$, for some $r \in X$. One builds an unravelling \mathcal{A} of \mathcal{X} such that it contains all the paths in \mathcal{X} from r where no self-loop is ever taken and such that one never immediately returns to the preceding state, regardless of the relation used. This unravelling gives us a quasi-tree and one then shows it to be bisimilar \mathcal{X} .

We are now ready to introduce the announced tableau construction. In this context, we will restrict our attention to $\mathcal{F}_2(K_n, I)$ -formulas in nnf subject to the following restriction: if me occurs *free* in ϕ , then it does so *under the scope of some modality*. Notice that for such a formula $\Box_i \phi$, ϕ need not satisfy this condition. We solve this by defining, for an *arbitrary* formula ϕ , $\phi(\top)$ and $\phi(\perp)$ as the result of replacing *every free occurrence of me that is not under a modality* by \top and \perp , respectively. E.g., for $\phi = me \wedge \Box_i me$, $\phi(\perp) = \perp \wedge \Box_i me$; the former does not satisfy the restriction above, the latter does.

Given a set of formulas Σ , its closure $Cl(\Sigma)$, is the smallest set that is closed under nnf negation and *pseudo-subformulas*, that is, if $\phi \in Cl(\Sigma)$ and ψ is a subformula of ϕ , then $\psi(\top)$ and $\psi(\perp)$ are in $Cl(\Sigma)$ as well. $Cl^*(\Sigma)$ denotes the set of sentences in $Cl(\Sigma)$.

Let ϕ be a $\mathcal{F}_2(K_n, I)$ -sentence. A *tableau for ϕ* is then a labelled quasi-tree $\mathcal{T} = \langle T, R, \cdot^\mathcal{T} \rangle$ with root r , where each directed edge $(s, t) \in R$ is labelled with a *non-empty* set of relation indices $(s, t)^\mathcal{T} \subseteq \{1 \dots n\}$ (we write sR_it if $(s, t) \in \mathcal{T}$ and $i \in (s, t)^\mathcal{T}$) and each node is labelled with a tuple $s^\mathcal{T} = \langle H_*, H_{**}, H_{f*}, H_{*f} \rangle$ where H_* is a Hintikka set over (some subset of) $Cl^*(\{\phi\})$ and all the others are Hintikka sets over (subsets of) $Cl(\{\phi\})$ (for an $s \in T$, we will denote these sets by $s_*^\mathcal{T}$, $s_{**}^\mathcal{T}$, $s_{f*}^\mathcal{T}$ and $s_{*f}^\mathcal{T}$; or even s_* , s_{**} , s_{f*} and s_{*f} if \mathcal{T} is clear from context).

Intuitively, $s_*^\mathcal{T}$ contains sentences that hold at s (where the remembered state is irrelevant); $s_{**}^\mathcal{T}$ contains formulas that hold at s when the remembered state is

Table 3. Conditions for a tableau \mathcal{T} for ϕ with root r . For $s \in T$, f_s is the father of s (if $s \neq r$); and $t_1, \dots, t_k \in T$ are the children of s . We use t to designate some existentially quantified children of s . Also $L = \{*, **, f*, *f\}$.

1. $\phi \in r_*$ and $r_{f_*} = r_{*f} = \emptyset$.
2. $\bigcup_{x \in L} s_x \subseteq Cl(\{\phi\})$ and $rank(\bigcup_{x \in L} s_x) > rank(\bigcup_{x \in L} t_{j_x})$ for all j .
3. For a sentence ψ , if $\psi \in s_{**} \cup s_{*f}$ or $\psi \in t_{*f}$, then $\psi \in s_*$.
4. If $\Diamond_i \psi \in s_*$ or $I.\Diamond_i \psi \in s_*$, then *some* of the following must hold:
 - i) $sR_i f_s, \psi(\perp) \in s_{*f}$; ii) $sR_i s, \psi(\top) \in s_{**}$; iii) $sR_i t, \psi(\perp) \in t_{*f}$.
5. If $\Box_i \psi \in s_*$ or $I.\Box_i \psi \in s_*$, then *all* of the following must hold:
 - i) $sR_i f_s \implies \psi(\perp) \in s_{*f}$; ii) $sR_i s \implies \psi(\top) \in s_{**}$; iii) $sR_i t \implies \psi(\perp) \in t_{*f}$.
6. If $\Diamond_i \psi \in s_{**}$ (with me free in ψ), then *some* of the following must hold:
 - i) $sR_i f_s, \psi(\perp) \in f_{s_*}$; ii) $sR_i s, \psi(\top) \in s_*$; iii) $sR_i t, \psi(\perp) \in t_*$.
7. If $\Box_i \psi \in s_{**}$ (with me free in ψ), then *all* of the following must hold:
 - i) $sR_i f_s \implies \psi(\perp) \in f_{s_*}$; ii) $sR_i s \implies \psi(\top) \in s_*$; iii) $sR_i t \implies \psi(\perp) \in t_*$.
8. If $\Diamond_i \psi \in s_{*f}$ (with me free in ψ), then *some* of the following must hold:
 - i) $sR_i f_s, \psi(\top) \in f_{s_*}$; ii) $sR_i s, \psi(\perp) \in s_*$; iii) $sR_i t, \psi(\perp) \in t_*$.
9. If $\Box_i \psi \in s_{*f}$ (with me free in ψ), then *all* of the following must hold:
 - i) $sR_i f_s \implies \psi(\top) \in f_{s_*}$; ii) $sR_i s \implies \psi(\perp) \in s_*$; iii) $sR_i t \implies \psi(\perp) \in t_*$.
10. If $\Diamond_i \psi \in t_{j_{*f}}$ (with me free in ψ), then *some* of the following must hold:
 - i) $sR_i f_s, \psi(\perp) \in f_{s_*}$; ii) $sR_i s, \psi(\perp) \in s_*$;
 - iii) $sR_i t_j, \psi(\top) \in t_{j_*}$; iv) $sR_i t_k, j \neq k, \psi(\perp) \in t_{k_*}$.
11. If $\Box_i \psi \in t_{j_{*f}}$ (with me free in ψ), then *all* of the following must hold:
 - i) $sR_i f_s \implies \psi(\perp) \in f_{s_*}$; ii) $sR_i s \implies \psi(\perp) \in s_*$;
 - iii) $sR_i t_j \implies \psi(\top) \in t_{j_*}$; iv) $sR_i t_k$ with $j \neq k \implies \psi(\perp) \in t_{k_*}$.

$s, s_{f_*}^{\mathcal{T}}$ the ones that hold when the remembered state is the father of s and $s_{*f}^{\mathcal{T}}$ the formulas that hold *at the father of s* when the remembered state is s . This intuition suggests the conditions in Table 3. Notice that conditions 7–11 heavily use the fact that, because I is assumed to occur always in front of a modality, me cannot occur free under a modality in ψ .

Lemma 22. *There exists a tableau for a sentence ϕ iff ϕ is satisfiable.*

Lemma 23. *Let ϕ be a satisfiable sentence. Every tableau for ϕ has depth bounded by $rank(\phi)$. Moreover, ϕ has a tableau with breadth polynomial in $|\phi|$.*

Theorem 24. *The satisfiability problem for $\mathcal{F}_2(K_n, I)$ is PSPACE-complete.*

Remark 25. Unlike the depth-1 case (Remark 16) this result does not hold in the presence of nominals, were we can form a spypoint using the formula: $s \wedge \Box(\Diamond s \wedge \bigwedge_i \Box_i I.\Diamond(s \wedge \Diamond me))$, internalizing global reasoning. This gives us the possibility of defining functions and inverses, which in combination with nominals cause NEXPTIME-hardness and break the finite model property.

4.2 $\mathcal{F}_2(K_n, I, @)$ Is PSPACE-Complete

It is not hard to see that $\mathcal{F}_2(K_n, I, @)$ is as expressive as $\mathcal{F}_2(K_n, I)$, although perhaps more succinctly so. Indeed, as an example, consider the form

$\Box_i I. \Box_j (p \vee \Box_k (@_{me} \Diamond_l me \rightarrow p))$). A trivial transformation leads to the equivalent formula $\Box_i I. ((\Diamond_l me \wedge \Box_j (p \vee \Box_k (\top \rightarrow p))) \vee (\neg \Diamond_l me \wedge \Box_j (p \vee \Box_k (\perp \rightarrow p))))$. It is not hard to generalize this example into a truth-preserving translation from $\mathcal{F}_2(K_n, I, @)$ to $\mathcal{F}_2(K_n, I)$ but with an exponential blow-up in size.

Lemma 26. *There exists a polynomial, satisfiability preserving translation from $\mathcal{F}_2(K_n, I, @)$ to $\mathcal{F}_2(K_n, I)$.*

Proof. We use the standard technique of replacing subformulas by fresh propositional symbols. We illustrate the idea translating the example used above and leave the formal details to the reader. The translated formula we obtain in this case is $\Box_i I. (\Diamond_l me \leftrightarrow q \wedge (q \rightarrow \Box_j \Box_k q) \wedge (\neg q \rightarrow \Box_j \Box_k \neg q) \wedge \Box_j (p \vee \Box_k (q \rightarrow p)))$, where q is fresh. Notice that formulas $(\neg)q \rightarrow \Box_j \Box_k (\neg)q$ are used to propagate the value of q to the modal context in which the $@$ -formula originally occurred.

Theorem 27. *Satisfiability for $\mathcal{F}_2(K_n, I, @)$ is PSPACE-complete.*

4.3 $\mathcal{F}_2(K_n, I, @, A)$ Is EXPTIME-Complete

We begin by observing that the proof of Lemma 26 can be used to show also that $\mathcal{F}_2(K_n, I, @, A)$ is polynomially reducible to $\mathcal{F}_2(K_n, I, A)$, so the latter is the logic we will consider. Moreover, we reduce A to TBoxes (Remark 4).

Now, it is not hard to adapt conditions 2 in Table 3 in order to obtain a tableau for $\mathcal{F}_2(K_n, I, A)$ such that an analogue of Lemma 22 holds. However, we will not be able to give a bound for the depth of this tableau. In fact, since $\mathcal{F}_2(K_n, I, A)$ encodes inverse functional roles, standard examples show

Theorem 28. *$\mathcal{F}_2(K_2, I, A)$ lacks the finite model property.*

We will show that we can nevertheless effectively decide existence of infinite tableaux by encoding the problem in the description logic $\mathcal{ALCHI}Q$, whose satisfiability problem with respect to general TBoxes is EXPTIME-complete [12]. As a modal language (i.e. avoiding the more common DL syntax), $\mathcal{ALCHI}Q$ contains graded modalities $\Diamond_i^{>n}$ and $\Box_i^{\leq n}$ and inverse modalities (denoted by \blacklozenge_i and \blacksquare_i). Additionally it features *role inclusion axioms* $R_i \sqsubseteq R_j$, which roughly correspond to $AI.\Box_i \blacklozenge_j me$. For an introduction to description logics, see [4].

Remark 29. In terms of descriptive power, $\mathcal{ALCHI}Q$ is not more expressive than $\mathcal{F}_2(K_n, I)$; for instance $\Box I. \Diamond me$ has no $\mathcal{ALCHI}Q$ equivalent.

Assume we are given a $\mathcal{F}_2(K_n, I)$ -formula ϕ and a general TBox χ . The encoding into $\mathcal{ALCHI}Q$ will use relation symbols R_1, \dots, R_n and R_f , with role inclusion axioms $R_i^{-1} \sqsubseteq R_f$ for $1 \leq i \leq n$ (where R_i^{-1} denotes the inverse relation of R_i). Moreover, we impose a TBox axiom $\Box_f^{\leq 1} \top$, so that R_f is interpreted as a partial function. This essentially means that we work with tree-models, where R_f represents the “father-of” relation. Notice that a node may reach its children by more than one R_i . Since models will be proper trees, we will need to make explicit provisions for quasi-trees in the encoding.

Table 4. Encoding of conditions [3](#)–[11](#) in Table [3](#) as global axioms in $\mathcal{ALC}HIQ$

3. $\bigwedge_{\psi \in \Sigma_0} (\rho_{**:\psi} \rightarrow \rho_{*:\psi}) \wedge (\rho_{f*:\psi} \rightarrow \rho_{*:\psi}) \wedge (\rho_{*f:\psi} \rightarrow \diamond_f \rho_{*:\psi})$
4. $\bigwedge_{(I.)\diamond_i \psi \in \Sigma_0} \rho_{*:(I.)\diamond_i \psi} \rightarrow ((\uparrow_i \wedge \rho_{*f:\psi(\perp)}) \vee (\odot_i \wedge \rho_{**:\psi(\top)}) \vee \diamond_i \rho_{f*:\psi(\perp)})$
5. $\bigwedge_{(I.)\square_i \psi \in \Sigma_0} \rho_{*:(I.)\square_i \psi} \rightarrow ((\uparrow_i \rightarrow \rho_{*f:\psi(\perp)}) \wedge (\odot_i \rightarrow \rho_{**:\psi(\top)}) \wedge \square_i \rho_{f*:\psi(\perp)})$
6. $\bigwedge_{\diamond_i \psi \in \Sigma_1} \rho_{**:\diamond_i \psi} \rightarrow ((\uparrow_i \wedge \square_f \rho_{*:\psi(\perp)}) \vee (\odot_i \wedge \rho_{*:\psi(\top)}) \vee \diamond_i \rho_{*:\psi(\perp)})$
7. $\bigwedge_{\square_i \psi \in \Sigma_1} \rho_{**:\square_i \psi} \rightarrow ((\uparrow_i \rightarrow \square_f \rho_{*:\psi(\perp)}) \wedge (\odot_i \rightarrow \rho_{*:\psi(\top)}) \wedge \square_i \rho_{*:\psi(\perp)})$
8. $\bigwedge_{\diamond_i \psi \in \Sigma_1} \rho_{f*:\diamond_i \psi} \rightarrow ((\uparrow_i \wedge \square_f \rho_{*:\psi(\top)}) \vee (\odot_i \wedge \rho_{*:\psi(\perp)}) \vee \diamond_i \rho_{*:\psi(\perp)})$
9. $\bigwedge_{\square_i \psi \in \Sigma_1} \rho_{f*:\square_i \psi} \rightarrow ((\uparrow_i \rightarrow \square_f \rho_{*:\psi(\top)}) \wedge (\odot_i \rightarrow \rho_{*:\psi(\perp)}) \wedge \square_i \rho_{*:\psi(\perp)})$
10. $\bigwedge_{\diamond_i \psi \in \Sigma_1} \rho_{*f:\diamond_i \psi} \rightarrow \left(\begin{array}{l} \diamond_f (\uparrow_i \wedge \rho_{*:\psi(\perp)}) \vee \diamond_f (\odot_i \wedge \rho_{*:\psi(\perp)}) \vee (\blacklozenge_i \top \wedge \rho_{*:\psi(\top)}) \\ \vee (\rho_{*:\neg\psi(\perp)} \wedge \diamond_f \diamond_i \rho_{*:\psi(\perp)}) \vee (\rho_{*:\psi(\perp)} \wedge \diamond_f \diamond_i^{>1} \rho_{*:\psi(\perp)}) \end{array} \right)$
11. $\bigwedge_{\square_i \psi \in \Sigma_1} \rho_{*f:\square_i \psi} \rightarrow \left(\begin{array}{l} \diamond_f (\uparrow_i \rightarrow \square_f \rho_{*:\psi(\perp)}) \wedge \diamond_f (\odot_i \rightarrow \rho_{*:\psi(\perp)}) \\ \wedge (\blacksquare_i \perp \rightarrow \diamond_f \square_i \rho_{*:\psi(\perp)}) \\ \wedge \blacklozenge_i \top \rightarrow \left(\rho_{*:\psi(\top)} \wedge \left(\rho_{*:\psi(\perp)} \rightarrow \diamond_f \square_i \rho_{*:\psi(\perp)} \wedge \right) \right) \\ \left(\rho_{*:\neg\psi(\perp)} \rightarrow \diamond_f \square_i^{\leq 1} \rho_{*:\psi(\perp)} \right) \end{array} \right)$

Let $L = \{*, **, f*, *f\}$, $\Sigma = Cl(\{\chi, \psi\})$, $\Sigma_0 = Cl^*(\{\chi, \psi\})$ and $\Sigma_1 = \Sigma \setminus \Sigma_0$. We use the set $V = \{\rho_{l:\psi} \mid l \in L, \psi \in \Sigma\} \cup \{\odot_i \mid 1 \leq i \leq n\} \cup \{\uparrow_i \mid 1 \leq i \leq n\}$ of proposition symbols. Intuitively we want $\rho_{l:\psi}$ to hold at a state s of a (quasi-)tree if $\psi \in s_l$. Moreover \odot_i and \uparrow_i are used to denote that state s has a self-loop or reaches his father, respectively.

Since we want χ to hold globally, we impose the TBox axiom $\rho_{*:\chi}$. Also, we require nodes to be labelled with Hintikka sets by means of the TBox axiom $\bigwedge_{l \in L} (\neg \rho_{l:\perp} \wedge \bigwedge_{\psi \in \Sigma} \neg(\rho_{l:\psi} \wedge \rho_{l:\neg\psi}) \wedge \bigwedge_{\psi_1 * \psi_2 \in \Sigma} (\rho_{l:\psi_1 * \psi_2} \rightarrow \rho_{l:\psi_1} * \rho_{l:\psi_2}))$.

To ensure a correct distinction between the root and non-root nodes, we impose TBox axioms $\uparrow_i \rightarrow \diamond_f \top$ which state that \uparrow_i can only hold at non-root nodes. Moreover, conditions [3](#) to [11](#) are encoded by the TBox axioms shown in Table [4](#). The encoding is straightforward, except for conditions [10](#) and [11](#), since $\mathcal{ALC}HIQ$ provides no direct way of expressing the requirement $t_j \neq t_k$ (cf. Table [3](#)). In the case of condition [10](#), for instance, we overcome this by splitting into cases: if $t_j \not\models \psi(\perp)$, then any t_k that satisfies $\psi(\perp)$ will be different from t_i ; otherwise, we use the graded modality $\diamond^{>1}$ to ensure another one exists.

Existence of a tableau for $\mathcal{A}\chi \wedge \phi$ then amounts to the satisfiability, over the TBox described above, of the formula $\rho_{*\phi} \wedge \square_f \perp \wedge \bigwedge_{\psi \in \Sigma_0} (\neg \rho_{f*:\psi} \wedge \neg \rho_{*f:\psi})$.

Theorem 30. *Satisfiability for $\mathcal{F}_2(K_n, I, @, A)$ is EXPTIME-complete.*

5 Embedding the Guarded Fragment

Modal correspondence theory studies the relation between modal logic and classical logics, most notably, first-order logic (cf. [23](#)). The link is typically established through the *modal correspondence language*: a first-order language with only one-place and two-place relation symbols; the former stand for proposition

symbols on the modal side, the latter for (relational) modalities. Kripke models can then be seen as relational models, and well-known translations allow embedding many modal logics into first-order logic (FO).

The *guarded fragment* of FO [11], characterized by its *guarded quantification* pattern, generalizes the modal fragment of FO. It enjoys good computational properties: both the finite model property and a form of tree-model property, and an EXPTIME-complete satisfiability problem (under bounded arity) [11].

We will see that the guarded fragment over the modal correspondence language can be almost perfectly (i.e. using fresh relation symbols whose interpretation is uniquely determined by that of the old symbols) embedded into $\mathcal{F}_2(K_n, I, @, A)$. While they agree on the complexity of satisfiability, because the latter lacks the finite model property (Theorem 28), the containment is proper.

We need to make precise what we mean by an almost perfect embedding. For $K_n = \{\square_1, \dots, \square_n\}$, let *tense* $\mathcal{F}_2(K_n, I, @, A)$ to be the logic $\mathcal{F}_2(K_{2n}, I, @, A)$, with $K_{2n} = \{\square_1, \dots, \square_n, \blacksquare_1, \dots, \blacksquare_n\}$, restricted to the class of models that satisfy the formula $A \wedge_{i \leq n} (I.\square_i \blacklozenge_i me \wedge I.\blacksquare_i \blacklozenge_i me)$, i.e. those where \square_i and \blacksquare_i are interpreted as inverse relations. There is a bijection between models for $\mathcal{F}_2(K_n, I, @, A)$ and models for *tense* $\mathcal{F}_2(K_n, I, @, A)$.

Theorem 31. *Given a guarded formula $\alpha(x_1, x_2)$ in the correspondence language for K_n , there is a ϕ in *tense* $\mathcal{F}_2(K_n, I, @, A)$ such that for every \mathcal{C} , $\mathcal{C} \models_{fo} \alpha[x_1 \mapsto c_1, x_2 \mapsto c_2]$ iff $c_1, c_2 \models \phi$.*

Proof. First, observe that every guarded formula with two free variables over the correspondence language is equivalent to a guarded formula of FO2. Moreover, the set of guarded formulas of FO2 can be split into four overlapping sets G^\emptyset , G^{x_1} , G^{x_2} and $G^{x_1x_2}$, such that $\alpha \in G^\star$ iff the free variables of α are among \star . These sets can be described by simple, mutually recursive grammars, as shown in Table 5, where x ranges over $\{x_1, x_2\}$, $\dot{x}_1 = x_2$ and $\dot{x}_2 = x_1$. Finally, Table 6 exhibits translation functions (Booleans omitted) for each of these sets. An induction over $\alpha \in G^\star$ shows that $\mathcal{C} \models_{fo} \alpha[\dot{x} \mapsto c_1, x \mapsto c_2]$ iff $c_1, c_2 \models T_{\dot{x}x}^\star(\alpha)$. □

Remark 32. One point to observe about Table 6 is that modal operators appear without a preceding I only in formulas of the form $\blacklozenge_i me$ and $\blacklozenge_i me$; contrastingly, $\mathcal{F}_2(K_n, I)$ would also allow formulas such as $\blacklozenge_i(\neg me \wedge \phi)$. Indeed one can show that the guarded fragment is essentially the extension of $\mathcal{F}_1(K_n, I, @, A)$ with

Table 5. The guarded fragment of FO2 is the union of G^\emptyset , G^{x_1} , G^{x_2} and G^{x_1, x_2}

$G^\emptyset \ni \alpha, \beta$	$G^x \ni \alpha(x), \beta(x)$	$G^{x\dot{x}} \ni \alpha(x, \dot{x}), \beta(x, \dot{x})$
$\neg\alpha \mid \alpha \vee \beta$	$x = x \mid Px \mid Rxx \mid \alpha$	$x = \dot{x} \mid Rxx \mid R\dot{x}x$
$\exists x.x = x \wedge \alpha(x)$	$\neg\alpha(x) \mid \alpha(x) \vee \beta(x)$	$\alpha \mid \alpha(x) \mid \alpha(\dot{x})$
$\exists x.Px \wedge \alpha(x)$	$\exists \dot{x}.x = \dot{x} \wedge \alpha(x, \dot{x})$	$\neg\alpha(x, \dot{x})$
$\exists x.Rxx \wedge \alpha(x)$	$\exists \dot{x}.Rxx \wedge \alpha(x, \dot{x})$	$\alpha(x, \dot{x}) \vee \beta(x, \dot{x})$
$\exists x\exists \dot{x}.R\dot{x}x \wedge \alpha(x, \dot{x})$	$\exists \dot{x}.R\dot{x}x \wedge \alpha(x, \dot{x})$	

Table 6. Translations from the guarded fragment of FO2 to *tense* $\mathcal{F}_2(K_n, I, @, A)$

$T_{\dot{x}\dot{x}}^x(x = x) = \top$ $T_{\dot{x}\dot{x}}^x(Px) = p$ $T_{\dot{x}\dot{x}}^x(R_i x x) = I.\diamond_i \text{me}$ $T_{\dot{x}\dot{x}}^x(\alpha) = T_{\dot{x}\dot{x}}^\emptyset(\alpha), \text{ if } \alpha \in G^\emptyset$ $T_{\dot{x}\dot{x}}^x(\exists \dot{x}.x = \dot{x} \wedge \alpha) = I.T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^x(\exists \dot{x}.R_i x \dot{x} \wedge \alpha) = I.\diamond_i T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^x(\exists \dot{x}.R_i x \dot{x} \wedge \alpha) = I.\blacklozenge_i T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^\emptyset(\exists x.x = x \wedge \alpha) = \text{EI}.T_{\dot{x}\dot{x}}^x(\alpha)$ $T_{\dot{x}\dot{x}}^\emptyset(\exists x.Px \wedge \alpha) = \text{EI}.(p \wedge T_{\dot{x}\dot{x}}^x(\alpha))$ $T_{\dot{x}\dot{x}}^\emptyset(\exists x.R_i x x \wedge \alpha) = \text{E}(I.\diamond_i \text{me} \wedge I.T_{\dot{x}\dot{x}}^x(\alpha))$ $T_{\dot{x}\dot{x}}^\emptyset(\exists x \exists \dot{x}.R_i x \dot{x} \wedge \alpha) = \text{EI}.\diamond_i T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$	$T_{\dot{x}\dot{x}}^{\dot{x}}(\dot{x} = \dot{x}) = \top$ $T_{\dot{x}\dot{x}}^{\dot{x}}(P\dot{x}) = @_{\text{me}} p$ $T_{\dot{x}\dot{x}}^{\dot{x}}(R_i \dot{x} \dot{x}) = @_{\text{me}} I.\diamond_i \text{me}$ $T_{\dot{x}\dot{x}}^{\dot{x}}(\alpha) = T_{\dot{x}\dot{x}}^\emptyset(\alpha), \text{ if } \alpha \in G^\emptyset$ $T_{\dot{x}\dot{x}}^{\dot{x}}(\exists x.x = \dot{x} \wedge \alpha) = @_{\text{me}} T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^{\dot{x}}(\exists x.R_i x \dot{x} \wedge \alpha) = @_{\text{me}} I.\diamond_i T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^{\dot{x}}(\exists x.R_i x \dot{x} \wedge \alpha) = @_{\text{me}} I.\blacklozenge_i T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha)$ $T_{\dot{x}\dot{x}}^{x_1 x_2}(x = \dot{x}) = \text{me}$ $T_{\dot{x}\dot{x}}^{x_1 x_2}(R_i \dot{x} x) = \diamond_i \text{me}$ $T_{\dot{x}\dot{x}}^{x_1 x_2}(R_i x \dot{x}) = \blacklozenge_i \text{me}$ $T_{\dot{x}\dot{x}}^{x_1 x_2}(\alpha) = T_{\dot{x}\dot{x}}^*(\alpha) \ (\alpha \in G^*)$
--	---

$\diamond_i \text{me}$ (noting that also the definition of inverses as in Table 1 needs only $\diamond_i \text{me}$). Thus, one might call the guarded fragment the ‘depth-1.5’ fragment — use of me is unrestricted at depth 1, and limited to positive occurrences at depth 2 when \diamond is taken as the basic modal operator. In fact, formulas of the form $\diamond_i \text{me}$ can be encoded in the guarded 2-variable fragment with counting quantifiers (which is also known to be in EXPTIME [17]), while it does not seem easily possible to encode formulas of the more general form $\diamond_i(\text{me} \wedge \phi)$.

6 Conclusion

Modal logics extended with the I-me operators (or alternatively, \downarrow with only one variable) are robustly undecidable. In this paper we have identified decidable fragments, based on the modal distance (depth) between the binder I and the bound variable me . We have shown that already for depth 3 the logic becomes undecidable (previous undecidability proofs required depth 4). However, for depth less than 3 we obtain well-behaved logics with a relatively high descriptive power. Indeed, for depth 2 we arrive at a logic that is EXPTIME-complete and strictly more expressive than the (constant-free) guarded fragment of the correspondence language. When restricted to local satisfiability (i.e., without TBoxes nor a universal modality), the problem was shown to be PSPACE-complete.

For depth 1 we obtained a very general result: coalgebraic modal logics extended in this way have an exponential model property, even with general TBoxes and nominals. Generalizing our results for depth 2 to the coalgebraic case is the subject of ongoing work; recent results [10] show that they do extend to the graded case. Unlike for depth 1, nominals cannot be added to the depth-2 logic without losing most of the good properties even in the relational case: even one nominal makes the logic NEXPTIME-hard and causes infinite branching. We conjecture that the depth-2 logic with nominals is NEXPTIME-complete.

References

1. Andréka, H., van Benthem, J., Németi, I.: Modal languages and bounded fragments of predicate logic. *J. Phil. Log.* 27(3), 217–274 (1998)
2. Areces, C., Figueira, D., Figueira, S., Mera, S.: The expressive power of memory logics. *Rev. Symb. Log.* 2, 290–318 (2011)
3. Areces, C., ten Cate, B.: Hybrid logics. In: *Handbook of Modal Logics*. Elsevier (2006)
4. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): *The Description Logic Handbook*. Cambridge University Press (2003)
5. Barr, M.: Terminal coalgebras in well-founded set theory. *Theoret. Comput. Sci.* 114, 299–315 (1993)
6. Börger, E., Grädel, E., Gurevich, Y.: *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, Heidelberg (1997)
7. D’Agostino, G., Visser, A.: Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Log.* 41, 267–298 (2002)
8. Fine, K.: In so many possible worlds. *Notre Dame J. Form. Log.* 13, 516–520 (1972)
9. Goré, R., Kupke, C., Pattinson, D.: Optimal Tableau Algorithms for Coalgebraic Logics. In: Esparza, J., Majumdar, R. (eds.) *TACAS 2010*. LNCS, vol. 6015, pp. 114–128. Springer, Heidelberg (2010)
10. Gorín, D., Schröder, L.: Celebrities don’t follow their followers: Binding and qualified number restrictions. Technical report, DFKI GmbH (2011)
11. Grädel, E.: On the restraining power of guards. *J. Symb. Log.* 64 (1999)
12. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. *J. Log. Comput.* 9, 385–410 (1999)
13. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. *Inform. Comput.* 94(1), 1–28 (1991)
14. Marx, M.: Narcissists, stepmothers and spies. In: *International Workshop on Description Logics, CEUR Workshop Proceedings (2002)*, CEUR-WS.org
15. Myers, R., Pattinson, D., Schröder, L.: Coalgebraic Hybrid Logic. In: de Alfaro, L. (ed.) *FOSSACS 2009*. LNCS, vol. 5504, pp. 137–151. Springer, Heidelberg (2009)
16. Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.* 309, 177–193 (2003)
17. Pratt-Hartmann, I.: Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.* 17(1), 133–155 (2007)
18. Schneider, T.: *The Complexity of Hybrid Logics over Restricted Classes of Frames*. PhD thesis, Univ. of Jena (2007)
19. Schröder, L.: A finite model construction for coalgebraic modal logic. *J. Log. Algebr. Prog.* 73, 97–110 (2007)
20. Schröder, L., Pattinson, D.: How many toes do I have? Parthood and number restrictions in description logics. In: *Principles of Knowledge Representation and Reasoning, KR 2008*, pp. 307–218. AAAI Press (2008)
21. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. *ACM Trans. Comput. Log.* 10, 13:1–13:33 (2009)
22. ten Cate, B., Franceschet, M.: On the Complexity of Hybrid Logics with Binders. In: Ong, L. (ed.) *CSL 2005*. LNCS, vol. 3634, pp. 339–354. Springer, Heidelberg (2005)
23. van Benthem, J.: Correspondence theory. In: *Handbook of Philosophical Logic*, vol. 3, pp. 325–408. Kluwer (2001)

On Nominal Regular Languages with Binders^{*}

Alexander Kurz, Tomoyuki Suzuki^{**}, and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

Abstract. We investigate regular languages on infinite alphabets where words may contain binders on names. To this end, classical regular expressions and automata are extended with binders. We prove the equivalence between finite automata on binders and regular expressions with binders and investigate closure properties and complementation of regular languages with binders.

1 Introduction

Automata over infinite alphabets have been receiving an increasing amount of attention, see e.g. [16, 22, 4, 26]. In these approaches, the countably infinite alphabet \mathcal{N} can be considered as a set of ‘names’, which can be tested only for equality.

Typically, in this context languages of interest such as

$$\mathcal{L} = \{n_1 \dots n_k \in \mathcal{N}^* \mid \exists j > i. n_i = n_j\}$$

from [16] are invariant under name-permutations: If e.g. nmn is in the language, then so is $n'mn' = (nn') \cdot nmn$, where (nn') stands for the application of the transposition $(n n')$ to the word nmn . This suggests to think of names as being bound and languages to be closed under α -equivalence.

On the other hand, we may fix a name n_0 and consider the language

$$\mathcal{L}_{n_0} = \{n_0 n_1 \dots n_k \in \mathcal{N}^* \mid \exists j > i > 0. n_i = n_j\}$$

so that we can think of n_0 as a free name and of n_1, \dots, n_k as bound names. This suggests to study not only words over names, but also words which contain binders and allow us to make explicit the distinction between bound and free names. For example, we might then model \mathcal{L}_{n_0} by a regular expression

$$n_0 \langle n. \langle m.m \rangle^* n \langle k.k \rangle^* n \rangle \quad (1)$$

where $\langle n.e \rangle$ binds n in e and, in the reading above, $\langle n.n \rangle^*$ is interpreted as \mathcal{N}^* .

In this paper, we consider languages with explicit binders on names in words and study regular expressions such as (1) together with the associated notion of finite automata. We prove a Kleene style theorem relating finite automata and regular expressions (cf. § 4) and show that regular languages are closed under intersection, union, and *resource-sensitive complement* (defined in § 5).

^{*} This work has been partially supported by the Aut. Region of Sardinia under grant L.R.7/2007 CRP2-120 (Project TESLA).

^{**} The author’s PhD research was supported by the Yoshida Scholarship Foundation.

Regular expressions for words with binders could be used for the design and analysis of programming languages (as in [23] or [21]). For instance, to check that a ‘variable’ (i.e. name) is declared before it is used, consider the finite set of ‘letters’ $\mathcal{S} = \{\text{DECL}, \text{USE}\}$ and the nominal regular expression on the alphabet $\mathcal{N} \cup \mathcal{S}$

$$e_1 \langle n.\text{DECL } n \ e_2 \ \text{USE } n \ e_3 \rangle^*$$

where with $n \in \mathcal{N}$ and e being some other regular expression not involving n .

Another motivation for words with binders comes from verification and testing. To this aim, in §2 we consider a scenario based on transactional computations to show how regular languages with binders can suitably represent computations that classical regular languages can only approximate. More precisely, we give examples throughout the paper that illustrate how the naming policies used to implement transactional behaviours can be captured in terms of regular languages with binders. Then our Kleene theorem yields in this context an automata that recognises the language and our resource-sensitive complement operation can be used to obtain a transactional monitor, namely an automaton that recognises anomalous transactional behaviours.

Related Work. Automata on words with binders already appear in [24] in the study of the λ -calculus. In this paper we begin the systematic study of words with binders from the point of view of the classical theory of formal languages and automata. This builds on the recent convergence of three lines of research: languages over infinite alphabets, HD-automata, and nominal sets, as we will explain now in more detail. As emphasised above, the languages over infinite alphabets typically considered are *equivariant*, that is, they are invariant under *permutations*. On the other hand, history-dependent (HD) automata [19,18] have been developed in order to check π -calculus expressions for bisimilarity. This research highlighted that minimization requires to keep track of permutations of local names. It is also known that HD-automata are automata internal (in the sense of [3]) in the category of *named sets* [6,7]. Independently of the work on HD-automata, nominal sets [13] were introduced as a framework of doing mathematics in a set theory where every set comes equipped with a permutation action. In particular, in nominal sets, binders arise naturally as *name abstractions*. Later it was shown that the categories of nominal sets and named sets are equivalent [11,14].

Recently, these three developments have been coming together, see [8,26,4,12] and in particular [5]. In these works, the idea of automata and language theory internal in nominal sets takes shape, but, although binders are the *raison d’être* of nominal sets they are, so far, not present in the words themselves.

Structure of the Paper. In §2 we describe a transactional scenario used throughout the paper. In §3 we define regular languages with binders, nominal regular expressions, and automata on binders. §4 and §5 contain the main technical contributions of the paper.

2 Representing Transactions

Our running example is centred around the notion of *nested transactions* which are paramount in information systems [27] because they feature data consistency in the presence of concurrent or distributed accesses. A transaction is a logically atomic computation made of several steps. A transaction either commits when all its steps are

successful or rolls partial computations back when failures occur before completion. Nested transactions are transactions that may possibly contain inner transactions so that the failure of an inner transaction is confined and does not affect outer transactions. For instance, let $A, B,$ and C be basic activities subject to failure in the nested transaction

$$\text{beginTX } A ; \text{beginTX } B \text{ endTX} ; C \text{ endTX} \tag{2}$$

where C is executed even if B fails and the outer transaction is successful if A and C do not fail; on the contrary, a failure of C would require that also B is rolled-back.

Inner transactions can abort outer ones. This is typically implemented by using *transaction identities* that allow inner transactions to invoke abort operations on outer transactions. Using identities transaction (2), becomes

$$\text{beginTX}_2 A ; \text{beginTX}_1 B \text{ endTX}_1 ; C \text{ endTX}_2 \tag{3}$$

and B can execute an abort operation (say abt_2) that makes transaction (3) fail.

We will consider *i-bound* nested transactions, namely transactions that can be nested only up to a fixed level i . To characterise correct executions of bound nested transactions, one could think of using regular expressions. For instance, take the alphabet

$$T = \{s_1, \dots, s_h\} \cup \{\checkmark, \times\} \cup \{[{}^i,]^i, \text{cmt}^i, \text{abt}^i\}_{1 \leq i \leq 2}$$

where symbols s_i represent basic activities and the others denote success (\checkmark), failure (\times), and — for each possible nesting level — begin ($[{}^i$), end ($]^i$), and the intention to commit (cmt^i) or abort (abt^i). Consider the following regular expressions (4), (5), and (6) on T where, for simplicity, we examine 2-bound transactions:

$$e_0 = \left(\sum_{i=1}^h s_i \right)^* \tag{4}$$

$$e_1 = \left(e_0 + [{}^1 e_0 \text{cmt}^1]^1 \checkmark + [{}^1 e_0 \text{abt}^1]^1 \times \right)^* \tag{5}$$

$$e_2 = \left(e_0 + [{}^2 e_1 \text{cmt}^2]^2 \checkmark + [{}^2 e_1 \text{abt}^2]^2 \times + [{}^2 e_1 [{}^1 e_0 \text{abt}^2]^1]^2 \times \right)^* \tag{6}$$

The language corresponding to e_2 characterises the correct executions of computations with transactions nested up to level two. Therefore, the automaton recognising the complement of the language of e_2 can be used as monitor of such transactions.

Although the expressions (4), (5), and (6) correctly capture the structure of correct executions of our transactions (balanced parenthesis up to level 2), a main drawback is that they do not suitably represent identities of transactions. For example,

$$\text{beginTX}_1 s_1 \text{ endTX}_1 \dots \text{beginTX}_k s_1 \text{ endTX}_k \tag{7}$$

where k is unbound and all the identifiers beginTX_i are pairwise distinct (and similarly for endTX_i), represents the sequential execution of k (non-nested) transactions. Translated in our alphabet T , computation (7) becomes the word $[{}^1 s_1]^1 \checkmark \dots [{}^1 s_1]^1 \checkmark$, where the identities of the transactions vanish. Note that the alternative $[{}^1 s_1]^1 \checkmark \dots [{}^k s_1]^k \checkmark$ requires an infinite alphabet because k is unbound.

We define nominal regular languages below and give a nominal regular expression that captures computations like (7) (cf. Example 1). We provide a class of automata with binders able to accept such language (cf. Example 2).

3 Languages, Automata and Regular Expressions, with Binders

In this section, we introduce languages, automata and regular expressions to present examples as above in a uniform and formal way.

Languages. The main idea is to handle *local names* by explicitly denoting the *binding scopes* to express locality. A binding scope takes the form $\langle n.\dots \rangle$ and represents the fact that the name n is bound between the scope delimiters \langle and \rangle . For instance, the word $\langle n.n m n \rangle$ has the occurrences of n bound while m is not affected by the binder (it occurs *free* in the word). Consequently, we consider words up to α -renaming for bound names, e.g. $\langle n.n m n \rangle$ is identified with $\langle n'.n' m n' \rangle$ for any $n' \neq m$.

Now, let \mathcal{N} be a countably infinite set (of ‘names’) and \mathcal{S} a finite set (of ‘letters’). We define *words* w according to

$$w ::= \varepsilon \mid n \mid s \mid w \circ w \mid \langle n.w \rangle$$

where n ranges over \mathcal{N} and s over \mathcal{S} . We do not consider equalities on words other than α -renaming and, as in the classical case, the monoidal laws of composition. Namely, every word is taken up to α -renaming and the concatenation operation \circ is associative and has the empty word ε as the neutral element. We often write $w v$ for $w \circ v$. We call a set of words (closed under α -renaming) a *nominal language*, or simply a *language*.

The occurrence in a word w of $n \in \mathcal{N}$ is *bound* (resp. *free*) if it is (resp. not) in the scope of a binder $\langle n.\dots \rangle$.

Regular Expressions. We define regular expressions with binders, or *nominal regular expressions* via the grammar

$$ne ::= 1 \mid 0 \mid n \mid s \mid ne \circ ne \mid ne + ne \mid \langle n.ne \rangle \mid ne^*$$

An occurrence of a name n in a nominal regular expression ne , is *bound* if it is in the scope of a binder $\langle n.\dots \rangle$, otherwise it is *free*; accordingly, we say that n is a bound (resp. free) name of ne if there are bound (resp. free) occurrences of n in ne and we let $FN(ne)$ be the set of free names in ne (since ne is a finite expression, $FN(ne)$ is finite).

Example 1. We describe the nominal regular expression addressing the problem from §2(7). Let $S_{tx} = \{\checkmark, \times, \text{cmt}, \text{abt}\} \cup \{s_1, \dots, s_h\}$ and $n_1, n_2 \in \mathcal{N}$ be distinct. Define

$$ne_1 = \left(e_0 + \langle n_1. e_0 \text{cmt } n_1 \rangle \checkmark + \langle n_1. e_0 \text{abt } n_1 \rangle \times \right)^* \quad (8)$$

$$ne_2 = \left(e_0 + \langle n_2. ne_1 \text{cmt } n_2 \rangle \checkmark + \langle n_2. ne_1 \text{abt } n_2 \rangle \times + \right. \\ \left. \langle n_2. ne_1 \langle n_1. e_0 \text{abt } n_2 \rangle \rangle \times \right)^* \quad (9)$$

where e_0 is defined in (4). The above equations are rather similar to (5), and (6); however, in (8) and (9), the binders delimit the scope of n_1 and n_2 and correspond to the beginning and ending of transactions.

In Example 1, identities of transactions are modelled as bound names. Computations of the form (7) are captured by ne_1 that simply requires to re-bind n_1 to beginTX_{i+1}

after it has been bound to `beginTXi`. This is made more precise by considering the interpretation of nominal regular expressions below.

The nominal language $L(\text{ne})$ of a nominal regular expression ne , is defined as

$$\begin{aligned}
 L(1) &\stackrel{\text{def}}{=} \{\varepsilon\} & L(0) &\stackrel{\text{def}}{=} \emptyset & L(n) &\stackrel{\text{def}}{=} \{n\} & L(s) &\stackrel{\text{def}}{=} \{s\} \\
 L(\text{ne}_1 + \text{ne}_2) &\stackrel{\text{def}}{=} L(\text{ne}_1) \cup L(\text{ne}_2) & L(\langle n.\text{ne} \rangle) &\stackrel{\text{def}}{=} \{\langle n.w \rangle \mid w \in L(\text{ne})\} \\
 L(\text{ne}_1 \circ \text{ne}_2) &\stackrel{\text{def}}{=} L(\text{ne}_1) \circ L(\text{ne}_2) = \{w \circ v \mid w \in L(\text{ne}_1), v \in L(\text{ne}_2)\} \\
 L(\text{ne}^*) &\stackrel{\text{def}}{=} \bigcup_{j \in \mathbb{N}} L(\text{ne})^j, \text{ where } L(\text{ne})^j \stackrel{\text{def}}{=} \begin{cases} \{\varepsilon\} & j = 0 \\ L(\text{ne}) \circ L(\text{ne})^{j-1} & j \neq 0 \end{cases}
 \end{aligned}$$

A language of the form $L(\text{ne})$ is called a *nominal regular language*.

Automata. To describe a mechanism to handle *local names* and binders, we let \mathbb{N} denote the set of natural numbers and define $\underline{i} = \{1, \dots, i\}$ for each $i \in \mathbb{N}$. We consider sets (of states) Q paired with a map $\|\cdot\| : Q \rightarrow \mathbb{N}$ and define the *local registers of* $q \in Q$ to be $\|q\|$. Definition 2 below explains how registers store names via maps $\sigma : \|q\| \rightarrow \mathcal{N}$.

Definition 1. Let $\mathcal{N}_{\text{fin}} \subseteq \mathcal{N}$ be a finite set of names. An automaton on binders over S and \mathcal{N}_{fin} — an $(S, \mathcal{N}_{\text{fin}})$ -automaton for short — is a tuple $\mathcal{H} = \langle Q, q_0, F, tr \rangle$ such that

- Q is a finite set (of states) equipped with a map $\|\cdot\| : Q \rightarrow \mathbb{N}$
- $q_0 \in Q$ is the initial state and $\|q_0\| = 0$
- $F \subseteq Q$ is the set of final states and $\|q\| = 0$ for each $q \in F$
- for each $q \in Q$ and $\alpha \in I^{\mathcal{N}_{\text{fin}}}(q) \cup \{\varepsilon\}$ — where $I^{\mathcal{N}_{\text{fin}}}(q) \stackrel{\text{def}}{=} S \cup \{\langle \cdot, \cdot \rangle\} \cup \|q\| \cup \mathcal{N}_{\text{fin}}$ is the set of possible inputs on q — we have a set $tr(q, \alpha) \subseteq Q$ of ‘successor states’; for all $q' \in tr(q, \alpha)$ the following must hold:

$$\begin{aligned}
 \alpha = \langle \cdot \rangle &\implies \|q'\| = \|q\| + 1 \\
 \alpha = \rangle &\implies \|q'\| = \|q\| - 1 \\
 \alpha \in I^{\mathcal{N}_{\text{fin}}}(q) \setminus \{\langle \cdot, \cdot \rangle\} \text{ or } \alpha = \varepsilon &\implies \|q'\| = \|q\|
 \end{aligned}$$

An α -transition is a triple (q, α, q') such that $q' \in tr(q, \alpha)$.

\mathcal{H} is deterministic if, for each $q \in Q$,

$$\begin{cases} |tr(q, \alpha)| = 0, & \text{if } (\alpha = \langle \cdot \rangle \text{ and } \|q\| = \max\{\|q'\| \mid q' \in Q\}) \text{ or } (\alpha = \rangle \text{ and } \|q\| = 0) \\ |tr(q, \alpha)| = 1, & \text{otherwise.} \end{cases}$$

The condition $\|q\| = 0$ for each $q \in F \cup \{q_0\}$ in Definition 1 can be removed at the cost of making the presentation technically more complex.

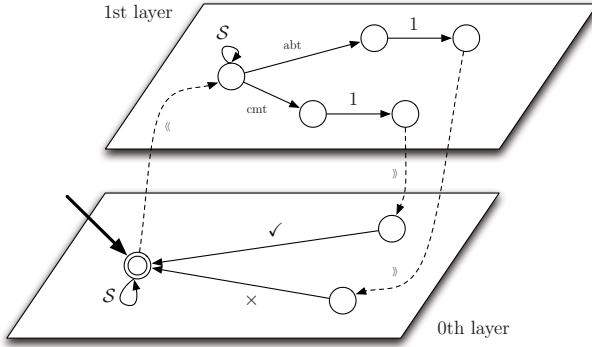
As in the classical case, we say that a $(S, \mathcal{N}_{\text{fin}})$ -automaton is *accessible* when all its states are reachable from the initial state. We tacitly assume that all $(S, \mathcal{N}_{\text{fin}})$ -automata in the current paper are accessible.

Fact 1. For any $(S, \mathcal{N}_{\text{fin}})$ -automaton $\mathcal{H} = \langle Q, q_0, F, tr \rangle$ and $q \in Q$ we have that $\|q\| = 0$ implies $tr(q, \rangle) = \emptyset$ and that $\|q\| = \max_{q' \in Q} \|q'\|$ implies $tr(q, \langle \cdot \rangle) = \emptyset$.

The notion of determinism in Definition 1 is slightly different from the classical one because it must consider the constraints between the registers of source and target states of transitions. We shall come back to this in § 5

Example 2 below exhibits an (S, \mathcal{N}_{fin}) -automaton for the nominal regular expression ne_1 in Example 1. Instead of the formal definition, we introduce a more appealing graphical notation, anticipating the notion of *layer* defined in § 4

Example 2. Let S_{tx} as defined in Example 1. The (S_{tx}, \emptyset) -automaton corresponding to ne_1 can be depicted as



which has a unique initial and final state (the circled one on the 0th layer).

In the figure of Example 2, dashed transitions denote \llcorner - and \lrcorner -transitions. The \llcorner -transition goes from a state on the 0th layer to a state on the 1st layer, whereas the two \lrcorner -transitions go in the opposite direction. Also note that within each layer the picture shows essentially a classical automaton. This is typical for (S, \mathcal{N}_{fin}) -automata, see § 4

Let us fix an (S, \mathcal{N}_{fin}) -automaton

$$\mathcal{H} \stackrel{\text{def}}{=} \langle Q, q_0, F, tr \rangle. \tag{10}$$

and define how \mathcal{H} processes words with free names in \mathcal{N}_{fin} . Hereafter, we denote the image of a map σ by $Im(\sigma)$ and the empty map by \emptyset .

A *configuration* of \mathcal{H} is a tuple $\langle q, w, \sigma \rangle$ consisting of a state q , a word w whose free names are in $\mathcal{N}_{fin} \cup Im(\sigma)$, and a map $\sigma: \llbracket q \rrbracket \rightarrow \mathcal{N}$. We call a configuration $\langle q, w, \sigma \rangle$ *initial* if $q = q_0$, w is a word whose free names are in \mathcal{N}_{fin} , and $\sigma = \emptyset$; we call $\langle q, w, \sigma \rangle$ *accepting* if $q \in F$, $w = \varepsilon$, and $\sigma = \emptyset$.

Definition 2. Given $q, q' \in Q$ and two configurations $t = \langle q, w, \sigma \rangle$ and $t' = \langle q', w', \sigma' \rangle$, \mathcal{H} as in (10) moves from t to t' (written $t \xrightarrow{\mathcal{H}} t'$) if there is $\alpha \in \mathcal{N} \cup \mathbb{N} \cup S \cup \{\llcorner, \lrcorner, \varepsilon\}$ such that $q' \in tr(q, \alpha)$ and

$$\left\{ \begin{array}{lll} \alpha \in \llbracket q \rrbracket, & w = \sigma(\alpha) w', & \sigma' = \sigma \\ \alpha \in \mathcal{N}_{fin} \setminus Im(\sigma), & w = \alpha w', & \text{and } \sigma' = \sigma \\ \alpha \in S, & w = \alpha w', & \text{and } \sigma' = \sigma \\ \alpha = \varepsilon, & w = w', & \text{and } \sigma' = \sigma \\ \alpha = \lrcorner, & w = \lrcorner w', & \text{and } \sigma' = \sigma|_{\llbracket q' \rrbracket} \\ \alpha = \llcorner, & w = \llcorner n.w', & \text{and } \sigma' = \sigma[\llbracket q' \rrbracket \mapsto n] \end{array} \right. \quad \text{and } \forall i > \alpha. \sigma(\alpha) \neq \sigma(i)$$

where $\sigma[\|q'\| \mapsto n]$ extends σ by allocating the maximum index in $\|q'\|$ to n .

The set $\text{reach}_{\mathcal{H}}(t)$ of states reached by \mathcal{H} from the configuration t is defined as

$$\text{reach}_{\mathcal{H}}(t) \stackrel{\text{def}}{=} \begin{cases} \{q\} & \text{if } t = \langle q, \varepsilon, \sigma \rangle \\ \bigcup_{t \xrightarrow{\mathcal{H}} t'} \text{reach}_{\mathcal{H}}(t') & \text{otherwise} \end{cases}$$

A run of \mathcal{H} on a word w is a sequence of moves of \mathcal{H} from $\langle q_0, w, \emptyset \rangle$.

Definition 3. The $(S, \mathcal{N}_{\text{fin}})$ -automaton \mathcal{H} in (10) accepts (or recognises) a word w if $F \cap \text{reach}_{\mathcal{H}}(\langle q_0, w, \emptyset \rangle) \neq \emptyset$. The language of \mathcal{H} is the set $L_{\mathcal{H}}$ of words accepted by \mathcal{H} .

Example 3. It is straightforward to observe that the $(S_{\text{ix}}, \emptyset)$ -automaton in Example 2 accepts $L(\text{ne}_1)$. The only interesting steps are from a configuration where the word starts with a binder; say $\langle n.w \rangle$. In our example, the automaton consumes the binder only if it is in the initial/final state; in this case, the (unique) register of the target state on the 1st layer is mapped to n and used in the transitions on the 1st layer. Observe that, if the right-most states on the 1st layer are reached by consuming w , then the automaton can “deallocate” n and possibly reach the final state.

A direct consequence of Definitions 2 and 3 is the following proposition.

Proposition 1. If \mathcal{H} accepts w and w' is α -equivalent to w then \mathcal{H} accepts w' .

Remark 1. The automata in Definition 1 can be envisaged either as an instantiation of basic history-dependent automata [19] or as a variant of finite-memory automata [16]. In fact, the constraint on local names imposed in Definition 1 allows us to treat names as “global” (as done in finite-memory automata). More precisely, the semantics of each index is uniformly fixed through our automata, once it has been allocated. A main difference wrt basic history-dependent and finite-memory automata though, is the “stack discipline” imposed by $\langle -$ and \rangle -transitions.

4 A Kleene Theorem

This section gives the details of the equivalence, in the setting with binders, of finite automata and regular expressions. The main results can be summarised as follows.

Theorem 1. For each $(S, \mathcal{N}_{\text{fin}})$ -automaton \mathcal{H} , there exists a deterministic $(S, \mathcal{N}_{\text{fin}})$ -automaton which recognises the same language as \mathcal{H} .

Theorem 2. Every language recognised by an $(S, \mathcal{N}_{\text{fin}})$ -automaton is representable by a nominal regular expression. Conversely, every language represented by a nominal regular expression ne is acceptable by an $(S, FN(\text{ne}))$ -automaton.

The proofs and constructions are obtained by extending the techniques of classical automata theory (see e.g. [15]) layer-wise: Given $\mathcal{H} = \langle Q, q_0, F, tr \rangle$ as in (10), $Q^i \stackrel{\text{def}}{=} \{q \in Q \mid \|q\| = i\}$ is the i -th layer of \mathcal{H} . Each layer of \mathcal{H} is very much like a classical automaton if all $\langle -$ and \rangle -transitions are ignored. In fact, the only way to move from

layer i to $i + 1$ is to read a \langle along a \langle -transition; vice-versa, moving from layer $i + 1$ to i is possible only by reading a \rangle along a \rangle -transition.

From $(\mathcal{S}, \mathcal{N}_{fin})$ -Automata to Regular Expressions. The first step to prove Theorem [1](#) is to construct a deterministic $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton for each $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton. Given an $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton \mathcal{H} , we first remove all ε -transitions. Note that ε -transitions are not allowed to connect states on different layers. For the ε -free non-deterministic $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton, we take *the powerset construction* for each layer, and make all layers deterministic except \langle - and \rangle -transitions. Finally, we define \langle -transitions and \rangle -transitions in a deterministic way: For each subset $Q' \subseteq Q^i$, we let

$$tr(Q', \langle) \stackrel{\text{def}}{=} \bigcup_{q \in Q'} tr(q, \langle) \quad \text{and} \quad tr(Q', \rangle) \stackrel{\text{def}}{=} \bigcup_{q \in Q'} tr(q, \rangle)$$

This construction allows us to claim Theorem [1](#).

There are two main reasons for applying the powerset construction layer-wise rather than to the whole automaton. A technical reason is that the definition of the function $\|_|_$ on sets of states taken from different layers could not be given in a consistent way. Secondly, since only \langle - and \rangle -transitions can move between layers, each layer must have a “sink” state (i.e. the empty set of states) to allow for transitions that reject words.

The following proposition yields one direction of Theorem [2](#).

Proposition 2. *For any deterministic $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton \mathcal{H} there is a nominal regular expression $ne_{\mathcal{H}}$ such that $L(ne_{\mathcal{H}})$ is the language recognised by \mathcal{H} .*

Proof (Sketch). Take \mathcal{H} as in [10](#) to be deterministic; Q can be decomposed into

$$Q^0 = \{q_1^0, \dots, q_{m_0}^0\}, \quad Q^1 = \{q_1^1, \dots, q_{m_1}^1\}, \quad \dots \quad Q^h = \{q_1^h, \dots, q_{m_h}^h\}$$

where $h = \max_{q \in Q} \|q\|$ is the highest layer of \mathcal{H} . Note that $q_0 \in Q^0$ and we assume that it is q_1^0 . Let ${}^sR_{i,j}^k$ denote the set of paths from q_i^s to q_j^s which visit only states on layers higher than s or states $q_r^s \in Q^s$ with $r \leq k$. Then, ${}^sR_{i,j}^k$ is defined on the highest layer h by

$${}^hR_{i,j}^0 \stackrel{\text{def}}{=} \{\alpha \mid q_j^h \in tr(q_i^h, \alpha)\} \cup E \quad {}^hR_{i,j}^k \stackrel{\text{def}}{=} {}^hR_{i,k}^{k-1} \left({}^hR_{k,k}^{k-1} \right)^* {}^hR_{k,j}^{k-1} \cup {}^hR_{i,j}^{k-1}$$

where, in the first clause, $E = \{\varepsilon\}$ if $i = j$ and $E = \emptyset$ if $i \neq j$ and, for layer $s < h$, letting $\Gamma_{i,j}^s \stackrel{\text{def}}{=} \{(i', j') \mid q_{i'}^{s+1} \in tr(q_i^s, \langle) \wedge q_j^s \in tr(q_{j'}^{s+1}, \rangle)\}$, by

$${}^sR_{i,j}^0 \stackrel{\text{def}}{=} \{\alpha \mid q_j^s \in tr(q_i^s, \alpha)\} \cup \bigcup_{(i', j') \in \Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_{s+1}} \cup E$$

$${}^sR_{i,j}^k \stackrel{\text{def}}{=} {}^sR_{i,k}^{k-1} \left({}^sR_{k,k}^{k-1} \right)^* {}^sR_{k,j}^{k-1} \cup {}^sR_{i,j}^{k-1} \cup \bigcup_{(i', j') \in \Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_{s+1}}$$

where, in the first clause, $E = \{\varepsilon\}$ if $i = j$ and $E = \emptyset$ if $i \neq j$.

Hence, $\bigcup_{(i', j') \in \Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_{s+1}}$ is the collection of all paths from q_i^s to q_j^s visiting only states on the higher layers. Finally, we translate all paths from the initial state to final states into a nominal regular expression, but this is analogous to the classical theory. \square

From Nominal Regular Expressions to $(\mathcal{S}, \mathcal{N}_{fin})$ -Automata. We now turn our attention to the construction of $(\mathcal{S}, \mathcal{N}_{fin})$ -automata from nominal regular expressions.

Proposition 3. *Given a nominal regular expression ne , there is an $(\mathcal{S}, FN(ne))$ -automaton \mathcal{H} which recognises $L(ne)$.*

We prove the above proposition by induction on the structure of ne . Let $\mathcal{H}_{(ne)}$ denote the $(\mathcal{S}, FN(ne))$ -automaton defined by the following construction.

We start with the constructions for the base cases.

- $ne = 1$: let $\mathcal{H}_{(1)}$ be $\langle \{q_0\}, q_0, \{q_0\}, tr \rangle$ where $\|q_0\| = 0$ and $tr(q_0, \alpha) = \emptyset$ for each $\alpha \in I^{\mathcal{N}_{fin}}(q_0)$.
- $ne = 0$: let $\mathcal{H}_{(0)}$ be $\langle \{q_0\}, q_0, \emptyset, tr \rangle$ where $\|q_0\| = 0$ and $tr(q_0, \alpha) = \emptyset$ for each $\alpha \in I^{\mathcal{N}_{fin}}(q_0)$.
- $ne = n$: let $\mathcal{H}_{(n)}$ be $\langle \{q_0, q_1\}, q_0, \{q_1\}, tr \rangle$ where, for $j \in \{0, 1\}$ $\|q_j\| = 0$ and

$$\begin{aligned} tr(q_0, n) &= \{q_1\} \\ tr(q_j, \alpha) &= \emptyset, \quad \text{for } \alpha \in I^{\mathcal{N}_{fin}}(q_j) \text{ and } \alpha \neq n \text{ if } j = 0. \end{aligned}$$

Note that, as $FN(n) = \{n\}$, each state may have a transition with the label n .

- $ne = s$: let $\mathcal{H}_{(s)}$ be $\langle \{q_0, q_1\}, q_0, \{q_1\}, tr \rangle$ where, for $j \in \{0, 1\}$, $\|q_j\| = 0$ and

$$\begin{aligned} tr(q_0, s) &= \{q_1\} \\ tr(q_j, \alpha) &= \emptyset, \quad \text{for } \alpha \in I^{\mathcal{N}_{fin}}(q_j) \text{ and } \alpha \neq s \text{ if } j = 0. \end{aligned}$$

Lemma 1. $\mathcal{H}_{(1)}$, $\mathcal{H}_{(0)}$, $\mathcal{H}_{(n)}$ and $\mathcal{H}_{(s)}$ recognise, respectively, $L(1)$, $L(0)$, $L(n)$ and $L(s)$. Further, $\mathcal{H}_{(1)}$, $\mathcal{H}_{(0)}$ and $\mathcal{H}_{(s)}$ are (\mathcal{S}, \emptyset) -automata, and $\mathcal{H}_{(n)}$ is an $(\mathcal{S}, \{n\})$ -automaton, i.e. $\mathcal{H}_{(1)}$, $\mathcal{H}_{(0)}$, $\mathcal{H}_{(n)}$ and $\mathcal{H}_{(s)}$ are all $(\mathcal{S}, FN(ne))$ -automata.

Union $ne_1 + ne_2$: For $j \in \{1, 2\}$, let $\mathcal{H}_{(ne_j)} = \langle Q_j, q_{0,j}, F_j, tr_j \rangle$ be an $(\mathcal{S}, FN(ne_j))$ -automaton which recognises $L(ne_j)$. The union $\mathcal{H}_{(ne_1 + ne_2)}$ is a tuple $\langle Q^+, q_0^+, F^+, tr^+ \rangle$:

- $Q^+ \stackrel{\text{def}}{=} Q_1 \uplus Q_2 \uplus \{q_0^+\}$ (where \uplus stands for disjoint union);
- q_0^+ is a new state with $\|q_0^+\| = 0$;
- $F^+ \stackrel{\text{def}}{=} F_1 \uplus F_2$;
- $tr^+(q, \alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q, \alpha), & \text{if } j \in \{1, 2\}, q \in Q_j, \text{ and } \alpha \in I_j^{\mathcal{N}_{fin}}(q) \\ \{q_{0,1}, q_{0,2}\}, & \text{if } q = q_0^+ \text{ and } \alpha = \varepsilon \\ \emptyset, & \text{otherwise} \end{cases}$

where $I_1^{\mathcal{N}_{fin}}(q)$ is the set of possible inputs on q in $\mathcal{H}_{(ne_1)}$ and $I_2^{\mathcal{N}_{fin}}(q)$ is the set of possible inputs on q in $\mathcal{H}_{(ne_2)}$. Notice that possible inputs of free names are extended from $FN(ne_1)$ or $FN(ne_2)$ to $FN(ne_1) \cup FN(ne_2)$ in $\mathcal{H}_{(ne_1 + ne_2)}$, but the above definition of transitions says that each state q in Q_1 has no transition with labels in $FN(ne_2) \setminus I_1^{\mathcal{N}_{fin}}(q)$, and each state q in Q_2 has no transition with labels in $FN(ne_1) \setminus I_2^{\mathcal{N}_{fin}}(q)$.

Lemma 2. $\mathcal{H}_{(\text{ne}_1 + \text{ne}_2)}$ is an $(S, FN(\text{ne}_1 + \text{ne}_2))$ -automaton recognising $L(\text{ne}_1 + \text{ne}_2)$.

Concatenation $\text{ne}_1 \circ \text{ne}_2$: For $j \in \{1, 2\}$, let $\mathcal{H}_{(\text{ne}_j)} = \langle Q_j, q_{0,j}, F_j, tr_j \rangle$ be an $(S, FN(\text{ne}_j))$ -automaton which recognises $L(\text{ne}_j)$. The concatenation $\mathcal{H}_{(\text{ne}_1 \circ \text{ne}_2)}$ is a tuple $\langle Q^\circ, q_0^\circ, F^\circ, tr^\circ \rangle$:

$$\begin{aligned} & - Q^\circ \stackrel{\text{def}}{=} Q_1 \uplus Q_2; \\ & - q_0^\circ \stackrel{\text{def}}{=} q_{0,1}; \\ & - F^\circ \stackrel{\text{def}}{=} F_2; \\ & - tr^\circ(q, \alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q, \alpha), & \text{if } q \in Q_1 \setminus F_1, \text{ and } \alpha \in I_1^{\mathcal{N}_{fin}}(q) \\ tr_2(q, \alpha), & \text{if } q \in Q_2 \text{ and } \alpha \in I_2^{\mathcal{N}_{fin}}(q) \\ tr_1(q, \alpha) \cup \{q_{0,2}\}, & \text{if } q \in F_1 \text{ and } \alpha = \varepsilon \\ tr_1(q, \alpha), & \text{if } q \in F_1 \text{ and } \alpha \in I_1^{\mathcal{N}_{fin}}(q) \\ \emptyset, & \text{otherwise} \end{cases} \end{aligned}$$

Intuitively, we just add ε -transitions to $q_{0,2}$ from the final states of $\mathcal{H}_{(\text{ne}_1)}$.

Lemma 3. $\mathcal{H}_{(\text{ne}_1 \circ \text{ne}_2)}$ is an $(S, FN(\text{ne}_1 \circ \text{ne}_2))$ -automaton recognising $L(\text{ne}_1 \circ \text{ne}_2)$.

Iteration ne^* : Given a nominal regular expression ne and an $(S, FN(\text{ne}))$ -automaton $\mathcal{H}_{(\text{ne})} = \langle Q, q_0, F, tr \rangle$ which recognises $L(\text{ne})$, the iteration $\mathcal{H}_{(\text{ne}^*)}$ is defined by a tuple $\langle Q^*, q_0^*, F^*, tr^* \rangle$:

$$\begin{aligned} & - Q^* \stackrel{\text{def}}{=} Q; \\ & - q_0^* \stackrel{\text{def}}{=} q_0; \\ & - F^* \stackrel{\text{def}}{=} \{q_0\}; \\ & - tr^*(q, \alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q, \alpha), & \text{if } q \in Q \setminus F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \\ tr(q, \alpha) \cup \{q_0^*\}, & \text{if } q \in F \text{ and } \alpha = \varepsilon \\ tr(q, \alpha), & \text{if } q \in F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \setminus \{\varepsilon\} \end{cases} \end{aligned}$$

Notice that, since the possible inputs on q in $\mathcal{H}_{(\text{ne}^*)}$ and $\mathcal{H}_{(\text{ne})}$ are the same, here we do not need to consider the ‘‘otherwise’’ case.

Lemma 4. $\mathcal{H}_{(\text{ne}^*)}$ is an $(S, FN(\text{ne}^*))$ -automaton recognising $L(\text{ne}^*)$.

Name-abstraction $\langle n.\text{ne} \rangle$: Given a nominal regular expression ne and an $(S, FN(\text{ne}))$ -automaton $\mathcal{H}_{(\text{ne})} = \langle Q, q_0, F, tr \rangle$ which recognises $L(\text{ne})$, the name-abstraction $\mathcal{H}_{(\langle n.\text{ne} \rangle)}$ is defined by a tuple $\langle Q^\diamond, q_0^\diamond, F^\diamond, tr^\diamond \rangle$:

$$\begin{aligned} & - q_s and q_t are new states with $\|q_s\| = 0$ and $\|q_t\| = 0$; \\ & - $Q^\diamond \stackrel{\text{def}}{=} Q \uplus \{q_s, q_t\}$ where we increase $\|q\|$ by 1 for each state $q \in Q$ (hence q_s and q_t are the only states on the 0th layer); \\ & - $q_0^\diamond \stackrel{\text{def}}{=} q_s$; \\ & - $F^\diamond \stackrel{\text{def}}{=} \{q_t\}$; \end{aligned}$$

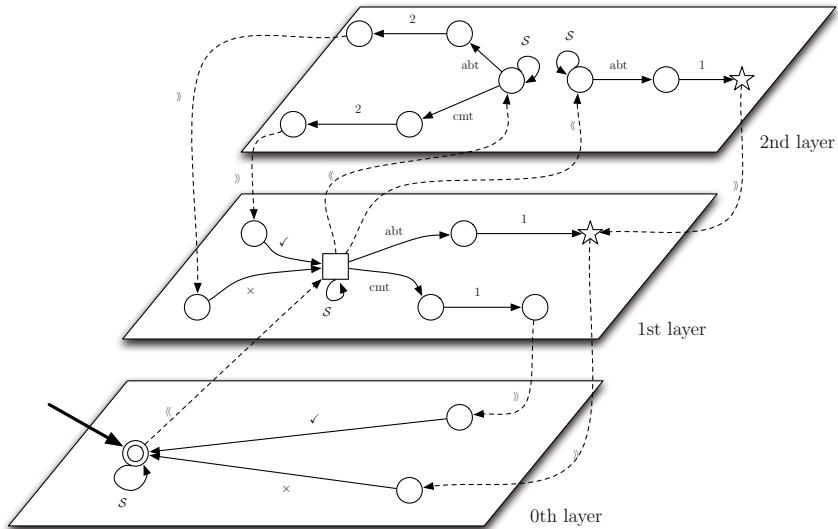
$$- tr^\circ(q, \alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q, n), & \text{if } q \in Q \text{ and } \alpha = 1 \in \underline{\|q\|} \\ tr(q, \alpha - 1), & \text{if } q \in Q \text{ and } \alpha \in \underline{\|q\|} \setminus \{1\} \\ tr(q, \alpha), & \text{if } q \in Q \setminus F \text{ and } \alpha \in I^{\mathcal{N}_{fn}}(q) \setminus \underline{\|q\|} \\ \{q_i\}, & \text{if } q \in F \text{ and } \alpha = \gg \\ tr(q, \alpha), & \text{if } q \in F \text{ and } \alpha \in I^{\mathcal{N}_{fn}}(q) \setminus (\underline{\|q\|} \cup \{\gg\}) \\ \{q_0\}, & \text{if } q = q_s \text{ and } \alpha = \ll \\ \emptyset, & \text{otherwise} \end{cases}$$

Note that on an $(S, FN(\langle n, ne \rangle))$ -automaton, n cannot be an input on any state. Intuitively, to bind the free name n , we first increase all numbers in all states in $\mathcal{H}_{(ne)}$ (accordingly for the labels on transitions) by 1, and then we allocate the new number 1 in each state in $\mathcal{H}_{(ne)}$ for the new local name obtained by binding n and rename all labels n on each transition in $\mathcal{H}_{(ne)}$ (if they exist) with the number 1.

Lemma 5. $\mathcal{H}_{(n, ne)}$ is an $(S, FN(\langle n, ne \rangle))$ -automaton recognising $L(\langle n, ne \rangle)$.

Lemmas 1, 2, 3, 4 and 5 complete the proof of Proposition 3.

Example 4. We give an account of how the nominal regular expression ne_2 in Example 7 is translated to an (S_x, \emptyset) -automaton. Below is a simplified graphical representation of the automaton $\mathcal{H}_{(ne_2)}$.



(where “box” and “star” shaped states are used to ease the textual description below). This automaton is basically obtained by the above inductive construction but for the following simplifications:

1. we removed all ε -transitions in the obvious way;
2. three equivalent states of $\mathcal{H}_{(ne_2)}$ accessible from the initial state with \ll -transitions are now unified as the single box state on the 1st layer;

3. the star state on the 2nd layer which had a \gg -transition to a distinct state on the 1st layer in $\mathcal{H}_{(ne_2)}$ is now connected to the star state on the 1st layer. Accordingly, non-accessible states are deleted.

5 Closure Properties

Here we shall discuss the closure properties of nominal languages summarised in

Theorem 3. *Nominal languages are closed under union, intersection, and resource sensitive complementation.*

The notion of resource-sensitive complementation is given in Definition 4 below. Closure under unions is immediate and the construction is the same as the classical one. Similarly, closure under intersections is shown by taking a product of the respective automata; the only difference wrt the analogous construction in the classical theory is that we must take the product layer-wise (otherwise there is no meaningful way to define the values of $\|_-\|$). It remains to discuss complementation.

In our nominal languages, brackets \langle and \rangle are explicitly expressed as syntax. So, for example, $\langle n.w \rangle$ is different from $\langle m.\langle n.w \rangle \rangle$, even when m does not freely occur in $\langle n.w \rangle$. This is important for complementing nominal regular languages since every word has a maximum *depth* of nested binders determined by the regular expression. Define $\partial(ne)$, the *depth of a nominal regular expression* ne as

$$\begin{aligned} ne \in \{\varepsilon, 1, 0\} \cup \mathcal{N} \cup \mathcal{S} &\implies \partial(ne) = 0 \\ ne = ne_1 + ne_2 \text{ or } ne_1 \circ ne_2 &\implies \partial(ne) = \max(\partial(ne_1), \partial(ne_2)) \\ ne = \langle n.ne \rangle &\implies 1 + \partial(ne) \\ ne = ne^* &\implies \partial(ne) \end{aligned}$$

For example, if $ne = \langle n.(m.s m n)^* n \circ s \rangle \circ \langle l.s l \rangle \circ s$ then $\partial(ne) = 2$, hence no word in $L(ne)$ can have more than 2 nested binders; therefore the complement of $L(ne)$ has to include words which have finite but unbounded depth, e.g. $\langle n_1.\langle n_2.\dots\langle n_k.n_1 \dots n_k \rangle \dots \rangle \rangle$ for any natural number $k > \partial(ne)$. But, it is impossible to accept all these words on any finite $(\mathcal{S}, \mathcal{N}_{fin})$ -automaton. Therefore, nominal regular languages are not closed under the standard complementation.

On the other hand, when contemplating nominal languages, is the notion of the standard complementation suitable? We claim that there are two distinct conditions for rejecting words on $(\mathcal{S}, \mathcal{N}_{fin})$ -automata:

1. The word is consumed and the automaton finishes in a non-accepting state.
2. The automaton is in a configuration whose word is of the form $\langle n.w \rangle$ and its state is in the highest layer.

The first is the usual non-acceptance condition, while the second one, which we call *overflow* condition, is necessary for $(\mathcal{S}, \mathcal{N}_{fin})$ -automata. Informally, the distinction of the two conditions of rejection above can be rephrased as follows:

Rejection by non-acceptance takes place when the word represents a ‘wrong behaviour’; instead, rejection by overflow happens when we do not have enough resources to process the word.

The considerations above lead to the notion of *resource-sensitive complementation*:

Definition 4. Let ne be a nominal regular expression. The resource sensitive complementation of $L(ne)$ is the set $\{w \notin L(ne) \mid \partial(w) \leq \partial(ne)\}$ (where the depth of a word is defined as the depth the corresponding expression).

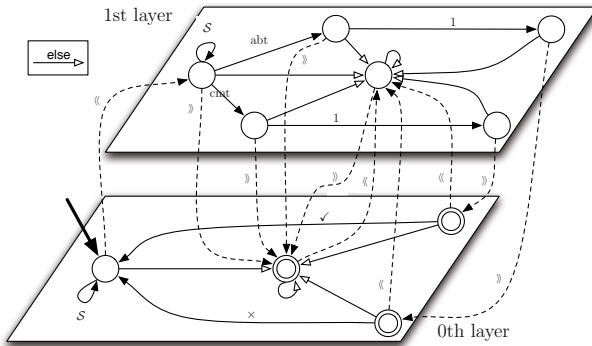
The algebraic structure of union, intersection, and resource sensitive complementation is that of a generalised Boolean algebra [25], that is, of a distributive lattice with bottom and relative complement (but no top).

For the proof that nominal regular expressions are closed under resource-sensitive complementation, note first that the overflow condition characterises the configurations where a deterministic automaton (Definition 1) can get stuck. Further, recall that, by Proposition 3, for each nominal regular expression ne , there is an (S, \mathcal{N}_{fin}) -automaton $\mathcal{H}_{(ne)}$ which accepts the language $L(ne)$. By Theorem 1 we can assume, without loss of generality, that $\mathcal{H}_{(ne)}$ is deterministic. Since configurations can get stuck only by overflow, for any word in $L(ne)$, $\mathcal{H}_{(ne)}$ has a run to a final state which, by construction, is on the 0-th layer. Hence, if we swap the final states with the non-final states on the 0-th layer, the automaton recognises the resource-sensitive complementation of $L(ne)$. Finally, by Proposition 2, we obtain

Proposition 4. *Nominal regular expressions are closed under resource-sensitive complementation.*

We give an example of how to construct an (S, \mathcal{N}_{fin}) -automaton which accepts the resource-sensitive complementation of a nominal regular expression in Example 1.

Example 5. An automaton which recognises the resource-sensitive complementation of $L(ne_1)$ from Example 1 is given below:



In this automaton, the transitions with non-filled heads represent “complement transitions”, that is transitions taken when the source state does not have any filled-head transition with a label matching the input symbol. Let $\mathcal{H}_{(ne_1)}$ be the (S, \mathcal{N}_{fin}) -automaton in Example 2: the automaton above is constructed from $\mathcal{H}_{(ne_1)}$ as follows:

1. firstly $\mathcal{H}_{(ne_1)}$ is determinised, by means of the layer-wise powerset construction; note that this adds deadlock states (in the automaton in the picture above, we just added two deadlock states, one per layer);

2. *secondly, all non-accessible states are removed;*
3. *finally, accepting and non-accepting states on the 0th layer are swapped.*

The automaton in Example 5 can be used as a *transactional monitor* of the transactions characterised by ne_2 in Example 1. Namely, it accepts words representing computations of 2-level nested transactions that diverge from the expected behaviour, e.g., transactions that starts but do not explicitly commit or abort.

6 Conclusion

Our long-term aim is to develop a theory of nominal languages with binders. In this paper we looked at the most basic case where the binders do not interact with the monoid operations. But there is a range of other interesting possibilities. For example, one may impose the additional equation $\langle n.w \rangle \circ v = \langle n.w \circ v \rangle$ for n not free in v . This is known as scope extrusion in the π -calculus and would have as a consequence that an automaton recognising $\langle n.n \rangle^*$ would need to be able to keep track of an *unbounded* number of local names (for an analysis of the interplay between binders and name locality see e.g. [20,10]). A first sketch of some of the arising landscape is drafted in [17].

Although the use of binders in this paper is rather restricted, it is expressive enough to represent interesting computational phenomena and it guarantees the properties in § 4 and 5. Increasing the expressiveness of our regular expressions by adding permutations is the natural step we are currently investigating. For instance, we are considering languages where the Kleene-star operator interplays with name automorphisms.

It will be interesting to explore the connections with tree-walking pebble automata [9]. The idea is that the configuration of the automaton is given by a pair (q, v) where q is a state of the automaton and v a node of the input tree. A run is obtained by letting the automaton to change its state and mode to parent/children nodes of v according to the label of v and the fact that v is the root, a leaf, or an internal node. Our approach is reminiscent of this pushdown mechanism of tree-walking pebble automata, with the difference that the decision of dropping/lifting pebbles is driven by binders in the word.

With an eye on applications to verification it is of interest to pursue further developments in the direction of the work [1] on Kleene algebra with local scope. Another direction for future work starts from the observation that, due to the last two clauses of Definition 2, automata with binders do not process words but nested words (with dangling brackets) [2]. This suggests extend the work of [2] to the nominal setting.

Acknowledgements. The authors thank the reviewers for their criticisms and comments which helped to greatly improve the paper.

References

1. Aboul-Hosn, K., Kozen, D.: Local variable scoping and kleene algebra with tests. *J. Log. Algebr. Program.* 76(1), 3–17 (2008)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. *J. ACM* 56(3) (2009)

3. Arbib, M.A., Manes, E.G.: Machines in a category: An expository introduction. *SIAM Review* 16(163-192), 285–302 (1974)
4. Bojanczyk, M.: Data monoids. In: *STACS*, pp. 105–116 (2011)
5. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: *IEEE Symposium on Logic in Computer Science*, pp. 355–364 (2011)
6. Ciancia, V.: *Nominal Sets, Accessible Functors and Final Coalgebras for Named Sets*. PhD thesis, Dipartimento di Informatica, Università di Pisa (2008) forthcoming
7. Ciancia, V., Montanari, U.: Symmetries, local names and dynamic (de)-allocation of names. *Inf. Comput.* 208(12), 1349–1367 (2010)
8. Ciancia, V., Tuosto, E.: A novel class of automata for languages on infinite alphabets. Technical Report CS-09-003, Leicester (2009)
9. Engelfriet, J., Hooġeboom, H.J.: Tree-walking pebble automata. In: *Jewels are Forever*, pp. 72–83 (1999)
10. Fernández, M., Gabbay, M.: Nominal rewriting. *Inf. Comput.* 205(6), 917–965 (2007)
11. Fiore, M.P., Staton, S.: Comparing operational models of name-passing process calculi. *Inf. Comput.* 204(4), 524–560 (2006)
12. Gabbay, M., Ciancia, V.: Freshness and Name-Restriction in Sets of Traces with Names. In: Hofmann, M. (ed.) *FOSSACS 2011*. LNCS, vol. 6604, pp. 365–380. Springer, Heidelberg (2011)
13. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: *Symbolic on Logics in Comput Science*, pp. 214–224 (1999)
14. Gadducci, F., Miculan, M., Montanari, U.: About permutation algebras (pre)sheaves and named sets. *Higher-Order and Symbolic Computation* 19(2-3), 283–304 (2006)
15. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*, 2nd edn. Addison Wesley (2000)
16. Kaminski, M., Francez, N.: Finite-memory automata. *TCS* 134(2), 329–363 (1994)
17. Kurz, A., Suzuki, T., Tuosto, E.: Towards nominal formal languages. *CoRR*, abs/1102.3174 (2011)
18. Montanari, U., Pistore, M.: π -Calculus, Structured Coalgebras, and Minimal HD-Automata. In: Nielsen, M., Rovan, B. (eds.) *MFCS 2000*. LNCS, vol. 1893, pp. 569–578. Springer, Heidelberg (2000)
19. Pistore, M.: *History Dependent Automata*. PhD thesis, Dip. di Informatica - Pisa (1999)
20. Pitts, A., Stark, I.: Observable Properties of Higher Order Functions that Dynamically Create Local Names, or What’s New? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) *MFCS 1993*. LNCS, vol. 711, pp. 122–141. Springer, Heidelberg (1993)
21. Pouillard, N., Pottier, F.: A fresh look at programming with names and binders. In: *Proceeding of the 15th ACM SIGPLAN International Conference on Functional Programming*, pp. 217–228 (2010)
22. Segoufin, L.: Automata and Logics for Words and Trees over an Infinite Alphabet. In: Ésik, Z. (ed.) *CSL 2006*. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
23. Shinwell, M., Pitts, A., Gabbay, M.: Freshml: programming with binders made simple. In: *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pp. 263–274 (2003)
24. Stirling, C.: Dependency Tree Automata. In: de Alfaro, L. (ed.) *FOSSACS 2009*. LNCS, vol. 5504, pp. 92–106. Springer, Heidelberg (2009)
25. Stone, M.H.: Postulates for boolean algebras and generalized boolean algebras. *American Journal of Mathematics* 57(4), 703–732 (1935)
26. Tzevelekos, N.: Fresh-Register Automata. In: *Symposium on Principles of Programming Languages*, pp. 295–306. ACM (2011)
27. Weikum, G., Vossen, G.: *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann (2002)

Robustness of Structurally Equivalent Concurrent Parity Games^{*}

Krishnendu Chatterjee

IST Austria (Institute of Science and Technology Austria)

Abstract. We consider two-player stochastic games played on a finite state space for an infinite number of rounds. The games are *concurrent*: in each round, the two players (player 1 and player 2) choose their moves independently and simultaneously; the current state and the two moves determine a probability distribution over the successor states. We also consider the important special case of turn-based stochastic games where players make moves in turns, rather than concurrently. We study concurrent games with ω -regular winning conditions specified as *parity* objectives. The value for player 1 for a parity objective is the maximal probability with which the player can guarantee the satisfaction of the objective against all strategies of the opponent. We study the problem of continuity and robustness of the value function in concurrent and turn-based stochastic parity games with respect to imprecision in the transition probabilities. We present quantitative bounds on the difference of the value function (in terms of the imprecision of the transition probabilities) and show the value continuity for structurally equivalent concurrent games (two games are structurally equivalent if the supports of the transition functions are the same and the probabilities differ). We also show robustness of optimal strategies for structurally equivalent turn-based stochastic parity games. Finally, we show that the value continuity property breaks without the structural equivalence assumption (even for Markov chains) and show that our quantitative bound is asymptotically optimal. Hence our results are tight (the assumption is both necessary and sufficient) and optimal (our quantitative bound is asymptotically optimal).

1 Introduction

Concurrent stochastic games are played by two players on a finite state space for an infinite number of rounds. In every round, the two players simultaneously and independently choose moves (or actions), and the current state and the two chosen moves determine a probability distribution over the successor states. The outcome of the game (or a *play*) is an infinite sequence of states. These games were introduced by Shapley [25], and have been one of the most fundamental and well studied game models in stochastic graph games. We consider ω -regular objectives specified as parity objectives; that is, given an ω -regular set Φ of infinite state sequences, player 1 wins if the outcome of the game lies in Φ . Otherwise, player 2 wins, i.e., the game is zero-sum.

^{*} The research was supported by Austrian Science Fund (FWF) Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

The class of concurrent stochastic games subsumes many other important classes of games as sub-classes: (1) *turn-based stochastic* games, where in every round only one player chooses moves (i.e., the players make moves in turns); and (2) *Markov decision processes (MDPs)* (one-player stochastic games). Concurrent games and the sub-classes provide a rich framework to model various classes of dynamic reactive systems, and ω -regular objectives provide a robust specification language to express all commonly used properties in verification, and all ω -regular objectives can be expressed as parity objectives. Thus concurrent games with parity objectives provide the mathematical framework to study many important problems in the synthesis and verification of reactive systems [7,24,22] (see also [11,5,2]).

The player-1 *value* $v_1(s)$ of the game at a state s is the limit probability with which player 1 can ensure that the outcome of the game lies in Φ ; that is, the value $v_1(s)$ is the maximal probability with which player 1 can guarantee Φ against all strategies of player 2. Symmetrically, the player-2 *value* $v_2(s)$ is the limit probability with which player 2 can ensure that the outcome of the game lies outside Φ . The problem of studying the computational complexity of MDPs, turn-based stochastic games, and concurrent games with parity objectives has received a lot of attention in literature. Markov decision processes with ω -regular objectives have been studied in [9,10,5] and the results show existence of pure (deterministic) memoryless (stationary) optimal strategies for parity objectives and the problem of value computation is achievable in polynomial time. Turn-based stochastic games with the special case of reachability objectives have been studied in [8] and existence of pure memoryless optimal strategies has been established and the decision problem of whether the value at a state is at least a given rational value lies in $\text{NP} \cap \text{coNP}$. The existence of pure memoryless optimal strategies for turn-based stochastic games with parity objectives was established in [6,29], and again the decision problem lies in $\text{NP} \cap \text{coNP}$. Concurrent parity games have been studied in [11,13,4,16] and for concurrent parity games optimal strategies need not exist, and ε -optimal strategies (for $\varepsilon > 0$) require both infinite memory and randomization in general, and the decision problem can be solved in PSPACE.

Almost all results in the literature consider the problem of computing values and optimal strategies when the game model is given precisely along with the objective. However, it is often unrealistic to know the precise probabilities of transition which are only estimated through observation. Since the transition probabilities are not known precisely, an extremely important question is how robust is the analysis of concurrent games and its sub-classes with parity objectives with respect to small changes in the transition probabilities. This question has been largely ignored in the study of concurrent and turn-based stochastic parity games. In this paper we study the following problems related to continuity and robustness of values: (1) (*continuity of values*): under what conditions can continuity of the value function be proved for concurrent parity games; (2) (*robustness of values*): can quantitative bounds be obtained on the difference of the value function in terms of the difference of the transition probabilities; and (3) (*robustness of optimal strategies*): do optimal strategies of a game remain ε -optimal, for $\varepsilon > 0$, if the transition probabilities are slightly changed.

Our contributions. Our contributions are as follows:

1. We consider *structurally equivalent* game structures, where the supports of the transition probabilities are the same, but the precise transition probabilities may differ. We show the following results for structurally equivalent concurrent parity games:
 - (a) *Quantitative bound.* We present a quantitative bound on the difference of the value functions of two structurally equivalent game structures in terms of the difference of the transition probabilities. We show when the difference in the transition probabilities are small, our bound is asymptotically optimal. Our example to show the matching lower bound is on a Markov chain, and thus our result shows that the bound for a Markov chain can be generalized to concurrent games.
 - (b) *Value continuity.* We show *value continuity* for structurally equivalent concurrent parity games, i.e., as the difference in the transition probabilities goes to 0, the difference in value functions also goes to 0. We then show that the structural equivalence assumption is necessary: we show a family of Markov chains (that are not structurally equivalent) where the difference of the transition probabilities goes to 0, but the difference in the value functions is 1. It follows that the structural equivalence assumption is both necessary (even for Markov chains) and sufficient (even for concurrent games).

It follows from above that our results are both optimal (quantitative bounds) as well as tight (assumption both necessary and sufficient). Our result for concurrent parity games is also a significant quantitative generalization of a result for concurrent parity games of [11] which shows that the set of states with value 1 remains same if the games are structurally equivalent. We also argue that the structural equivalence assumption is not unrealistic in many cases: a reactive system consists of many state variables, and given a state (valuation of variables) it is typically known which variables are possibly updated, and what is unknown is the precise transition probabilities (which are estimated by observation). Thus the system that is obtained for analysis is structurally equivalent to the underlying original system and it only differs in precise transition probabilities.

2. For turn-based stochastic parity games the value continuity and the quantitative bounds are same as for concurrent games. We also prove a stronger result for structurally equivalent turn-based stochastic games that shows that along with continuity of the value function, there is also robustness property for pure memoryless optimal strategies. More precisely, for all $\varepsilon > 0$, we present a bound $\beta > 0$, such that any pure memoryless optimal strategy in a turn-based stochastic parity game is an ε -optimal strategy in every structurally equivalent turn-based stochastic game such that the transition probabilities differ by at most β . Our result has deep significance as it allows the rich literature of work on turn-based stochastic games to carry over robustly for structurally equivalent turn-based stochastic games. As argued before the model of turn-based stochastic game obtained to analyze may differ slightly in precise transition probabilities, and our results shows that the analysis on the slightly imprecise model using the classical results carry over to the underlying original system with small error bounds.

Our results are obtained as follows. The result of [12] shows that the value function for concurrent parity games can be characterized as the limit of the value function of concurrent multi-discounted games (concurrent discounted games with different discount factors associated with every state). There exists bound on difference on value function of discounted games [17], however, the bound depends on the discount factor, and in the limit gives trivial bounds (and in general this approach does not work as value continuity cannot be proven in general and the structural equivalence assumption is necessary). We use a classical result on Markov chains by Friedlin and Wentzell [18] and generalize a result of Solan [26] from Markov chains with single discount to Markov chains with multi-discounted objective to obtain a bound that is independent of the discount factor for structurally equivalent games. Then the bound also applies when we take the limit of the discount factors, and gives us the desired bound.

Our paper is organized as follows: in Section 2 we present the basic definitions, in Section 3 we consider Markov chains with multi-discounted and parity objectives; in Section 4 (Subsection 4.1) we prove the results related to turn-based stochastic games (item (2) of our contributions) and finally in Subsection 4.2 we present the quantitative bound and value continuity for concurrent games along with the two examples to illustrate the asymptotic optimality of the bound and the structural equivalence assumption is necessary. Detailed proofs available in [3].

2 Definitions

In this section we define game structures, strategies, objectives, values and present other preliminary definitions.

Probability Distributions. For a finite set A , a *probability distribution* on A is a function $\delta : A \mapsto [0, 1]$ such that $\sum_{a \in A} \delta(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. Given a distribution $\delta \in \mathcal{D}(A)$, we denote by $\text{Supp}(\delta) = \{x \in A \mid \delta(x) > 0\}$ the *support* of the distribution δ .

Concurrent Game Structures. A (two-player) *concurrent stochastic game structure* $G = \langle S, A, \Gamma_1, \Gamma_2, \delta \rangle$ consists of the following components.

- A finite state space S and a finite set A of moves (or actions).
- Two move assignments $\Gamma_1, \Gamma_2 : S \mapsto 2^A \setminus \emptyset$. For $i \in \{1, 2\}$, assignment Γ_i associates with each state $s \in S$ the nonempty set $\Gamma_i(s) \subseteq A$ of moves available to player i at state s .
- A probabilistic transition function $\delta : S \times A \times A \mapsto \mathcal{D}(S)$, which associates with every state $s \in S$ and moves $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$ a probability distribution $\delta(s, a_1, a_2) \in \mathcal{D}(S)$ for the successor state.

Plays. At every state $s \in S$, player 1 chooses a move $a_1 \in \Gamma_1(s)$, and simultaneously and independently player 2 chooses a move $a_2 \in \Gamma_2(s)$. The game then proceeds to the successor state t with probability $\delta(s, a_1, a_2)(t)$, for all $t \in S$. For all states $s \in S$ and moves $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$, we indicate by $\text{Dest}(s, a_1, a_2) = \text{Supp}(\delta(s, a_1, a_2))$ the set of possible successors of s when moves a_1, a_2 are selected. A *path* or a *play* of G is an infinite sequence $\omega = \langle s_0, s_1, s_2, \dots \rangle$ of states in S

such that for all $k \geq 0$, there are moves $a_1^k \in \Gamma_1(s_k)$ and $a_2^k \in \Gamma_2(s_k)$ such that $s_{k+1} \in \text{Dest}(s_k, a_1^k, a_2^k)$. We denote by Ω the set of all paths. We denote by θ_i the random variable that denotes the i -th state of a path. For a play $\omega = \langle s_0, s_1, s_2, \dots \rangle \in \Omega$, we define $\text{Inf}(\omega) = \{s \in S \mid s_k = s \text{ for infinitely many } k \geq 0\}$ to be the set of states that occur infinitely often in ω .

Special Classes of Games. We consider the following special classes of concurrent games.

1. *Turn-based stochastic games.* A game structure G is *turn-based stochastic* if at every state at most one player can choose among multiple moves; that is, for every state $s \in S$ there exists at most one $i \in \{1, 2\}$ with $|\Gamma_i(s)| > 1$.
2. *Markov decision processes.* A game structure is a *player-1 Markov decision process (MDP)* if for all $s \in S$ we have $|\Gamma_2(s)| = 1$, i.e., only player 1 has choice of actions in the game. Similarly, a game structure is a *player-2 MDP* if for all $s \in S$ we have $|\Gamma_1(s)| = 1$.
3. *Markov chains.* A game structure is a Markov chain if for all $s \in S$ we have $|\Gamma_1(s)| = 1$ and $|\Gamma_2(s)| = 1$. Hence in a Markov chain the players do not matter, and for the rest of the paper a Markov chain consists of a tuple (S, δ) where $\delta : S \mapsto \mathcal{D}(S)$ is the probabilistic transition function.

Strategies. A *strategy* for a player is a recipe that describes how to extend a play. Formally, a strategy for player $i \in \{1, 2\}$ is a mapping $\pi_i : S^+ \mapsto \mathcal{D}(A)$ that associates with every nonempty finite sequence $x \in S^+$ of states, representing the past history of the game, a probability distribution $\pi_i(x)$ used to select the next move. The strategy π_i can prescribe only moves that are available to player i ; that is, for all sequences $x \in S^*$ and states $s \in S$, we require that $\text{Supp}(\pi_i(x \cdot s)) \subseteq \Gamma_i(s)$. We denote by Π_i the set of all strategies for player $i \in \{1, 2\}$.

Given a state $s \in S$ and two strategies $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$, we define $\text{Outcome}(s, \pi_1, \pi_2) \subseteq \Omega$ to be the set of paths that can be followed by the game, when the game starts from s and the players use the strategies π_1 and π_2 . Formally, $\langle s_0, s_1, s_2, \dots \rangle \in \text{Outcome}(s, \pi_1, \pi_2)$ if $s_0 = s$ and if for all $k \geq 0$ there exist moves $a_1^k \in \Gamma_1(s_k)$ and $a_2^k \in \Gamma_2(s_k)$ such that (i) $\pi_1(s_0, \dots, s_k)(a_1^k) > 0$; (ii) $\pi_2(s_0, \dots, s_k)(a_2^k) > 0$; and (iii) $s_{k+1} \in \text{Dest}(s_k, a_1^k, a_2^k)$. Once the starting state s and the strategies π_1 and π_2 for the two players have been chosen, the probabilities of events are uniquely defined [28], where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of paths¹. For an event $\mathcal{A} \subseteq \Omega$, we denote by $\text{Pr}_s^{\pi_1, \pi_2}(\mathcal{A})$ the probability that a path belongs to \mathcal{A} when the game starts from s and the players use the strategies π_1 and π_2 .

Classification of Strategies. We consider the following special classes of strategies.

1. (*Pure*). A strategy π is *pure (deterministic)* if for all $x \in S^+$ there exists $a \in A$ such that $\pi(x)(a) = 1$. Thus, deterministic strategies are equivalent to functions $S^+ \mapsto A$.

¹ To be precise, we should define events as measurable sets of paths *sharing the same initial state*, and we should replace our events with families of events, indexed by their initial state. However, our (slightly) improper definition leads to more concise notation.

2. (*Finite-memory*). Strategies in general are *history-dependent* and can be represented as follows: let M be a set called *memory* to remember the history of plays (the set M can be infinite in general). A strategy with memory can be described as a pair of functions: (a) a *memory update* function $\pi_u : S \times M \mapsto M$, that given the memory M with the information about the history and the current state updates the memory; and (b) a *next move* function $\pi_n : S \times M \mapsto \mathcal{D}(A)$ that given the memory and the current state specifies the next move of the player. A strategy is *finite-memory* if the memory M is finite.
3. (*Memoryless*). A *memoryless* strategy is independent of the history of play and only depends on the current state. Formally, for a memoryless strategy π we have $\pi(x \cdot s) = \pi(s)$ for all $s \in S$ and all $x \in S^*$. Thus memoryless strategies are equivalent to functions $S \mapsto \mathcal{D}(A)$.
4. (*Pure memoryless*). A strategy is *pure memoryless* if it is both pure and memoryless. Pure memoryless strategies neither use memory, nor use randomization and are equivalent to functions $S \mapsto A$.

Qualitative Objectives. We specify *qualitative* objectives for the players by providing the set of *winning plays* $\Phi \subseteq \Omega$ for each player. In this paper we study only zero-sum games [23,17], where the objectives of the two players are complementary. A general class of objectives are the Borel objectives [20]. A *Borel objective* $\Phi \subseteq S^\omega$ is a Borel set in the Cantor topology on S^ω . In this paper we consider ω -regular objectives, which lie in the first $2^{1/2}$ levels of the Borel hierarchy (i.e., in the intersection of Σ_3 and Π_3) [27]. All ω -regular objectives can be specified as parity objectives, and hence in this work we focus on parity objectives, and they are defined as follows.

- *Parity objectives.* For $c, d \in \mathbb{N}$, we let $[c..d] = \{c, c + 1, \dots, d\}$. Let $p : S \mapsto [0..d]$ be a function that assigns a *priority* $p(s)$ to every state $s \in S$, where $d \in \mathbb{N}$. The *Even parity objective* requires that the minimum priority visited infinitely often is even. Formally, the set of winning plays is defined as $\text{Parity}(p) = \{\omega \in \Omega \mid \min(p(\text{Inf}(\omega))) \text{ is even}\}$.

Quantitative Objectives. *Quantitative* objectives are measurable functions $f : \Omega \mapsto \mathbb{R}$. We will consider *multi-discounted* objective functions, as there is a close connection established between concurrent games with multi-discounted objectives and concurrent games with parity objectives. Given a concurrent game structure with state space S , let λ be a *discount vector* that assigns for all $s \in S$ a discount factor $0 < \lambda(s) < 1$ (unless otherwise mentioned we will always consider discount vectors λ such that for all $s \in S$ we have $0 < \lambda(s) < 1$). Let $r : S \mapsto \mathbb{R}$ be a reward function that assigns a real-valued reward $r(s)$ to every state $s \in S$. The multi-discounted objective function $\text{MDT}(\lambda, r) : \Omega \mapsto \mathbb{R}$ maps every path to the mean-discounted reward of the path. Formally, the function is defined as follows: for a path $\omega = s_0 s_1 s_2 \dots$ we have

$$\text{MDT}(\lambda, r)(\omega) = \frac{\sum_{j=0}^{\infty} (\prod_{i=0}^j \lambda(s_i)) \cdot r(s_j)}{\sum_{j=0}^{\infty} (\prod_{i=0}^j \lambda(s_i))}.$$

Also note that a parity objective Φ can be interpreted as a function $\Phi : \Omega \mapsto \{0, 1\}$ by simply considering the characteristic function that assigns 1 to paths that belong to Φ and 0 otherwise.

Values, Optimality, ε -Optimality. Given an objective Φ which is a measurable function $\Phi : \Omega \mapsto \mathbb{R}$, we define the *value* for player 1 of game G with objective Φ from the state $s \in S$ as $\text{Val}(G, \Phi)(s) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \mathbb{E}_s^{\pi_1, \pi_2}(\Phi)$; i.e., the value is the maximal expectation with which player 1 can guarantee the satisfaction of Φ against all player 2 strategies. Given a player-1 strategy π_1 , we use the notation $\text{Val}^{\pi_1}(G, \Phi)(s) = \inf_{\pi_2 \in \Pi_2} \mathbb{E}_s^{\pi_1, \pi_2}(\Phi)$. A strategy π_1 for player 1 is *optimal* for an objective Φ if for all states $s \in S$, we have $\text{Val}^{\pi_1}(G, \Phi)(s) = \text{Val}(G, \Phi)(s)$. For $\varepsilon > 0$, a strategy π_1 for player 1 is *ε -optimal* if for all states $s \in S$, we have $\text{Val}^{\pi_1}(G, \Phi)(s) \geq \text{Val}(G, \Phi)(s) - \varepsilon$. The notion of values, optimal and ε -optimal strategies for player 2 are defined analogously. The following theorem summarizes the results in literature related to determinacy and memory complexity of concurrent games and its sub-classes for parity and multi-discounted objectives.

Theorem 1. *The following assertions hold:*

1. (Determinacy [21]). *For all concurrent game structures and for all parity and multi-discounted objectives Φ we have $\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \mathbb{E}_s^{\pi_1, \pi_2}(\Phi) = \inf_{\pi_2 \in \Pi_2} \sup_{\pi_1 \in \Pi_1} \mathbb{E}_s^{\pi_1, \pi_2}(\Phi)$.*
2. (Memory complexity). *For all concurrent game structures and for all multi-discounted objectives Φ , randomized memoryless optimal strategies exist [25]. For all turn-based stochastic game structures and for all multi-discounted objectives Φ , pure memoryless optimal strategies exist [17]. For all turn-based stochastic game structures and for all parity objectives Φ , pure memoryless optimal strategies exist [629]. In general optimal strategies need not exist in concurrent games with parity objectives, and ε -optimal strategies, for $\varepsilon > 0$, need both randomization and infinite memory in general [11].*

The results of [12] established that the value of concurrent games with certain special multi-discounted objectives can be characterized as valuations of quantitative discounted μ -calculus formula. In the limit, the value function of the discounted μ -calculus formula characterizes the value function of concurrent games with parity objectives. An elegant interpretation of the result was given in [19], and from the interpretation we obtain the following theorem.

Theorem 2 ([12,19]). *Let G be a concurrent game structure with a parity objective Φ defined by a priority function p . Let r be a reward function that assigns reward 1 to even priority states and reward 0 to odd priority states. Then there exists an order $s_1 s_2 \dots s_n$ on the states (where $S = \{s_1, s_2, \dots, s_n\}$) dependent only on the priority function p such that $\text{Val}(G, \Phi) = \lim_{\lambda(s_1) \rightarrow 1} \lim_{\lambda(s_2) \rightarrow 1} \dots \lim_{\lambda(s_n) \rightarrow 1} \text{Val}(G, \text{MDT}(\lambda, r))$; in other words, if we consider the value function $\text{Val}(G, \text{MDT}(\lambda, r))$ with the multi-discounted objective and take the limit of the discount factors to 1 in the order of the states we obtain the value function for the parity objective.*

We now present notions related to *structure equivalent* game structures and distances.

Structure Equivalent Game Structures. Given two game structures $G_1 = \langle S, A, \Gamma_1, \Gamma_2, \delta_1 \rangle$ and $G_2 = \langle S, A, \Gamma_1, \Gamma_2, \delta_2 \rangle$ on the same state and action space, with different transition function, we say that G_1 and G_2 are *structure equivalent*

(denoted $G_1 \equiv G_2$) if for all $s \in S$ and all $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$ we have $\text{Supp}(\delta_1(s, a_1, a_2)) = \text{Supp}(\delta_2(s, a_1, a_2))$. Similarly, two Markov chains $G_1 = (S, \delta_1)$ and $G_2 = (S, \delta_2)$ are structurally equivalent (denoted $G_1 \equiv G_2$) if for all $s \in S$ we have $\text{Supp}(\delta_1(s)) = \text{Supp}(\delta_2(s))$. For a game structure G (resp. Markov chain G), we denote by $\llbracket G \rrbracket_{\equiv}$ the set of all game structures (resp. Markov chains) that are structurally equivalent to G .

Ratio and Absolute Distances. Given two game structures $G_1 = \langle S, A, \Gamma_1, \Gamma_2, \delta_1 \rangle$ and $G_2 = \langle S, A, \Gamma_1, \Gamma_2, \delta_2 \rangle$, the *absolute distance* of the game structures is maximum absolute difference in the transition probabilities. Formally, $\text{dist}_A(G_1, G_2) = \max_{s, t \in S, a \in \Gamma_1(s), b \in \Gamma_2(s)} |\delta_1(s, a, b)(t) - \delta_2(s, a, b)(t)|$. The absolute distance for two Markov chains $G_1 = (S, \delta_1)$ and $G_2 = (S, \delta_2)$ is $\text{dist}_A(G_1, G_2) = \max_{s, t \in S} |\delta_1(s)(t) - \delta_2(s)(t)|$. We now define the ratio distance between two structurally equivalent game structures and Markov chains. Let G_1 and G_2 be two structurally equivalent game structures. The *ratio distance* is defined on the ratio of the transition probabilities. Formally,

$$\text{dist}_R(G_1, G_2) = \max \left\{ \frac{\delta_1(s, a, b)(t)}{\delta_2(s, a, b)(t)}, \frac{\delta_2(s, a, b)(t)}{\delta_1(s, a, b)(t)} \mid s \in S, a \in \Gamma_1(s), b \in \Gamma_2(s), t \in \text{Supp}(\delta_1(s, a, b)) = \text{Supp}(\delta_2(s, a, b)) \right\} - 1$$

The ratio distance between two structurally equivalent Markov chains G_1 and G_2 is $\max \left\{ \frac{\delta_1(s)(t)}{\delta_2(s)(t)}, \frac{\delta_2(s)(t)}{\delta_1(s)(t)} \mid s \in S, t \in \text{Supp}(\delta_1(s)) = \text{Supp}(\delta_2(s)) \right\} - 1$.

Remarks about the Distance Functions. We first remark that the ratio distance is not necessarily a metric. Consider the Markov chain with state space $S = \{s, s'\}$ and let $\varepsilon \in (0, 1/7)$. For $k = 1, 2, 5$ consider the transition functions δ_k such that $\delta_k(t)(s) = 1 - \delta_k(t)(s') = k \cdot \varepsilon$, for all $t \in S$. Let G_k be the Markov chain with transition function δ_k . Then we have $\text{dist}_R(G_1, G_2) = 1$, $\text{dist}_R(G_2, G_5) = \frac{3}{2}$ and $\text{dist}_R(G_1, G_5) = 4$, and hence $\text{dist}_R(G_1, G_2) + \text{dist}_R(G_2, G_5) < \text{dist}_R(G_1, G_5)$. The above example is from [26]. Also note that dist_R is only defined for structurally equivalent game structures, and without the assumption dist_R is ∞ . We also remark that the absolute distance that measures the difference in the transition probabilities is the most intuitive measure for the difference of two game structures.

Proposition 1. *Let G_1 be a game structure (resp. Markov chain) such that the minimum positive transition probability is $\eta > 0$. For all game structures (resp. Markov chains) $G_2 \in \llbracket G_1 \rrbracket_{\equiv}$ we have $\text{dist}_R(G_1, G_2) \leq \frac{\text{dist}_A(G_1, G_2)}{\eta}$.*

Notation for Fixing Strategies. Given a concurrent game structure $G = \langle S, A, \Gamma_1, \Gamma_2, \delta \rangle$, let π_1 be a randomized memoryless strategy. Fixing the strategy π_1 in G we obtain a player-2 MDP, denoted as $G \upharpoonright \pi_1$, defined as follows: (1) the state space is S ; (2) for all $s \in S$ we have $\Gamma_1(s) = \{\perp\}$ (hence it is a player-2 MDP); (3) the new transition function δ_{π_1} is defined as follows: for all $s \in S$ and all $b \in \Gamma_2(s)$ we have $\delta_{\pi_1}(s, \perp, b)(t) = \sum_{a \in \Gamma_1(s)} \pi_1(s)(a) \cdot \delta(s, a, b)(t)$. Similarly if we fix a

randomized memoryless strategy π_1 in an MDP G we obtain a Markov chain, denoted as $G \upharpoonright \pi_1$. The following proposition is straightforward to verify from the definitions.

Proposition 2. *Let G_1 and G_2 be two concurrent game structures (resp. MDPs) that are structurally equivalent. Let π_1 be a randomized memoryless strategy. Then $dist_A(G_1 \upharpoonright \pi_1, G_2 \upharpoonright \pi_1) \leq dist_A(G_1, G_2)$ and $dist_R(G_1 \upharpoonright \pi_1, G_2 \upharpoonright \pi_1) \leq dist_R(G_1, G_2)$.*

3 Markov Chains with Multi-discounted and Parity Objectives

In this section we consider Markov chains with multi-discounted and parity objectives. We present a bound on the difference of value functions of two structurally equivalent Markov chains that is dependent on the distance between the Markov chains and is *independent* of the discount factors. The result for parity objectives is then a consequence of our result for multi-discounted objectives and Theorem 2. Our result crucially depends on a result of Friedlin and Wentzell for Markov chains and we present this result below, and then use it to obtain the main result of the section.

Result of Friedlin and Wentzell. Let (S, δ) be a Markov chain and let s_0 be the initial state. Let $C \subset S$ be a proper subset of S and let us denote by $ex_C = \inf\{n \in \mathbb{N} \mid \theta_n \notin C\}$ the first hitting time to the set $S \setminus C$ of states (or the first exit time from set C) (recall that θ_n is the random variable to denote the n -th state of a path). Let $\mathcal{F}(C, S) = \{f : C \mapsto S\}$ denote the set of all functions from C to S . For every $f \in \mathcal{F}(C, S)$ we define a directed graph $G_f = (S, E_f)$ where $(s, t) \in E_f$ iff $f(s) = t$. Let $\alpha_f = 1$ if the directed graph G_f has no directed cycles (i.e., G_f is a directed acyclic graph); and $\alpha_f = 0$ otherwise. Observe that since f is a function, for every $s \in C$ there is exactly one path that starts at s . For every $s \in C$ and every $t \in S$, let $\beta_f(s, t) = 1$ if the directed path that leaves s in G_f reaches t , otherwise $\beta_f(s, t) = 0$. We now state a result that can be obtained as a special case of the result from Friedlin and Wentzell [18]. Below we use the formulation of the result as presented in [26] (Lemma 2 of [26]).

Theorem 3 (Friedlin-Wentzell result [18]). *Let (S, δ) be a Markov chain, and let $C \subset S$ be a proper subset of S such that $\Pr_s(ex_C < \infty) > 0$ for every $s \in C$ (i.e., from all $s \in C$ with positive probability the first hitting time to the complement set is finite). Then for every initial state $s_1 \in C$ and for every $t \notin C$ we have*

$$\Pr_{s_1}(\theta_{ex_C} = t) = \frac{\sum_{f \in \mathcal{F}(C, S)} (\beta_f(s_1, t) \cdot \prod_{s \in C} \delta(s)(f(s)))}{\sum_{f \in \mathcal{F}(C, S)} (\alpha_f \cdot \prod_{s \in C} \delta(s)(f(s)))}, \tag{1}$$

in other words, the probability that the exit state is t when the starting state is s_1 is given by the expression on the right hand side (very informally the right hand side is the normalized polynomial expression for exit probabilities).

Value Function Difference for Markov Chains. We will use the result of Theorem 3 to obtain bounds on the value functions of Markov chains. We start with the notion of mean-discounted time.

Mean-Discounted Time. Given a Markov chain (S, δ) and a discount vector λ , we define for every state $s \in S$, the *mean-discounted time* the process is in the state s . We first define the mean-discounted time function $\text{MDT}(\lambda, s) : \Omega \mapsto \mathbb{R}$ that maps every path to the mean-discounted time that the state s is visited, and the function is formally defined as follows: for a path $\omega = s_0 s_1 s_2 \dots$ we have

$$\text{MDT}(\lambda, s)(\omega) = \frac{\sum_{j=0}^{\infty} (\prod_{i=0}^j \lambda(s_i)) \cdot \mathbf{1}_{s_j=s}}{\sum_{j=0}^{\infty} (\prod_{i=0}^j \lambda(s_i))};$$

where $\mathbf{1}_{s_j=s}$ is the indicator function. The expected mean-discounted time function for a Markov chain G with transition function δ is defined as follows: $\text{MT}(s_1, s, G, \lambda) = \mathbb{E}_{s_1}[\text{MDT}(\lambda, s)]$, i.e., it is the expected mean-discounted time for s when the starting state is s_1 , where the expectation measure is defined by the Markov chain with transition function δ . We now present a lemma that shows the value function for multi-discounted Markov chains can be expressed as ratio of two polynomials (the result is obtained as a simple extension of a result of Solan [26]).

Lemma 1. *For Markov chains defined on state space S , for all initial states s_0 , for all states s , for all discount vectors λ , there exists two polynomials $g_1(\cdot)$ and $g_2(\cdot)$ in $|S|^2$ variables $x_{t,t'}$, where $t, t' \in S$ such that the following conditions hold:*

1. *the polynomials have degree at most $|S|$ with non-negative coefficients; and*
2. *for all transition functions δ over S we have $\text{MT}(s_0, s, G, \lambda) = \frac{g_1(\delta)}{g_2(\delta)}$, where $G = (S, \delta)$, $g_1(\delta)$ and $g_2(\delta)$ denote the values of the function g_1 and g_2 such that all the variables $x_{t,t'}$ is instantiated with values $\delta(t)(t')$ as given by the transition function δ .*

Proof. (Sketch). We present a sketch of the proof (details in [3]). Fix a discount vector λ . We construct a Markov chain $\bar{G} = (\bar{S}, \bar{\delta})$ as follows: $\bar{S} = S \cup S_1$, where S_1 is a copy of states of S (and for a state $s \in S$ we denote its corresponding copy as s_1); and the transition function $\bar{\delta}$ is defined below

1. $\bar{\delta}(s_1)(s_1) = 1$ for all $s_1 \in S_1$ (i.e., all copy states are absorbing);
2. for $s \in S$ we have

$$\bar{\delta}(s)(t) = \begin{cases} (1 - \lambda(s)) & t = s_1; \\ \lambda(s) \cdot \delta(s)(t) & t \in S; \\ 0 & t \in S_1 \setminus s_1; \end{cases}$$

i.e., it goes to the copy with probability $(1 - \lambda(s))$, it follows the transition δ in the original copy with probabilities multiplied by $\lambda(s)$.

We first show that for all s_0 and s we have $\text{MT}(s_0, s, G, \lambda) = \text{Pr}_{s_0}^{\bar{\delta}}(\theta_{\text{ex}_S} = s_1)$; i.e., the expected mean-discounted time in s when the original Markov chain starts in s_0 is the probability in the Markov chain $(\bar{S}, \bar{\delta})$ that the first hitting state out of S is the copy s_1 of the state s . The claim is easy to verify as both $(\text{MT}(s_0, s, G, \lambda))_{s_0 \in S}$

and $(\Pr_{s_0}^{\bar{\delta}}(\theta_{\text{ex}_S} = s_1))_{s_0 \in S}$ are the unique solution of the following system of linear equations: for all $t \in S$ we have $y_t = (1 - \lambda(t)) \cdot \mathbf{1}_{t=s} + \sum_{z \in S} \lambda(t) \cdot \delta(t)(z) \cdot y_z$.

We now claim that $\Pr_{s_0}^{\bar{\delta}}(\text{ex}_S < \infty) > 0$ for all $s_0 \in S$. This follows since for all $s \in S$ we have $\bar{\delta}(s)(s_1) = (1 - \lambda(s)) > 0$ and since $s_1 \notin S$ we have $\Pr_{s_0}^{\bar{\delta}}(\text{ex}_S = 2) = (1 - \lambda(s_0)) > 0$. Now we observe that we can apply Theorem 3 on the Markov chain $\bar{G} = (\bar{S}, \bar{\delta})$ with S as the set C of states of Theorem 3, and obtain the result. Indeed the terms α_f and $\beta_f(s, t)$ are independent of δ , and the two products of Equation (1) each contains at most $|S|$ terms of the form $\bar{\delta}(s)(t)$ for $s, t \in \bar{S}$. Thus the desired result follows. ■

Lemma 2. *Let $h(x_1, x_2, \dots, x_k)$ be a polynomial function with non-negative coefficients of degree at most n . Let $\varepsilon > 0$ and $y, y' \in \mathbb{R}^k$ be two non-negative vectors such that for all $i = 1, 2, \dots, k$ we have $\frac{1}{1+\varepsilon} \leq \frac{y_i}{y'_i} \leq 1 + \varepsilon$. Then we have $(1 + \varepsilon)^{-n} \leq \frac{h(y)}{h(y')} \leq (1 + \varepsilon)^n$.*

Lemma 3. *Let $G_1 = (S, \delta)$ and $G_2 = (S, \delta')$ be two structurally equivalent Markov chains. For all non-negative reward functions $r : S \mapsto \mathbb{R}$ such that the reward function is bounded by 1, for all discount vectors λ , for all $s \in S$ we have $|\text{Val}(G_1, \text{MDT}(\lambda, r))(s) - \text{Val}(G_2, \text{MDT}(\lambda, r))(s)| \leq (1 + \text{dist}_R(G_1, G_2))^{2 \cdot |S|} - 1$; i.e., the absolute difference of the value functions for the multi-discounted objective is bounded by $(1 + \text{dist}_R(G_1, G_2))^{2 \cdot |S|} - 1$.*

The proof of Lemma 3 uses Lemma 1 and Lemma 2 and details available in [3].

Theorem 4. *Let $G_1 = (S, \delta)$ and $G_2 = (S, \delta')$ be two structurally equivalent Markov chains. Let η be the minimum positive transition probability in G_1 . The following assertions hold:*

1. *For all non-negative reward functions $r : S \mapsto \mathbb{R}$ such that the reward function is bounded by 1, for all discount vectors λ , for all $s \in S$ we have*

$$\begin{aligned} |\text{Val}(G_1, \text{MDT}(\lambda, r))(s) - \text{Val}(G_2, \text{MDT}(\lambda, r))(s)| &\leq (1 + \varepsilon_R)^{2 \cdot |S|} - 1 \\ &\leq (1 + \varepsilon_A)^{2 \cdot |S|} - 1 \end{aligned}$$

2. *For all parity objectives Φ and for all $s \in S$ we have*

$$|\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2, \Phi)(s)| \leq (1 + \varepsilon_R)^{2 \cdot |S|} - 1 \leq (1 + \varepsilon_A)^{2 \cdot |S|} - 1$$

where $\varepsilon_R = \text{dist}_R(G_1, G_2)$ and $\varepsilon_A = \frac{\text{dist}_A(G_1, G_2)}{\eta}$.

Proof. The first part follows from Lemma 3 and Proposition 1. The second part follows from part 1, the fact the value function for parity objectives is obtained as the limit of multi-discounted objectives (Theorem 2), and the fact the bound for part 1 is independent of the discount factors (hence independent of taking the limit). ■

Remark on Structural Assumption in the Proof. The result of the previous theorem depends on the structural equivalence assumption in two crucial ways. They are as

follows: (1) Proposition [1](#) that establishes the relation of $dist_R$ and $dist_A$ only holds with the assumption of structural equivalence; and (2) without the structural equivalence assumption $dist_R$ is ∞ , and hence without the assumption the bound of the previous theorem is ∞ , which is a trivial bound. We will later show (in Example [1](#)) that the structural equivalence assumption is necessary.

4 Value Continuity for Parity Objectives

In this section we show two results: first we show robustness of strategies and present quantitative bounds on value functions for turn-based stochastic games and then we show continuity for concurrent parity games.

4.1 Bounds for Structurally Equivalent Turn-Based Stochastic Parity Games

In this section we present quantitative bounds for robustness of optimal strategies in structurally equivalent turn-based stochastic games. For every $\varepsilon > 0$, we present a bound $\beta > 0$, such that if the distance of the structurally equivalent turn-based stochastic games differs by at most β , then any pure memoryless optimal strategy in one game is ε -optimal in the other. The result is first shown for MDPs and then extended to turn-based stochastic games (both proofs are in [\[3\]](#)).

Theorem 5. *Let G_1 be a turn-based stochastic game such that the minimum positive transition probability is $\eta > 0$. The following assertions hold:*

1. *For all turn-based stochastic games $G_2 \in \llbracket G_1 \rrbracket_{\equiv}$, for all parity objectives Φ and for all $s \in S$ we have*

$$\begin{aligned} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2, \Phi)(s)| &\leq (1 + dist_R(G_1, G_2))^{2 \cdot |S|} - 1 \\ &\leq \left(1 + \frac{dist_A(G_1, G_2)}{\eta}\right)^{2 \cdot |S|} - 1 \end{aligned}$$

2. *For $\varepsilon > 0$, let $\beta \leq \frac{\eta}{2} \cdot \left(1 + \frac{\varepsilon}{2}\right)^{\frac{1}{2 \cdot |S|}} - 1$. For all $G_2 \in \llbracket G_1 \rrbracket_{\equiv}$ such that $dist_A(G_1, G_2) \leq \beta$, for all parity objectives Φ , every pure memoryless optimal strategy π_1 in G_1 is an ε -optimal strategy in G_2 .*

4.2 Value Continuity for Concurrent Parity Games

In this section we show value continuity for structurally equivalent concurrent parity games, and show with an example on Markov chains that the continuity property breaks without the structural equivalence assumption. Finally with an example on Markov chains we show the our quantitative bounds are asymptotically optimal for small distance values. We start with a lemma for MDPs.

Lemma 4. *Let G_1 and G_2 be two structurally equivalent MDPs. Let η be the minimum positive transition probability in G_1 . For all non-negative reward functions $r : S \mapsto \mathbb{R}$ such that the reward function is bounded by 1, for all discount vectors λ , for all $s \in S$ we have*

$$\begin{aligned} |\text{Val}(G_1, \text{MDT}(\lambda, r))(s) - \text{Val}(G_2, \text{MDT}(\lambda, r))(s)| &\leq (1 + \text{dist}_R(G_1, G_2))^{2 \cdot |S|} - 1 \\ &\leq \left(1 + \frac{\text{dist}_A(G_1, G_2)}{\eta}\right)^{2 \cdot |S|} - 1 \end{aligned}$$

The main idea of the proof of the above lemma is to fix a pure memoryless optimal strategy and then use the results for Markov chains. Using the same proof idea, along with randomized memoryless optimal strategies for concurrent game structures and the above lemma, we obtain the following lemma (the result is identical to the previous lemma, but for concurrent game structures instead of MDPs).

Lemma 5. *Let G_1 and G_2 be two structurally equivalent concurrent game structures. Let η be the minimum positive transition probability in G_1 . For all non-negative reward functions $r : S \mapsto \mathbb{R}$ such that the reward function is bounded by 1, for all discount vectors λ , for all $s \in S$ we have*

$$\begin{aligned} |\text{Val}(G_1, \text{MDT}(\lambda, r))(s) - \text{Val}(G_2, \text{MDT}(\lambda, r))(s)| &\leq (1 + \text{dist}_R(G_1, G_2))^{2 \cdot |S|} - 1 \\ &\leq \left(1 + \frac{\text{dist}_A(G_1, G_2)}{\eta}\right)^{2 \cdot |S|} - 1 \end{aligned}$$

We now present the main theorem that depends on Lemma 5.

Theorem 6. *Let G_1 and G_2 be two structurally equivalent concurrent game structures. Let η be the minimum positive transition probability in G_1 . For all parity objectives Φ and for all $s \in S$ we have*

$$\begin{aligned} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2, \Phi)(s)| &\leq (1 + \text{dist}_R(G_1, G_2))^{2 \cdot |S|} - 1 \\ &\leq \left(1 + \frac{\text{dist}_A(G_1, G_2)}{\eta}\right)^{2 \cdot |S|} - 1 \end{aligned}$$

Proof. The result follows from Theorem 2, Lemma 5 and the fact that the bound of Lemma 5 are independent of the discount factors and hence independent of taking the limits. ■

In the following theorem we show that for structurally equivalent game structures, for all parity objectives, the value function is continuous in the absolute distance between the game structures. We have already remarked (after Theorem 4) that the structural equivalence assumption is required in our proofs, and we show in Example 1 that this assumption is necessary.

Theorem 7. *For all concurrent game structures G_1 , for all parity objectives Φ*

$$\lim_{\varepsilon \rightarrow 0} \sup_{G_2 \in \llbracket G_1 \rrbracket_{\equiv, \text{dist}_A(G_1, G_2) \leq \varepsilon}} \sup_{s \in S} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2, \Phi)(s)| = 0.$$

Proof. Let $\eta > 0$ be the minimum positive transition probability in G_1 . By Theorem 6 we have

$$\lim_{\varepsilon \rightarrow 0} \sup_{G_2 \in \llbracket G_1 \rrbracket_{\equiv}, \text{dist}_A(G_1, G_2) \leq \varepsilon} \sup_{s \in S} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2, \Phi)(s)| \leq \lim_{\varepsilon \rightarrow 0} \left(1 + \frac{\varepsilon}{\eta}\right)^{2 \cdot |S|} - 1$$

The above limit equals to 0, and the desired result follows. ■

Example 1 (Structurally equivalence assumption necessary). In this example we show that in Theorem 7 the structural equivalence assumption is necessary, and thereby show that the result is tight. We show an Markov chain G_1 and a family of Markov chains G_2^ε , for $\varepsilon > 0$, such that $\text{dist}_A(G_1, G_2^\varepsilon) \leq \varepsilon$ (but G_1 is not structurally equivalent to G_2^ε) with a parity objective Φ and we have $\lim_{\varepsilon \rightarrow 0} \sup_{s \in S} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2^\varepsilon, \Phi)(s)| = 1$. The Markov chains G_1 and G_2^ε are defined over the state space $\{s_0, s_1\}$, and in G_1 both states have self-loops with probability 1, and in G_2^ε the self-loop at s_0 has probability $1 - \varepsilon$ and the transition probability from s_0 to s_1 is ε (for details see [3]). Clearly, $\text{dist}_A(G_1, G_2^\varepsilon) = \varepsilon$. The parity objective Φ requires to visit the state s_1 infinitely often (i.e., assign priority 2 to s_1 and priority 1 to s_0). Then we have $\text{Val}(G_1, \Phi)(s_0) = 0$ as the state s_0 is never left, whereas in G_2^ε the state s_1 is the only closed recurrent set of the Markov chain and hence reached with probability 1 from s_0 . Hence $\text{Val}(G_2^\varepsilon, \Phi)(s_0) = 1$. It follows that $\lim_{\varepsilon \rightarrow 0} \sup_{s \in S} |\text{Val}(G_1, \Phi)(s) - \text{Val}(G_2^\varepsilon, \Phi)(s)| = 1$. ■

Example 2 (Asymptotically tight bound for small distances). We now show that our quantitative bound for the value function difference is asymptotically optimal for small distances. Let us denote the absolute distance as ε , and the quantitative bound we obtain in Theorem 6 is $(1 + \frac{\varepsilon}{\eta})^{2 \cdot |S|} - 1$, and if ε is small, then we obtain the following approxi-

mate bound: $\left(1 + \frac{\varepsilon}{\eta}\right)^{2 \cdot |S|} - 1 \approx 1 + 2 \cdot |S| \cdot \frac{\varepsilon}{\eta} - 1 = 2 \cdot |S| \cdot \frac{\varepsilon}{\eta}$. We now illustrate with an example (on structurally equivalent Markov chains) where the difference in the value function is $O(|S| \cdot \varepsilon)$, for small ε . Consider the Markov chain defined on state space $S = \{s_0, s_1, \dots, s_{2n-1}, s_{2n}\}$ as follows: states s_0 and s_{2n} are absorbing (states with self-loops of probability 1) and for a state $1 \leq i \leq 2n - 1$ we have $\delta(s_i)(s_{i-1}) = \frac{1}{2} + \varepsilon$; and $\delta(s_i)(s_{i+1}) = \frac{1}{2} - \varepsilon$; i.e., we have a Markov chain defined on a line from 0 to $2n$ (with 0 and $2n$ absorbing states) and the chain moves towards 0 with probability $\frac{1}{2} + \varepsilon$ and towards $2n$ with probability $\frac{1}{2} - \varepsilon$ (for complete details see [3]). Our goal is to estimate the probability to reach the state s_0 , and let v_i denote the probability to reach s_0 from the starting state s_i . We show (details in [3]) that if $\varepsilon = 0$, then $v_n = \frac{1}{2}$ and for $0 < \varepsilon < \frac{1}{2}$, such that ε is close to 0, we have $v_n = \frac{1}{2} + n \cdot \varepsilon$. Observe that the Markov chain obtained for $\varepsilon = 0$ and $\frac{1}{2} > \varepsilon > 0$ are structurally equivalent. Thus the desired result follows. ■

5 Conclusion

In this work we studied the robustness and continuity property of concurrent and turn-based stochastic parity games with respect to small imprecision in the transition probabilities. We presented (i) quantitative bounds on difference of the value functions and

proved value continuity for concurrent parity games under the structural equivalence assumption, and (ii) showed robustness of all pure memoryless optimal strategies for structurally equivalent turn-based stochastic parity games. We also showed that the structural equivalence assumption is necessary and that our quantitative bounds are asymptotically optimal for small imprecision. We believe our results will find applications in robustness analysis of various other classes of stochastic games.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and Unrealizable Specifications of Reactive Systems. In: Ronchi Della Rocca, S., Ausiello, G., Dezanı-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. *Journal of the ACM* 49, 672–713 (2002)
3. Chatterjee, K.: Robustness of structurally equivalent concurrent parity games. *CoRR* 1107.2009 (2011)
4. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: The complexity of quantitative concurrent parity games. In: SODA 2006, pp. 678–687. ACM-SIAM (2004)
5. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: LICS 2004, pp. 160–169. IEEE (2004)
6. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA 2004, pp. 121–130. SIAM (2004)
7. Church, A.: Logic, arithmetic, and automata. In: *Proceedings of the International Congress of Mathematicians*, pp. 23–35. Institut Mittag-Leffler (1962)
8. Condon, A.: The complexity of stochastic games. *Information and Computation* 96(2), 203–224 (1992)
9. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. *Journal of the ACM* 42(4), 857–907 (1995)
10. de Alfaro, L.: *Formal Verification of Probabilistic Systems*. PhD thesis. Stanford University (1997)
11. de Alfaro, L., Henzinger, T.A.: Concurrent omega-regular games. In: LICS 2000, pp. 141–154. IEEE (2000)
12. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the Future in Systems Theory. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1022–1037. Springer, Heidelberg (2003)
13. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. In: STOC 2001, pp. 675–683. ACM Press (2001)
14. Derman, C.: *Finite State Markovian Decision Processes*. Academic Press (1970)
15. Dill, D.L.: *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits*. The MIT Press (1989)
16. Etesami, K., Yannakakis, M.: Recursive Concurrent Stochastic Games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 324–335. Springer, Heidelberg (2006)
17. Filar, J., Vrieze, K.: *Competitive Markov Decision Processes*. Springer, Heidelberg (1997)
18. Friedlin, M.I., Wentzell, A.D.: *Random perturbations of dynamical systems*. Springer, Heidelberg (1984)
19. Gimbert, H., Zielonka, W.: Discounting infinite games but how and why? *Electr. Notes Theor. Comput. Sci.* 119(1), 3–9 (2005)
20. Kechris, A.: *Classical Descriptive Set Theory*. Springer, Heidelberg (1995)

21. Martin, D.A.: The determinacy of Blackwell games. *The Journal of Symbolic Logic* 63(4), 1565–1581 (1998)
22. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: *POPL 1989*, pp. 179–190. ACM Press (1989)
23. Raghavan, T.E.S., Filar, J.A.: Algorithms for stochastic games — a survey. *ZOR — Methods and Models of Op. Res.* 35, 437–472 (1991)
24. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization* 25(1), 206–230 (1987)
25. Shapley, L.S.: Stochastic games. *Proc. Nat. Acad. Sci. USA* 39, 1095–1100 (1953)
26. Solan, E.: Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability* 16, 831–845 (2003)
27. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, vol. B, pp. 133–191. Elsevier (1990)
28. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state systems. In: *FOCS 1985*, pp. 327–338. IEEE Computer Society Press (1985)
29. Zielonka, W.: Perfect-Information Stochastic Parity Games. In: Walukiewicz, I. (ed.) *FOSSACS 2004*. LNCS, vol. 2987, pp. 499–513. Springer, Heidelberg (2004)

Subgame Perfection for Equilibria in Quantitative Reachability Games

Thomas Brihaye¹, Véronique Bruyère¹, Julie De Pril¹, and Hugo Gimbert²

¹ University of Mons - UMONS

Place du Parc 20, 7000 Mons, Belgium

{thomas.brihaye, veronique.bruyere, julie.depril}@umons.ac.be

² LaBRI & CNRS, Bordeaux, France

hugo.gimbert@labri.fr

Abstract. We study turn-based quantitative multiplayer non zero-sum games played on finite graphs with reachability objectives. In such games, each player aims at reaching his own goal set of states as soon as possible. A previous work on this model showed that Nash equilibria (resp. secure equilibria) are guaranteed to exist in the multiplayer (resp. two-player) case. The existence of secure equilibria in the multiplayer case remained, and is still an open problem. In this paper, we focus our study on the concept of subgame perfect equilibrium, a refinement of Nash equilibrium well-suited in the framework of games played on graphs. We also introduce the new concept of subgame perfect secure equilibrium. We prove the existence of subgame perfect equilibria (resp. subgame perfect secure equilibria) in multiplayer (resp. two-player) quantitative reachability games. Moreover, we provide an algorithm deciding the existence of secure equilibria in the multiplayer case.

1 Introduction

General framework. The construction of correct and efficient computer systems (hardware or software) is recognized as an extremely difficult task. To support the design and verification of such systems, mathematical logic, automata theory [16] and more recently model-checking [13] have been intensively studied. The efficiency of the model-checking approach is widely recognized when applied to systems that can be accurately modeled as a finite-state automaton. In contrast, the application of these techniques to more complex systems like embedded systems or distributed systems has been less successful. This could be partly explained by the following reasons: classical automata-based models do not faithfully capture the complex behavior of modern computational systems that are usually composed of several interacting components, also interacting with an environment that is only partially under control. One recent trend to improve the automata models used in the classical approach of verification is to generalize these models with the more flexible and mathematically deeper game-theoretic framework [22,23].

The first steps to extend computational models with concepts from game theory were done with the so-called two-player zero-sum games played on graphs [14].

Those games are adequate to model controller-environment interaction problems [26,27]. Moves of player 1 model actions of the controller whereas moves of player 2 model the uncontrollable actions of the environment, and a winning strategy for player 1 is an abstract form of a control program that enforces the control objective. However, only purely antagonist interactions between a controller and a hostile environment can be modeled in this framework. In order to study more complex systems with more than two components and objectives that are not necessarily antagonist, we need multiplayer non zero-sum games. Moreover, we do not look for winning strategies, but rather try to find relevant notions of equilibria, like the famous notion of *Nash equilibrium* [22]. We also consider the more recent concept of *secure equilibrium* [9] which is especially well-suited for assume-guarantee synthesis [11,12]. On the other hand, only qualitative objectives have been considered so far to specify, for example, that a player must be able to reach a target set of states in the underlying game graph. But, in line with the previous point, we also want to express and solve games for quantitative objectives where each player wants to force the play to reach a particular set of states within a given time bound, or within a given energy consumption limit. In summary, we need to study *equilibria* for *multiplayer non zero-sum* games played on graphs with *quantitative* objectives. This article provides some new results in this research direction, in particular it is another step in the quest for solution concepts well-suited for the computer-aided synthesis and verification of multi-agent systems.

Our contribution. We study turn-based multiplayer non zero-sum games played on finite graphs with quantitative reachability objectives, continuing work initiated in [6]. In this framework each player aims at reaching his own goal as soon as possible. In [6], among other results, it has been proved that a finite-memory Nash (resp. secure) equilibria always exists in multiplayer (resp. 2-player) games.

In this paper we consider alternative solution concepts to the classical notion of Nash equilibria. In particular, in the present framework of games on graphs, it is very natural to consider the notion of *subgame perfect equilibrium* [25]: a choice of strategies is not only required to be optimal for the initial vertex but also for every possible initial history of the game. Indeed if the initial state or the initial history of the system is not known, then a robust controller should be subgame perfect. We introduce a new and even stronger solution concept with the notion of subgame perfect *secure* equilibrium, which gathers both the sequential nature of subgame perfect equilibria and the verification-oriented aspects of secure equilibria. These different notions of equilibria are precisely defined in Section 2.

In this paper, we address the following problems:

Problem 1. *Given a multiplayer quantitative reachability game \mathcal{G} , does there exist a Nash (resp. secure, subgame perfect, subgame perfect secure) equilibrium in \mathcal{G} ?*

Problem 2. *Given a Nash (resp. secure, subgame perfect, subgame perfect secure) equilibrium in a multiplayer quantitative reachability game \mathcal{G} , does there exist such an equilibrium with finite memory?*

These questions have been positively solved by some of the authors in [6] for Nash equilibria in multiplayer games, and for secure equilibria in two-player games. Notice that these problems and related ones have been investigated a lot in the qualitative framework (see [15]).

Here we go a step further and establish the following results about subgame perfect and secure equilibria:

- in every multiplayer quantitative reachability game, there exists a subgame perfect equilibrium (Theorem 10),
- in every two-player quantitative reachability game, there exists a subgame perfect secure equilibrium (Theorem 13),
- in every multiplayer quantitative reachability game, one can decide whether there exists a secure equilibrium in ExpSpace (Theorem 14),
- if there exists a secure equilibrium in a multiplayer quantitative reachability game, then there exists one that is finite-memory (Theorem 15).

Related work. Several recent papers have considered *two-player zero-sum* games played on finite graphs with regular objectives enriched by some *quantitative* aspects. Let us mention some of them: games with finitary objectives [10], games with prioritized requirements [1], request-response games where the waiting times between the requests and the responses are minimized [17,28], and games whose winning conditions are expressed via quantitative languages [2].

Other works concern *qualitative non zero-sum* games. In [9] where the notion of secure equilibrium has been introduced, it is proved that a unique maximal payoff profile of secure equilibria always exists for two-player non zero-sum games with regular objectives. In [15], general criteria ensuring existence of Nash equilibria and subgame perfect equilibria (resp. secure equilibria) are provided for multiplayer (resp. 2-player) games, as well as complexity results. In [4], the existence of Nash equilibria is studied for timed games with qualitative reachability objectives. Complexity issues are discussed in [5] about Nash equilibria in multiplayer concurrent games with Büchi objectives.

Finally, let us mention works that combine both *quantitative* and *non zero-sum* aspects. In [3], the authors study games played on graphs with terminal vertices where quantitative payoffs are assigned to the players. These games may have cycles but all the infinite plays form a single outcome (like in chess where every infinite play is a draw). That paper gives criteria that ensure the existence of Nash (and subgame perfect) equilibria in pure and memoryless strategies. In [19], the studied games are played on priced graphs similar to the ones considered in this article, however in a concurrent way. In this concurrent framework, Nash equilibria are not guaranteed to exist anymore. The authors provide an algorithm to decide existence of Nash equilibria, thanks to a Büchi automaton accepting all Nash equilibria outcomes. The complexity of some related decision problems is also studied. In [24], the authors study Muller games on finite graphs where players have a preference ordering on the sets of the Muller table. They show that Nash equilibria always exist for such games, and that it is decidable whether there exists a subgame perfect equilibrium. In both cases they give a procedure to compute an equilibrium strategy profile (when it exists).

2 Preliminaries

2.1 Games, Strategy Profiles and Equilibria

We consider here *quantitative* games played on a graph where all the players have *reachability objectives*. It means that, given a certain set of vertices Goal_i , each player i wants to reach one of these vertices as soon as possible. We recall the basic notions about these games and we introduce different kinds of equilibria, like Nash equilibria. This section is inspired from [6].

Definition 1. An infinite turn-based multiplayer quantitative reachability game is a tuple $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, v_0, E, (\text{Goal}_i)_{i \in \Pi})$ where

- Π is a finite set of players,
- $G = (V, (V_i)_{i \in \Pi}, v_0, E)$ is a finite directed graph where V is the set of vertices, $(V_i)_{i \in \Pi}$ is a partition of V into the state sets of each player, $v_0 \in V$ is the initial vertex, and $E \subseteq V \times V$ is the set of edges, and
- $\text{Goal}_i \subseteq V$ is the non-empty goal set of player i .

From now on we often use the term *game* to denote a multiplayer quantitative reachability game according to Definition 1.

We assume that each vertex has at least one outgoing edge. The game is played as follows. A token is first placed on the vertex v_0 . Player i , such that $v_0 \in V_i$, has to choose one of the outgoing edges of v_0 and put the token on the vertex v_1 reached when following this edge. Then, it is the turn of the player who owns v_1 . And so on.

A *play* $\rho \in V^\omega$ (resp. a *history* $h \in V^+$) of \mathcal{G} is an *infinite* (resp. a *finite*) path through the graph G starting from vertex v_0 . Note that a history is always non-empty because it starts with v_0 . The set $H \subseteq V^+$ is made up of all the histories of \mathcal{G} , and for $i \in \Pi$, the set H_i is the set of all histories $h \in H$ whose last vertex belongs to V_i .

For any play $\rho = \rho_0\rho_1 \dots$ of \mathcal{G} , we define $\text{Cost}_i(\rho)$ the *cost* of player i as:

$$\text{Cost}_i(\rho) = \begin{cases} l & \text{if } l \text{ is the least index such that } \rho_l \in \text{Goal}_i, \\ +\infty & \text{otherwise.} \end{cases}$$

We note $\text{Cost}(\rho) = (\text{Cost}_i(\rho))_{i \in \Pi}$ the *cost profile* for the play ρ . Each player i aims to *minimize* the cost he has to pay, i.e. reach his goal set as soon as possible. The cost profile for a history h is defined similarly.

A *prefix* (resp. *proper prefix*) α of a history $h = h_0 \dots h_k$ is a finite sequence $h_0 \dots h_l$, with $l \leq k$ (resp. $l < k$), denoted by $\alpha \leq h$ (resp. $\alpha < h$). We similarly consider a prefix α of a play ρ , denoted by $\alpha < \rho$. The function **Last** returns, given a history $h = h_0 \dots h_k$, the last vertex h_k of h , and the *length* $|h|$ of h is the number k of its *edges*¹. Given a play $\rho = \rho_0\rho_1 \dots$, we denote by $\rho_{\leq l}$ the prefix of ρ of length l , i.e. $\rho_{\leq l} = \rho_0\rho_1 \dots \rho_l$. Similarly, $\rho_{< l} = \rho_0\rho_1 \dots \rho_{l-1}$.

¹ Note that the length is not defined as the number of vertices.

We say that a play $\rho = \rho_0\rho_1 \dots$ *visits* a set $S \subseteq V$ (resp. a vertex $v \in V$) if there exists $l \in \mathbb{N}$ such that ρ_l is in S (resp. $\rho_l = v$). The same terminology also stands for a history h .

A *strategy* of player i in \mathcal{G} is a function $\sigma : H_i \rightarrow V$ assigning to each history $h \in H_i$, a next vertex $\sigma(h)$ such that $(\text{Last}(h), \sigma(h))$ belongs to E . We say that a play $\rho = \rho_0\rho_1 \dots$ of \mathcal{G} is *consistent* with a strategy σ of player i if $\rho_{k+1} = \sigma(\rho_0 \dots \rho_k)$ for all $k \in \mathbb{N}$ such that $\rho_k \in V_i$. The same terminology is used for a history h of \mathcal{G} . A *strategy profile* of \mathcal{G} is a tuple $(\sigma_i)_{i \in \Pi}$ where σ_i is a strategy for player i . It determines a unique play of \mathcal{G} consistent with each strategy σ_i , called the *outcome* of $(\sigma_i)_{i \in \Pi}$ and denoted by $\langle (\sigma_i)_{i \in \Pi} \rangle$. We write σ_{-j} for $(\sigma_i)_{i \in \Pi \setminus \{j\}}$, the set of strategies σ_i for all the players except for player j .

A strategy σ of player i is *memoryless* if σ depends only on the current vertex, i.e. $\sigma(hv) = \sigma(v)$ for all $h \in H$ and $v \in V_i$. More generally, σ is a *finite-memory strategy* if the equivalence relation \approx_σ on H defined by $h \approx_\sigma h'$ if $\sigma(h\delta) = \sigma(h'\delta)$ for all $\delta \in H_i$ has finite index. In other words, a finite-memory strategy is a strategy that can be implemented by a finite automaton with output. A strategy profile $(\sigma_i)_{i \in \Pi}$ is called *memoryless* or *finite-memory* if each σ_i is a memoryless or a finite-memory strategy, respectively.

For a strategy profile $(\sigma_i)_{i \in \Pi}$ with outcome ρ and a strategy σ'_j of player j , we say that *player j deviates from ρ* if there exists a prefix h of ρ , consistent with σ'_j , such that $h \in H_j$ and $\sigma'_j(h) \neq \sigma_j(h)$.

We now introduce different notions of equilibria in the *quantitative* framework and give several examples to make clear the presented concepts. We first begin with the definition of *Nash equilibrium*.

Definition 2. A strategy profile $(\sigma_i)_{i \in \Pi}$ of a game \mathcal{G} is a Nash equilibrium if for all player $j \in \Pi$ and for all strategy σ'_j of player j , we have:

$$\text{Cost}_j(\rho) \leq \text{Cost}_j(\rho')$$

where $\rho = \langle (\sigma_i)_{i \in \Pi} \rangle$ and $\rho' = \langle \sigma'_j, \sigma_{-j} \rangle$.

This definition means that for all $j \in \Pi$, player j has no incentive to deviate since he can not strictly decrease his cost when using σ'_j instead of σ_j . Keeping notations of Definition 2 in mind, a strategy σ'_j such that $\text{Cost}_j(\rho) > \text{Cost}_j(\rho')$ is called a *profitable deviation* for player j w.r.t. $(\sigma_i)_{i \in \Pi}$. In this case, either player j pays an infinite cost for ρ and a finite cost for ρ' (i.e. ρ' visits Goal_j , but ρ does not), or player j pays a finite cost for ρ and a strictly lower cost for ρ' (i.e. ρ' visits Goal_j for the first time earlier than ρ does).

We now define the concept of *secure equilibrium* [2]. We first need to associate a binary relation \prec_j on cost profiles with each player j . Given two cost profiles $(x_i)_{i \in \Pi}$ and $(y_i)_{i \in \Pi}$:

$$(x_i)_{i \in \Pi} \prec_j (y_i)_{i \in \Pi} \quad \text{iff} \quad (x_j > y_j) \vee (x_j = y_j \wedge (\forall i \in \Pi \ x_i \leq y_i) \wedge (\exists i \in \Pi \ x_i < y_i)).$$

² Our definition naturally extends the notion of *secure equilibrium* proposed in [9] to the quantitative framework.

We then say that *player j prefers $(y_i)_{i \in \Pi}$ to $(x_i)_{i \in \Pi}$* . In other words, player j prefers a cost profile to another one either if he has a strictly lower cost, or if he keeps the same cost, the other players have a greater cost, and at least one has a strictly greater cost.

Definition 3. A strategy profile $(\sigma_i)_{i \in \Pi}$ of a game \mathcal{G} is a secure equilibrium if for all player $j \in \Pi$, there does not exist any strategy σ'_j of player j such that:

$$\text{Cost}(\rho) \prec_j \text{Cost}(\rho')$$

where $\rho = \langle (\sigma_i)_{i \in \Pi} \rangle$ and $\rho' = \langle \sigma'_j, \sigma_{-j} \rangle$.

In other words, player j has no incentive to deviate w.r.t. the relation \prec_j . A strategy σ'_j such that $\text{Cost}(\rho) \prec_j \text{Cost}(\rho')$ is called a \prec_j -profitable deviation for player j w.r.t. $(\sigma_i)_{i \in \Pi}$. Clearly, any secure equilibrium is a Nash equilibrium.

We now introduce a third type of equilibrium: the *subgame perfect equilibrium*. In this case, a strategy profile is not only required to be optimal for the initial vertex, but also after every possible history of the game. Before giving the definition, we introduce the concept of *subgame* and explain some notations.

Given a game $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, v_0, E, (\text{Goal}_i)_{i \in \Pi})$ and a history hv of \mathcal{G} , with $v \in V$, the *subgame* $\mathcal{G}|_h$ of \mathcal{G} is the game $(\Pi, V, (V_i)_{i \in \Pi}, v, E, (\text{Goal}_i)_{i \in \Pi})$ with initial vertex v . Given a strategy σ_i for player i in \mathcal{G} , we define the strategy $\sigma_i|_h$ in $\mathcal{G}|_h$ by $\sigma_i|_h(h') = \sigma_i(hh')$ for all history h' of $\mathcal{G}|_h$ such that $\text{Last}(h') \in V_i$. Let σ be the strategy profile $(\sigma_i)_{i \in \Pi}$, we write $\sigma|_h$ for $(\sigma_i|_h)_{i \in \Pi}$, and $h(\sigma|_h)$ for the play in \mathcal{G} with prefix h that is consistent with $\sigma|_h$ from v .

Then, we say that $(\sigma_i|_h)_{i \in \Pi}$ is a Nash equilibrium in $\mathcal{G}|_h$ if for all player $j \in \Pi$ and for all strategy σ'_j of player j , we have that $\text{Cost}_j(\rho) \leq \text{Cost}_j(\rho')$, where $\rho = h\langle (\sigma_i|_h)_{i \in \Pi} \rangle$ and $\rho' = h\langle \sigma'_j|_h, \sigma_{-j}|_h \rangle$. Let us stress on the fact that plays ρ and ρ' both include the history h as their prefix, and that the related costs $\text{Cost}_j(\rho)$ and $\text{Cost}_j(\rho')$ thus depend on h (the goal set Goal_j could have already been visited by h). The definition of a secure equilibrium in $\mathcal{G}|_h$ is given similarly.

A subgame perfect equilibrium is a strategy profile that is a Nash equilibrium after every possible history of the game, i.e. in every subgame. In particular, a subgame perfect equilibrium is also a Nash equilibrium.

Definition 4. A strategy profile $(\sigma_i)_{i \in \Pi}$ of a game \mathcal{G} is a subgame perfect equilibrium if for all history h of \mathcal{G} , $(\sigma_i|_h)_{i \in \Pi}$ is a Nash equilibrium in $\mathcal{G}|_h$.

We now introduce the last kind of equilibrium that we study. It is a new notion that combines both concepts of subgame perfect equilibrium and secure equilibrium in the following way.

Definition 5. A strategy profile $(\sigma_i)_{i \in \Pi}$ of a game \mathcal{G} is a subgame perfect secure equilibrium if for all history h of \mathcal{G} , $(\sigma_i|_h)_{i \in \Pi}$ is a secure equilibrium in $\mathcal{G}|_h$.

Notice that a subgame perfect secure equilibrium is a secure equilibrium, as well as a subgame perfect equilibrium.

In order to understand the differences between the various notions of equilibria, we provide three simple examples of games limited to two players and to finite trees. Some examples of equilibria involving cycles in two-player games can be found in [8]. They provide further motivations for the introduction of subgame perfect secure equilibria.

Example 6. Let $\mathcal{G} = (V, V_1, V_2, A, E, \text{Goal}_1, \text{Goal}_2)$ be the two-player game depicted in Fig. 1. The vertices of player 1 (resp. 2) are represented by circles (resp. squares), that is, $V_1 = \{A, D, E, F\}$ and $V_2 = \{B, C\}$. The initial vertex v_0 is A . The vertices of Goal_1 are shaded whereas the vertices of Goal_2 are doubly circled; thus $\text{Goal}_1 = \{D, F\}$ and $\text{Goal}_2 = \{F\}$. The number 2 labeling the edge (B, D) is a shortcut to indicate that there are two consecutive edges from B to D (through one intermediate vertex).

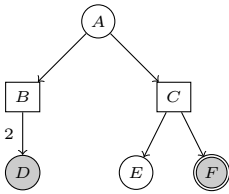


Fig. 1. Game \mathcal{G}

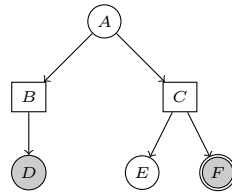


Fig. 2. Game \mathcal{G}'

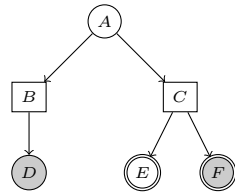


Fig. 3. Game \mathcal{G}''

In the games \mathcal{G} , \mathcal{G}' and \mathcal{G}'' of Fig. 1, 2 and 3 (played on the same graph), we define two strategies σ_1, σ'_1 of player 1 and two strategies σ_2, σ'_2 of player 2 in the following way: $\sigma_1(A) = B, \sigma'_1(A) = C, \sigma_2(C) = E$ and $\sigma'_2(C) = F$.

In \mathcal{G} , one can easily check that the strategy profile (σ_1, σ_2) is a secure equilibrium (and thus a Nash equilibrium) with cost profile is $(3, +\infty)$. Such a secure equilibrium exists because player 2 threatens player 1 to go to vertex E in the case where vertex C is reached. This threat is not credible in this case since by acting this way, player 2 gets an infinite cost instead of a cost of 2 (that he could obtain by reaching F). For this reason, (σ_1, σ_2) is not a subgame perfect equilibrium (and thus not a subgame perfect secure equilibrium). However, one can check that the strategy profile (σ'_1, σ'_2) is a subgame perfect secure equilibrium.

Let us now consider the game \mathcal{G}' depicted in Fig. 2 (notice that the number 2 has disappeared from the edge (B, D)). One can verify that the strategy profile (σ'_1, σ'_2) is a subgame perfect equilibrium which is not a secure equilibrium (and thus not a subgame perfect secure equilibrium). A subgame perfect secure equilibrium for \mathcal{G}' is given by the strategy profile (σ_1, σ'_2) .

Finally, for the game \mathcal{G}'' depicted in Fig. 3, one can check that the strategy profile (σ_1, σ'_2) is both a subgame perfect equilibrium and a secure equilibrium. However it is not a subgame perfect secure equilibrium. In particular, this shows that being a subgame perfect secure equilibrium is *not* equivalent to be a subgame perfect equilibrium and a secure equilibrium. On the other hand, (σ_1, σ_2) is a subgame perfect secure equilibrium in \mathcal{G}'' .

2.2 Unraveling

In the proofs of this article, it will be often useful to *unravel* the graph $G = (V, (V_i)_{i \in \Pi}, v_0, E)$ from the initial vertex v_0 , which ends up in an *infinite tree*, denoted by T . This tree can be seen as a new graph where the set of vertices is the set H of histories of \mathcal{G} , the initial vertex is v_0 , and a pair $(h, hv) \in H \times H$ is an edge of T if $(\text{Last}(h), v) \in E$. A history h is a vertex of player i in T if $h \in H_i$, and h belongs to the goal set of player i if $\text{Last}(h) \in \text{Goal}_i$.

We denote by \mathcal{T} the related game. This game \mathcal{T} played on the unraveling T of G is equivalent to the game \mathcal{G} that is played on G in the following sense. A play $(\rho_0)(\rho_0\rho_1)(\rho_0\rho_1\rho_2) \dots$ in \mathcal{T} induces a unique play $\rho = \rho_0\rho_1\rho_2 \dots$ in \mathcal{G} , and conversely. Thus, we denote a play in \mathcal{T} by the respective play in \mathcal{G} . The bijection between plays of \mathcal{G} and plays of \mathcal{T} allows us to use the same cost function Cost , and to transform easily strategies in \mathcal{G} to strategies in \mathcal{T} (and conversely).

We also need to study the tree T limited to a certain depth $d \in \mathbb{N}$: we denote by $\text{Trunc}_d(T)$ the *truncated tree of T of depth d* and $\text{Trunc}_d(\mathcal{T})$ the *finite game played on $\text{Trunc}_d(T)$* . More precisely, the set of vertices of $\text{Trunc}_d(T)$ is the set of histories $h \in H$ of length $\leq d$; the edges of $\text{Trunc}_d(T)$ are defined in the same way as for T , except that for the histories h of length d , there exists no edge (h, hv) . A play ρ in $\text{Trunc}_d(\mathcal{T})$ corresponds to a history of \mathcal{G} of length *equal to d* . The notions of cost and strategy are defined exactly like in the game \mathcal{T} , but limited to the depth d . For instance, a player pays an infinite cost for a play ρ (of length d) if his goal set is not visited by ρ .

2.3 Kuhn’s Theorem

This section is devoted to the classical Kuhn’s theorem [20]. It claims the existence of a subgame perfect equilibrium (resp. subgame perfect secure equilibrium) in multiplayer games played on *finite trees*.

A *preference relation* is a total, reflexive and transitive binary relation.

Theorem 7 (Kuhn’s theorem). *Let Γ be a finite tree and \mathcal{G} a game played on Γ . For each player $i \in \Pi$, let \succsim_i be a preference relation on cost profiles. Then there exists a strategy profile $(\sigma_i)_{i \in \Pi}$ such that for all history h of \mathcal{G} , all player $j \in \Pi$, and all strategy σ'_j of player j in \mathcal{G} , we have*

$$\text{Cost}(\rho') \succsim_j \text{Cost}(\rho)$$

where $\rho = h \langle (\sigma_i|_h)_{i \in \Pi} \rangle$ and $\rho' = h \langle \sigma'_j|_h, \sigma_{-j}|_h \rangle$.

One can easily be convinced that the binary relation on cost profiles used to define the notion of Nash equilibrium (see Definition 2) is total, reflexive and transitive. We thus have the following corollary.

Corollary 8. *Let \mathcal{G} be a game and T be the unraveling of G . Let $\text{Trunc}_d(\mathcal{T})$ be the game played on the truncated tree of T of depth $d \in \mathbb{N}$. Then there exists a subgame perfect equilibrium in $\text{Trunc}_d(\mathcal{T})$.*

Let \preceq_j be the relation defined by $x \preceq_j y$ iff $x \prec_j y$ or $x = y$, where \prec_j is the relation used in Definition 3. We notice that in the two-player case, this relation is total, reflexive and transitive. However when there are more than two players, \preceq_j is no longer total. Nevertheless, it is proved in [21] that Kuhn’s theorem remains true when \preceq_j is only transitive. So, the next corollary holds.

Corollary 9. *Let \mathcal{G} be a game and T be the unraveling of G . Let $\text{Trunc}_d(\mathcal{T})$ be the game played on the truncated tree of T of depth $d \in \mathbb{N}$. Then there exists a subgame perfect secure equilibrium in $\text{Trunc}_d(\mathcal{T})$.*

3 Subgame Perfection

In this section, we positively solve Problem 1 for subgame perfect equilibria, and for subgame perfect secure equilibria in the two-player case.

Theorem 10. *In every multiplayer quantitative reachability game, there exists a subgame perfect equilibrium.*

The proof uses techniques completely different from the ones given in [6,7] for the existence of Nash equilibria, and secure equilibria in two-player games.

Let \mathcal{G} be a game and \mathcal{T} be the infinite game played on the unraveling T of \mathcal{G} . Kuhn’s theorem (and in particular Corollary 8) guarantees the existence of a subgame perfect equilibrium in each finite game $\text{Trunc}_n(\mathcal{T})$ for all depth $n \in \mathbb{N}$. Given a sequence of such equilibria, the keypoint is to derive the existence of a subgame perfect equilibrium in the infinite game \mathcal{T} . This is possible by the following lemma.

Lemma 11. *Let $(\sigma^n)_{n \in \mathbb{N}}$ be a sequence of strategy profiles such that for every $n \in \mathbb{N}$, σ^n is a strategy profile in the truncated game $\text{Trunc}_n(\mathcal{T})$. Then there exists a strategy profile σ^* in the game \mathcal{T} with the property:*

$$\forall d \in \mathbb{N}, \exists n \geq d, \sigma^* \text{ and } \sigma^n \text{ coincide on histories of length up to } d. \quad (1)$$

Proof. This result is a direct consequence of the compactness of the set of infinite trees with bounded outdegree [18]. An alternative proof is as follows. We give a tree structure, denoted by Γ , to the set of all strategy profiles in the games $\text{Trunc}_n(\mathcal{T})$, $n \in \mathbb{N}$: the nodes of Γ are the strategy profiles, and we draw an edge from a strategy profile σ in $\text{Trunc}_n(\mathcal{T})$ to a strategy profile σ' in $\text{Trunc}_{n+1}(\mathcal{T})$ if and only if σ is the restriction of σ' to histories of length less than n . It means that the nodes at depth d correspond to strategy profiles of $\text{Trunc}_d(\mathcal{T})$. We then consider the tree Γ' derived from Γ where we only keep the nodes σ^n , $n \in \mathbb{N}$, and their ancestors. Since Γ' has finite outdegree, it has an infinite path by König’s lemma. This path goes through infinitely many nodes that are ancestors of nodes in the set $\{\sigma^n, n \in \mathbb{N}\}$. Therefore there exists a strategy profile σ^* in the infinite game \mathcal{T} (given by the previous infinite path in Γ') with property (1). \square

Proof (of Theorem 10). Let $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, v_0, E, (\text{Goal}_i)_{i \in \Pi})$ be a multi-player quantitative reachability game and \mathcal{T} the game played on the unraveling of \mathcal{G} . For all $n \in \mathbb{N}$, we consider the finite game $\text{Trunc}_n(\mathcal{T})$ and get a subgame perfect equilibrium $\sigma^n = (\sigma_i^n)_{i \in \Pi}$ in this game by Corollary 8. According to Lemma 11, there exists a strategy profile σ^* in the game \mathcal{T} with property 11.

It remains to show that σ^* is a subgame perfect equilibrium in \mathcal{T} , and thus in \mathcal{G} . Let $h \in H$ be a history of the game. We have to prove that $\sigma^*|_h$ is a Nash equilibrium in $\mathcal{T}|_h$. As a contradiction, suppose that there exists a profitable deviation σ'_j for some player $j \in \Pi$ w.r.t. $\sigma^*|_h$ in $\mathcal{T}|_h$. This means that $\text{Cost}_j(\rho) > \text{Cost}_j(\rho')$ for $\rho = h\langle \sigma^*|_h \rangle$ and $\rho' = h\langle \sigma'_j|_h, \sigma^*_{-j}|_h \rangle$, that is ρ' visits Goal_j for the first time at a certain depth d , such that $|h| < d < +\infty$, and ρ visits Goal_j at a depth strictly greater than d (see Figure 4). Thus:

$$\text{Cost}_j(\rho) > \text{Cost}_j(\rho') = d.$$

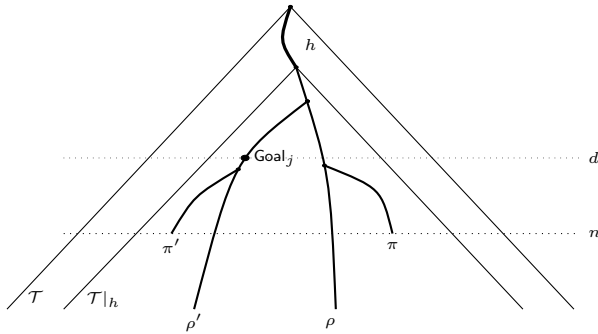


Fig. 4. The game \mathcal{T} with its subgame $\mathcal{T}|_h$

According to property 11, there exists $n \in \mathbb{N}$ such that σ^* coincide with σ^n on histories of length up to d . It follows that for $\pi = h\langle \sigma^n|_h \rangle$ and $\pi' = h\langle \sigma'_j|_h, \sigma^n_{-j}|_h \rangle$, we have that (see Figure 4)

$$\text{Cost}(\pi') = \text{Cost}(\rho') = d \quad \text{and} \quad \text{Cost}(\pi) > d.$$

And so, σ'_j is a profitable deviation for player j w.r.t. $\sigma^n|_h$ in $\text{Trunc}_n(\mathcal{T})|_h$, which leads to a contradiction with the fact that σ^n is a subgame perfect equilibrium in $\text{Trunc}_n(\mathcal{T})$ by hypothesis. \square

As an extension, we consider *multiplayer quantitative reachability games with tuples of costs on edges* (see [7, Definition 31]). In these games, we assume that edges are labelled with tuples of strictly positive costs (one cost for each player). Here we do not only count the number of edges to reach the goal of a player, but we sum up his costs along the path until his goal is reached. His aim is still to minimize his global cost for a play. In this framework, we can also prove

the existence of a subgame perfect equilibrium. The proof is similar to the one of Theorem 10, the only difference lies in the choice of the different considered depths.

Theorem 12. *In every multiplayer quantitative reachability game with tuples of costs on edges, there exists a subgame perfect equilibrium.*

Regarding subgame perfect *secure* equilibria, we positively solve Problem 1 but only in the case of two-player games.

Theorem 13. *In every two-player quantitative reachability game, there exists a subgame perfect secure equilibrium.*

The main ideas of the proof are similar to the ones for Theorem 10 (see 8). Unfortunately the proof does not seem to extend to the multiplayer case. Indeed we face the same kind of problems encountered in 6,7, where the existence of secure equilibria is proved for two-player games, and left open for multiplayer games. (See 7 for further discussion on these problems).

4 Decidability of the Existence of a Secure Equilibrium

In this section, we study Problems 1 and 2 in the context of secure equilibria. Both problems have been positively solved in 6 for two-player games only. To the best of our knowledge, the existence of secure equilibria in the multiplayer framework is still an open problem. We here provide an algorithm that *decides* the existence of a secure equilibrium. We also show that if there exists a secure equilibrium, then there exists one that is finite-memory.

Theorem 14. *In every multiplayer quantitative reachability game, one can decide whether there exists a secure equilibrium in ExpSpace .*

Theorem 15. *If there exists a secure equilibrium in a multiplayer quantitative reachability game, then there exists one that is finite-memory.*

The proof of Theorem 14 is inspired from ideas developed in 6,7. It is rather technical and can be found in 8. Nevertheless, let us give some flavor of the proof. The keypoint is to show that the existence of a secure equilibrium in a game \mathcal{G} is equivalent to the existence of a secure equilibrium (with two additional properties) in the finite game $\text{Trunc}_d(\mathcal{T})$ for a well-chosen depth d . The existence of the latter equilibrium is decidable. Notice that by Corollary 9 a secure equilibrium always exists in $\text{Trunc}_d(\mathcal{T})$; however we do not know if a secure equilibrium with the two required additional properties always exists in $\text{Trunc}_d(\mathcal{T})$. Let us now give some details about these two properties.

The first property requires that the secure equilibrium is *goal-optimized*, meaning that all the visited goal sets are visited for the first time *before a certain given depth*. Let $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, v_0, E, (\text{Goal}_i)_{i \in \Pi})$ be a game. We fix the following constant: $d_{\text{goal}} := 2 \cdot |\Pi| \cdot |V|$.

Definition 16. Given a strategy profile $(\sigma_i)_{i \in \Pi}$ in a game \mathcal{G} , with outcome ρ , we say that $(\sigma_i)_{i \in \Pi}$ is goal-optimized if and only if for all $i \in \Pi$ such that $\text{Cost}_i(\rho) < +\infty$, we have that $\text{Cost}_i(\rho) < d_{goal}$.

The second property asks for a secure equilibrium that is *deviation-optimized*, meaning that whenever a player deviates, he realizes *within a certain given number of steps* that his deviation is not profitable for him.

Definition 17. Given a secure equilibrium $(\sigma_i)_{i \in \Pi}$ in a game \mathcal{G} , we say that $(\sigma_i)_{i \in \Pi}$ is deviation-optimized if and only if for all player $j \in \Pi$, for all strategy σ'_j of player j ,

$$\text{Cost}(\rho_{<d_{dev}}) \not\leq_j \text{Cost}(\rho'_{<d_{dev}}),$$

where $\rho = \langle (\sigma_i)_{i \in \Pi} \rangle$, $\rho' = \langle \sigma'_j, \sigma_{-j} \rangle$, and $d_{dev} = \max\{\text{Cost}_i(\rho) \mid \text{Cost}_i(\rho) < +\infty\} + |V|$.

We can now state the key proposition.

Proposition 18. Given a game \mathcal{G} , there exists a secure equilibrium in \mathcal{G} iff there exists a goal-optimized and deviation-optimized secure equilibrium in $\text{Trunc}_d(\mathcal{T})$, for $d = d_{goal} + 3 \cdot |V|$.

The proof of this proposition is detailed in [8]. It is here difficult to give the significance of choosing constants d_{goal} , d_{dev} and $d = d_{goal} + 3 \cdot |V|$ as is done. We just propose a sketch of proof.

Proof (of Proposition 18 - Sketch).

Suppose that there exists a secure equilibrium $(\sigma_i)_{i \in \Pi}$ in \mathcal{G} . The first step consists in transforming $(\sigma_i)_{i \in \Pi}$ into a goal-optimized and deviation-optimized secure equilibrium in \mathcal{G} . To get a goal-optimized equilibrium, the idea is to eliminate unnecessary cycles between two successively visited goal sets. Such an idea is already developed in [7, Lemma 19] for Nash equilibria. Unfortunately, this lemma cannot be applied for secure equilibria. We need to adapt it to the context of secure equilibria, by modifying the strategies of the coalition against a deviating player. In this way, we get a goal-optimized equilibrium that is also deviation-optimized due to this particular form of the coalitions strategies.

Once we have a goal-optimized and deviation-optimized secure equilibrium in \mathcal{G} , the second step consists in showing that its restriction to $\text{Trunc}_d(\mathcal{T})$ with $d = d_{goal} + 3 \cdot |V|$ is still a goal-optimized and deviation-optimized secure equilibrium in $\text{Trunc}_d(\mathcal{T})$.

Suppose now that there exists a goal-optimized and deviation-optimized secure equilibrium $(\sigma_i)_{i \in \Pi}$ in $\text{Trunc}_d(\mathcal{T})$, for $d = d_{goal} + 3 \cdot |V|$. To get from $(\sigma_i)_{i \in \Pi}$ a secure equilibrium in \mathcal{G} , we inspire from a construction proposed in [7, Proposition 25] where it is shown, in the context of two-player games, how to extend a secure equilibrium in a finite truncation of \mathcal{G} to a secure equilibrium in \mathcal{G} . The rough idea is as follows. Due to the hypotheses, the outcome π of $(\sigma_i)_{i \in \Pi}$ has a prefix $\alpha\beta$ such that all goal sets visited by π are already visited by α , and such that β is a cycle. The required secure equilibrium is specified such that its

outcome is equal to $\alpha\beta^\omega$ and any deviating player is punished by the coalition of the other players in a way that this deviation is not profitable for him³ \square

We are now able to prove the two theorems of this section.

Proof (of Theorem 14). By Proposition 18, there exists a secure equilibrium in \mathcal{G} iff there exists a goal-optimized and deviation-optimized secure equilibrium in $\text{Trunc}_d(\mathcal{T})$, with $d = d_{\text{goal}} + 3 \cdot |V|$. The latter property is decidable in NExpSpace (in $|V|$ and $|I|$). Indeed, $\text{Trunc}_d(\mathcal{T})$ has an exponential size. Guessing a strategy profile $(\sigma_i)_{i \in I}$ in this tree also needs an exponential size. Then we can test in exponential size whether $(\sigma_i)_{i \in I}$ is a goal-optimized and deviation-optimized secure equilibrium in $\text{Trunc}_d(\mathcal{T})$. By Savitch's theorem, deciding the existence of a secure equilibria is thus in ExpSpace . \square

Proof (of Theorem 15). This theorem is a direct consequence of Proposition 18. Indeed consider a secure equilibrium in a game \mathcal{G} . We first apply Proposition 18 to this strategy profile to get a goal-optimized and deviation-optimized secure equilibrium $(\sigma_i)_{i \in I}$ in $\text{Trunc}_d(\mathcal{T})$, for $d = d_{\text{goal}} + 3 \cdot |V|$. Then we apply Proposition 18, in the other direction, to the equilibrium $(\sigma_i)_{i \in I}$, to get a secure equilibrium back in \mathcal{G} . The latter equilibrium can be supposed to be finite-memory as explained in the proof of Proposition 18 (see Footnote 5). \square

5 Conclusion and Perspectives

In this paper, we study the concept of subgame perfect equilibrium, a refinement of Nash equilibrium well-suited in the framework of games played on graphs. We also introduce the new concept of subgame perfect secure equilibrium. We prove the existence of subgame perfect equilibria in multiplayer quantitative reachability games. We also prove the existence of subgame perfect secure equilibria, but only in the two-player framework. Finally, we provide an algorithm deciding in ExpSpace the existence of secure equilibria in the multiplayer case. On one hand, the first two results have been obtained by topological techniques, that are completely different from the techniques used in [6,7]. On the other hand, proofs of the last result are strongly inspired by proofs developed in these references, but have required new ideas about the coalition strategies.

There are several interesting directions for future research. We are currently working on the model of quantitative game, enriched by allowing n -tuples of positive weights on edges (see Theorem 12). We do believe that our results remain true in this context. The case of Nash equilibria is already treated in [7]. Notice that our results trivially generalize to the particular case where the weights of the edges are of the form (c, \dots, c) with $c \in \mathbb{N}_0$. Indeed it is enough to replace each such edge by a path of length c composed of c new edges (of cost 1).

To the best of our knowledge, the existence of secure equilibria in the multiplayer framework is still an open problem. We prove that the existence of a secure

³ It should be noted that this secure equilibrium can be constructed in a way to be finite-memory.

equilibrium in an infinite game is equivalent to the existence of a goal-optimized and deviation-optimized secure equilibrium in a finite game. This open problem could be positively solved if Corollary 9 could be adapted in a way to get a goal-optimized and deviation-optimized secure equilibrium in the finite game, and then by applying Proposition 18. A deeper understanding of equilibria with unnecessary cycles could also be helpful. For the moment, we are not able to solve this problem with more than two players. The same kind of question is also open for subgame perfect secure equilibria.

Another research direction concerns a deeper study of the memory needed in the different kinds of equilibria. In the case of subgame perfect equilibria and subgame perfect secure equilibria, the topological techniques give no results on the memory needed. However, in the case of secure equilibria, we prove that we can limit to finite-memory equilibria.

Acknowledgements. This work has been partly supported by the ESF project GASICS and a grant from the National Bank of Belgium. The third author is supported by a grant from L'Oréal-UNESCO/F.R.S.-FNRS.

References

1. Alur, R., Kanade, A., Weiss, G.: Ranking Automata and Games for Prioritized Requirements. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 240–253. Springer, Heidelberg (2008)
2. Bloem, R., Chatterjee, K., Henzinger, T., Jobstmann, B.: Better Quality in Synthesis through Quantitative Objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
3. Boros, E., Gurvich, V.: Why chess and back gammon can be solved in pure positional uniformly optimal strategies. Rutcor Research Report 21-2009. Rutgers University (2009)
4. Bouyer, P., Brenguier, R., Markey, N.: Nash Equilibria for Reachability Objectives in Multi-Player Timed Games. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 192–206. Springer, Heidelberg (2010)
5. Bouyer, P., Brenguier, R., Markey, N., Ummels, M.: Nash equilibria in concurrent games with Büchi objectives. In: Foundations of Software Technology and Theoretical Computer Science, FSTTCS. LIPIcs, vol. 13, pp. 375–386. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
6. Brihaye, T., Bruyère, V., De Pril, J.: Equilibria in Quantitative Reachability Games. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 72–83. Springer, Heidelberg (2010)
7. Brihaye, T., Bruyère, V., De Pril, J.: On equilibria in quantitative games with reachability/safety objectives. Technical report (2011), <http://www.ulb.ac.be/di/gasics/equilibria.pdf>
8. Brihaye, T., Bruyère, V., De Pril, J., Gimbert, H.: Subgame Perfection for Equilibria in Quantitative Reachability Games. Technical report (2011), <http://www.ulb.ac.be/di/gasics/subgame-perfection.pdf>
9. Chatterjee, K., Henzinger, T., Jurdziński, M.: Games with secure equilibria. Theoretical Computer Science 365(1-2), 67–82 (2006)

10. Chatterjee, K., Henzinger, T.A.: Finitary Winning in Omega-Regular Games. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 257–271. Springer, Heidelberg (2006)
11. Chatterjee, K., Henzinger, T.A.: Assume-Guarantee Synthesis. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 261–275. Springer, Heidelberg (2007)
12. Chatterjee, K., Raman, V.: Assume-guarantee synthesis for digital contract signing. CoRR, abs/1004.2697 (2010)
13. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
14. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
15. Grädel, E., Ummels, M.: Solution concepts and algorithms for infinite multiplayer games. In: New Perspectives on Games and Interaction. Texts in Logic and Games, vol. 4, pp. 151–178. Amsterdam University Press (2008)
16. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Co., Reading (1979)
17. Horn, F., Thomas, W., Wallmeier, N.: Optimal Strategy Synthesis in Request-Response Games. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 361–373. Springer, Heidelberg (2008)
18. Kechris, A.: Classical descriptive set theory. Springer, Heidelberg (1995)
19. Klimoš, M., Larsen, K.G., Štefaňák, F., Thaarup, J.: Nash Equilibria in Concurrent Priced Games. In: Dediu, A.-H. (ed.) LATA 2012. LNCS, vol. 7183, pp. 363–376. Springer, Heidelberg (2012)
20. Kuhn, H.: Extensive games and the problem of information. *Classics in Game Theory*, 46–68 (1953)
21. Le Roux, S.: Acyclic Preferences and Existence of Sequential Nash Equilibria: A Formal and Constructive Equivalence. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 293–309. Springer, Heidelberg (2009)
22. Nash, J.: Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America* 36(1), 48–49 (1950)
23. Osborne, M., Rubinstein, A.: A course in game theory. MIT Press, Cambridge (1994)
24. Paul, S., Simon, S., Kannan, R., Kumar, K.: Nash equilibrium in generalised muller games. In: Foundations of Software Technology and Theoretical Computer Science, FSTTCS. LIPIcs, vol. 4, pp. 335–346. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
25. Selten, R.: Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrage-trägheit. *Zeitschrift für die gesamte Staatswissenschaft* 121, 301–324, 667–689 (1965)
26. Thomas, W.: On the Synthesis of Strategies in Infinite Games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
27. Thomas, W.: Church’s Problem and a Tour through Automata Theory. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 635–655. Springer, Heidelberg (2008)
28. Zimmermann, M.: Time-Optimal Winning Strategies for Poset Games. In: Maneth, S. (ed.) CIAA 2009. LNCS, vol. 5642, pp. 217–226. Springer, Heidelberg (2009)

Concurrent Games with Ordered Objectives

Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels

LSV, CNRS & ENS Cachan, France

{bouyer,brenguier,markey,ummels}@lsv.ens-cachan.fr

Abstract. We consider concurrent games played on graphs, in which each player has several qualitative (e.g. reachability or Büchi) objectives, and a preorder on these objectives (for instance the counting order, where the aim is to maximise the number of objectives that are fulfilled).

We study two fundamental problems in that setting: (1) the *value problem*, which aims at deciding the existence of a strategy that ensures a given payoff; (2) the *Nash equilibrium problem*, where we want to decide the existence of a Nash equilibrium (possibly with a condition on the payoffs). We characterise the exact complexities of these problems for several relevant preorders, and several kinds of objectives.

1 Introduction

Games (and especially games played on graphs) have been intensively used in computer science as a powerful way of modelling interactions between several computerised systems [15,6]. Until recently, more focus had been put on the study of purely antagonistic games (a.k.a. zero-sum games), useful for modelling systems evolving in a (hostile) environment.

Over the last ten years, non-zero-sum games have come into the picture: they are convenient for modelling complex infrastructures where each individual system tries to fulfill its objectives, while still being subject to uncontrollable actions of the surrounding systems. As an example, consider a wireless network in which several devices try to send data: each device can modulate its transmit power, in order to maximise its bandwidth and reduce energy consumption as much as possible. In that setting, focusing only on optimal strategies for one single agent may be too narrow, and several other solution concepts have been defined and studied in the literature, of which Nash equilibrium [11] is the most prominent. A Nash equilibrium is a strategy profile where no player can improve her payoff by unilaterally changing her strategy, resulting in a configuration of the network that is satisfactory to everyone. Notice that Nash equilibria need not exist or be unique, and are not necessarily optimal: Nash equilibria where all players lose may coexist with more interesting Nash equilibria.

Our contributions. In this paper, we extend our previous study of pure-strategy Nash equilibria in concurrent games with qualitative objectives [24] to a (semi-) quantitative setting: we assume that each player is given a set S of qualitative objectives (reachability, for instance), together with a preorder on 2^S . This preorder

defines a preference relation (or payoff), and the aim of a player is to maximise her payoff. For instance, the counting order compares the number of objectives which are fulfilled. As another example, we will consider the lexicographic order, defined in an obvious way once we have ordered the simple objectives. More generally, preorders will be defined by Boolean circuits.

We characterise the exact complexity of deciding the existence of a Nash equilibrium, for reachability and Büchi objectives, under arbitrary preorders. Our techniques also provide us with solutions to the value problem, which corresponds to the purely antagonistic setting described above. We prove for instance that both problems are PSPACE-complete for reachability objectives together with a lexicographic order on these objectives (or for the more general class of preorders defined by Boolean circuits). On the other hand, we show that for sets of Büchi objectives (assumed to be indexed) ordered by the maximum index they contain, both problems are solvable in PTIME.

Related work. Even though works on concurrent games go back to the fifties, the complexity of computing Nash equilibria in games played on graphs has only recently been addressed [5,16]. Most of the works so far have focused on turn-based games and on qualitative objectives, but have also considered the more general setting of stochastic games or strategies. Our restriction to pure strategies is justified by the undecidability of computing Nash equilibria in concurrent games with qualitative reachability or Büchi objectives, when strategies can be randomised [17]. Although their setting is turn-based, the most relevant related work is [14], where a first step towards quantitative objectives is made: they consider generalised Muller games (with a preference order on the set of states that are visited infinitely often), show that pure Nash equilibria always exist, and give a doubly-exponential algorithm for computing a Nash equilibrium. Generalised Muller conditions can be expressed using Büchi conditions and Boolean circuits (which in the worst-case can be exponential-size): from our results we derive an EXPSPACE upper bound.

For lack of space, the technical proofs are omitted, and can be found in [3].

2 Preliminaries

2.1 Concurrent Games

Definition 1 ([1]). *A (finite) concurrent game is a tuple $\mathcal{G} = \langle \text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab} \rangle$, where States is a (finite) set of states, Agt is a finite set of players, Act is a finite set of actions, and*

- Mov: $\text{States} \times \text{Agt} \rightarrow 2^{\text{Act}} \setminus \{\emptyset\}$ is a mapping indicating the actions available to a given player in a given state;
- Tab: $\text{States} \times \text{Act}^{\text{Agt}} \rightarrow \text{States}$ associates with a given state and a given move¹ of the players the resulting state.

¹ A move is an element of Act^{Agt} .

Fig. 1 displays an example of a concurrent game. Transitions are labelled with the moves that trigger them. We say that a move $m_{\text{Agt}} = \langle m_A \rangle_{A \in \text{Agt}} \in \text{Act}^{\text{Agt}}$ is *legal* at s if $m_A \in \text{Mov}(s, A)$ for all $A \in \text{Agt}$. A game is *turn-based* if for each state the set of allowed moves is a singleton for all but at most one player.

In a concurrent game \mathcal{G} , whenever we arrive at a state s , the players simultaneously select an available action, which results in a legal move m_{Agt} ; the next state of the game is then $\text{Tab}(s, m_{\text{Agt}})$. The same process repeats *ad infinitum* to form an infinite sequence of states.

A path π in \mathcal{G} is a sequence $(s_i)_{0 \leq i < n}$ (where $n \in \mathbb{N}_{>0} \cup \{\infty\}$) of states. The length of π , denoted by $|\pi|$, is $n - 1$. The set of finite paths (also called *histories*) of \mathcal{G} is denoted by $\text{Hist}_{\mathcal{G}}$, the set of infinite paths (also called *plays*) of \mathcal{G} is denoted by $\text{Play}_{\mathcal{G}}$, and $\text{Path}_{\mathcal{G}} = \text{Hist}_{\mathcal{G}} \cup \text{Play}_{\mathcal{G}}$ is the set of paths of \mathcal{G} . Given a path $\pi = (s_i)_{0 \leq i < n}$ and an integer $j < n$, the j -th prefix (resp. j -th suffix, j -th state) of π , denoted by $\pi_{\leq j}$ (resp. $\pi_{\geq j}$, $\pi_{=j}$), is the finite path $(s_i)_{0 \leq i < j+1}$ (resp. $(s_i)_{j \leq i < n}$, state s_j). If $\pi = (s_i)_{0 \leq i < n}$ is a history, we write $\text{last}(\pi) = s_{|\pi|}$. In the sequel, we write $\text{Hist}_{\mathcal{G}}(s)$, $\text{Play}_{\mathcal{G}}(s)$ and $\text{Path}_{\mathcal{G}}(s)$ for the respective subsets of paths starting in state s . If π is a play, $\text{Occ}(\pi) = \{s \mid \exists j. \pi_{=j} = s\}$ is the sets of states that appear at least once along π , and $\text{Inf}(\pi) = \{s \mid \forall i. \exists j \geq i. \pi_{=j} = s\}$ is the sets of states that appear infinitely often along π .

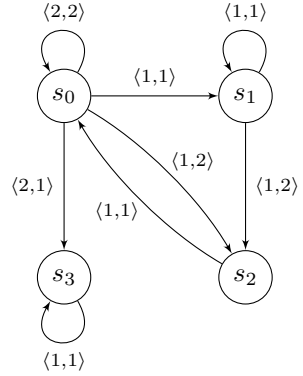


Fig. 1. Example of a two-player concurrent game \mathcal{A}

Definition 2. Let \mathcal{G} be a concurrent game, and $A \in \text{Agt}$. A strategy for A is a mapping $\sigma_A: \text{Hist}_{\mathcal{G}} \rightarrow \text{Act}$ such that $\sigma_A(\pi) \in \text{Mov}(\text{last}(\pi), A)$ for all $\pi \in \text{Hist}_{\mathcal{G}}$. A strategy σ_P for a coalition $P \subseteq \text{Agt}$ is a tuple of strategies, one for each player in P . We write $\sigma_P = (\sigma_A)_{A \in P}$ for such a strategy. A strategy profile is a strategy for Agt . We write $\text{Strat}_{\mathcal{G}}^P$ for the set of strategies of coalition P , and $\text{Prof}_{\mathcal{G}} = \text{Strat}_{\mathcal{G}}^{\text{Agt}}$.

Note that we only consider *pure* (i.e., non-randomised) strategies. Notice also that strategies are based on the sequences of visited states, and not on the sequences of actions played by the players. This is realistic when considering multi-agent systems, where only the global effect of the actions of the agents may be observable. When computing Nash equilibria, this restriction makes it more difficult to detect which players have deviated from their strategies.

Let \mathcal{G} be a game, P a coalition, and σ_P a strategy for P . A path π is *compatible* with the strategy σ_P if, for all $k < |\pi|$, there exists a move m_{Agt} such that

1. m_{Agt} is legal at $\pi_{=k}$,
2. $m_A = \sigma_A(\pi_{\leq k})$ for all $A \in P$, and
3. $\text{Tab}(\pi_{=k}, m_{\text{Agt}}) = \pi_{=k+1}$.

We write $\text{Out}_{\mathcal{G}}(\sigma_P)$ for the set of paths (called the *outcomes*) in \mathcal{G} which are compatible with strategy σ_P of P . We write $\text{Out}_{\mathcal{G}}^f$ (resp. $\text{Out}_{\mathcal{G}}^\infty$) for the finite (resp. infinite) outcomes, and $\text{Out}_{\mathcal{G}}(s, \sigma_P)$, $\text{Out}_{\mathcal{G}}^f(s, \sigma_P)$ and $\text{Out}_{\mathcal{G}}^\infty(s, \sigma_P)$ for the respective sets of outcomes of σ_P with initial state s . Notice that any strategy profile has a single infinite outcome from a given state.

2.2 Winning Objectives

Objectives and preference relations. An *objective* (or *winning condition*) is an arbitrary set of plays. With a set T of states, we associate an objective $\Omega(T)$ in three different ways:

$$\Omega(T) = \{\rho \in \text{Play}_{\mathcal{G}} \mid \text{Occ}(\rho) \cap T \neq \emptyset\} \quad (\text{Reachability})$$

$$\Omega(T) = \{\rho \in \text{Play}_{\mathcal{G}} \mid \text{Occ}(\rho) \cap T = \emptyset\} \quad (\text{Safety})$$

$$\Omega(T) = \{\rho \in \text{Play}_{\mathcal{G}} \mid \text{Inf}(\rho) \cap T \neq \emptyset\} \quad (\text{Büchi})$$

In our setting, each player A is assigned a tuple of such objectives $(\Omega_i)_{1 \leq i \leq n}$, together with a preorder \lesssim on $\{0, 1\}^n$. The *payoff vector* of a play ρ for player A is the vector $\mathbf{1}_{\{i \mid \rho \in \Omega_i\}} \in \{0, 1\}^n$ ($\mathbf{1}_S$ is the vector v such that $v_i = 1 \Leftrightarrow i \in S$; we write $\mathbf{1}$ for $\mathbf{1}_{[1, n]}$, and $\mathbf{0}$ for $\mathbf{1}_\emptyset$). The preorder \lesssim then defines another preorder \preceq on the set of plays of \mathcal{G} , called the *preference relation* of A , by ordering the plays according to their payoffs: $\rho' \preceq \rho$ if and only if $\mathbf{1}_{\{i \mid \rho' \in \Omega_i\}} \lesssim \mathbf{1}_{\{i \mid \rho \in \Omega_i\}}$. Intuitively, each player aims at a play that is preferred to most other plays.

Examples of preorders. We now describe some preorders on $\{0, 1\}^n$ that we consider in the sequel (Fig. 2(a)–2(d) display four of these preorders for $n = 3$). For the purpose of these definitions, we assume that $\max \emptyset = -\infty$.

- *Conjunction:* $v \lesssim w$ iff either $v_i = 0$ for some $0 \leq i \leq n$, or $w_i = 1$ for all $0 \leq i \leq n$. This corresponds to the case where a player wants to achieve all her objectives.
- *Disjunction:* $v \lesssim w$ iff either $v_i = 0$ for all $0 \leq i \leq n$, or $w_i = 1$ for some $0 \leq i \leq n$. The aim here is to satisfy at least one objective.
- *Counting:* $v \lesssim w$ iff $|\{i \mid v_i = 1\}| \leq |\{i \mid w_i = 1\}|$. The aim is to maximise the number of satisfied conditions.
- *Subset:* $v \lesssim w$ iff $\{i \mid v_i = 1\} \subseteq \{i \mid w_i = 1\}$: in this setting, a player will always struggle to satisfy a larger (for inclusion) set of objectives.
- *Maximise:* $v \lesssim w$ iff $\max\{i \mid v_i = 1\} \leq \max\{i \mid w_i = 1\}$. The aim is to maximise the highest index of the satisfied objectives.
- *Lexicographic:* $v \lesssim w$ iff either $v = w$, or there is an index i such that $v_i = 0$, $w_i = 1$ and $v_j = w_j$ for all $0 \leq j < i$.
- *Parity:* $v \lesssim w$ iff either $\max\{i \mid w_i = 1\}$ is even, or $\max\{i \mid v_i = 1\}$ is odd (or $-\infty$). Combined with reachability objectives, this corresponds to a *weak parity condition*; parity objectives as they are classically defined correspond to parity preorders over Büchi objectives.

- *Boolean Circuit*: given a Boolean circuit, with input from $\{0, 1\}^{2n}$, $v \lesssim w$ if and only if the circuit evaluates to 1 on input $v_1 \dots v_n w_1 \dots w_n$.
- *Monotonic Boolean Circuit*: Same as above, with the restriction that the input gates corresponding to v are negated, and no other negation appears in the circuit.

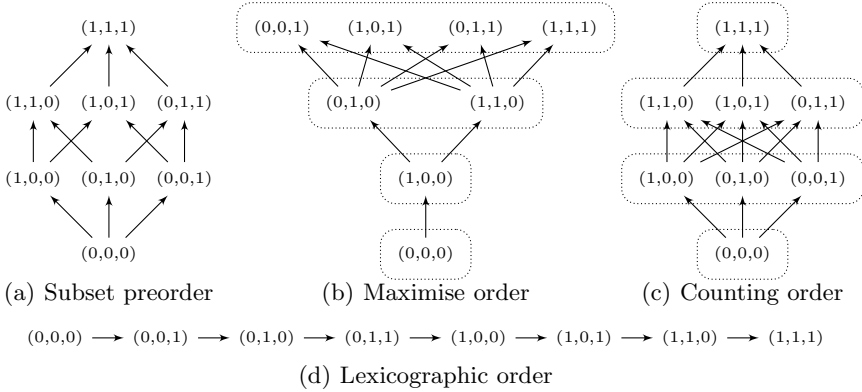


Fig. 2. Examples of preorders (for $n = 3$): dotted boxes represent equivalence classes for \sim ; arrows represent the preorder relation \lesssim , forgetting about \sim -equivalent elements

In terms of expressiveness, any preorder over $\{0, 1\}^n$ can be given as a Boolean circuit: for each pair (v, w) with $v \lesssim w$, it is possible to construct a circuit whose output is 1 if and only if the input is $v_1 \dots v_n w_1 \dots w_n$; taking the disjunction of all these circuits we obtain a Boolean circuit defining the preorder. Its size can be bounded by $2^{2n+3}n$, which is exponential in general, but all of our examples can be specified with a circuit of polynomial size.

A preorder \lesssim is *monotonic* if it is compatible with the subset ordering, i.e. if $\{i \mid v_i = 1\} \subseteq \{i \mid w_i = 1\}$ implies $v \lesssim w$. Hence, a preorder is monotonic if fulfilling more objectives never results in a lower payoff. All the above examples are monotonic, except the parity preorder and the general Boolean circuits. Moreover, any monotonic preorder can be expressed as a monotonic Boolean circuit.

2.3 Nash Equilibria

Given a move m_{Agt} and an action m' for some player B , we write $m_{\text{Agt}}[B \mapsto m']$ for the move n_{Agt} with $n_A = m_A$ when $A \neq B$ and $n_B = m'$. This is extended to strategies in the natural way.

Definition 3. Let \mathcal{G} be a concurrent game with preference relation $(\lesssim_A)_{A \in \text{Agt}}$, and let s be a state of \mathcal{G} . A Nash equilibrium of \mathcal{G} from s is a strategy profile $\sigma_{\text{Agt}} \in \text{Prof}_{\mathcal{G}}$ such that $\text{Out}(s, \sigma_{\text{Agt}}[B \mapsto \sigma']) \lesssim_B \text{Out}(s, \sigma_{\text{Agt}})$ for all players $B \in \text{Agt}$ and all strategies $\sigma' \in \text{Strat}^B$.

Hence, Nash equilibria are strategy profiles where no player has an incentive to unilaterally deviate from her strategy.

Remark 4. Another possible way of defining Nash equilibrium would be to require that either $\text{Out}(s, \sigma_{\text{Agt}}[B \mapsto \sigma']) \lesssim_B \text{Out}(s, \sigma_{\text{Agt}})$ or $\text{Out}(s, \sigma_{\text{Agt}}) \not\lesssim_B \text{Out}(s, \sigma_{\text{Agt}}[B \mapsto \sigma'])$. This definition is not equivalent to the one we adopted if the preorder is not total, but both can be meaningful. Notice that with our Definition 3, any Nash equilibrium σ_{Agt} for the subset preorder is also a Nash equilibrium for any monotonic preorder.

2.4 Decision Problems

Given a game $\mathcal{G} = \langle \text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab} \rangle$, a type of objective (Reachability, Safety or Büchi), for each player a list $(T_i^A)_{A \in \text{Agt}, i \in \{1, \dots, n_A\}}$ of targets and a preorder \lesssim_A on $\{0, 1\}^{n_A}$, and a state s , we consider the following problems:

- *Value:* Given a player A and a payoff vector v , can player A ensure payoff v , i.e., is there a strategy σ_A for player A such that any outcome of σ_A in \mathcal{G} from s with payoff v' for A satisfies $v \lesssim_A v'$?
- *Existence:* Does there exist a Nash equilibrium in \mathcal{G} from s ?
- *Constrained existence:* Given two vectors u^A and w^A for each player A , does there exist a Nash equilibrium in \mathcal{G} from s with some payoff $(v^A)_{A \in \text{Agt}}$ satisfying the constraint, i.e., $u^A \lesssim_A v^A \lesssim_A w^A$ for all $A \in \text{Agt}$?

2.5 Preliminary Lemma

We first characterise outcomes of Nash equilibria as ultimately periodic runs.

Lemma 5. *Assume that every player has a preference relation which only depends on the set of states that are visited, and the set of states that are visited infinitely often, i.e. if $\text{Inf}(\rho) = \text{Inf}(\rho')$ and $\text{Occ}(\rho) = \text{Occ}(\rho')$, then $\rho \sim_A \rho'$ for every player $A \in \text{Agt}$. If there is a Nash equilibrium with payoff v , then there is a Nash equilibrium with payoff v for which the outcome is of the form $\pi \cdot \tau^\omega$, where $|\pi|$ and $|\tau|$ are bounded by $|\text{States}|^2$.*

3 Reachability Objectives

Multiplexer games with one reachability objective per player have been studied in [2], where the existence and constrained existence are shown NP-complete.

We now assume that each player has several reachability objectives. In the general case where the preorders are given as Boolean circuits, we show that the various decision problems are PSPACE-complete, where the hardness result even holds for several simpler preorders. We then improve this result in a number of cases. The results are summarised in Table 1.

Table 1. Summary of the results for reachability objectives

	Preorder	Value problem	(Constrained) existence
Disjunction, Maximise	Parity	P-c	NP-c
	Subset	P-c [12]	NP-h and in PSPACE
		PSPACE-c	NP-c
Conjunction, Counting, Lexicographic (Monotonic) Boolean Circuit		PSPACE-c	PSPACE-c
		PSPACE-c	PSPACE-c

3.1 General Case

Theorem 6. *For reachability objectives with preorders given by Boolean circuits, the value, existence and constrained existence problems are in PSPACE. For preorders having $\mathbf{1}$ as a unique maximal element, the value problem is PSPACE-complete. If moreover there is an element $v \in \{0, 1\}^n$ such that $\mathbf{1} \not\lesssim v' \Leftrightarrow v' \lesssim v$, then the existence and constrained existence problems are PSPACE-complete (even for two-player games).*

Before proving these results, let us first discuss the above conditions. The conjunction, subset, counting and lexicographic preorders have unique maximal element $\mathbf{1}$. The conjunction, counting and lexicographic preorders have an element v such that $\mathbf{1} \not\lesssim v' \Leftrightarrow v' \lesssim v$.

As conjunction (for instance) can easily be encoded using a (monotonic) Boolean circuit in polynomial time, the hardness results are also valid if the order is given by a (monotonic) Boolean circuit. On the other hand, disjunction and maximise preorders do not have a unique maximal element, so we cannot apply the hardness result. In the same way, for the subset preorder there is no v such that $\mathbf{1} \not\lesssim v' \Leftrightarrow v' \lesssim v$, so the hardness result does not apply. We prove later (Section 3.2) that in these special cases, the complexity is actually lower.

Proof of the PSPACE upper bounds. We first focus on the constrained existence problem, and we fix a game \mathcal{G} with reachability objectives and a preorder for every player, and a constraint on the payoffs. The algorithm proceeds as follows. For every possible payoff vector that satisfies the given constraint, we will check whether there is an equilibrium with this payoff. Fix such a payoff tuple $\mathbf{v} = (v^A)_{A \in \text{Agt}}$. We construct a new game $\mathcal{G}(\mathbf{v})$: the structure of $\mathcal{G}(\mathbf{v})$ is identical to \mathcal{G} , but each player A has a single objective given by a 1-weak² deterministic Büchi automaton $\mathcal{A}(v^A)$. The new game satisfies the following property: there is a Nash equilibrium in \mathcal{G} with payoff \mathbf{v} iff there is a Nash equilibrium in $\mathcal{G}(\mathbf{v})$ with payoff $\mathbf{0}$ whose outcome has payoff \mathbf{v} in \mathcal{G} . Then, applying arguments similar to [4, Thm. 22], we easily design a polynomial-space algorithm for deciding the existence of a Nash equilibrium with a given payoff, and therefore more generally for the constrained existence problem.

² That is, each strongly connected component contains exactly one state.

The automata $\mathcal{A}(v^A)$ are obtained from a common structure \mathcal{A} (whose construction is illustrated by an example in Fig. 3) by adding the set of accepting states $F(v^A) = \{S \mid \mathbf{1}_{\{i \mid S \cap T_i^A \neq \emptyset\}} \not\leq v^A\}$, where T_i^A is the i -th target of player A in \mathcal{G} . While reading a word ρ from the initial state, the current state of $\mathcal{A}(v^A)$ is the set of states that have been seen so far. Hence, if v is the payoff of ρ for player A in game \mathcal{G} , then $\mathcal{A}(v^A)$ accepts v iff $v \not\leq v^A$. With this construction, the announced equivalence is straightforward.

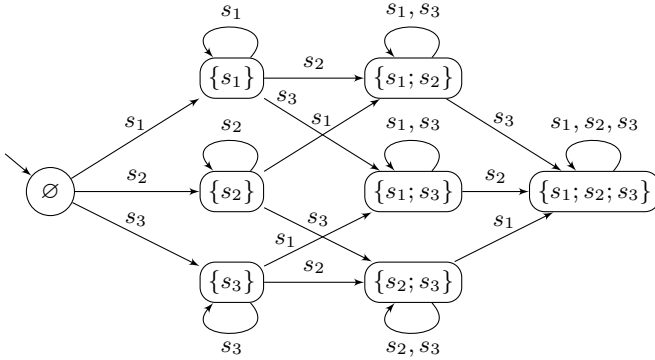


Fig. 3. Automaton \mathcal{A} for set of states $\{s_1, s_2, s_3\}$

We now turn to the proof for the value problem. Without loss of generality, we assume that we are given a two-player game \mathcal{G} , a player A and a threshold v . We define a new game (of polynomial size), by only changing the preferences of the players. Player A has now no objective, and her opponent wins if the payoff is not above v in the original game, i.e. if the run has payoff v' with $v \not\leq v'$. Then, there is a Nash equilibrium where the opponent loses iff there is a strategy for A that ensures v in the original game. We can thereby use the algorithm that decides constrained existence.

Hardness of the value problem. The proof is done by encoding an instance of QSAT. Given a formula of QSAT, we construct a two-player turn-based game with several reachability objectives for player A , such that the formula is valid iff player A has a strategy that visits all her target sets. We do not give details of the construction but better illustrate it on an example.

Example 7. We consider the formula

$$\phi = \forall x_1. \exists x_2. \forall x_3. \exists x_4. (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge \neg x_4 \quad (1)$$

The targets for player A are given by the clauses of ϕ : $T_1^A = \{x_1, \neg x_2, \neg x_3\}$, $T_2^A = \{x_1, x_2, x_4\}$, and $T_3^A = \{\neg x_4\}$. We fix any preorder with unique maximal element $(1, 1, 1)$. The structure of the game is represented in Fig. 4. In this example, player B has a strategy that falsifies one of the clauses whatever A does, which means that player A has no strategy to enforce all its target sets, which means that the formula ϕ is not valid.

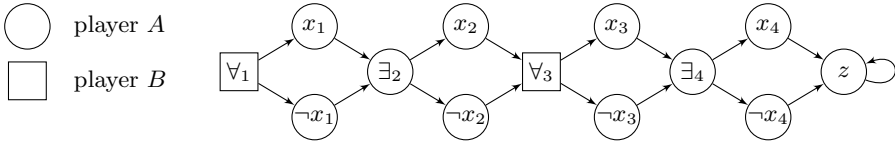


Fig. 4. Reachability game associated with the formula (Π)

Hardness of the (constrained) existence problem. The previous hardness proof applies in particular to conjunctions of reachability objectives. We use a reduction from this problem to prove that the constrained existence problem is PSPACE-hard, under the conditions specified in the statement of Theorem 6. Let \mathcal{G} be a turn-based game with a conjunction of reachability objectives for player A and v be a threshold for player A. We construct a new game \mathcal{G}' as follows. We add to \mathcal{G} an initial state s'_0 , and a sink state z . In the initial state s'_0 , the two players A and B play the matching-pennies game, to either go to z or s_0 .³ We modify the targets of player A so that, in \mathcal{G}' , reaching z exactly gives her payoff v . The new sink state is the unique target of player B. We can check that if there is no strategy for A ensuring v in \mathcal{G} , then there is a Nash equilibrium in game \mathcal{G}' , which consists in going to z . And conversely if there is a Nash equilibrium in \mathcal{G}' then its outcome goes to z , which means that A cannot ensure $\mathbf{1}$ in game \mathcal{G} .

3.2 Simple Cases

For some preorders, the preference relation can (efficiently) be reduced to a single reachability objective. For instance, a disjunction of several reachability objectives can obviously be reduced to a single reachability objective, by forming the union of the targets. Formally, we say that a preorder \lesssim is *reducible* to a single (reachability) objective if, given any payoff vector v , we can construct in polynomial time a target \hat{T}^A such that $v \lesssim \mathbf{1}_{\{i | \text{Occ}(\rho) \cap T_i^A \neq \emptyset\}}$ iff $\text{Occ}(\rho) \cap \hat{T}^A \neq \emptyset$. It means that *securing* the payoff corresponds to ensuring a visit to the new target. Similarly, we say that the preorder is *co-reducible* to a single reachability objective, if for any vector v we can construct \hat{T}^A such that $\mathbf{1}_{\{i | \text{Occ}(\rho) \cap T_i^A \neq \emptyset\}} \lesssim v$ if, and only if $\text{Occ}(\rho) \cap \hat{T}^A \neq \emptyset$. It means that *improving* the payoff corresponds to reaching the new target. The disjunction and maximise preorders are examples of preorders that are reducible to single reachability objectives. The disjunction, maximise and subset preorders are co-reducible.

Proposition 8. *For reachability objectives with a (non-trivial) preorder reducible to a single reachability objective, the value problem is P-complete. For a (non-trivial) preorder co-reducible to a single reachability objective, the existence and constrained existence problems are NP-complete.*

³ That is, A and B play with two actions 0 and 1, and for instance moves (0, 0) and (1, 1) lead to z whereas moves (0, 1) and (1, 0) lead to s_0 .

4 Safety Objectives

The results for safety objectives are summarised in Table 2. We begin with a polynomial-space algorithm when the preorder is given as a Boolean circuit, and characterise classes of preorders for which PSPACE-hardness holds. We then consider preorders outside those classes and establish the complexity of the associated problems.

Table 2. Summary of the results for safety

	Preorder	Value problem	(Constr.) existence
	Conjunction	P-c	NP-c
	Subset	P-c	PSPACE-c
	Disjunction, Parity	PSPACE-c	PSPACE-c
	Counting, Maximise, Lexicographic	PSPACE-c	PSPACE-c
	(Monotonic) Boolean Circuit	PSPACE-c	PSPACE-c

Theorem 9. *For safety objectives with preorders given as Boolean circuits, the value, existence and constrained existence problems are in PSPACE. For preorders having $\mathbf{0}$ as a unique minimal element, the existence and constrained existence problems are PSPACE-complete, even for two players. If additionally there is a vector $v \in \{0, 1\}^n$ satisfying the equivalence $v \not\leq v' \Leftrightarrow v' = \mathbf{0}$, then the value problem is PSPACE-complete.*

Proof. In the most general case (Boolean circuits), safety objectives are dual to reachability objectives, hence the PSPACE algorithm.

The hardness proof for the existence problem and preorders with a unique minimal element uses the same arguments as in the proof of Theorem 6. We need to insert a matching-pennies game at the beginning however, because we are interested in Nash equilibria here. Hardness for the value problem is obtained by dualizing the result of Theorem 6 for conjunctions of reachability objectives.

Disjunction, counting, maximise, and lexicographic preorders are examples of preorders that satisfy this condition, and have a unique minimal element. The subset preorder also has a unique element. \square

Note that the hardness results extends to parity, as it can encode disjunction.

We now consider simpler cases. As for reachability, the simple cases are for the preference relations that are *reducible* or *co-reducible* to a single safety objective. For a (non-trivial) preorder reducible to a single safety objective, the value problem retains the same complexity as in the single objective case, namely P-completeness. In the same way, for a (non-trivial) preorder co-reducible to a single safety objective, the existence and constrained existence problems remain NP-complete. The conjunction order is reducible and co-reducible to a single safety objective. The subset preorder is reducible to a single safety objective (but not co-reducible).

5 Büchi Objectives

We now turn to Büchi objectives, for which we prove the results listed in Table 3. (For the definition of the class $P_{\parallel}^{\text{NP}}$, see [13, Chapter 17].)

Table 3. Summary of the results for Büchi objectives

	Preorder	Value	Existence	Constr. exist.
Maximise, Disjunction, Subset		P-c	P-c	P-c
Conjunction, Lexicographic		P-c	P-h, in NP	NP-c
Counting		coNP-c	NP-c	NP-c
Monotonic Boolean Circuit		coNP-c	NP-c	NP-c
Parity		$UP \cap \text{coUP}$ [8]	coNP-h [4], in $P_{\parallel}^{\text{NP}}$	$P_{\parallel}^{\text{NP-c}}$
Boolean Circuit		PSPACE-c	PSPACE-c	PSPACE-c

5.1 Reduction to Zero-Sum Games

In this section, we show how, from a multiplayer game \mathcal{G} , we can construct a two-player game \mathcal{H} , such that there is a correspondence between Nash equilibria in \mathcal{G} and certain winning strategies in \mathcal{H} . This allows us to reuse algorithmic techniques for zero-sum games to solve our problems.

We begin with introducing a few extra definitions. We say that a strategy profile σ_{Agt} is a *trigger strategy* for payoff $(v^A)_{A \in \text{Agt}}$ from state s if for any strategy σ'_A of any player $A \in \text{Agt}$, the outcome ρ of $\sigma_{\text{Agt}}[A \mapsto \sigma'_A]$ from s satisfies $\mathbf{1}_{\{i|\rho \in \Omega_i^A\}} \lesssim v^A$.

Remark 10. A Nash equilibrium is a trigger strategy for the payoff of its outcome. Reciprocally, if the outcome of σ_{Agt} has payoff $(v^A)_{A \in \text{Agt}}$ and σ_{Agt} is a trigger strategy for $(v^A)_{A \in \text{Agt}}$, then σ_{Agt} is a Nash equilibrium.

Given two states s and s' , and a move m_{Agt} , the set of *suspect players* [2] for (s, s') and m_{Agt} , denoted with $\text{Susp}((s, s'), m_{\text{Agt}})$, is the set

$$\{A \in \text{Agt} \mid \exists m' \in \text{Mov}(s, A). \text{Tab}(s, m_{\text{Agt}}[A \mapsto m']) = s'\}.$$

Intuitively, player $A \in \text{Agt}$ is a suspect for transition (s, s') and move m_{Agt} if she can unilaterally change her action to trigger the transition to s' . Notice that if $\text{Tab}(s, m_{\text{Agt}}) = s'$, then $\text{Susp}((s, s'), m_{\text{Agt}}) = \text{Agt}$. Also notice that, given a strategy profile σ_{Agt} , player A is a suspect along all the transitions of a play ρ (i.e., for all index i , player A is in $\text{Susp}((\rho_i, \rho_{i+1}), \sigma_{\text{Agt}}(\rho_{\leq i}))$) iff there is a strategy σ'_A such that $\text{Out}(s, \sigma_{\text{Agt}}[A \mapsto \sigma'_A]) = \rho$.

With a game \mathcal{G} and a payoff $(v^A)_{A \in \text{Agt}}$, we associate a two-player turn-based game $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$. The set V_1 of states (*configurations*) of $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$

controlled by player A_1 is (a subset of) $\text{States} \times 2^{\text{Agt}}$, and the set V_2 of configurations controlled by player A_2 is (a subset of) $\text{States} \times 2^{\text{Agt}} \times \text{Act}^{\text{Agt}}$. The game is played in the following way: from a configuration (s, P) in V_1 , player A_1 chooses a legal move m_{Agt} from s ; the next configuration is (s, P, m_{Agt}) , in which A_2 choose some state $s' \in \text{States}$, and the new configuration is $(s', P \cap \text{Susp}((s, s'), m_{\text{Agt}}))$. In particular, when the state s' chosen by player A_2 satisfies $s' = \text{Tab}(s, m_{\text{Agt}})$ (we say that A_2 obeys A_1), then the new configuration is (s', P) .

We define projections π_1 and π_2 from V_1 on States and 2^{Agt} , respectively, in the natural way. We extend these projections to plays, but only using player A_1 states to avoid stutter, by setting $\pi_1((s_0, P_0)(s_0, P_0, m_0)(s_1, P_1) \dots) = s_0 s_1 \dots$. For any run ρ , $\pi_2(\rho)$ (seen as a sequence of sets of players) is decreasing, therefore its limit $L(\rho)$ is well defined. An outcome ρ is *winning for player A_1* if, for all $A \in L(\rho)$, $\mathbf{1}_{\{i | \pi(\rho) \in \Omega_i^A\}} \lesssim v^A$. In general, since each Ω_i^A is a Büchi objective, the winning condition for A_1 can be represented using a (possibly exponential-size) Muller condition. The *winning region* is the set of configurations (s, P) from which A_1 has a winning strategy. Intuitively, player A_1 tries to have the players play a Nash equilibrium, and player A_2 tries to disprove that the played strategy profile is a Nash equilibrium, by finding a possible deviation that improves the payoff for one of the original players.

At first sight, the number of states in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ is exponential (in the number of players). However, since the transition table Tab is given explicitly [9], the size of \mathcal{G} is $\sum_{s \in \text{States}} \prod_{A \in \text{Agt}} |\text{Mov}(s, A)|$, and we have the following result:

Lemma 11. *The number of reachable configurations from $\text{States} \times \{\text{Agt}\}$ in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ is polynomial in the size of \mathcal{G} .*

The next two lemmas state the correctness of our construction, establishing a correspondence between winning strategies in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ and Nash equilibria in \mathcal{G} .

Lemma 12. *Let $(v^A)_{A \in \text{Agt}}$ be a payoff vector, and ρ be an infinite path in \mathcal{G} . The following two conditions are equivalent:*

- *player A_1 has a winning strategy in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ from (s, Agt) , and its outcome ρ' from (s, Agt) when A_2 obeys A_1 is such that $\pi_1(\rho') = \rho$;*
- *there is a trigger strategy for $(v^A)_{A \in \text{Agt}}$ in \mathcal{G} from state s whose outcome from s is ρ .*

Proof. Assume there is a winning strategy σ^1 for player A_1 in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ from (s, Agt) . We define the strategy profile σ_{Agt} according to the actions played by A_1 . Pick a history $g = s_1 s_2 \dots s_{k+1}$ with $s_1 = s$. Let h be the outcome of σ^1 from s ending in a player A_1 state and such that $\pi_1(h) = s_1 \dots s_k$. This history is uniquely defined as follows: the first state of h is (s_1, Agt) , and if its $(2i + 1)$ -th state is (s_i, P_i) , then its $(2i + 2)$ -th state is $(s_i, P_i, \sigma^1(h_{\leq 2i+1}))$ and its $(2i + 3)$ -th state is $(s_{i+1}, P_i \cap \text{Susp}((s_i, s_{i+1}), \sigma^1(h_{\leq 2i+1})))$. Now, write (s_k, P_k) for the last state of h , and let $h' = h \cdot (s_k, P_k, \sigma^1(h)) \cdot (s_{k+1}, P_k \cap \text{Susp}((s_k, s_{k+1}), \sigma^1(h)))$. Then we define $\sigma_{\text{Agt}}(g) = \sigma^1(h')$. Notice that when $g \cdot s$ is a prefix of $\pi_1(\rho')$

(where ρ' is the outcome of σ^1 from s when A_2 obeys A_1), then $g \cdot s \cdot \sigma_{\text{Agt}}(g \cdot s)$ is also a prefix of $\pi_1(\rho')$.

We now prove that σ_{Agt} is a trigger strategy for $(v^A)_{A \in \text{Agt}}$. Pick a player $A \in \text{Agt}$, a strategy σ'_A for A , and an infinite play g in $\text{Out}(s, \sigma_{\text{Agt}}[A \mapsto \sigma'_A])$. With g , we associate an infinite play h in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ in the same way as above. Then player A is a suspect along all the transitions of g , so that she belongs to $L(h)$. Now, as σ^1 is winning, the payoff for A of $g = \pi_1(h)$ is less than v^A , which proves that σ_{Agt} is a trigger strategy.

Conversely, assume that σ_{Agt} is a trigger strategy for $(v^A)_{A \in \text{Agt}}$, and define the strategy σ^1 by $\sigma^1(h) = \sigma_{\text{Agt}}(\pi_1(h))$. Notice that the outcome ρ' of σ^1 when A_2 obeys A_1 satisfies $\pi_1(\rho') = \rho$.

Let η be an outcome of σ^1 from s , and $A \in L(\eta)$. Then A is a suspect for each transition along $\pi_1(\eta)$, which means that for all i there is a move m_i^A such that $\pi_1(\eta)_{=i+1} = \text{Tab}(\pi_1(\eta)_{=i}, \sigma_{\text{Agt}}(\pi_1(\eta)_{\leq i})[A \mapsto m_i^A])$. Therefore there is a strategy σ'_A such that $\pi_1(\eta) = \text{Out}(s, \sigma_{\text{Agt}}[A \mapsto \sigma'_A])$. Since σ_{Agt} is a trigger strategy for $(v^A)_{A \in \text{Agt}}$, the payoff for player A of $\pi_1(\eta)$ is less than v_A . As this holds for any $A \in L(\eta)$, σ^1 is winning. \square

Lemma 13. *Let ρ be an infinite path in \mathcal{G} with payoff $(v^A)_{A \in \text{Agt}}$. The following two conditions are equivalent:*

- *there is a path ρ' from (s, Agt) in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ that never leaves the winning region of A_1 and along which A_2 obeys A_1 , such that $\pi_1(\rho') = \rho$;*
- *there is a Nash equilibrium σ_{Agt} from s in \mathcal{G} whose outcome is ρ .*

Proof. Let ρ be a path in the winning region of A_1 in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$. We define a strategy σ^1 that follows ρ when A_2 obeys. Along ρ , this strategy is defined as follows: $\sigma^1(\rho_{\leq 2i}) = m_{\text{Agt}}$ such that $\text{Tab}(\pi_1(\rho)_{=i}, m_{\text{Agt}}) = \pi_1(\rho)_{=i+1}$. Such a legal move must exist since A_2 obeys A_1 along ρ . Now, if player A_2 deviates from the obeying strategy, we make σ^1 follow a winning strategy of A_1 : given a finite outcome η of σ^1 that is not a prefix of ρ , we let j be the largest index such that $\eta_{\leq j}$ is a prefix of ρ . In particular, $\eta_{=j}$ belongs to the winning region W of A_1 , and belongs to player A_2 (otherwise $\eta_{\leq j+1}$ would also be a prefix of ρ). Hence, all the successors of $\eta_{=j}$ are in W . Thus player A_1 has a winning strategy $\hat{\sigma}^1$ from $\eta_{=j+1}$. We then define $\sigma^1(\eta_{\leq j} \cdot \eta') = \hat{\sigma}^1(\eta')$ for any outcome η' of $\hat{\sigma}^1$ from $\eta_{=j+1}$.

Each outcome of σ^1 is either the path ρ or a path that, from some point onwards, is compatible with a winning strategy. Since $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$ has a Muller winning condition, it follows that σ^1 is winning. Applying Lemma 12, we obtain a strategy profile σ_{Agt} in \mathcal{G} that is a trigger strategy for $(v^A)_{A \in \text{Agt}}$. Moreover, the outcome of σ_{Agt} from s equals $\pi_1(\rho)$, so that σ_{Agt} is a Nash equilibrium.

Conversely, the Nash equilibrium is a trigger strategy, and from Lemma 12, we get a winning strategy σ^1 in $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$. The outcome ρ of σ^1 from s when A_2 obeys A_1 is such that $\pi_1(\rho)$ is the outcome of the Nash equilibrium, so that its payoff is $(v^A)_{A \in \text{Agt}}$. Since σ^1 is winning, ρ never leaves the winning region, which concludes the proof. \square

5.2 Applications of the Reduction

General Case. As noticed in [7], the algorithm from [10] to find the winning states in a game can be adapted to the case where the winning conditions are given as a Boolean circuit (the circuit has as many input gates as the number of states, and a path is declared winning if the circuit evaluates to 1 when setting the input gates to 1 for the states that are visited infinitely often). It uses polynomial space. Using such an algorithm we get the following result.

Proposition 14. *For Büchi objectives with preorders given as Boolean circuits, the value, existence and constrained existence problems are PSPACE-complete.*

Reduction to a Single Büchi Objective. The preorders that were reducible to a single reachability objectives in the case of reachability can also be reduced to a single Büchi objective in the Büchi case: just replace Occ with Inf. The same holds for co-reducibility. The algorithm from [4] can then be adapted.

Proposition 15. *For Büchi objectives with a monotonic preorder reducible to a single objective, the value problem is P-complete. For Büchi objectives with a preorder co-reducible to a single objective, the existence and constrained existence problems are P-complete.*

Reduction to a Deterministic Büchi Automaton. For some preorders, given any payoff u , it is possible to construct (in polynomial time) a deterministic Büchi automaton that recognises the plays whose payoff v for player A is higher than u (i.e. $u \lesssim v$). When this is the case, we say that the preorder is *reducible to a deterministic Büchi automaton*.

Proposition 16. *For Büchi objectives and a preorder reducible to a deterministic Büchi automaton, the value problem is in P. In particular, it is P-complete for conjunction, lexicographic and subset preorders.*

The idea of the algorithm is to compute the product of the game with the Büchi automaton to which the given payoff v^A reduces. Notice that *reachability* objectives with the parity order are also reducible to a deterministic Büchi automaton; we thus recover the complexity result about weak parity games from [12].

Monotonic Preorders. When the preorder is monotonic, our problems are also simpler than in the general case. This is because we can find suspect-based trigger strategies, corresponding to memoryless strategies in the game $\mathcal{H}(\mathcal{G}, (v^A)_{A \in \text{Agt}})$.

Proposition 17. *For Büchi objectives with preorders given by a monotonic circuits, the value problem is coNP-complete, and the existence and constrained existence problem are NP-complete. For the counting order, the value problem is coNP-complete, and existence and constrained existence are NP-complete. For monotonic preorders with an element v such that $u \not\lesssim v \Leftrightarrow u = \mathbf{1}$, the constrained existence problem is NP-complete.*

Parity Games. Finally, for Büchi objectives with the parity preorder, we have:

Proposition 18. *For Büchi objectives with the parity preorder, the constrained existence problem is $\text{P}_{\parallel}^{\text{NP}}$ -complete.*

6 Conclusion

We have contributed to the algorithmics of Nash equilibria computation in concurrent games with ordered objectives. We believe the game abstraction proposed in Section 5.1 can be used in other contexts, which we are currently investigating. The algorithms presented in this paper have partly been implemented in the tool PRALINE (<http://www.lsv.ens-cachan.fr/Software/praline/>).

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. *J. ACM* 49(5), 672–713 (2002)
2. Bouyer, P., Brenguier, R., Markey, N.: Nash Equilibria for Reachability Objectives in Multi-player Timed Games. In: Gastin, P., Laroussinie, F. (eds.) *CONCUR 2010*. LNCS, vol. 6269, pp. 192–206. Springer, Heidelberg (2010)
3. Bouyer, P., Brenguier, R., Markey, N., Ummels, M.: Concurrent games with ordered objectives. Research Report LSV-11-22, LSV, ENS Cachan, France (2011)
4. Bouyer, P., Brenguier, R., Markey, N., Ummels, M.: Nash equilibria in concurrent games with Büchi objectives. In: *FSTTCS 2011*. LIPIcs, vol. 13, pp. 375–386. Leibniz-Zentrum für Informatik, LZI (2011)
5. Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash Equilibria in Stochastic Games. In: Marcinkowski, J., Tarlecki, A. (eds.) *CSL 2004*. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)
6. Henzinger, T.A.: Games in system design and verification. In: *TARK 2005*, pp. 1–4 (2005)
7. Hunter, P., Dawar, A.: Complexity Bounds for Regular Games. In: Jędrzejowicz, J., Szepietowski, A. (eds.) *MFCS 2005*. LNCS, vol. 3618, pp. 495–506. Springer, Heidelberg (2005)
8. Jurdzinski, M.: Deciding the winner in parity games is in $UP \cap co-UP$. *Inf. Process. Lett.* 68(3), 119–124 (1998)
9. Laroussinie, F., Markey, N., Oreiby, G.: On the expressiveness and complexity of ATL. *Logical Methods in Computer Science* 4(2) (2008)
10. McNaughton, R.: Infinite games played on finite graphs. *Annals of Pure and Applied Logic* 65(2), 149–184 (1993)
11. Nash Jr., J.F.: Equilibrium points in n -person games. *Proc. National Academy of Sciences of the USA* 36(1), 48–49 (1950)
12. Neumann, J., Szepietowski, A., Walukiewicz, I.: Complexity of weak acceptance conditions in tree automata. *Inf. Process. Lett.* 84(4), 181–187 (2002)
13. Papadimitriou, C.H.: *Complexity Theory*. Addison-Wesley (1994)
14. Paul, S., Simon, S.: Nash equilibrium in generalised Muller games. In: *FSTTCS 2009*. LIPIcs, vol. 4, pp. 335–346. LZI (2010)
15. Thomas, W.: Infinite Games and Verification (Extended abstract of a tutorial). In: Brinksma, E., Larsen, K.G. (eds.) *CAV 2002*. LNCS, vol. 2404, pp. 58–64. Springer, Heidelberg (2002)
16. Ummels, M.: The Complexity of Nash Equilibria in Infinite Multiplayer Games. In: Amadio, R.M. (ed.) *FOSSACS 2008*. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)
17. Ummels, M., Wojtczak, D.: The Complexity of Nash Equilibria in Limit-Average Games. In: Katoen, J.-P., König, B. (eds.) *CONCUR 2011 – Concurrency Theory*. LNCS, vol. 6901, pp. 482–496. Springer, Heidelberg (2011)

Full Abstraction for Set-Based Models of the Symmetric Interaction Combinators

Damiano Mazza¹ and Neil J. Ross²

¹ LIPN, UMR 7030 CNRS-Université Paris 13
Damiano.Mazza@lipn.univ-paris13.fr

<http://www-lipn.univ-paris13.fr/~mazza>

² Department of Mathematics and Statistics, Dalhousie University
neil.jr.ross@dal.ca

Abstract. The symmetric interaction combinators are a model of distributed and deterministic computation based on Lafont’s interaction nets, a special form of graph rewriting. The interest of the symmetric interaction combinators lies in their universality, that is, the fact that they may encode all other interaction net systems; for instance, several implementations of the lambda-calculus in the symmetric interaction combinators exist, related to Lamping’s sharing graphs for optimal reduction. A certain number of observational equivalences were introduced for this system, by Lafont, Fernandez and Mackie, and the first author. In this paper, we study the problem of full abstraction with respect to one of these equivalences, using a class of very simple denotational models based on pointed sets.

Keywords: Interaction nets, Observational equivalence, Denotational semantics.

1 Introduction

The symmetric interaction combinators. Interaction nets are a model of distributed, deterministic computation introduced by Lafont [7]. By *distributed* we mean that computation, which is based on graph rewriting, is such that elementary rewriting steps may be applied at different places in the graph, in a completely asynchronous way. By *deterministic* we mean that such elementary steps never overlap, so that the order in which they are executed does not matter, and the computation is essentially unique.

In terms of expressiveness, interaction nets are extremely versatile; for instance, Turing machines may be seen as special interaction nets, in which parallelism is absent. Also, the optimal implementation of λ -calculus evaluation uses certain systems of interaction nets, known as *sharing graphs* [9,10].

Among all systems of interaction nets, the *interaction combinators* [8] are of special interest, because of their simplicity and universality: in spite of being composed of only 3 combinators with 6 rewriting rules, any system of interaction nets may be encoded in the interaction combinators, preserving the parallelism

of computations. Additionally, the λ -calculus under certain reduction strategies (such as head reduction) may be directly represented in the interaction combinators [11].

The symmetric interaction combinators are a variant of the interaction combinators, also introduced by Lafont [8], having essentially the same expressiveness (for instance, the results of [11] may be immediately transported to the symmetric interaction combinators). The advantage of considering this system is that it is even simpler: although there are still 3 combinators and 6 rules, these latter may actually be arranged in just 2 patterns (annihilations and commutations), and the systems lends itself to a simpler study from the point of view of denotational semantics [12].

Contextual observational equivalences. As an abstract programming language, the symmetric interaction combinators are not too far from the λ -calculus: computation is based on a confluent rewriting relation (yielding the analog of β -equivalence), there is a notion of normal form, a relation analogous to η -equivalence, and so on. Additionally, one may also easily formulate the concept of *context*, from which a rich class of *observational equivalences* may be defined.

As a matter of fact, after Morris [14], we have a general way of defining observational equivalences in a language with an internal notion of context: given a set of syntactic objects S , one defines, for any two syntactic objects t, u ,

$$t \simeq_S u \text{ iff, for every context } C, C[t] \in S \text{ iff } C[u] \in S.$$

In languages like the λ -calculus, one usually considers S to be closed under β -equivalence, so that this latter is automatically contained in \simeq_S . Morris himself introduced the first, still widely used, observational equivalence for the λ -calculus, by taking S as the set of normalizable λ -terms; we denote such equivalence by \simeq_n . Following this pattern, other interesting equivalences may be defined; we refer the reader to [3] for a detailed survey.

The presence of contexts allows more generally to consider the notion of *congruence* on the syntax, *i.e.*, an equivalence relation \sim such that $t \sim u$ implies $C[t] \sim C[u]$, for all objects t, u and context C . Obviously, a Morris observational equivalence is a congruence. Given a congruence \sim , it is natural to ask whether and how it relates to a particular observational equivalence \simeq . We say that \sim is an *abstraction* of \simeq when $\simeq \subseteq \sim$; it is *adequate* with respect to \simeq when $\sim \subseteq \simeq$. These two properties are most frequently investigated when \sim is induced by a denotational model; in particular, one seeks a model whose induced congruence enjoys both of them, in which case the model is said to be *fully abstract*.

The observational equivalence \simeq_n on λ -terms has been widely studied and characterized in several different ways: it is the congruence $=_{\mathfrak{B}\mathfrak{T}_n}$ induced by equality of η -normal Bhöm trees [6]; and it is the congruence induced by equality in Coppo, Dezani-Ciancaglini and Zacchi's filter model defined in [2].

Back to the symmetric interaction combinators. It is fairly natural to attempt reformulating all of the above congruences, and the questions concerning them, in the symmetric interaction combinators. After Lafont's initial work [8], Fernández

and Mackie were the first to formulate a notion of observational equivalence for interaction nets [4]. More recently, the first author [13] introduced a notion of solvability for nets of symmetric interaction combinators, together with the concept of *observable axiom*, corresponding to a “head variable”. With these, comes a notion of *edifice*, which is a sort of infinite normal form for nets, analogous to a Böhm tree. According to these notions, Fernández and Mackie’s observational equivalence turns out to be quite strong: it is not *sensible*, *i.e.*, it does not identify all unsolvable nets. Alternatively, in analogy with the λ -calculus, it is possible to consider the following two congruences on nets: one which we still denote by \simeq_n , the Morris observational equivalence induced by normalizable nets; and $=_{\mathfrak{E}_\eta}$, the congruence induced by equality of η -expanded edifices, analogous to $=_{\mathfrak{B}\mathfrak{T}_\eta}$.

Perhaps surprisingly, it turns out that these two congruences are distinct: in fact, in the symmetric interaction combinators, \simeq_n is not even *semi-sensible*, *i.e.*, it equates a solvable and an unsolvable net; this is due to the purely local nature of reduction in interaction nets, in which diverging computations cannot be erased.

Interestingly, equality of η -expanded edifices does correspond to an observational equivalence on nets, called *finitary axiom-equivalence*, also introduced in [13]. Rephrased in terms of Böhm trees, this would correspond to the Morris equivalence \simeq_{B_f} , where B_f is the set of λ -terms whose Böhm tree is finite. In the λ -calculus, it coincides with \simeq_n (to prove $\simeq_n \subseteq \simeq_{B_f}$, simply observe that $=_{\mathfrak{B}\mathfrak{T}_\eta}$ obviously discriminates between terms in and not in B_f ; to prove $\simeq_{B_f} \subseteq \simeq_n$, consider the contrapositive, and use Böhm’s theorem to find a context discriminating with respect to B_f); in the symmetric interaction combinators, it is a wholly different equivalence.

The contribution of this paper. Our present objective is to study the question of full abstraction for finitary axiom-equivalence in terms of set-based models, which have a more “abstract” flavor than edifices (just like, say, Plotkin’s model $P\omega$ [1] is more “abstract” than Böhm trees). These models, which were introduced by the first author [12], interpret nets as pointed sets, and are based on the notion of *experiment*, first used by Girard for linear logic [5]. The interest of set-based models lies in their simplicity and concreteness: in many cases, the denotational interpretation of a net may be explicitly computed with ease.

Technically, these models are given by an *interaction set*, which is a pointed set X together with two pointed bijections onto $X \oplus X$ (the product of X with itself), satisfying a certain commutation property. We give sufficient conditions for an interaction set to be fully abstract with respect to finitary axiom-equivalence, and we provide a concrete example of such an interaction set, which thus plays the role, in interaction nets, that Coppo, Dezani-Ciancaglini and Zacchi’s filter model [2] plays in the λ -calculus. The conditions we give are all related to some kind of *approximation property*, *i.e.*, the fact that the denotation of a net is entirely determined by the denotation of its approximations.

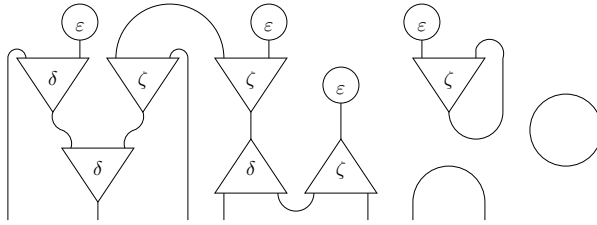


Fig. 1. A net

2 The Symmetric Interaction Combinators

2.1 Nets

Computation in the symmetric interaction combinators (or, henceforth, simply the symmetric combinators) is a special form of graph rewriting. The graphs on which computation is performed are called *nets*, which are composed by *cells* and *wires*.

Each cell carries a *symbol*, and has a number of *ports*, exactly one of which is *principal*, the other being *auxiliary*. There are three symbols: δ , ε , and ζ . We shall use the letters α, β to range over symbols. Cells of type δ or ζ have two auxiliary ports, hence are called *binary*, and are represented by a triangle; the principal port is represented by one of the “tips” of the triangle, while the auxiliary ports are on the opposite edge. Auxiliary ports are numbered: port 1 (resp. 2) is the “left” (resp. “right”) auxiliary port when the cell is represented with its principal port pointing “down”, and the numbering is preserved by rotations. Cells of type ε have no auxiliary port, hence are called *nullary*, and are represented by a circle.

A *wire* has exactly two extremities, which may be connected to the ports of cells. A *loop* is a wire with its extremities connected together. A wire which is not a loop is called *proper*.

A *net* is a finite (possibly empty) collection of cells and wires, such that each port of each cell is attached to the extremity of a wire. Nets will be ranged over by μ, ν . A net may contain proper wires with one or both extremities not connected to any cell; these are called the *free ports* of the net. If a net has n free ports, they are supposed to be numbered by the integers $1, \dots, n$. As an example, the net in Fig. 1 has 11 cells, of which 4 nullary, 1 loop, 16 proper wires, and 7 free ports, assumed to be numbered increasingly from “left” to “right”.

Let us introduce some remarkable nets, which will be useful in the sequel. A *wiring* is a net containing no cell and no loop. Wirings are permutations of free ports; they are ranged over by ω . We shall often use ω also to denote a single wire.

The net with n free ports consisting of n ε cells is denoted by \mathbf{E}_n .

A *tree* is a net defined by induction as follows. A single ε cell is a tree with no leaf, denoted by ε ; a proper wire is a tree with one leaf, denoted by \bullet ; if τ_1, τ_2 are two trees with resp. n_1, n_2 leaves, and if α is a binary symbol, the net obtained by

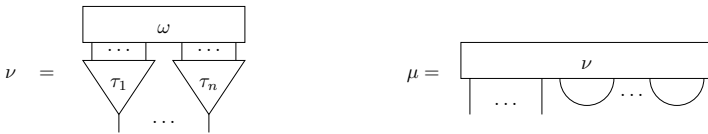
plugging the root of τ_i into the auxiliary port i of an α cell, with $i \in \{1, 2\}$, is a tree with $n_1 + n_2$ leaves, denoted by $\alpha(\tau_1, \tau_2)$. Trees are represented adopting the same graphical notations as cells. We shall avoid possible ambiguities by never using $\delta, \varepsilon, \zeta$ to denote trees, and by using α, β exclusively to range over cell symbols, so that a triangle annotated with α or β will unambiguously represent a single cell.

An *active pair* is a net consisting of two cells whose principal ports are connected by a wire.

It is also useful to define some special sorts of wires in nets. An *axiom* is a proper wire such that none of its extremities is the principal port of a cell. A *cut* is a proper wire connecting two principal ports. An *axiom-cut* is either a loop, or a proper wire connecting the root of a tree to one of its own leaves. We say that a net is *cut-free* if it contains no cuts and no axiom-cuts. Note that cuts are in one-to-one correspondence with active pairs. As an example, consider the net in Fig. 11, in which the reader should find 7 axioms, 2 cuts (or active pairs), and 2 axiom-cuts.

The following result is proved by induction on the number of cells.

Lemma 1 (Shape). *Let ν be a cut-free net with n free ports. Then, for each $1 \leq i \leq n$ there exist a unique tree τ_i , and there exists a unique wiring ω such that the equality below on the left holds. Let μ be a net with n free ports and k cuts and axiom-cuts. Then, there exists a cut-free net ν with $n + 2k$ free ports such that the equality below on the right holds.*



Note that we use rectangles to represent generic nets, including wirings. However, ω will always denote a wiring. Observe that all wires in the wiring ω of the left equality are axioms; in fact, that is the shape of a generic cut-free proof net of multiplicative linear logic [5], with the axiom links in ω and the logical links in τ_1, \dots, τ_n , whence our terminology. Also observe that the net ν of the right equality is unique as soon as μ does not contain axiom-cuts.

We conclude this section with the essential notion of *context*:

Definition 1 (Context, test). *Let μ be a net with n free ports. A context for μ is a net C with at least n free ports. We denote by $C[\mu]$ the application of C to μ , which is the net obtained by plugging the free port i of μ to the free port i of C , with $i \in \{1, \dots, n\}$. A test for μ is a particular context consisting of n trees τ_1, \dots, τ_n such that the root of each τ_i is the free port i .*

In the sequel, when we use the notation $C[\mu]$ we implicitly assume that C has enough ports so that μ can be plugged into it. Moreover, we shall say that μ' is a *subnet* of μ if there exists C such that $\mu = C[\mu']$.

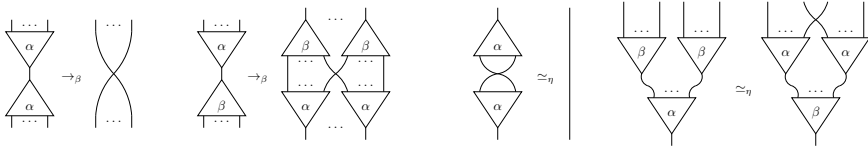


Fig. 2. The rules and equations defining β -reduction and η -equivalence. We assume $\alpha \neq \beta$. In the left β -rule, the right member is empty in case $\alpha = \varepsilon$. In the left η -equation, α is binary.

2.2 Reductions and Equivalences

Computation in the symmetric combinators is performed by rewriting active pairs. We define \rightarrow_β as the contextual closure of the rules of Fig. 2, and denote by \rightarrow_β^* its reflexive-transitive closure. Since active pairs are always disjoint, the relation \rightarrow_β trivially satisfies the diamond property, i.e., reduction is strongly confluent. A confluent rewriting relation always induces an equivalence relation (in this case, a congruence): we define $\mu \simeq_\beta \mu'$ iff there is ν such that $\mu \rightarrow_\beta^* \nu$ and $\mu' \rightarrow_\beta^* \nu$. As usual, a net is *normal* when no β -reduction applies to it. Note that cut-free nets are always normal, but normal nets need not be cut-free.

There is also a notion of η -equivalence, first introduced by Lafont [8] and Fernández and Mackie [4]. Unlike in the λ -calculus, it cannot be presented as the symmetrization of a rewriting relation, because the rightmost equation of Fig. 2 when both α, β are binary is intrinsically non orientable. We define \simeq_η as the reflexive, transitive, and contextual closure of the equations of Fig. 2. Then, we define $\beta\eta$ -equivalence as $\simeq_{\beta\eta} = (\simeq_\beta \cup \simeq_\eta)^+$.

The following definitions were introduced by the first author in [13], and are inspired by similar notions in the λ -calculus [1].

Definition 2 (Solvability). A quasi-wire is a net of the following shape:



where μ_0 is any net with no free ports (including the empty net). A net μ is solvable if there exists a test θ such that $\theta[\mu] \rightarrow_\beta^* W$, where W is a quasi-wire. A net is unsolvable if it is not solvable.

We write $\mu \rightarrow_\varepsilon \mu'$ if $\mu = C[\mu_0]$, $\mu' = C[\mathbf{E}_n]$, and μ_0 is an unsolvable net with n free ports different from \mathbf{E}_n . We then define $\rightarrow_{\beta\varepsilon}$ as the union of \rightarrow_β and \rightarrow_ε , and denote by $\rightarrow_{\beta\varepsilon}^*$ its reflexive-transitive closure. The relation $\rightarrow_{\beta\varepsilon}^*$ too may be proved to enjoy the diamond property [13], so we may consider its induced congruence $\simeq_{\beta\varepsilon}$, and set $\simeq_{\beta\eta\varepsilon} = (\simeq_{\beta\varepsilon} \cup \simeq_\eta)^+$ (the transitive closure of union).

The following Morris observational equivalence was also introduced in [13].

Definition 3 (Finitary axiom-equivalence). Let μ be a net with n free ports. We write $\mu \Downarrow$ if μ is $\beta\varepsilon$ -normalizable; otherwise, we write $\mu \Uparrow$. Let μ, μ' be two nets with the same number of free ports. We say that they are finitarily axiom-equivalent, and we write $\mu \cong \mu'$, if, for every context C , $C[\mu] \Downarrow$ iff $C[\mu'] \Downarrow$.

3 Denotational Semantics

Informally, a denotational semantics of a programming language with a notion of evaluation (denoted by \rightarrow_β) and a notion of context is an interpretation $\llbracket \cdot \rrbracket$ of the syntax into some kind of mathematical structure (which might be the syntax itself) which satisfies the following, for all syntactic objects t, u :

Invariance: $t \rightarrow_\beta u$ implies $\llbracket t \rrbracket = \llbracket u \rrbracket$;

Congruence: for every context C , $\llbracket t \rrbracket = \llbracket u \rrbracket$ implies $\llbracket C[t] \rrbracket = \llbracket C[u] \rrbracket$.

By definition, a denotational semantics induces a congruence on the syntax, by setting $t \sim u$ iff $\llbracket t \rrbracket = \llbracket u \rrbracket$. We make the following important remark, concerning the adequacy property. If \simeq_S is the Morris observational equivalence based on a set S , it is easy to see that it is the greatest congruence contained in $(S \times S) \cup (\mathbb{C}S \times \mathbb{C}S)$; therefore, the semantic congruence is adequate w.r.t. \simeq_S iff $\llbracket t \rrbracket = \llbracket u \rrbracket$ implies that either both t, u are in S , or neither is.

3.1 Interaction Sets

The first set-based denotational semantics for the symmetric combinators was introduced in [12]. It is based on *pointed sets*, *i.e.*, sets with a distinguished element, which we denote by $\mathbf{0}$. A morphism of pointed sets, or *pointed function*, is a function between the underlying sets which preserves the distinguished element. Pointed sets and their morphisms form a category, which is equivalent to the category of sets and *partial* functions. This category has a zero object (the pointed set $\{\mathbf{0}\}$) and biproducts: given a family of pointed sets $(X_i)_{i \in I}$, their biproduct $\bigoplus_{i \in I} X_i$ is the pointed set whose underlying set is the product of the underlying sets of the family, and whose distinguished element is the everywhere-zero sequence.

Definition 4 (Interaction set). *An interaction set is a triple $(X, \langle \cdot, \cdot \rangle, [\cdot, \cdot])$ where X is a pointed set, and $\langle \cdot, \cdot \rangle$ and $[\cdot, \cdot]$ are isomorphisms between $X \oplus X$ and X satisfying*

$$\langle [x, y], [z, w] \rangle = [\langle x, z \rangle, \langle y, w \rangle],$$

for all $x, y, z, w \in X$. An interaction set is non-trivial if $X \neq \{\mathbf{0}\}$.

Let $\varphi, \psi : X \rightarrow X \oplus X$ be the inverses of $\langle \cdot, \cdot \rangle$ and $[\cdot, \cdot]$, respectively, and let $\pi_1, \pi_2 : X \oplus X \rightarrow X$ be the projections associated with the biproduct. Then, for $i \in \{1, 2\}$, we define $\delta_i = \pi_i \circ \varphi$ and $\zeta_i = \pi_i \circ \psi$. In other words, for all $x \in X$, $\delta_1(x)$ and $\delta_2(x)$ are the unique elements of X such that $\langle \delta_1(x), \delta_2(x) \rangle = x$, and similarly for ζ_i and $[\cdot, \cdot]$.

In the sequel, we shall denote interaction sets by specifying only their underlying pointed set, leaving the bijections implicit.

Every interaction set X induces a denotational semantics of the symmetric combinators. A net with n free ports is interpreted as a pointed subset of $X^n = X \oplus \cdots \oplus X$ (n times), as follows.

Definition 5 (Experiment, interpretation). Let X be an interaction set, and let μ be a net with n free ports. An experiment on μ in X is a function e from the ports of μ (including the free ports) to X such that:

- if p, q are two ports connected by a wire, $e(p) = e(q)$;
- if p_1, p_2 are auxiliary ports number 1 and 2 of a δ (resp. ζ) cell whose principal port is q , then $e(q) = \langle e(p_1), e(p_2) \rangle$ (resp. $e(q) = [e(p_1), e(p_2)]$);
- if q is the principal port of a ε cell, then $e(q) = \mathbf{0}$.

If $p_1 < \dots < p_n$ are the free ports of μ , the sequence $(e(p_1), \dots, e(p_n))$ is said to be the result of e . We use the notation $e : \mu$ as a shorthand for “ e is an experiment on μ ” (the interaction set will always be clear from the context), and we denote by $|e|$ the result of e .

The interpretation of μ in X , denoted by $\llbracket \mu \rrbracket$, is the set containing all results of all experiments on μ in X .

Note that $\llbracket \mathbf{E}_n \rrbracket = \{(\mathbf{0}, \dots, \mathbf{0})\}$. The fact that $\llbracket \mu \rrbracket$ is a pointed set is then immediate:

Lemma 2. For every net μ with n free ports, $\llbracket \mathbf{E}_n \rrbracket \subseteq \llbracket \mu \rrbracket$.

Proof. The function assigning $\mathbf{0}$ to all ports of μ is always an experiment. □

Proposition 1. For all nets μ, μ' , $\mu \simeq_{\beta\eta} \mu'$ implies $\llbracket \mu \rrbracket = \llbracket \mu' \rrbracket$. Moreover, $\llbracket \cdot \rrbracket$ enjoys the congruence property.

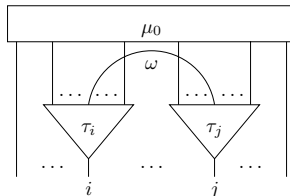
Proof. To prove invariance under reduction, given $\mu \rightarrow_{\beta} \mu'$, it is enough to show that, for every experiment on μ , there is an experiment on μ' yielding the same result, and vice versa. Invariance under η -equivalence is proved in the same way. The congruence property is an immediate consequence of the definition of experiment. Details may be found in [12]. □

3.2 Edifices

The following notion, introduced in [13], is analogous to that of a head variable in the λ -calculus.

Definition 6 (Observable axiom). Let μ be a net. An observable axiom of μ is an axiom connecting the leaves of two trees τ_i, τ_j whose respective roots i, j are both free ports of μ . We say that such observable axiom is based at i, j .

It is perhaps useful to visualize observable axioms. A net μ contains an observable axiom ω iff it is of the following shape:



If $i = j$, then $\tau_i = \tau_j$, and ω connects two leaves of the same tree. Note also that one or both of τ_i, τ_j may be equal to a wire; in particular, a wire whose both extremities are free is an observable axiom.

The following result is proved in [13]. It is analogous to the λ -calculus result stating that solvability is equivalent to having a head normal form.

Proposition 2. *A net μ is solvable iff there exists a net μ' containing an observable axiom such that $\mu \rightarrow_{\beta}^* \mu'$.*

The observable axioms appearing during the reduction of a net may be collected, just like the head variables of a λ -term, to form the analogous of a Böhm tree. We first need to assign a unique identifier to each observable axiom within a net. In what follows, \mathbb{W} denotes the set of finite words over $\{1, 2\}$. We use a, b to range over \mathbb{W} and denote the empty word by ϵ . Concatenation of words is denoted by juxtaposition.

Definition 7 (Address of a leaf). *Let τ be a tree of cells, and let l be a leaf of τ . We define the δ -address and ζ -address of l in τ , denoted by $\text{addr}_{\delta}^{\tau}(l)$ and $\text{addr}_{\zeta}^{\tau}(l)$, respectively, by induction on τ :*

- if $\tau = \bullet$, then $\text{addr}_{\delta}^{\tau}(l) = \text{addr}_{\zeta}^{\tau}(l) = \epsilon$;
- if $\tau = \delta(\tau_1, \tau_2)$, and l belongs to τ_i (with $i \in \{1, 2\}$), then $\text{addr}_{\delta}^{\tau}(l) = i\text{addr}_{\delta}^{\tau_i}(l)$, and $\text{addr}_{\zeta}^{\tau}(l) = \text{addr}_{\zeta}^{\tau_i}(l)$;
- if $\tau = \zeta(\tau_1, \tau_2)$, and l belongs to τ_i (with $i \in \{1, 2\}$), then $\text{addr}_{\delta}^{\tau}(l) = \text{addr}_{\delta}^{\tau_i}(l)$, and $\text{addr}_{\zeta}^{\tau}(l) = i\text{addr}_{\zeta}^{\tau_i}(l)$.

Definition 8 (Arch). *A pillar is an element of $\mathbb{W} \times \mathbb{W} \times \mathbb{N}$, denoted by $(a, b)@i$; an arch is a set containing exactly two pillars, denoted by $(a, b)@i \frown (a', b')@i'$ (which is the same as $(a', b')@i' \frown (a, b)@i$). Arches are ranged over by \mathbf{a} .*

Let ω be an observable axiom of the net μ . By definition, ω connects two leaves l_i, l_j of two trees τ_i, τ_j whose roots i, j are free ports of μ . Then, we represent ω by the arch

$$(\text{addr}_{\delta}^{\tau_i}(l_i), \text{addr}_{\zeta}^{\tau_i}(l_i))@i \frown (\text{addr}_{\delta}^{\tau_j}(l_j), \text{addr}_{\zeta}^{\tau_j}(l_j))@j.$$

Note that different observable axioms are necessarily represented by different arches; even more, two different observable axioms have no pillar in common. This is because each leaf of a tree in a net may only be connected to one extremity of one observable axiom.

Definition 9 (Edifice). *Let μ be a net. We denote by $\text{ax}(\mu)$ the set of all arches representing all observable axioms of μ . Then, we define the edifice of μ to be the following set of arches:*

$$\mathfrak{E}(\mu) = \bigcup_{\mu \rightarrow_{\beta}^* \mu'} \text{ax}(\mu').$$

The η -expanded edifice of μ is defined by

$$\mathfrak{E}_{\eta}(\mu) = \{(ac, bd)@i \frown (a'c, b'd)@i' \mid (a, b)@i \frown (a', b')@i' \in \mathfrak{E}(\mu), c, d \in \mathbb{W}\}.$$

In other words, $\mathfrak{E}(\mu)$ is the set of all arches representing all observable axioms appearing during the reduction of μ , and $\mathfrak{E}_\eta(\mu)$ is obtained by “ η -expanding” the arches in all possible ways.

As mentioned above, edifices are to nets what Böhm trees are to λ -terms. Indeed, they too yield a denotational semantics, whose induced congruence contains $\simeq_{\beta\eta\varepsilon}$ and is fully abstract with respect to finitary axiom-equivalence:

Proposition 3 (Full abstraction for edifices). *For all nets $\mu, \mu', \mu \cong \mu'$ iff $\mathfrak{E}_\eta(\mu) = \mathfrak{E}_\eta(\mu')$.*

Proof. The result is non-trivial; we refer the reader to [13]. □

Finally, the following result justifies the name given to \cong : $\mu \Downarrow$ means that μ generates a finite number of observable axioms.

Proposition 4. *For all $\mu, \mu \Downarrow$ iff $\mathfrak{E}(\mu)$ is finite.*

Proof. Suppose $\mu \Downarrow$; by definition $\mu \rightarrow_{\beta\varepsilon}^* \nu$ with ν $\beta\varepsilon$ -normal. Now, it is easy to verify that $\beta\varepsilon$ -normal nets are all cut-free, hence the result follows immediately from Lemma 11. For the converse, $\mathfrak{E}(\mu)$ finite means that all observable axioms are generated in a finite number of steps, i.e., $\mu \rightarrow_{\beta}^* C[\mu_0]$, with μ_0 unsolvable by Proposition 2; but then μ is $\beta\varepsilon$ -normalizable to $C[\mathbf{E}_k]$. □

4 Approximations and Full Abstraction

Definition 10 (Approximation). *An approximation of a net μ is a cut-free net ν of the form $C[\mathbf{E}_k]$ such that $\mu \rightarrow_{\beta}^* C[\mu_0]$ for some net μ_0 . In that case, we write $\nu \sqsubseteq \mu$, and we denote by $\text{apx}(\mu)$ the set of all approximations of μ .*

Note that $\text{apx}(\mu)$ is never empty; in fact, if μ has n free ports, we always have $\mathbf{E}_n \sqsubseteq \mu$. Moreover, it is not hard to see that $\text{apx}(\mu)$ is a directed set w.r.t. \sqsubseteq . It is then natural to look for interaction sets X in which the interpretation enjoys the following *approximation property*, inspired by algebraicity in domain theory: for every net μ , we ask

$$\llbracket \mu \rrbracket = \bigcup_{\nu \sqsubseteq \mu} \llbracket \nu \rrbracket,$$

that is, the interpretation of μ in X is the supremum of the interpretations of its approximations.

The congruence induced on nets by an interaction set X , which we denote by $=_X$, may be “located” quite precisely in case X satisfies the approximation property (we mention this result without proof, as we shall not need it in the sequel). Indeed, if we let \simeq denote the Morris equivalence induced by unsolvable nets (i.e., $\mu \simeq \mu'$ iff, for every $C, C[\mu], C[\mu']$ are either both solvable, or both unsolvable), which is the greatest semi-sensible theory, we have:

Proposition 5. *Let X be an interaction set enjoying the approximation property. Then, $\cong \subseteq =_X \subseteq \simeq$.*

Observe that the approximation property is trivially enjoyed by edifices (it is an immediate consequence of the definition). The key to full abstraction will be another approximation property, this time connected to the way the set-based semantics “sees” observable axioms.

If X is an interaction set, recall the “projections” $\delta_1, \delta_2, \zeta_1, \zeta_2 : X \rightarrow X$ introduced in Definition 4. If $w = i_1 \cdots i_n \in \mathbb{W}$, with $n > 0$, and if $\alpha \in \{\delta, \zeta\}$ and $x \in X$, in the sequel we shall use the notation $\alpha_w(x) = \alpha_{i_1}(\dots \alpha_{i_n}(x)\dots)$.

Definition 11 (Semantic axioms). *Let X be an interaction set; we denote the diagonal of X^2 by Δ_X . Let $z \in X^n$ with $n > 0$; we denote by z_j the j -th component of z . Given the arch $\mathbf{a} = (a, b)@i \frown (a', b')@i'$, with $1 \leq i, i' \leq n$, we define the projection of z onto \mathbf{a} as*

$$\pi_{\mathbf{a}}(z) = (\delta_a(\zeta_b(z_i)), \delta_{a'}(\zeta_{b'}(z_{i'}))).$$

If $U \subseteq X^n$, we denote by $\pi_{\mathbf{a}}(U)$ the set resulting from the pointwise application of $\pi_{\mathbf{a}}$ to the elements of U . Then, given a net μ , we define

$$\text{sax}(\mu) = \{\mathbf{a} \mid \pi_{\mathbf{a}}(\llbracket \mu \rrbracket) = \Delta_X\}.$$

We say that an interaction set enjoys the *axiom approximation property* if, for every net μ ,

$$\text{sax}(\mu) = \bigcup_{\nu \sqsubseteq \mu} \text{sax}(\nu).$$

Approximation and axiom approximation do not coincide; we know of interaction sets satisfying the first but not the second, and we believe the converse implication to be false as well. However, we shall now establish that an interaction set satisfying both is fully abstract with respect to finitary axiom-equivalence.

Lemma 3. *Let ν be a cut-free net with 2 free ports such that $\llbracket \nu \rrbracket = \Delta_X$, where X is a non-trivial interaction set. Then, ν is η -equivalent to a wire.*

Proof. Observe that any cut-free net $\nu' \simeq_{\eta} \nu$ decomposes, by Lemma 11, into two trees τ_1, τ_2 and a wiring ω . Now, suppose none of these nets (including ν) is such that $\tau_1 = \tau_2 = \tau$ with ω connecting exactly the matching occurrences of the leaves of the two copies of τ (i -th leaf with i -th leaf). Then, since X is non-trivial, we may easily find an experiment showing that $(x, x') \in \llbracket \nu \rrbracket$ for some $x \neq x'$, a contradiction. Hence, there is an η -equivalent form of ν which is as above; now, if τ contained any ε cell, it is easy to see that we would have $\llbracket \nu \rrbracket \subsetneq \Delta_X$, another contradiction. But a net as the one obtained may be shown to be η -equivalent to a wire by a straightforward induction on τ . \square

Lemma 4. *In a non-trivial interaction set satisfying the axiom approximation property, we have $\text{sax}(\mu) = \mathfrak{E}_{\eta}(\mu)$, for every net μ .*

Proof. The inclusion $\mathfrak{E}_{\eta}(\mu) \subseteq \text{sax}(\mu)$ always holds, with no need of the axiom approximation property. In fact, if $\mathbf{a} \in \mathfrak{E}_{\eta}(\mu)$ by definition $\mathbf{a} = (ac, bd)@i \frown (a'c, b'd)@i'$ for some $c, d \in \mathbb{W}$ and with $(a, b)@i \frown (a', b')@i'$ being the arch

of an observable axiom ω of a reduct μ' of μ . Then, it is enough to consider experiments on μ' which label every port with $\mathbf{0}$ except those “descending” from ω to see that $\mathbf{a} \in \text{sax}(\mu)$, by invariance under reduction.

For what concerns the converse, we first consider a cut-free net ν , with n free ports. Let $\mathbf{a} = (a, b)@i \frown (a', b')@i' \in \text{sax}(\nu)$. It can be shown [13] that, for every sequence of trees τ_1, \dots, τ_n , there exists a cut-free $\nu' \simeq_{\beta\eta} \nu$ such that, for all $1 \leq i \leq n$, we find τ_i rooted at the free port i of ν' . We may then choose $\tau_i, \tau_{i'}$ (the case $i = i'$ changes nothing to the argument) so that two of their resp. leaves l, l' have δ -addresses and ζ -addresses given by a, a' , and b, b' . Now, by invariance under $\simeq_{\beta\eta}$, we still have $\mathbf{a} \in \text{sax}(\nu')$, which means that every experiment on ν' is forced to assign the same element of X to l, l' , and all elements of X may be assigned to them. But ν' must therefore contain a subnet ν_0 whose free ports coincide with l, l' , and which is detached from the rest of the net, for otherwise, by the non-triviality of X , it would be easy to define an experiment which sets one of l, l' to $\mathbf{0}$ and the other to a non-zero value. Then, we apply Lemma 3 to ν_0 and infer $\mathbf{a} \in \mathfrak{E}_\eta(\nu') = \mathfrak{E}_\eta(\nu)$. The general statement follows immediately from the axiom approximation property, and from the approximation property for edifices. \square

Lemma 5. *In a non-trivial interaction set satisfying both the approximation and axiom approximation property, we have $\llbracket \mu \rrbracket = \llbracket \mu' \rrbracket$ iff $\text{sax}(\mu) = \text{sax}(\mu')$, for all nets μ, μ' .*

Proof. The implication from left to right is obvious and does not depend on any approximation property. It is then enough to show that $\text{sax}(\mu) \subseteq \text{sax}(\mu')$ implies $\llbracket \mu \rrbracket \subseteq \llbracket \mu' \rrbracket$. By Lemma 4 and the approximation property, this amounts to show that, under the hypothesis $\mathfrak{E}_\eta(\mu) \subseteq \mathfrak{E}_\eta(\mu')$, $\llbracket \nu \rrbracket \subseteq \llbracket \mu' \rrbracket$ for all $\nu \sqsubseteq \mu$. So let $\nu \sqsubseteq \mu$ and $z \in \llbracket \nu \rrbracket$. By Lemma 2, we may suppose $z \neq (\mathbf{0}, \dots, \mathbf{0})$. Then, z is the result of an experiment assigning non-null points to $p > 0$ axioms of ν . These axioms induce p edifices $\mathfrak{A}_1, \dots, \mathfrak{A}_p \subseteq \mathfrak{E}_\eta(\mu) \subseteq \mathfrak{E}_\eta(\mu')$, which means that the “same” axioms (modulo \simeq_η) appear during the reduction of μ' , from which we immediately infer $z \in \llbracket \mu' \rrbracket$, as desired. \square

As announced, combining Proposition 3, Lemma 4, and Lemma 5, we obtain

Theorem 1 (Full abstraction for interaction sets). *If X is a non-trivial interaction set satisfying the approximation and axiom approximation properties, then for all nets $\mu, \mu', \mu \cong \mu'$ iff $\llbracket \mu \rrbracket = \llbracket \mu' \rrbracket$.*

4.1 A Fully Abstract Model

We now proceed to give an example of interaction set satisfying both the approximation and axiom approximation properties.

Let \mathbb{W}^∞ be the set of infinite words over $\{1, 2\}$. We shall consider the pointed set $X = \mathcal{P}_{\text{fin}}(\mathbb{W}^\infty \times \mathbb{W}^\infty)$ of finite subsets of pairs of infinite words, with $\mathbf{0} = \emptyset$. If $x \in X$ and $(a, b) \in \mathbb{W} \times \mathbb{W}$, we set $(a, b) \cdot x = \{(au, bv) \in \mathbb{W}^\infty \times \mathbb{W}^\infty \mid (u, v) \in x\}$, and define the pointed bijections $\langle x, y \rangle = (1, \epsilon) \cdot x \cup (2, \epsilon) \cdot y$ and $[x, y] = (\epsilon, 1) \cdot x \cup (\epsilon, 2) \cdot y$. It is easy to see that $(X, \langle \cdot, \cdot \rangle, [\cdot, \cdot])$ is an interaction set.

In the sequel, all denotational interpretations are assumed to be in X . We shall keep denoting by $\delta_1, \delta_2, \zeta_1, \zeta_2$ the “projections” associated with $\langle \cdot, \cdot \rangle$ and $[\cdot, \cdot]$, and we also use the generalized notations δ_w, ζ_w , with $w \in \mathbb{W}$, introduced just before Definition [11](#).

Definition 12 (Mean and nice elements). *We say that $x \in X$ is δ -mean (resp. ζ -mean), if the set $\pi_1(x)$ (resp. $\pi_2(x)$), i.e., the set of first (resp. second) projections of the elements of x , is empty or a singleton. If $\alpha \in \{\delta, \zeta\}$, and if $\bar{\alpha}$ is “the other” binary symbol, we introduce the following notations and terminology:*

- M_α is the set of α -mean elements;
- $M = M_\delta \cap M_\zeta$ is the set of mean elements (which are singletons or empty);
- $N = X \setminus (M_\delta \cup M_\zeta)$ is the set of nice elements
- $M_\alpha^* = M_\alpha \setminus M_{\bar{\alpha}}$ is the set of strictly α -mean elements.

In the sequel, if $x \in X$, we shall denote by $\|x\|$ its cardinality. Moreover, we will always have $\alpha \in \{\delta, \zeta\}$, with $\bar{\alpha}$ denoting “the other” binary symbol.

Lemma 6. *For all $x \in X$, the following properties hold:*

Positivity: $\|\mathbf{0}\| = 0$, and $\|x\| \neq 0$ for all $x \neq \mathbf{0}$.

Additivity: $\|\delta_1(x)\| + \|\delta_2(x)\| = \|x\| = \|\zeta_1(x)\| + \|\zeta_2(x)\|$.

Heredity: For all $\beta \in \{\delta_1, \delta_2, \zeta_1, \zeta_2\}$, $x \in M_\alpha$ implies $\beta(x) \in M_\alpha$.

Proof. A simple verification. □

In the following, we write $e : \mu$ to mean “ e is an experiment on μ ”.

Lemma 7. *Let τ be a tree entirely composed of α cells (resp. an arbitrary tree). If $e : \tau$ assigns $x \in M_\alpha$ (resp. $x \in M$) to the root of τ , then e assigns $x' \in M_\alpha$ (resp. $x' \in M$) with $\|x'\| = \|x\|$ to exactly one leaf of τ , and $\mathbf{0}$ to all remaining leaves.*

Proof. By heredity, additivity, and the fact that, if $x \in M_\alpha$ and $i \in \{1, 2\}$, then $\alpha_i(x)$ is either empty, or has the same cardinality as x . □

We say that an experiment assigns an element x to an active pair when it assigns x to both extremities of the cut associated with the active pair.

Lemma 8. *Let μ be a net with $n > 0$ free ports. If $e : \mu$ is such that the labels assigned by e to the active pairs of μ all belong to M then there exists $\nu \sqsubseteq \mu$ and $e_0 : \nu$ such that $|e_0| = |e|$.*

Proof. The idea of the proof is to take each active pair labelled by some $x \in M$, and replace the active pair with a net admitting an experiment of identical result. The case $x = \mathbf{0}$ is easy; otherwise, Lemma [7](#) is used to extract only two leaves of subtrees of μ on which e assigns non-zero points; then, the net formed by the two trees rooted at the cut is wholly reduced, and the zero-labelled subnets in the result are carefully removed. We omit the technical details. □

Proposition 6. *X enjoys the approximation property.*

Proof. The right-to-left inclusion holds for all interaction sets, as a corollary of the fact that $\nu \sqsubseteq \mu$ implies $\llbracket \nu \rrbracket \subseteq \llbracket \mu \rrbracket$. Indeed, by definition, $\nu = C[\mathbf{E}_n]$ and $\mu \rightarrow_{\beta}^* C[\mu_0]$. Then, by Lemma 2 and the congruence property, $\llbracket \nu \rrbracket \subseteq \llbracket C[\mu_0] \rrbracket$, so we conclude by invariance.

For the converse, we give the idea of the proof, which is fairly straightforward. We start by defining two functions $\#_{\delta}, \#_{\zeta} : X \rightarrow \mathbb{N}$. Let $x \in X$, and let n_{δ} (resp. n_{ζ}) be the length of the longest common prefix between two different words in $\pi_1(x)$ (resp. $\pi_2(x)$); then, we define $\#_{\alpha}(x)$ to be 0 if $x = \mathbf{0}$, 1 if $x \in M_{\alpha} \setminus \{\mathbf{0}\}$, or $n_{\alpha} + 2$ otherwise. We define the *measure* of x as the integer $m(x) = \|x\| \cdot \#_{\delta}(x) \cdot \#_{\zeta}(x)$. Now, by a simple case inspection, one checks that:

- (i) if $x \in N$, then for all $\beta \in \{\delta_1, \delta_2, \zeta_1, \zeta_2\}$, $m(\beta(x)) < m(x)$;
- (ii) if $x \in M_{\alpha}^*$, then for all $i \in \{1, 2\}$, $m(\bar{\alpha}_i(x)) < m(x)$.

We now show in three steps that by obstinately reducing the active pairs of μ labelled by elements belonging to N first, M_{δ}^* then, and M_{ζ}^* finally, we obtain a net μ' and $e' : \mu'$ such that $\mu \rightarrow_{\beta}^* \mu'$, $|e'| = |e|$, and all the labels assigned to the active pairs of μ' by e' belong to M . The result then follows by Lemma 8.

1. The first step is shown by induction on the multiset $H(\mu)$ (under the usual well-ordering of multisets of integers) containing the measures of all nice elements labelling the active pairs of μ in e , using property (i). At the end of this step we obtain a reduct μ_1 of μ and an experiment $e_1 : \mu_1$ such that $|e_1| = |e|$, and no active pair is labelled by a nice element.
2. The second step is shown by induction on the multiset $J(\mu_1)$ containing the measures of all strictly δ -mean elements labelling the active pairs of μ_1 . Here, we use Lemma 6, Lemma 7 and property (ii). At the end of this step we obtain a reduct μ_2 of μ_1 and an experiment $e_2 : \mu_2$ such that $|e_2| = |e_1|$, and no active pair is labelled by a strictly δ -mean element.
3. The third step is similar to the second.

□

The fact that X enjoys the axiom approximation property is a consequence of the following

Lemma 9. *Any net μ such that $\llbracket \mu \rrbracket = \Delta_X$ is $\beta\eta$ -equivalent to a wire.*

Proof. It is enough to show that $\text{apx}(\mu)$ contains a net η -equivalent to a wire. Suppose, for the sake of contradiction, that this is not the case. If μ is $\beta\varepsilon$ -normalizable (necessarily to a cut-free form), we may immediately conclude by Lemma 3; therefore, we suppose that μ is not $\beta\varepsilon$ -normalizable. By Proposition 4, $\text{apx}(\mu)$ is infinite; since it is directed w.r.t. \sqsubseteq , we may take an ascending chain $\nu_0 \sqsubseteq \nu_1 \sqsubseteq \dots$ of approximations of μ . By the same arguments given in the proof of Lemma 3, every ν_k decomposes into two copies of a tree τ_k , whose matching occurrences of leaves are joined by axioms forming a wiring ω_k . Now, by hypothesis, every ν_k is not η -equivalent to a wire, which implies that every ν_k

contains at least one ε cell. Then, we may build a sequence of finite pairs of words (a_k, b_k) describing a leaf of τ_k where we find an ε cell, such that (a_k, b_k) is a prefix (not necessarily strict) of (a_{k+1}, b_{k+1}) . This increasing sequence defines some $(u, v) \in \mathbb{W}^\infty \times \mathbb{W}^\infty$ such that, by construction, $z = (\{(u, v)\}, \{(u, v)\}) \notin \llbracket \nu_k \rrbracket$ for all $k \in \mathbb{N}$. Then, by the approximation property, $z \notin \llbracket \mu \rrbracket$, in contradiction with $\Delta_X \subseteq \llbracket \mu \rrbracket$. \square

Proposition 7. *X enjoys the axiom approximation property.*

Proof. Let μ be a net, and suppose $\mathbf{a} \in \text{sax}(\mu)$. By the approximation property, using the same argument given in the second part of the proof of Lemma 4, we have that the arch \mathbf{a} defines two leaves l, l' of two trees rooted at two conclusions of a β -reduct μ' of μ , such that l, l' are the free ports of a subnet μ'_0 of μ' whose interpretation is exactly Δ_X . Then, if we apply Lemma 9, reduce μ'_0 , and replace everything else in μ' by ε cells, we obtain an approximation ν of μ such that $\mathbf{a} \in \text{sax}(\nu)$. \square

Acknowledgments. This work was partially supported by ANR project LOGOI.

References

1. Barendregt, H.P.: The Lambda Calculus, revised edn. North Holland (1984)
2. Coppo, M., Dezani-Ciancaglini, M., Zacchi, M.: Type theories, normal forms, and D_∞ -lambda-models. *Information and Computation* 72(2), 85–116 (1987)
3. Dezani-Ciancaglini, M., Giovannetti, E.: From Bohm’s theorem to observational equivalences: an informal account. *Electronic Notes in Theoretical Computer Science* 50(2), 85–118 (2001)
4. Fernández, M., Mackie, I.: Operational equivalence for interaction nets. *Theoretical Computer Science* 297(1-3), 157–181 (2003)
5. Girard, J.Y.: Linear logic. *Theoretical Computer Science* 50(1), 1–102 (1987)
6. Hyland, M.: A syntactic characterization of the equality in some models of the lambda calculus. *Journal of the London Mathematical Society* 2(12), 361–370 (1976)
7. Lafont, Y.: Interaction nets. In: *Conference Record of POPL 1990*, pp. 95–108. ACM Press (1990)
8. Lafont, Y.: Interaction combinators. *Information and Computation* 137(1), 69–101 (1997)
9. Lamping, J.: An algorithm for optimal lambda calculus reduction. In: *Conference Record of POPL 1990*, pp. 16–30. ACM Press (1990)
10. Mackie, I.: Efficient lambda Evaluation with Interaction Nets. In: van Oostrom, V. (ed.) *RTA 2004. LNCS*, vol. 3091, pp. 155–169. Springer, Heidelberg (2004)
11. Mackie, I., Pinto, J.S.: Encoding linear logic with interaction combinators. *Information and Computation* 176(2), 153–186 (2002)
12. Mazza, D.: A denotational semantics for the symmetric interaction combinators. *Mathematical Structures in Computer Science* 17(3), 527–562 (2007)
13. Mazza, D.: Observational equivalence and full abstraction in the symmetric interaction combinators. *Logical Methods in Computer Science* 5(4:6) (2009)
14. Morris, J.H.: Lambda calculus models of programming languages. Ph.D. Thesis, Massachusetts Institute of Technology (1968)

On Distributability of Petri Nets

(Extended Abstract)*

Rob van Glabbeek^{1,2}, Ursula Goltz³ and Jens-Wolfhard Schicke-Uffmann³

¹ NICTA, Sydney, Australia

² School of Computer Sc. and Engineering, University of New South Wales, Sydney, Australia

³ Institute for Programming and Reactive Systems, TU Braunschweig, Germany

rvg@cs.stanford.edu

goltz@ips.cs.tu-bs.de, drahflow@gmx.de

Abstract. We formalise a general concept of distributed systems as sequential components interacting asynchronously. We define a corresponding class of Petri nets, called LSGA nets, and precisely characterise those system specifications which can be implemented as LSGA nets up to branching ST-bisimilarity with explicit divergence.

1 Introduction

The aim of this paper is to contribute to a fundamental understanding of the concept of a distributed reactive system and the paradigms of synchronous and asynchronous interaction. We start by giving an intuitive characterisation of the basic features of distributed systems. In particular we assume that distributed systems consist of components that reside on different locations, and that any signal from one component to another takes time to travel. Hence the only interaction mechanism between components is asynchronous communication.

Our aim is to characterise which system specifications may be implemented as distributed systems. In many formalisms for system specification or design, synchronous communication is provided as a basic notion; this happens for example in process algebras. Hence a particular challenge is that it may be necessary to simulate synchronous communication by asynchronous communication.

Trivially, any system specification may be implemented distributedly by locating the whole system on one single component. Hence we need to pose some additional requirements. One option would be to specify locations for system activities and then to ask for implementations satisfying this distribution and still preserving the behaviour of the original specification. This is done in [1]. Here we pursue a different approach. We add another requirement to our notion of a distributed system, namely that its components only allow sequential behaviour. We then ask whether an arbitrary system specification may be implemented as a distributed system consisting of sequential components in an optimal way, that is without restricting the concurrency of the original specification. This is a particular challenge when synchronous communication interacts with concurrency in the specification of the original system. We will give a precise characterisation of the class of distributable systems, which answers in particular under which conditions synchronous communication may be implemented in a distributed setting.

* This work was partially supported by the DFG (German Research Foundation).

For our investigations we need a model which is expressive enough to represent concurrency. It is also useful to have an explicit representation of the distributed state space of a distributed system, showing in particular the local control states of components. We choose Petri nets, which offer these possibilities and additionally allow finite representations of infinite behaviours. We work within the class of *structural conflict nets* [4] —a proper generalisation of the class of one-safe place/transition systems, where conflict and concurrency are clearly separated.

For comparing the behaviour of systems with their distributed implementation we need a suitable equivalence notion. Since we think of open systems interacting with an environment, and since we do not want to restrict concurrency in applications, we need an equivalence that respects branching time and concurrency to some degree. Our implementations use transitions which are invisible to the environment, and this should be reflected in the equivalence by abstracting from such transitions. However, we do not want implementations to introduce divergence. In the light of these requirements we work with two semantic equivalences. *Step readiness equivalence* is one of the weakest equivalences that captures branching time, concurrency and divergence to some degree; whereas *branching ST-bisimilarity with explicit divergence* fully captures branching time, divergence, and those aspects of concurrency that can be represented by concurrent actions overlapping in time. We obtain the same characterisation for both notions of equivalence, and thus implicitly for all notions in between these extremes.

We model distributed systems consisting of sequential components as an appropriate class of Petri nets, called *LSGA nets*. These are obtained by composing nets with sequential behaviour by means of an asynchronous parallel composition. We show that this class corresponds exactly to a more abstract notion of distributed systems, formalised as *distributed nets* [5].

We then consider distributability of system specifications which are represented as structural conflict nets. A net N is *distributable* if there exists a distributed implementation of N , that is a distributed net which is semantically equivalent to N . In the implementation we allow unobservable transitions, and labellings of transitions, so that single actions of the original system may be implemented by multiple transitions. However, the system specifications for which we search distributed implementations are *plain* nets without these features.

We give a precise characterisation of distributable nets in terms of a semi-structural property. This characterisation provides a formal proof that the interplay between choice and synchronous communication is a key issue for distributability.

2 Basic Notions

We consider here general labelled place/transition nets with arc weights. Arc weights are not necessary for the results of the paper, but are included for the sake of generality.

We will employ the following notations for multisets.

Definition 1. Let X be a set.

- A *multiset* over X is a function $A: X \rightarrow \mathbb{N}$, i.e. $A \in \mathbb{N}^X$.
- $x \in X$ is an *element* of a multiset $A \in \mathbb{N}^X$, notation $x \in A$, iff $A(x) > 0$.
- For multisets A and B over X we write $A \leq B$ iff $A(x) \leq B(x)$ for all $x \in X$;

$A + B$ denotes the multiset over X with $(A + B)(x) := A(x) + B(x)$,
 $A \setminus B$ denotes the multiset over X with $(A - B)(x) := \max(A(x) - B(x), 0)$, and
for $k \in \mathbb{N}$ the multiset $k \cdot A$ is given by $(k \cdot A)(x) := k \cdot A(x)$.

- The function $\emptyset: X \rightarrow \mathbb{N}$, given by $\emptyset(x) := 0$ for all $x \in X$, is the *empty* multiset.
- If A is a multiset over X and $Y \subseteq X$ then $A \upharpoonright Y$ denotes the multiset over Y defined by $(A \upharpoonright Y)(x) := A(x)$ for all $x \in Y$.
- The cardinality $|A|$ of a multiset A over X is given by $|A| := \sum_{x \in X} A(x)$.
- A multiset A over X is *finite* iff $\{x \mid x \in A\}$ is finite, i.e., iff $|A| < \infty$.

Two multisets $A: X \rightarrow \mathbb{N}$ and $B: Y \rightarrow \mathbb{N}$ are *extensionally equivalent* iff $A \upharpoonright (X \setminus Y) = \emptyset$, $B \upharpoonright (Y \setminus X) = \emptyset$, and $A \upharpoonright (X \cap Y) = B \upharpoonright (X \cap Y)$. In this paper we often do not distinguish extensionally equivalent multisets. This enables us, for instance, to use $A + B$ even when A and B have different underlying domains.

A multiset A with $A(x) \in \{0, 1\}$ for all x is identified with the set $\{x \mid A(x) = 1\}$.

Definition 2. Let Act be a set of *visible actions* and $\tau \notin \text{Act}$ be an *invisible action*.

A (*labelled*) *Petri net* (over $\text{Act} \dot{\cup} \{\tau\}$) is a tuple $N = (S, T, F, M_0, \ell)$ where

- S and T are disjoint sets (of *places* and *transitions*),
- $F: (S \times T \cup T \times S) \rightarrow \mathbb{N}$ (the *flow relation* including *arc weights*),
- $M_0: S \rightarrow \mathbb{N}$ (the *initial marking*), and
- $\ell: T \rightarrow \text{Act} \dot{\cup} \{\tau\}$ (the *labelling function*).

Petri nets are depicted by drawing the places as circles and the transitions as boxes, containing their label. Identities of places and transitions are displayed next to the net element. When $F(x, y) > 0$ for $x, y \in S \cup T$ there is an arrow (*arc*) from x to y , labelled with the *arc weight* $F(x, y)$. Weights 1 are elided. When a Petri net represents a concurrent system, a global state of this system is given as a *marking*, a multiset M of places, depicted by placing $M(s)$ dots (*tokens*) in each place s . The initial state is M_0 .

To compress the graphical notation, we also allow universal quantifiers of the form $\forall x. \phi(x)$ to appear in the drawing (cf. Fig. 3). A quantifier replaces occurrences of x in element identities with all concrete values for which $\phi(x)$ holds, possibly creating a set of elements instead of the depicted single one. An arc of which only one end is replicated by a given quantifier results in a fan of arcs, one for each replicated element. If both ends of an arc are affected by the same quantifier, an arc is created between pairs of elements corresponding to the same x , but not between elements created due to differing values of x .

The behaviour of a Petri net is defined by the possible moves between markings M and M' , which take place when a finite multiset G of transitions *fires*. In that case, each occurrence of a transition t in G consumes $F(s, t)$ tokens from each place s . Naturally, this can happen only if M makes all these tokens available in the first place. Next, each t produces $F(t, s)$ tokens in each s . Definition 4 formalises this notion of behaviour.

Definition 3. Let $N = (S, T, F, M_0, \ell)$ be a Petri net and $x \in S \cup T$.

The multisets $\bullet x, x^\bullet: S \cup T \rightarrow \mathbb{N}$ are given by $\bullet x(y) = F(y, x)$ and $x^\bullet(y) = F(x, y)$ for all $y \in S \cup T$. If $x \in T$, the elements of $\bullet x$ and x^\bullet are called *pre-* and *postplaces* of x , respectively. These functions extend to multisets $X: S \cup T \rightarrow \mathbb{N}$ as usual, by $\bullet X := \sum_{x \in S \cup T} X(x) \cdot \bullet x$ and $X^\bullet := \sum_{x \in S \cup T} X(x) \cdot x^\bullet$.

Definition 4. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $G \in \mathbb{N}^T$, G non-empty and finite, and $M, M' \in \mathbb{N}^S$. G is a *step* from M to M' , written $M [G]_N M'$, iff $\bullet G \subseteq M$ (G is *enabled*) and $M' = (M \setminus \bullet G) + G^\bullet$.

Note that steps are (finite) multisets, thus allowing self-concurrency, i.e. the same transition can occur multiple times in a single step.

In our nets transitions are labelled with *actions* drawn from a set $\text{Act} \dot{\cup} \{\tau\}$. A transition t can be thought of as the occurrence of the action $\ell(t)$. If $\ell(t) \in \text{Act}$, this occurrence can be observed and influenced by the environment, but if $\ell(t) = \tau$, it cannot and t is an *internal* or *silent* transition. Transitions whose occurrences cannot be distinguished by the environment carry the same label. In particular, since the environment cannot observe the occurrence of internal transitions at all, they are all labelled τ .

To simplify statements about behaviours of nets, we use some abbreviations.

Definition 5. Let $N = (S, T, F, M_0, \ell)$ be a Petri net.

We write $M_1 \xrightarrow{\alpha}_N M_2$, for $\alpha \in \text{Act} \dot{\cup} \{\tau\}$, when $\exists t \in T. \alpha = \ell(t) \wedge M_1 [\{t\}]_N M_2$. Furthermore, for $a_1 a_2 \cdots a_n \in \text{Act}^*$ we write $M_1 \xrightarrow{a_1 a_2 \cdots a_n}_N M_2$ when

$$M_1 \Longrightarrow_N \xrightarrow{a_1}_N \Longrightarrow_N \xrightarrow{a_2}_N \Longrightarrow_N \cdots \Longrightarrow_N \xrightarrow{a_n}_N \Longrightarrow_N M_2$$

where \Longrightarrow_N denotes the reflexive and transitive closure of $\xrightarrow{\tau}_N$.

For $\alpha \in \text{Act} \dot{\cup} \{\tau\}$, we write $M_1 \xrightarrow{(\alpha)}_N M_2$ for $M_1 \xrightarrow{\alpha}_N M_2 \vee (\alpha = \tau \wedge M_1 = M_2)$, meaning that in case $\alpha = \tau$ performing a τ -transition is optional. We write $M_1 \xrightarrow{\alpha}_N$ for $\exists M_2. M_1 \xrightarrow{\alpha}_N M_2$, and $M_1 \not\xrightarrow{\alpha}_N$ for $\nexists M_2. M_1 \xrightarrow{\alpha}_N M_2$. Likewise $M_1 [G]_N$ abbreviates $\exists M_2. M_1 [G]_N M_2$. We omit the subscript N if clear from context.

Definition 6. Let $N = (S, T, F, M_0, \ell)$ be a Petri net.

- A marking $M \in \mathbb{N}^S$ is said to be *reachable in N* iff there is a $\sigma \in \text{Act}^*$ such that $M_0 \xrightarrow{\sigma}_N M$. The set of all *reachable markings of N* is denoted by $[M_0]_N$.
- N is *one-safe* iff $M \in [M_0]_N \Rightarrow \forall x \in S. M(x) \leq 1$.
- The *concurrency relation* $\smile \subseteq T^2$ is given by $t \smile u \Leftrightarrow \exists M \in [M_0]. M[\{t, u\}]$.
- N is a *structural conflict net* iff for all $t, u \in T$ with $t \smile u$ we have $\bullet t \cap \bullet u = \emptyset$.

We use the term *plain nets* for Petri nets where ℓ is injective and no transition has the label τ , i.e. essentially unlabelled nets.

This paper first of all aims at studying finite Petri nets: nets with finitely many places and transitions. However, our work also applies to infinite nets with the properties that $\bullet t \neq \emptyset$ for all transitions $t \in T$, and any reachable marking (a) is finite, and (b) enables only finitely many transitions. Henceforth, we call such nets *finitary*. Finitariness can be ensured by requiring $|M_0| < \infty \wedge \forall t \in T. \bullet t \neq \emptyset \wedge \forall x \in S \cup T. |x^\bullet| < \infty$.

We use the following variant of readiness semantics [11] to compare behaviour.

Definition 7. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $\sigma \in \text{Act}^*$ and $X \subseteq \mathbb{N}^{\text{Act}}$.

$\langle \sigma, X \rangle$ is a *step ready pair* of N iff

$$\exists M. M_0 \xrightarrow{\sigma} M \wedge M \not\xrightarrow{\tau} \wedge X = \{\ell(G) \mid M[G]\}.$$

Here we extend the labelling function ℓ to finite multisets of transitions elementwise.

We write $\mathcal{R}(N)$ for the set of all step ready pairs of N .

Two Petri nets N_1 and N_2 are *step readiness equivalent*, $N_1 \approx_{\mathcal{R}} N_2$, iff $\mathcal{R}(N_1) = \mathcal{R}(N_2)$.

ST-bisimilarity was proposed in [7] as a non-interleaved version of bisimilarity that respects causality to the extent that it can be expressed in terms of the possibility of durational actions to overlap in time. It was extended to a setting with internal actions in [15], based on the notion of *weak bisimilarity* of [10]. Here we apply the same idea, but based on *branching bisimilarity* [8], which unlike weak bisimilarity fully respects the branching structure of related systems.

An *ST-marking* of a net $N = (S, T, F, M_0, \ell)$ is a pair $(M, U) \in \mathbb{N}^S \times T^*$ of a normal marking, together with a sequence of transitions *currently firing*. The *initial ST-marking* is $\mathfrak{M}_0 := (M_0, \varepsilon)$. The elements of $\text{Act}^\pm := \{a^+, a^{-n} \mid a \in \text{Act}, n > 0\}$ are called *visible action phases*, and $\text{Act}_\tau^\pm := \text{Act}^\pm \dot{\cup} \{\tau\}$. For $U \in T^*$, we write $t \in^{(n)} U$ if t is the n^{th} element of U . Furthermore U^{-n} denotes U after removal of the n^{th} transition.

Definition 8. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, labelled over $\text{Act} \dot{\cup} \{\tau\}$.

The *ST-transition relations* $\xrightarrow{\eta}$ for $\eta \in \text{Act}_\tau^\pm$ between ST-markings are given by

$$\begin{aligned} (M, U) &\xrightarrow{a^+} (M', U') \text{ iff } \exists t \in T. \ell(t) = a \wedge M[\{t\}] \wedge M' = M - \bullet t \wedge U' = Ut. \\ (M, U) &\xrightarrow{a^{-n}} (M', U') \text{ iff } \exists t \in^{(n)} U. \ell(t) = a \wedge U' = U^{-n} \wedge M' = M + t^\bullet. \\ (M, U) &\xrightarrow{\tau} (M', U') \text{ iff } M \xrightarrow{\tau} M' \wedge U' = U. \end{aligned}$$

Now *branching ST-bisimilarity* is *branching bisimilarity* [8], applied to the labelled transition system made up of ST-markings of nets and the ST-transitions between them.

Definition 9. Two Petri nets N_1 and N_2 are *branching ST-bisimilar* iff there exists a relation \mathcal{R} between the ST-markings of N_1 and N_2 such that, for all $\eta \in \text{Act}_\tau^\pm$:

1. $\mathfrak{M}_{01} \mathcal{R} \mathfrak{M}_{02}$;
2. if $\mathfrak{M}_1 \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}_1 \xrightarrow{\eta} \mathfrak{M}'_1$ then $\exists \mathfrak{M}_2^\dagger, \mathfrak{M}'_2$ such that $\mathfrak{M}_2 \Longrightarrow \mathfrak{M}_2^\dagger \xrightarrow{\eta} \mathfrak{M}'_2$, $\mathfrak{M}_1 \mathcal{R} \mathfrak{M}_2^\dagger$ and $\mathfrak{M}'_1 \mathcal{R} \mathfrak{M}'_2$;
3. if $\mathfrak{M}_1 \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}_2 \xrightarrow{\eta} \mathfrak{M}'_2$ then $\exists \mathfrak{M}_1^\dagger, \mathfrak{M}'_1$ such that $\mathfrak{M}_1 \Longrightarrow \mathfrak{M}_1^\dagger \xrightarrow{\eta} \mathfrak{M}'_1$, $\mathfrak{M}_1^\dagger \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}'_1 \mathcal{R} \mathfrak{M}'_2$.

If a system has the potential to engage in an infinite sequence of internal actions, one speaks of *divergence*. *Branching bisimilarity with explicit divergence* [8], is a variant of branching bisimilarity that fully respects the diverging behaviour of related systems. Since here we only compare systems of which one admits no divergence at all, the definition simplifies to the requirement that the other system may not diverge either. We write $N_1 \approx_{bSTb}^\Delta N_2$ iff N_1 and N_2 are branching ST-bisimilar with explicit divergence.

3 Distributed Systems

In this section, we stipulate what we understand by a distributed system, and subsequently formalise a model of distributed systems in terms of Petri nets.

- A distributed system consists of components residing on different locations.
- Components work concurrently.
- Interactions between components are only possible by explicit communications.
- Communication between components is time consuming and asynchronous.

Asynchronous communication is the only interaction mechanism in a distributed system for exchanging signals or information.

- The sending of a message happens always strictly before its receipt (there is a causal relation between sending and receiving a message).
- A sending component sends without regarding the state of the receiver; in particular there is no need to synchronise with a receiving component. After sending the sender continues its behaviour independently of receipt of the message.

As explained in the introduction, we will add another requirement to our notion of a distributed system, namely that its components only allow sequential behaviour.

Formally, we model distributed systems as nets consisting of component nets with sequential behaviour and interfaces in terms of input and output places.

Definition 10. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $I, O \subseteq S$, $I \cap O = \emptyset$ and $O^\bullet = \emptyset$.

1. (N, I, O) is a *component with interface* (I, O) .
2. (N, I, O) is a *sequential component with interface* (I, O) iff $\exists Q \subseteq S \setminus (I \cup O)$ with $\forall t \in T. | \bullet t \uparrow Q | = 1 \wedge | t^\bullet \uparrow Q | = 1$ and $| M_0 \uparrow Q | = 1$.

An input place $i \in I$ of a component \mathcal{C} can be regarded as a mailbox of \mathcal{C} for a specific type of messages. An output place $o \in O$, on the other hand, is an address outside \mathcal{C} to which \mathcal{C} can send messages. Moving a token into o is like posting a letter. The condition $o^\bullet = \emptyset$ says that a message, once posted, cannot be retrieved by the component.

A set of places like Q above is called an *S-invariant*. The requirements guarantee that the number of tokens in these places remains constant, in this case 1. It follows that no two transitions can ever fire concurrently (in one step). Conversely, whenever a net is sequential, in the sense that no two transitions can fire in one step, it is easily converted into a behaviourally equivalent net with the required *S-invariant*, namely by adding a single marked place with a self-loop to all transitions. This modification preserves virtually all semantic equivalences on Petri nets from the literature, including \approx_{bSTb}^A .

Next we define an operator for combining components with asynchronous communication by fusing input and output places.

Definition 11. Let \mathfrak{K} be an index set.

Let $((S_k, T_k, F_k, M_{0k}, \ell_k), I_k, O_k)$ with $k \in \mathfrak{K}$ be components with interface such that $(S_k \cup T_k) \cap (S_l \cup T_l) = (I_k \cup O_k) \cap (I_l \cup O_l)$ for all $k, l \in \mathfrak{K}$ with $k \neq l$ (components are disjoint except for interface places) and moreover $I_k \cap I_l = \emptyset$ for all $k, l \in \mathfrak{K}$ with $k \neq l$ (mailboxes cannot be shared; the recipient of a message is always unique).

Then the *asynchronous parallel composition* of these components is defined by

$$\bigsqcup_{i \in \mathfrak{K}} ((S_k, T_k, F_k, M_{0k}, \ell_k), I_k, O_k) = ((S, T, F, M_0, \ell), I, O)$$

with $S = \bigcup_{k \in \mathfrak{K}} S_k$, $T = \bigcup_{k \in \mathfrak{K}} T_k$, $F = \bigcup_{k \in \mathfrak{K}} F_k$, $M_0 = \sum_{k \in \mathfrak{K}} M_{0k}$, $\ell = \bigcup_{k \in \mathfrak{K}} \ell_k$ (componentwise union of all nets), $I = \bigcup_{k \in \mathfrak{K}} I_k$ (we accept additional inputs from outside), and $O = \bigcup_{k \in \mathfrak{K}} O_k \setminus \bigcup_{k \in \mathfrak{K}} I_k$ (once fused with an input, $o \in O_I$ is no longer an output).

Observation 1. \bigsqcup is associative.

This follows directly from the associativity of the (multi)set union operator. \square

We are now ready to define the class of nets representing systems of asynchronously communicating sequential components.

Definition 12. A Petri net N is an *LSGA net* (a *locally sequential globally asynchronous net*) iff there exists an index set \mathfrak{K} and sequential components with interface \mathcal{C}_k , $k \in \mathfrak{K}$, such that $(N, I, O) = \bigsqcup_{k \in \mathfrak{K}} \mathcal{C}_k$ for some I and O .

Up to \approx_{bSTb}^A —or any reasonable equivalence preserving causality and branching time but abstracting from internal activity—the same class of LSGA systems would have been obtained if we had imposed, in Def. 10, that I, O and Q form a partition of S and that $\bullet I = \emptyset$. However, it is essential that our definition allows multiple transitions of a component to read from the same input place.

In the remainder of this section we give a more abstract characterisation of Petri nets representing distributed systems, namely as *distributed* Petri nets, which we introduced in [5]. This will be useful in Section 4, where we investigate distributability using this more semantic characterisation. We show below that the concrete characterisation of distributed systems as LSGA nets and this abstract characterisation agree.

Following [1], to arrive at a class of nets representing distributed systems, we associate *localities* to the elements of a net $N = (S, T, F, M_0, \ell)$. We model this by a function $D : S \cup T \rightarrow \text{Loc}$, with Loc a set of possible locations. We refer to such a function as a *distribution* of N . Since the identity of the locations is irrelevant for our purposes, we can just as well abstract from Loc and represent D by the equivalence relation \equiv_D on $S \cup T$ given by $x \equiv_D y$ iff $D(x) = D(y)$.

Following [5], we impose a fundamental restriction on distributions, namely that when two transitions can occur in one step, they cannot be co-located. This reflects our assumption that at a given location actions can only occur sequentially.

In [5] we observed that Petri nets incorporate a notion of synchronous interaction, in that a transition can fire only by synchronously taking the tokens from all of its preplaces. In general the behaviour of a net would change radically if a transition would take its input tokens one by one—in particular deadlocks may be introduced. Therefore we insist that in a distributed Petri net, a transition and all its input places reside on the same location. There is no reason to require the same for the output places of a transition, for the behaviour of a net would not change significantly if transitions were to deposit their output tokens one by one [5].

This leads to the following definition of a distributed Petri net.

Definition 13 ([5]). A Petri net $N = (S, T, F, M_0, \ell)$ is *distributed* iff there exists a distribution D such that

- (1) $\forall s \in S, t \in T. s \in \bullet t \Rightarrow t \equiv_D s,$
- (2) $\forall t, u \in T. t \smile u \Rightarrow t \not\equiv_D u.$

N is *essentially distributed* if (2) is weakened to $\forall t, u \in T. t \smile u \wedge \ell(t) \neq \tau \Rightarrow t \not\equiv_D u.$

A typical example of a net which is not distributed is shown in Fig. 1 on Page 339. Transitions t and v are concurrently executable and hence should be placed on different locations. However, both have preplaces in common with u which would enforce putting all three transitions on the same location. In fact, distributed nets can be characterised in the following semi-structural way.

Observation 2. A Petri net is distributed iff there is no sequence t_0, \dots, t_n of transitions with $t_0 \smile t_n$ and $\bullet t_{i-1} \cap \bullet t_i \neq \emptyset$ for $i = 1, \dots, n.$ □

It turns out that the classes of LSGA nets and distributable nets essentially coincide. Moreover, up to \approx_{bSTb}^A these classes also coincide with the more liberal notion of essentially distributed nets, permitting concurrency of internal transitions at the same location. We will make use of that in proving our main theorem.

Theorem 1. *Any LSGA net is distributed, and for any essentially distributed net N there is an LSGA net N' with $N' \approx_{bSTb}^{\Delta} N$.*

Proof. In the full version of this paper [6]. □

Observation 3. *Every distributed Petri net is a structural conflict net.* □

Corollary 1. *Every LSGA net is a structural conflict net.* □

4 Distributable Systems

We now consider Petri nets as specifications of concurrent systems and ask the question which of those specifications can be implemented as distributed systems. This question can be formalised as

Which Petri nets are semantically equivalent to distributed nets?

Of course the answer depends on the choice of a suitable semantic equivalence. Here we will answer this question using the two equivalences introduced in Section 2. We will give a precise characterisation of those nets for which we can find semantically equivalent distributed nets. For the negative part of this characterisation, stating that certain nets are not distributable, we will use step readiness equivalence, which is one of the simplest and least discriminating equivalences imaginable that abstracts from internal actions, but preserves branching time, concurrency and divergence to some small degree. As explained in [5], giving up on any of these latter three properties would make any Petri net distributable, but in a rather trivial and unsatisfactory way. For the positive part, namely that all other nets are indeed distributable, we will use the most discriminating equivalence for which our implementation works, namely branching ST-bisimilarity with explicit divergence, which is finer than step readiness equivalence. Hence we will obtain the strongest possible results for both directions and it turns out that the concept of distributability is fairly robust w.r.t. the choice of a suitable equivalence: any equivalence notion between step readiness equivalence and branching ST-bisimilarity with explicit divergence will yield the same characterisation.

Definition 14. A Petri net N is *distributable* up to an equivalence \approx iff there exists a distributed net N' with $N' \approx N$.

Formally we give our characterisation of distributability by classifying which finitary plain structural conflict nets can be implemented as distributed nets, and hence as LSGA nets. In such implementations, we use invisible transitions. We study the concept “distributable” for plain nets only, but in order to get the largest class possible we allow non-plain implementations, where a given transition may be split into multiple transitions carrying the same label.

It is well known that sometimes a global protocol is necessary to implement synchronous interaction present in system specifications. In particular, this may be needed for deciding choices in a coherent way, when these choices require agreement of multiple components. The simple net in Fig. 1 shows a typical situation of this kind.

Independent decisions of the two choices might lead to a deadlock. As remarked in [5], for this particular net there exists no satisfactory distributed implementation that fully respects the reactive behaviour of the original system. Indeed such M-structures, representing interference between concurrency and choice, turn out to play a crucial rôle for characterising distributability.

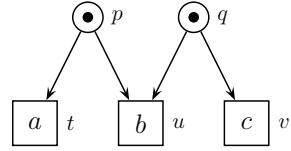


Fig. 1. A fully marked M

Definition 15. Let $N = (S, T, F, M_0, \ell)$ be a Petri net. N has a *fully reachable pure M* iff $\exists t, u, v \in T. \bullet t \cap \bullet u \neq \emptyset \wedge \bullet u \cap \bullet v \neq \emptyset \wedge \bullet t \cap \bullet v = \emptyset \wedge \exists M \in [M_0]. \bullet t \cup \bullet u \cup \bullet v \subseteq M$.

Note that Definition [15] implies that $t \neq u, u \neq v$ and $t \neq v$.

We now give an upper bound on the class of distributable nets by adopting a result from [5].

Theorem 2. *Let N be a plain structural conflict Petri net. If N has a fully reachable pure M, then N is not distributable up to step readiness equivalence.*

Proof. In [5] this theorem was obtained for plain one-safe nets [1]. The proof applies verbatim to plain structural conflict nets as well. □

Since \approx_{bSTb}^Δ is finer than $\approx_{\mathcal{R}}$, this result holds also for distributability up to \approx_{bSTb}^Δ (and any equivalence between $\approx_{\mathcal{R}}$ and \approx_{bSTb}^Δ).

In the following, we establish that this upper bound is tight, and hence a finitary plain structural conflict net is distributable iff it has no fully reachable pure M. For this, it is helpful to first introduce macros in Petri nets for reversibility of transitions.

4.1 Petri Nets with Reversible Transitions

A *Petri net with reversible transitions* generalises the notion of a Petri net; its semantics is given by a translation to an ordinary Petri net, thereby interpreting the reversible transitions as syntactic sugar for certain net fragments. It is defined as a tuple $(S, T, \Omega, \iota, F, M_0, \ell)$ with S a set of places, T a set of (reversible) transitions, labelled by $\ell : T \rightarrow \text{Act} \cup \{\tau\}$, Ω a set of *undo interfaces* with the relation $\iota \subseteq \Omega \times T$ linking interfaces to transitions, $M_0 \in \mathbb{N}^S$ an initial marking, and

$$F : (S \times T \times \{in, early, late, out, far\}) \rightarrow \mathbb{N}$$

the flow relation. For $t \in T$ and $type \in \{in, early, late, out, far\}$, the multiset of places $t^{type} \in \mathbb{N}^S$ is given by $t^{type}(s) = F(s, t, type)$. When $s \in t^{type}$ for $type \in \{in, early, late\}$, the place s is called a *preplace* of t of type $type$; when $s \in t^{type}$ for $type \in \{out, far\}$, s is called a *postplace* of t of type $type$. For each undo interface $\omega \in \Omega$ and transition t with $\iota(\omega, t)$ there must be places $\text{undo}_\omega(t)$, $\text{reset}_\omega(t)$ and $\text{ack}_\omega(t)$ in S . A transition with a nonempty set of interfaces is called *reversible*; the other (*standard*) transitions may have pre- and postplaces of types *in* and *out* only—for these transitions $t^{in} = \bullet t$ and $t^{out} = t \bullet$. In case $\Omega = \emptyset$, the net is just a normal Petri net.

¹ In [5] the theorem was claimed and proven only for plain nets with a fully reachable *visible* pure M; however, for plain nets the requirement of visibility is irrelevant.

A global state of a Petri net with reversible transitions is given by a marking $M \in \mathbb{N}^S$, together with the state of each reversible transition “currently in progress”. Each transition in the net can fire as usual. A reversible transition can moreover take back (some of) its output tokens, and be *undone* and *reset*. When a transition t fires, it consumes $\sum_{type \in \{in, early, late\}} F(s, t, type)$ tokens from each of its preplaces s and produces $\sum_{type \in \{out, far\}} F(s, t, type)$ tokens in each of its postplaces s . A reversible transition t that has fired can start its reversal by consuming a token from $undo_\omega(t)$ for one of its interfaces ω . Subsequently, it can take back one by one a token from its postplaces of type *far*. After it has retrieved all its output of type *far*, the transition is undone, thereby returning $F(s, t, early)$ tokens in each of its preplaces s of type *early*. Afterwards, by consuming a token from $reset_\omega(t)$, for the same interface ω that started the undo-process, the transition terminates its chain of activities by returning $F(s, t, late)$ tokens in each of its *late* preplaces s . At that occasion it also produces a token in $ack_\omega(t)$. Alternatively, two tokens in $undo_\omega(t)$ and $reset_\omega(t)$ can annihilate each other without involving the transition t ; this also produces a token in $ack_\omega(t)$. The latter mechanism comes in action when trying to undo a transition that has not yet fired.

Fig. 2 shows the translation of a reversible transition t with $\ell(t) = a$ into an ordinary net fragment. The arc weights on the green (or grey) arcs are inherited from the untranslated net; the other arcs have weight 1.

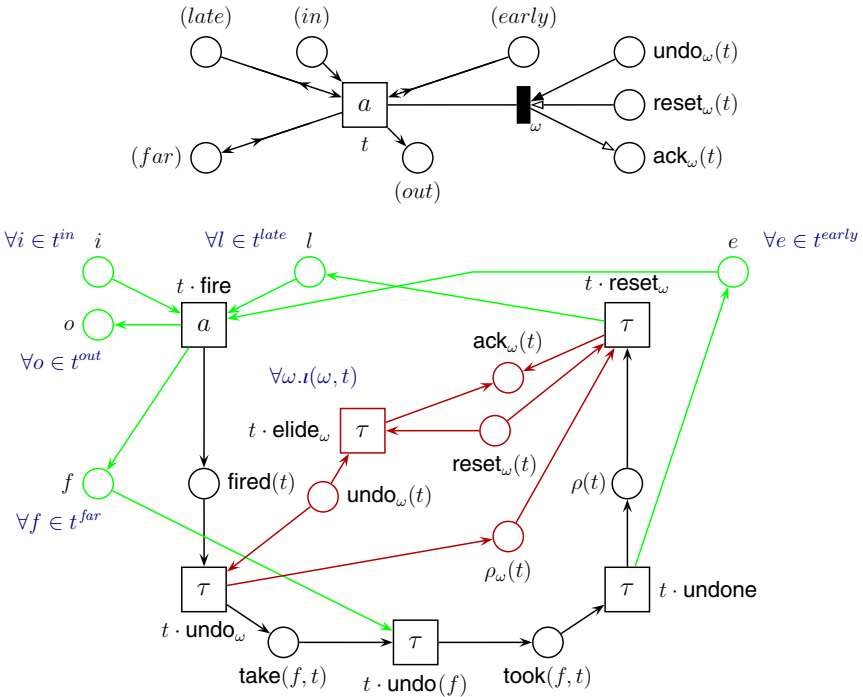


Fig. 2. A reversible transition and its macro expansion

4.2 The Conflict Replicating Implementation

Now we establish that a finitary plain structural conflict net that has no fully reachable pure M is distributable. We do this by proposing the *conflict replicating implementation* of any such net, and show that this implementation is always (a) essentially distributed, and (b) equivalent to the original net. In order to get the strongest possible result, for (b) we use branching ST-bisimilarity with explicit divergence.

To define the conflict replicating implementation of a net $N = (S, T, F, M_0, \ell)$ we fix an arbitrary well-ordering $<$ on its transitions. We let b, c, h, i, j, k, l range over these ordered transitions, and write

- $i \# j$ iff $i \neq j \wedge i \cap \bullet j \neq \emptyset$ (transitions i and j are in conflict), and $i \# \# j$ iff $i \# j \vee i = j$,
- $i < \# j$ iff $i < j \wedge i \# j$, and $i \leq \# j$ iff $i < \# j \vee i = j$.

Fig. 3 shows the conflict replicating implementation N . It is presented as a Petri net $\mathcal{I}(N) = (S', T', F', \Omega, \iota, M'_0, \ell')$ with reversible transitions. The set Ω of undo interfaces (not drawn) is T , and for $i \in \Omega$ we have $\iota(i, t)$ iff $t \in \Omega_i$, where the sets of transitions $\Omega_i \in \mathbb{N}^{T'}$ are specified in Fig. 3. The implementation $\mathcal{I}(N)$ inherits the places of N (i.e. $S' \supseteq S$), and we postulate that $M'_0 \upharpoonright S = M_0$. Given this, Fig. 3 is not merely an illustration of $\mathcal{I}(N)$ —it provides a complete and accurate description of it, thereby defining the conflict replicating implementation of any net. In interpreting this figure it is important to realise that net elements are completely determined by their name (identity), and exist only once, even if they show up multiple times in the figure. For instance, the place $\pi_{h\#j}$ with $h=2$ and $j=5$ (when using natural numbers for the transitions in T) is the same as the place $\pi_{j\#l}$ with $j=2$ and $l=5$; it is a standard preplace of execute_2^i (for all $i \leq \# 2$), a standard postplace of fetched_2^i , as well as a late preplace of transfer_5^2 .

The rôle of the transitions distribute_p for $p \in S$ is to distribute a token in p to copies p_j of p in the localities of all transitions $j \in T$ with $p \in \bullet j$. In case j is enabled in N , the transition initialise_j will become enabled in $\mathcal{I}(N)$. These transitions put tokens in the places pre_k^j , which are preconditions for all transitions execute_k^j , which model the execution of j at the location of k . When two conflicting transitions h and j are both enabled in N , the first steps initialise_h and initialise_j towards their execution in $\mathcal{I}(N)$ can happen in parallel. To prevent them from executing both, execute_j^j (of j at its own location) is only possible after transfer_j^h , which disables execute_h^h .

The main idea behind the conflict replicating implementation is that a transition $h \in T$ is primarily executed by a sequential component of its own, but when a conflicting transition j gets enabled, the sequential component implementing j may “steal” the possibility to execute h from the home component of h , and keep the options to do h and j open until one of them occurs. To prevent h and j from stealing each other’s initiative, which would result in deadlock, a global asymmetry is built in by ordering the transitions. Transition j can steal the initiative from h only when $h < j$.

In case j is also in conflict with a transition l , with $j < l$, the initiative to perform j may subsequently be stolen by l . In that case either h and l are in conflict too—then l takes responsibility for the execution of h as well—or h and l are concurrent—in that case h will not be enabled, due to the absence of fully reachable pure M s in N . The absence of fully reachable pure M s also guarantees that it cannot happen that two concurrent transitions j and k both steal the initiative from an enabled transition h .

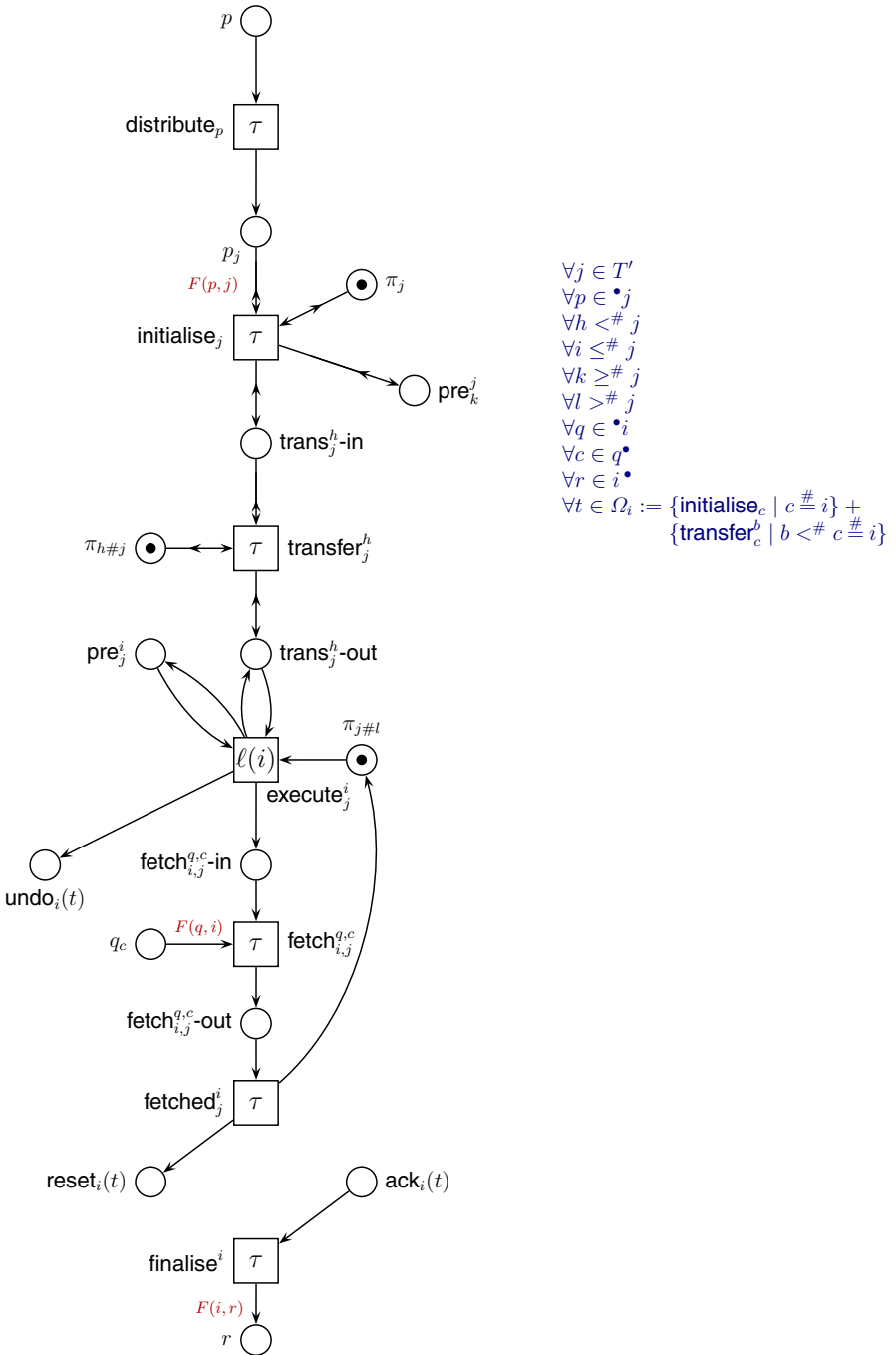


Fig. 3. The conflict replicating implementation

After the firing of execute_j^i all tokens that were left behind in the process of carefully orchestrating this firing will have to be cleaned up, to prepare the net for the next activity in the same neighbourhood. This is the reason for the reversibility of the transitions preparing the firing of execute_j^i . Hence there is an undo interface for each transition $i \in T'$, cleaning up the mess made in preparation of firing execute_j^i for some j . Ω_i is the multiset of all transitions t that could possibly have contributed to this. For each of them its interface i is activated, by execute_j^i depositing a token in $\text{undo}_i(t)$. When all preparatory transitions that have fired are undone, tokens appear in the places p_c for all $p \in \bullet i$ and $c \in p^\bullet$. These are collected by $\text{fetch}_{i,j}^{p,c}$, after which all $t \in \Omega_i$ get a reset signal. Those that have fired and were undone are reset, and those that never fired perform $\text{elide}_i(t)$. In either case a token appears in $\text{ack}_i(t)$. These are collected by finalise_i^i , which finishes the execution of i by depositing tokens in its postplaces.

Proposition 1. $\mathcal{I}(N)$ is essentially distributed for every Petri net N .

Proof sketch. We take the canonical distribution D of N , in which \equiv_D is the equivalence relation on places and transitions generated by Condition (1) of Definition 13. We need to show that D satisfies the weakened Condition 2. Any location that harbours an external transition execute_j^i for some $i \leq j \in T'$, also harbours $\text{initialise}_j \cdot \text{undo}(\text{pre}_j^i)$, $\text{transfer}_j^h \cdot \text{undo}(\text{trans}_j^h\text{-out})$ for all $h <^\# j$, execute_j^i for all $i \leq^\# j$, and, for all $l >^\# j$, $\text{transfer}_l^j \cdot \text{fire}$ and $\text{initialise}_l \cdot \text{undo}(\text{trans}_l^j\text{-in})$. In 6 we show that none of these transitions can happen concurrently with execute_j^i . ■

Theorem 3. Let N be a finitary plain structural conflict net without a fully reachable pure M . Then N is distributable up to \approx_{bSTb}^Δ .

Proof. In the full version of this paper 6. There we show that $\mathcal{I}(N) \approx_{bSTb}^\Delta N$. Hence $\mathcal{I}(N)$ is an essentially distributed implementation of N . Now apply Theorem 1. □

Given the complexity of our construction, no techniques known to us were adequate for performing this proof. We therefore had to develop an entirely new method for rigorously proving the equivalence of two Petri nets up to \approx_{bSTb}^Δ , one of which known to be plain. This method is presented in 6.

Corollary 2. Let N be a finitary plain structural conflict net. Then N is distributable iff it has no fully reachable pure M . □

5 Conclusion

In this paper, we have given a precise characterisation of distributable Petri nets in terms of a semi-structural property. Moreover, we have shown that our notion of distributability corresponds to an intuitive notion of a distributed system by establishing that any distributable net may be implemented as a network of asynchronously communicating components.

In order to formalise what qualifies as a valid implementation, we needed a suitable equivalence relation. We have chosen step readiness equivalence for showing the impossibility part of our characterisation, since it is one of the simplest and least discriminating semantic equivalences imaginable that abstracts from internal actions but preserves

branching time, concurrency and divergence to some small degree. For the positive part, stating that all other nets are implementable, we have introduced a combination of several well known rather discriminating equivalences, namely a divergence sensitive version of branching bisimulation adapted to ST-semantics. Hence our characterisation is rather robust against the chosen equivalence; it holds in fact for all equivalences between these two notions. However, ST-equivalence (and our version of it) preserves the causal structure between action occurrences only as far as it can be expressed in terms of the possibility of durational actions to overlap in time. Hence a natural question is whether we could have chosen an even stronger causality sensitive equivalence for our implementability result, respecting e.g. pomset equivalence or history preserving bisimulation. Our conflict replicating implementation does not fully preserve the causal behaviour of nets; we are convinced we have chosen the strongest possible equivalence for which our implementation works. It is an open problem to find a class of nets that can be implemented distributedly while preserving divergence, branching time and causality in full. Another line of research is to investigate which Petri nets can be implemented as distributed nets when relaxing the requirement of preserving the branching structure. If we allow linear time correct implementations (using a step trace equivalence), we conjecture that all Petri nets become distributable. However, also in this case it is problematic, in fact even impossible in our setting, to preserve the causal structure, as has been shown in [14]. A similar impossibility result has been obtained in the world of the π -calculus in [12].

The interplay between choice and synchronous communication has already been investigated in quite a number of approaches in different frameworks. We refer to [5] for a rather comprehensive overview and concentrate here on recent and closely related work.

The idea of modelling asynchronously communicating sequential components by sequential Petri nets interacting through buffer places has already been considered in [13]. There Wolfgang Reisig introduces a class of systems, represented as Petri nets, where the relative speeds of different components are guaranteed to be irrelevant. His class is a strict subset of our LSGA nets, requiring additionally, amongst others, that all choices in sequential components are free, i.e. do not depend upon the existence of buffer tokens, and that places are output buffers of only one component. Another quite similar approach was taken in [3], where transition labels are classified as being either input or output. There, asynchrony is introduced by adding new buffer places during net composition. This framework does not allow multiple senders for a single receiver.

Other notions of distributed and distributable Petri nets are proposed in [9][12]. In these works, given a distribution of the transitions of a net, the net is distributable iff it can be implemented by a net that is distributed w.r.t. that distribution. The requirement that concurrent transitions may not be co-located is absent; given the fixed distribution, there is no need for such a requirement. These papers differ from each other, and from ours, in what counts as a valid implementation. A comparison of our criterion with that of Hopkins [9] is provided in [5].

In [5] we have obtained a characterisation similar to Corollary 2 but for a much more restricted notion of distributed implementation (*plain distributability*), disallowing nontrivial transition labellings in distributed implementations. We also proved that fully reachable pure Ms are not implementable in a distributed way, even when using

transition labels (Theorem 2). However, we were not able to show that this upper bound on the class of distributable systems was tight. Our current work implies the validity of Conjecture 1 of [5]. While in [5] we considered only one-safe place/transition systems, the present paper employs a more general class of place/transition systems, namely structural conflict nets. This enables us to give a concrete characterisation of distributed nets as systems of sequential components interacting via non-safe buffer places.

References

1. Badouel, E., Caillaud, B., Darondeau, P.: Distributing Finite Automata Through Petri Net Synthesis. *Formal Aspects of Computing* 13(6), 447–470 (2002)
2. Best, E., Darondeau, P.: Petri Net Distributability. In: Voronkov, A. (ed.) *PSI 2011. LNCS*, vol. 7162, pp. 1–18. Springer, Heidelberg (2012)
3. El Hog Benzina, D., Haddad, S., Hennicker, R.: Process Refinement and Asynchronous Composition with Modalities. In: Sidorova, N., Serebrenik, A. (eds.) *Proceedings of the 2nd International Workshop on Abstractions for Petri Nets and Other Models of Concurrency (APNOC 2010)*, Braga, Portugal (2010), <http://www.lsv.ens-cachan.fr/Publications/PAPERS/PDF/EHH-apnoc10.pdf>
4. van Glabbeek, R.J., Goltz, U., Schicke, J.-W.: Abstract Processes of Place/Transition Systems. *Information Processing Letters* 111(13), 626–633 (2011), doi:10.1016/j.ipl.2011.03.013
5. van Glabbeek, R.J., Goltz, U., Schicke, J.-W.: On Synchronous and Asynchronous Interaction in Distributed Systems. In: Ochmański, E., Tyszkiewicz, J. (eds.) *MFCS 2008. LNCS*, vol. 5162, pp. 16–35. Springer, Heidelberg (2008)
6. van Glabbeek, R.J.: Goltz U J.-W. Schicke-Uffmann On Distributability of Petri Nets. Technical Report 2011-10, TU Braunschweig (2011) Full version of this paper (to appear), <http://theory.stanford.edu/~rvg/abstracts.html#95>
7. van Glabbeek, R.J., Vaandrager, F.W.: Petri Net Models for Algebraic Theories of Concurrency (Extended Abstract). In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) *PARLE 1987. LNCS*, vol. 259, pp. 224–242. Springer, Heidelberg (1987)
8. van Glabbeek, R.J., Weijland, W.P.: Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM* 43(3), 555–600 (1996), doi:10.1145/233551.233556
9. Hopkins, R.P.: Distributable Nets. In: Rozenberg, G. (ed.) *APN 1991. LNCS*, vol. 524, pp. 161–187. Springer, Heidelberg (1991), doi:10.1007/BFb0019974
10. Milner, R.: *Communication and Concurrency*. Prentice Hall, Englewood Cliffs (1989)
11. Olderog, E.-R., Hoare, C.A.R.: Specification-oriented semantics for communicating processes. *Acta Informatica* 23, 9–66 (1986), doi:10.1007/BF00268075
12. Peters, K., Schicke, J.-W., Nestmann, U.: Synchrony vs Causality in the Asynchronous Pi-Calculus. In: Luttkik, B., Valencia, F. (eds.) *Proceedings 18th International Workshop on Expressiveness in Concurrency*, Aachen, Germany, September 5. *Electronic Proceedings in Theoretical Computer Science*, vol. 64, pp. 89–103 (2011), doi:10.4204/EPTCS.64.7
13. Reisig, W.: Deterministic Buffer Synchronization of Sequential Processes. *Acta Informatica* 18, 115–134 (1982), doi:10.1007/BF00264434
14. Schicke, J.-W., Peters, K., Goltz, U.: Synchrony vs. Causality in Asynchronous Petri Nets. In: Luttkik, B., Valencia, F. (eds.) *Proceedings 18th International Workshop on Expressiveness in Concurrency*, Aachen, Germany, September 5. *Electronic Proceedings in Theoretical Computer Science*, vol. 64, pp. 119–131 (2011), doi:10.4204/EPTCS.64.9
15. Vogler, W.: Bisimulation and Action Refinement. *Theor. Comput. Sci.* 114(1), 173–200 (1993), doi:10.1016/0304-3975(93)90157-O

Functions as Session-Typed Processes

Bernardo Toninho^{1,2}, Luis Caires², and Frank Pfenning¹

¹ Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA

² CITI and Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Lisboa, Portugal

Abstract. We study type-directed encodings of the simply-typed λ -calculus in a session-typed π -calculus. The translations proceed in two steps: standard embeddings of simply-typed λ -calculus in a linear λ -calculus, followed by a standard translation of linear natural deduction to linear sequent calculus. We have shown in prior work how to give a Curry-Howard interpretation of the proofs in the linear sequent calculus as π -calculus processes subject to a session type discipline. We show that the resulting translations induce sharing and copying parallel evaluation strategies for the original λ -terms, thereby providing a new logically motivated explanation for these strategies.

1 Introduction

The goal of this paper is to study type-directed encodings of the simply-typed λ -calculus in a session-typed π -calculus, including a novel encoding that captures a reduction strategy in which shared sub-expressions are evaluated in parallel (providing a logical justification for *futures* [10]). Unlike other proposals, our encodings are canonically extracted from standard translations of linear natural deduction to linear sequent calculus in standard logical systems, based on the interpretation of session-typed processes as intuitionistic linear logic proofs we developed in [6].

Milner [17] presented two translations of the λ -calculus into the π -calculus: one capturing call-by-name and one capturing call-by-value. This provided evidence for the universality of the π -calculus for expressing sequential computation. Since then, a number of other translations have been developed (see [19]).

An interesting foundational question is if similar translations exist from *typed* λ -calculus to a *typed* π -calculus. Van Bakel and Vigliotti [21] give a partial answer by providing a conceptually different translation into the *asynchronous* π -calculus where the image of simply-typed λ -terms can be assigned types in Gentzen's classical sequent calculus LK. In this paper we provide another answer, also rooted in logic. We first translate from the simply-typed λ -calculus to a *linear* λ -calculus, using either of the two canonical embeddings proposed by Girard [8] as detailed by Maraist et al [15]. The linear λ -calculus is related to intuitionistic linear logic by a Curry-Howard isomorphism [22, 1], so we can now apply a standard translation from natural deduction to the dual intuitionistic linear sequent calculus. In prior work [6], we have shown how

to view this sequent calculus as a type assignment system for the π -calculus, capturing *session types* [12] in a purely logical manner. Viewed end-to-end, we have two type-directed translations from the simply-typed λ -calculus to the session-typed π -calculus, depending which of Girard’s embeddings of intuitionistic logic in linear logic we use.

Remarkably, the translations constructed in this type-directed manner are closely related to Milner’s two original translations that capture call-by-name and call-by-value. While our call-by-name translation behaves the same way, our analog of the call-by-value translation is subtly different in that the resulting terms behave like *futures* [10]: in an application $(\lambda x. M) N$ reductions in the function body M and the argument N can proceed in parallel, synchronizing only when the value of x is needed. If we always proceed with reducing N first, we obtain call-by-value. If we always proceed with the body M first we obtain call-by-need. So perhaps it is more appropriate to characterize the semantic endpoints of the two translations as *copying* (as in call-by-name) and *sharing* (as in futures, relaxing the sequentiality constraints of call-by-value and call-by-need). Remarkably, the sharing semantics thus obtained is reminiscent of sharing reductions as studied in the context of optimal reduction strategies [14,9], a study that goes back to Wadsworth’s thesis [23], although in our case it just comes out naturally from our logically grounded translations. Furthermore, to the best of the authors’ knowledge, such a sharing encoding of the λ -calculus in the π -calculus was not identified before this work.

Another remarkable property, due to the simple logical nature of the interpretations, is that the linear target type of the translation is *inhabited* by a π -calculus term if and only if the source type is inhabited by a λ -calculus term. In a sense, the linear session type discipline is a tight characterization for simple typing in the λ -calculus source, exposing intrinsic parallelism in its structure. We thus show that session types are powerful enough to type functional evaluation.

In the remainder of this paper we proceed as follows: We detail our translation of the λ -calculus to the π -calculus, by presenting the several intermediate steps (Sections 2.1, 2.2) and composing them in Section 2.4. In Section 3 we show the operational soundness and completeness of our translation. Finally, we present some concluding remarks and a discussion of related work in Section 4.

2 From the λ -Calculus to the π -Calculus

In this section we present several translations which, in combination, allow us to translate typed λ -calculus terms to *session-typed* π -calculus processes. We first review two well-known translations from the simply typed λ -calculus to the linear λ -calculus, introduced in Girard’s seminal paper [8] as translations from intuitionistic to linear logic, and hence by the Curry-Howard correspondence, from the λ -calculus to the linear λ -calculus [15,1]. We then show how to translate from the linear λ -calculus (logically, a system of natural deduction) to a sequent formulation of linear logic. The latter will yield the desired π -calculus terms by a Curry-Howard interpretation.

2.1 Interpreting the λ -Calculus in the Linear λ -Calculus

We consider the simply typed λ -calculus with function types $T \rightarrow S$ and base types b . Our translation naturally extends to products and sums, interpreted as the product and

sum types of the linear λ -calculus, but we refrain from presenting those here for the sake of simplicity. The simply-typed λ calculus is given by the following grammar of terms M, N and types T, S , where b denotes base types. The typing rules for the calculus are given below, inductively defining the judgment $\Gamma \vdash M : T$, where Γ denotes a finite set of unique typing assumptions of the form $x : T$, as usual.

Definition 2.1 (λ -calculus Terms and Typing)

$$\begin{array}{l}
 M, N ::= x \mid \lambda x : T. M \mid M N \\
 T, S ::= T \rightarrow S \mid b \\
 \frac{}{\Gamma, x : T \vdash x : T} \text{hyp} \quad \frac{\Gamma, x : T \vdash M : S}{\Gamma \vdash \lambda x : T. M : T \rightarrow S} \rightarrow I \\
 \frac{\Gamma \vdash M : S \rightarrow T \quad \Gamma \vdash N : S}{\Gamma \vdash M N : T} \rightarrow E
 \end{array}$$

The linear λ -calculus is a more fine-grained version of the λ -calculus, in which variables in a λ -abstraction must be used exactly once. The calculus is endowed with a modal type $!T$ (referred to as an exponential) that allows for unrestricted usage of variables, in order to obtain the full generality of the λ -calculus. We distinguish between the unrestricted and linear λ -abstraction by using $\hat{\lambda}$ for the latter. The linear function type is written as \multimap . The terms and types of the linear λ -calculus are given below.

Definition 2.2 (Linear λ -calculus Terms)

$$\begin{array}{l}
 M, N ::= x \mid u \mid \hat{\lambda} x : T. M \mid M N \mid !M \mid \text{let } !u = M \text{ in } N \\
 T, S ::= T \multimap S \mid !T \mid b
 \end{array}$$

We syntactically distinguish between linear variables x and unrestricted variables u (typing ensures linear variables are used exactly once, while unrestricted variables can be used arbitrarily often). The terms $!M$ and $\text{let } !u = M \text{ in } N$ are the introduction and elimination forms of the exponential type, respectively, where the variable u can occur in the term N in an unrestricted manner.

The typing rules for the linear calculus are given below, defining the judgment $\Gamma; \Delta \vdash M : T$ with Δ denoting *linear* (used exactly once) assumptions of the form $x : T$ (not subject to weakening or contraction) and Γ unrestricted assumptions $u : T$.

Definition 2.3 (Linear λ -calculus Typing)

$$\begin{array}{l}
 \frac{}{\Gamma; x : T \vdash x : T} \text{hyp} \quad \frac{}{\Gamma, u : T; \cdot \vdash u : T} \text{uhyp} \\
 \frac{\Gamma; \Delta, x : T \vdash M : S}{\Gamma; \Delta \vdash \hat{\lambda} x : T. M : T \multimap S} \multimap I \quad \frac{\Gamma; \cdot \vdash M : T}{\Gamma; \cdot \vdash !M : !T} !I \\
 \frac{\Gamma; \Delta \vdash M : T \multimap S \quad \Gamma; \Delta' \vdash N : T}{\Gamma; \Delta, \Delta' \vdash M N : S} \multimap E \\
 \frac{\Gamma; \Delta \vdash M : !T \quad \Gamma, u : T; \Delta' \vdash N : S}{\Gamma; \Delta, \Delta' \vdash \text{let } !u = M \text{ in } N : S} !E
 \end{array}$$

Given that the Curry-Howard correspondence allows us to identify the simply typed λ -calculus with intuitionistic logic and the linear λ -calculus with linear intuitionistic logic, we make use of two well-known embeddings of intuitionistic logic in linear logic to interpret the λ -calculus in the linear λ -calculus. These embeddings originate from Girard's seminal paper on linear logic [8] and are detailed by Maraist et al [15]. The first translation, which we will refer to as the $[\cdot]$ translation, is defined inductively on types, terms and contexts below.

Definition 2.4 (The $[\cdot]$ Translation)

$$\begin{aligned}
 [T \rightarrow S] &\triangleq (![T]) \multimap [S] \\
 [b] &\triangleq b \\
 [x] &\triangleq u_x \\
 [\lambda x : T. M] &\triangleq \hat{\lambda}x : ![T]. \text{let } !u_x = x \text{ in } [M] \\
 [M N] &\triangleq [M] (![N]) \\
 [I, x : T] &\triangleq [I], x : [T]
 \end{aligned}$$

Intuitively, this translation maps λ -abstractions to linear λ -abstractions, where the argument is forced to be of exponential type, and thus can be let-bound to a unrestricted variable u_x . We write u_x for a fresh unrestricted variable encoding the (linear) variable x . The translation of application enforces the invariant, resulting in an application of the translated terms where the argument is prefixed with $!$. The correctness of the translation can be shown by a straightforward induction on typing.

Theorem 2.5 (Correctness of the $[\cdot]$ Translation). *If $I \vdash M : T$ then $[I]; \cdot \vdash [M] : [T]$.*

The second translation, which we denote as the $(\cdot)^*$ translation (of which Girard curiously remarked: “*This boring translation is reminiscent of the modal translation of intuitionistic logic*”) is slightly more involved, being inductively defined using an auxiliary translation, denoted by $(\cdot)^+$. The idea behind this translation is that all components of composite types are prefixed with a $!$, which results in the following inductive definition on types, terms and contexts, given below.

Definition 2.6 (The $(\cdot)^*$ Translation)

$$\begin{aligned}
 (T)^* &\triangleq !T^+ \\
 (T \rightarrow S)^+ &\triangleq T^* \multimap S^* \\
 (\tau)^+ &\triangleq \tau \\
 (x)^* &\triangleq !u_x \\
 (\lambda x : T. M)^* &\triangleq !(\hat{\lambda}x : !T^+. \text{let } !u_x = x \text{ in } M^*) \\
 (M N)^* &\triangleq (\text{let } !u = M^* \text{ in } u) N^* \\
 (I, x : T)^+ &\triangleq (I)^+, x : T^+
 \end{aligned}$$

Similar to the previous translation, we can show the correctness of the translation through a straightforward induction on typing.

Theorem 2.7 (Correctness of the $(\cdot)^*$ Translation). *If $\Gamma \vdash M : T$ then $\Gamma^+; \cdot \vdash M^* : T^*$ and $\Gamma^+; \cdot \vdash M^+ : T^+$.*

Using the two translations given above, we can take any well-typed λ -calculus term and translate it to a corresponding well-typed linear λ -calculus term (we omit type annotations in abstractions for readability purposes). For instance, the redex $(\lambda x. M) N$ can be translated into the following linear terms (we denote by \longrightarrow the linear λ calculus operational semantics, as given in Def. 3.1 – by-name application, by-value let binding, where the values are λ -abstractions and terms of the form $!M$):

$$\begin{aligned}
 [(\lambda x. M) N] &= (\hat{\lambda}x. \text{let } !u_x = x \text{ in } [M]) (![N]) \\
 &\longrightarrow \text{let } !u_x = ![N] \text{ in } [M] \\
 &\longrightarrow [M]\{[N]/u_x\} \\
 ((\lambda x. M) N)^* &= (\text{let } !u = !(\hat{\lambda}x. \text{let } !u_x = x \text{ in } M^*) \text{ in } u) N^* \\
 &\longrightarrow (\hat{\lambda}x. \text{let } !u_x = x \text{ in } M^*) N^* \\
 &\longrightarrow \text{let } !u_x = N^* \text{ in } M^*
 \end{aligned}$$

As can be seen in the example reductions above, the $[\cdot]$ translation induces a call-by-name reduction strategy on λ terms, while the $(\cdot)^*$ translation induces call-by-value. These observations are made precise by Maraist et al. [15].

Our goal is to ultimately translate a λ -calculus term to a π -calculus process, which as we detail in the following sections, is achieved by translating a linear λ -calculus term into a sequent calculus proof and then interpreting the result as a session-typed process.

2.2 From Natural Deduction to Sequent Calculus

The Curry-Howard correspondence of linear logic and the linear λ -calculus allows us to move interchangeably between typing derivations and linear logic proofs in natural deduction form. Another typical form of presenting logic is through a sequent calculus. A sequent calculus consists of a collection of so-called left and right rules which define how to use and prove a particular proposition, respectively. We can further equip a sequent calculus with a faithful proof term assignment, where proof terms serve as compact notation for proofs. The rules for a sequent calculus for linear logic (in particular, the fragment with implication and exponential) are given below, defining the judgment $\Gamma; \Delta \Rightarrow D : A$, meaning that D is a proof term for the proposition A , under the linear assumptions recorded in Δ and the unrestricted assumptions recorded in Γ . This formulation is usually called Dual Intuitionistic Linear Logic (DILL) [1].

Definition 2.8 (Sequent Calculus for DILL)

$$\begin{array}{c}
 \frac{}{\Gamma; x : A \Rightarrow \text{id } x : A} \text{id} \quad \frac{\Gamma; \Delta, x : A \Rightarrow D : B}{\Gamma; \Delta \Rightarrow \text{--oR } (x. D) : A \text{--o } B} \text{--oR} \\
 \\
 \frac{\Gamma; \cdot \Rightarrow D : A}{\Gamma; \cdot \Rightarrow \text{!R } D : \text{!}A} \text{!R} \quad \frac{\Gamma; \Delta \Rightarrow D : A \quad \Gamma; \Delta', y : B \Rightarrow E : C}{\Gamma; \Delta, \Delta', x : A \text{--o } B \Rightarrow \text{--oL } x D (y. E) : C} \text{--oL} \\
 \\
 \frac{\Gamma, u : A; \Delta \Rightarrow D : C}{\Gamma; \Delta, x : \text{!}A \Rightarrow \text{!L } x (u. D) : C} \text{!L} \quad \frac{\Gamma, u : A; \Delta, x : A \Rightarrow D : C}{\Gamma, u : A; \Delta \Rightarrow \text{copy } u (x. D) : C} \text{copy} \\
 \\
 \frac{\Gamma; \Delta \Rightarrow D : A \quad \Gamma; \Delta', x : A \Rightarrow E : C}{\Gamma; \Delta, \Delta' \Rightarrow \text{cut } D (x. E) : C} \text{cut} \\
 \\
 \frac{\Gamma; \cdot \Rightarrow D : A \quad \Gamma, u : A; \Delta \Rightarrow E : C}{\Gamma; \Delta \Rightarrow \text{cut}^! D (u. E) : C} \text{cut}^!
 \end{array}$$

It is possible to translate proofs in natural deduction style to sequent calculus, and therefore by the Curry-Howard correspondence we can straightforwardly translate linear λ -terms to the sequent calculus proof terms given above. Intuitively, the introduction forms ($\hat{\lambda}$ abstractions and $!$) are directly translated to the proof terms corresponding to the respective right rules. The elimination forms (application and let) are translated to instances of cut with their respective left rules. This translation is defined below, written as $\llbracket \cdot \rrbracket$, and is the computational content of the following theorem (modulo α -equivalence).

Theorem 2.9. *If $\Gamma; \Delta \vdash M : A$ then $\Gamma; \Delta \Rightarrow \llbracket M \rrbracket : A$.*

Definition 2.10 (The $\llbracket \cdot \rrbracket$ Sequent Calculus Translation)

$$\begin{array}{l}
 \llbracket x \rrbracket \quad \triangleq \text{id } x \\
 \llbracket u \rrbracket \quad \triangleq \text{copy } u (x. \text{id } x) \\
 \llbracket \hat{\lambda}x. M \rrbracket \quad \triangleq \text{--oR } (x. \llbracket M \rrbracket) \\
 \llbracket M N \rrbracket \quad \triangleq \text{cut } \llbracket M \rrbracket (x. \text{--oL } x \llbracket N \rrbracket (y. \text{id } y)) \\
 \llbracket \text{!}M \rrbracket \quad \triangleq \text{!R } M \\
 \llbracket \text{let } !u = M \text{ in } N \rrbracket \triangleq \text{cut } \llbracket M \rrbracket (x. \text{!L } x (u. \llbracket N \rrbracket))
 \end{array}$$

The relevance of this translation step is made clear in the next section, where we detail a tight correspondence between sequent calculus proofs and session-typed processes in a π -calculus.

There is also a more complex translation from natural deduction to sequent calculus, which eliminates some unnecessary instances of cut. An instance cut in a proof corresponds to a process reduction and so this other translation can be seen as a more optimized version, with fewer administrative reduction steps.

2.3 Linear Logic and Session Types

It turns out that there exists a tight correspondence between linear logic proofs and session typed π -calculus processes [620]. We summarize it here in the following rules for

the fragment of linear logic we are interested in. Letters P, Q, R range over processes, as defined below.

Definition 2.11 (π -calculus Processes)

$$P, Q ::= \mathbf{0} \mid P|Q \mid (\nu y)P \mid x\langle y \rangle.P \mid x(y).P \mid !x(y).P \mid [y \leftrightarrow x]$$

The process calculus is mostly standard. $\mathbf{0}$ stands for the terminated process, $P|Q$ is the parallel composition of processes P and Q , $(\nu y)P$ denotes the binding of a channel name y whose scope is P , $x\langle y \rangle.P$ denotes the process that outputs y along channel x and continues as P , $x(y).P$ stands for the process that inputs along x and binds the received value to y in P , $!x(y).P$ denotes replicated input and $[x \leftrightarrow y]$ is a channel forwarding construct, that equates channel names x and y [20]. We consider it here as a primitive construct, even if in the typed language it may be encoded by a copycat process generated by expanding proofs of non-atomic identity axioms as shown in [6], in a way closely related to the internal mobility translation of [18].

The operational semantics of the processes given above are defined modulo a structural congruence relation, written $P \equiv Q$.

Definition 2.12 (Structural Congruence). *Structural congruence is the least congruence satisfying the following rules*

$$\begin{array}{ll} P \mid \mathbf{0} \equiv P & P \equiv_{\alpha} Q \Rightarrow P \equiv Q \\ P \mid (Q \mid R) \equiv (P \mid Q) \mid R & P \mid Q \equiv Q \mid P \\ x \notin \text{fn}(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) & (\nu x)\mathbf{0} \equiv \mathbf{0} \\ (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & [y \leftrightarrow x] \equiv [x \leftrightarrow y] \end{array}$$

All structural congruence rules given above are standard. The reduction rules for processes are as follows.

Definition 2.13 (Operational Semantics of Processes)

$$\begin{array}{l} x\langle y \rangle.Q \mid x(z).P \rightarrow Q \mid P\{y/z\} \\ x\langle y \rangle.Q \mid !x(z).P \rightarrow Q \mid P\{y/z\} \mid !x(z).P \\ (\nu x)([x \leftrightarrow y] \mid P) \rightarrow P\{y/x\} \quad (\text{provided } x \neq y) \\ Q \rightarrow Q' \Rightarrow P \mid Q \rightarrow P \mid Q' \\ P \rightarrow Q \Rightarrow (\nu y)P \rightarrow (\nu y)Q \\ P \equiv P', P' \rightarrow Q', Q' \equiv Q \Rightarrow P \rightarrow Q \end{array}$$

The typing judgment for processes is $\Gamma; \Delta \Rightarrow P :: z : A$, where P is a process that implements a session of type A on channel z , when composed with processes implementing the sessions in Γ and Δ . All channel names in Γ , Δ and z must be mutually distinct, and we may implicitly rename bound names to maintain this invariant.

$$\begin{array}{c}
 \frac{}{\Gamma; x : A \Rightarrow [x \leftrightarrow z] :: z : A} \text{id} \quad \frac{\Gamma; \Delta, x : A \Rightarrow P :: z : B}{\Gamma; \Delta \Rightarrow z(x).P :: z : A \multimap B} \multimap\text{R} \\
 \\
 \frac{\Gamma; \cdot \Rightarrow P :: x : A}{\Gamma; \cdot \Rightarrow !z(x).P :: z : !A} !\text{R} \quad \frac{\Gamma, u : A; \Delta \Rightarrow P :: z : C}{\Gamma; \Delta, x : !A \Rightarrow P\{x/u\} :: z : C} !\text{L} \\
 \\
 \frac{\Gamma; \Delta \Rightarrow P :: y : A \quad \Gamma; \Delta', x : B \Rightarrow Q :: z : C}{\Gamma; \Delta, \Delta', x : A \multimap B \Rightarrow (\nu y)x\langle y \rangle.(P \mid Q) :: z : C} \multimap\text{L} \\
 \\
 \frac{\Gamma, u : A; \Delta, x : A \Rightarrow P :: z : C}{\Gamma, u : A; \Delta \Rightarrow (\nu x)u\langle x \rangle.P :: z : C} \text{copy} \\
 \\
 \frac{\Gamma; \Delta \Rightarrow P :: x : A \quad \Gamma; \Delta', x : A \Rightarrow Q :: z : C}{\Gamma; \Delta, \Delta' \Rightarrow (\nu x)(P \mid Q) :: z : C} \text{cut} \\
 \\
 \frac{\Gamma; \cdot \Rightarrow P :: x : A \quad \Gamma, u : A; \Delta \Rightarrow Q :: z : C}{\Gamma; \Delta \Rightarrow (\nu u)(!u(x).P \mid Q) :: z : C} \text{cut}^!
 \end{array}$$

$A \multimap B$ corresponds to the session input type (i.e. input a session of type A and proceed as B), $!A$ corresponds to a persistent session of type A , while cut corresponds to session composition (and cut elimination to reduction). The left rules indicate how to use a session of a given type. The translation of a sequent calculus proof term D to a well-typed process \hat{D}^z is a straightforward mapping of the sequent calculus rules to the appropriate typing rules, singling out a distinguished channel name z .

Definition 2.14 (Translation from Sequent to Type Derivations). *The translation of a proof term D to a process \hat{D}^z , written $D \rightsquigarrow \hat{D}^z$ is defined by:*

$$\begin{array}{ll}
 \text{id } x & \rightsquigarrow [x \leftrightarrow z] \\
 \multimap\text{R } (y. D) & \rightsquigarrow z(y).\hat{D}^z \\
 \multimap\text{L } x D (x. E) & \rightsquigarrow (\nu y)x\langle y \rangle.(\hat{D}^y \mid \hat{E}^z) \\
 !\text{R } D & \rightsquigarrow !z(y).\hat{D}^y \\
 !\text{L } x (u. D) & \rightsquigarrow \hat{D}^z\{x/u\} \\
 \text{copy } u (y. D) & \rightsquigarrow (\nu y)u\langle y \rangle.\hat{D}^z \\
 \text{cut } D (x. E) & \rightsquigarrow (\nu x)(\hat{D}^x \mid \hat{E}^z) \\
 \text{cut}^! D (u. E) & \rightsquigarrow (\nu u)((!u(y).\hat{D}^y) \mid \hat{E}^z)
 \end{array}$$

The correctness of the translation, and other results regarding the correspondence mentioned above, including some discussion on channel linking are detailed in [6].

Finally, by composing the $\llbracket \cdot \rrbracket$ and process translations, we can translate a linear λ -calculus term to a process (we refer to this composition as $\llbracket \cdot \rrbracket_z$, where z is the name on which the process implements the appropriate session).

Definition 2.15 (The $[\cdot]_z$ Translation)

$$\begin{aligned}
[[x]]_z &\triangleq [x \leftrightarrow z] \\
[[u]]_z &\triangleq (\nu x)u\langle x \rangle.[x \leftrightarrow z] \\
[[\lambda x.M]]_z &\triangleq z(x).[[M]]_z \\
[[MN]]_z &\triangleq (\nu x)([[M]]_x \mid (\nu y)x\langle y \rangle.([[N]]_y \mid [x \leftrightarrow z])) \\
[[!M]]_z &\triangleq !z(x).[[M]]_x \\
[[\text{let } !u = M \text{ in } N]]_z &\triangleq (\nu x)([[M]]_x \mid [[N]]_z\{x/u\})
\end{aligned}$$

And thus we have the following correctness theorem, which follows straightforwardly from the composition of Theorems 2.9 and the correctness of the process translation.

Theorem 2.16. *If $\Gamma; \Delta \vdash M : A$ then $\Gamma; \Delta \Rightarrow [[M]]_z :: z : A$*

2.4 Composing the Translations

We can now compose the translations from Sections 2.1 and 2.2 with the translation $[\cdot]_z$ defined above to give translations from the λ -calculus to the session typed process calculus of the previous section. Note that while our typing discipline requires channel linking (arising from the proof theory), this is done without loss of generality, given that it is possible to encode this behavior using a forwarding process [184].

Copying Translation. Let us first consider the composition of $[\cdot]$ with $[\cdot]_z$, which we will write as $[\cdot]_z$ (we will omit the translation of types and contexts since those are the same as in $[\cdot]$):

$$\begin{aligned}
[x]_z &\triangleq (\nu x)u_x\langle x \rangle.[x \leftrightarrow z] \\
[\lambda x.M]_z &\triangleq z(x).(\nu y)([x \leftrightarrow y] \mid [M]_z\{y/u_x\}) \\
[MN]_z &\triangleq (\nu w)([M]_w \mid (\nu y)w\langle y \rangle.(!y(x).[N]_x) \mid [w \leftrightarrow z])
\end{aligned}$$

By composing the translation correctness theorems (Theorems 2.5 and 2.16) we obtain the following correctness result.

Theorem 2.17. *If $\Gamma \vdash M : T$ then $[\Gamma]; \cdot \Rightarrow [M]_z :: z : [T]$.*

It is known that the $[\cdot]$ translation from the λ -calculus to the linear λ -calculus corresponds to a call-by-name evaluation strategy. We can observe the evaluation strategy induced by our $[\cdot]_z$ translation by considering the process in the image of the translation of a β -redex (we write \rightarrow^n for the n -fold iteration of \rightarrow):

$$\begin{aligned}
[(\lambda x.M)N]_z &= (\nu w)(w(x).(\nu y')([x \leftrightarrow y'] \mid [M]_w\{y'/u_x\}) \\
&\quad \mid (\nu y)w\langle y \rangle.(!y(x).[N]_x) \mid [w \leftrightarrow z]) \\
&\rightarrow (\nu w)(\nu y)(\nu y')([y \leftrightarrow y'] \mid [M]_w\{y'/u_x\} \\
&\quad \mid !y(x).[N]_x \mid [w \leftrightarrow z]) \\
&\rightarrow^2 (\nu y)([M]_z\{y/u_x\} \mid !y(x).[N]_x)
\end{aligned}$$

As we can see above, in the $[\cdot]_z$ translation, a β -redex is translated to a process that is the parallel composition of the translation of the body M of the λ -abstraction and a

replicated instance of the argument N (both sharing a private channel y , along which they can communicate). For each occurrence of the variable x in M , the translation will generate outputs along y that trigger the evaluation of a new copy of N . For this reason, we call the evaluation strategy induced by $[\cdot]_z$ a *copying* evaluation strategy.

This translation is surprisingly similar to that originally presented by Milner in [17] as the call-by-name translation of the λ -calculus. Milner's untyped translation is (we use a different notation than Milner's since it overlaps with our $\llbracket \cdot \rrbracket$ translation):

$$\begin{aligned} [x]_z &\triangleq x\langle z \rangle \\ [\lambda x.M]_z &\triangleq z(x).z(v).\llbracket M \rrbracket_v \\ [MN]_z &\triangleq (\nu w)(\llbracket M \rrbracket_w \mid (\nu y)w\langle y \rangle.w\langle z \rangle.(!y(x).\llbracket N \rrbracket_x)) \end{aligned}$$

Milner's additional communication steps fundamentally play the role of our channel links. In the translation of a variable, we output a fresh name which we then link to the free name z , while Milner's outputs z outright. In the translation of application, there is an output of a fresh name y (that will be used by the function to receive its argument) and then an output of the free name z , which is the distinguished channel of the translation. We avoid this second output by, after sending the fresh channel that will be used to communicate with the argument, linking the name w with the distinguished name z . In essence, Milner's translation is fundamentally the same as ours (modulo some minor syntactical issues). As mentioned above, it is possible to eliminate channel linking in the translation by replacing the explicit linking construct with a forwarder process in the style of [18,4], or by appealing to a natural notion of observational equivalence. Nonetheless, our typing discipline requires the channel links, thus typing modulo this notion of observational equivalence is left for future work. The fact that his untyped translation, in some sense, predicted the one that can be derived by proof theory, further acknowledges Milner's foresight.

However, we are motivated by the proof theoretical underpinnings of type preserving translations, which is in some sense a more canonical approach to the problem. Furthermore, Milner's goal was to encode call-by-name and call-by-value as closely as possible, using the π -calculus. Our approach just examines the result of standard proof-theoretic translations and observes that they are distinguished not so much by evaluation order, but by the degree of sharing.

Sharing Translation. Similarly, we can compose $(\cdot)^*$ with $\llbracket \cdot \rrbracket_z$, written $\llbracket \cdot \rrbracket_z^*$, as follows:

$$\begin{aligned} \llbracket x \rrbracket_z^* &\triangleq !z(a).(\nu x)u_x\langle x \rangle.[x \leftrightarrow a] \\ \llbracket \lambda x.M \rrbracket_z^* &\triangleq !z(a).a(x).(\nu y)([x \leftrightarrow y] \mid \llbracket M \rrbracket_a^*\{y/u_x\}) \\ \llbracket MN \rrbracket_z^* &\triangleq (\nu w)((\nu x)(\llbracket M \rrbracket_x^* \mid (\nu v)x\langle v \rangle.[v \leftrightarrow w]) \mid \\ &\quad (\nu y)w\langle y \rangle.(\llbracket N \rrbracket_y^* \mid [w \leftrightarrow z])) \end{aligned}$$

As for the previous translation, we can compose Theorems 2.7 and 2.16 to obtain the correctness of the translation.

Theorem 2.18. *If $\Gamma \vdash M : A$ then $\Gamma^+; \cdot \Rightarrow \llbracket M \rrbracket_z^* :: z : A^*$.*

The $(\cdot)^*$ translation from λ to linear λ terms corresponds to call-by-value. However, let us consider the translation of a β -redex in the $\llbracket \cdot \rrbracket_z$ translation:

$$\begin{aligned}
\llbracket (\lambda x.M) N \rrbracket_z^* &= (\nu w)((\nu x)(!x(a).a(b).(\nu y')([b \leftrightarrow y'] \mid \llbracket M \rrbracket_a^* \{y'/u_x\}) \\
&\quad \mid (\nu v)x\langle v \rangle.[v \leftrightarrow w] \mid (\nu y)w\langle y \rangle.(\llbracket N \rrbracket_y^* \mid [w \leftrightarrow z])) \\
&\rightarrow^2 (\nu w)(\nu y)(w(b).(\nu y')([b \leftrightarrow y'] \mid \llbracket M \rrbracket_w^* \{y'/u_x\}) \\
&\quad \mid w\langle y \rangle.(\llbracket N \rrbracket_y^* \mid [w \leftrightarrow z])) \\
&\rightarrow^3 (\nu y)(\llbracket M \rrbracket_z^* \{y/u_x\} \mid \llbracket N \rrbracket_y^*)
\end{aligned}$$

In the redex above, we obtain the translation of the abstraction body M in parallel with the translation of the argument N (sharing the private name y for communication). Unlike in the copying reduction strategy, the reductions of the argument and the abstraction body can proceed concurrently, generating no extra copies of N (note that this translation explicitly guards λ -abstractions and variables with replication). Since the reductions of N are *shared* by all variable occurrences in a λ -abstraction, we call this a *sharing* evaluation strategy. This strategy is that of *futures* in Multilisp [10], in which the body of a function can be evaluated in parallel with its argument, synchronizing upon the parameter variable.

One interesting fact about this viewpoint of *sharing vs copying*, is that in the sharing translation, we obtain what can be understood as call-by-value and call-by-need as two extremal execution traces of the resulting processes (i.e., on one extreme we reduce a function argument all the way to a value, and only after that we begin to reduce the abstraction body – call-by-value; on the other extreme we never reduce the argument until it is actually needed by the abstraction body – call-by-need).

As was previously mentioned, the $(\cdot)^*$ translation was discovered to induce a call-by-value reduction strategy at the linear λ -calculus level [15]. One may then wonder why our $\llbracket \cdot \rrbracket_z^*$ translation (which makes use of $(\cdot)^*$) is “looser”, in some sense, given that it does not force arguments to reduce to values before reductions in an abstraction body can take place. The answer to this question lies in the translation of the let $!u = M$ in N construct, since it does not guard the processes in the image of the translation of M or N , allowing for reductions to take place in both. A careful look at the terms in the image $(\cdot)^*$ reveals that the β redex $(\lambda x.M) N$ translates (after some administrative reduction steps) to let $!u = N^*$ in M^* , explaining the apparent discrepancy mentioned above.

Moreover, the logical basis of our approach provides the following property, relating type inhabitation in the λ -calculus and the session-typed π -calculus.

Theorem 2.19. *For any type T in the simply typed λ -calculus, T is inhabited iff $[T]$ and T^* are inhabited.*

Proof. The forward direction follows from Theorems 2.17 and 2.18, respectively, discarding the proof terms. The backward direction follows from the correctness of the logical translations, which are standard in the literature (the process to λ -term translations can be straightforwardly extracted from these proofs).

3 Soundness and Completeness

The correctness results we have presented are related to the well-formedness (through typing) of the terms in the image of each translation. We will now present a soundness and completeness result for the dynamic behavior of the translations. Our translation from the linear λ -calculus to process calculus is looser than one might initially expect,

that is, it allows for reductions that are not typical in the standard operational semantics of the linear λ -calculus. This means that w.r.t the standard operational semantics of the linear λ -calculus, our translation should be able to simulate all reductions, but the converse cannot possibly hold. We now make this remark precise by first stating the typical operational semantics for the linear λ -calculus and showing a completeness result for our translation (i.e., reductions in λ -terms can be matched by a small sequence of reductions in the processes in the image of the translation). We then extend the operational semantics with a rule that allows for reductions under the let-binder, showing the new operational semantics to be observationally equivalent to the standard reduction rules in a precise sense. With this extended rule, we show soundness of our translation with respect to the operational semantics.

Definition 3.1 (Linear λ -calculus Operational Semantics)

$$\frac{M \longrightarrow M'}{M N \longrightarrow M' N} \quad \frac{}{(\lambda x.M) N \longrightarrow M\{N/x\}}$$

$$\frac{M \longrightarrow M'}{\text{let } !u = M \text{ in } N \longrightarrow \text{let } !u = M' \text{ in } N} \quad \frac{}{\text{let } !u = !M \text{ in } N \longrightarrow N\{M/u\}}$$

To accurately state the desired theorems, we need to characterize the relation between the substitutive behavior on the λ -calculus, where variables are substituted by λ -terms, and the π -calculus, where names are substituted by names. The idea is that instead of plugging in a term for a variable, we can think of names as “references” to a term (a channel name along which we communicate with a parallel process that encodes a given term). More precisely, we want to capture the “equivalence” of $\llbracket M\{N/x\} \rrbracket_z$ and $(\nu x)(\llbracket M \rrbracket_z \mid \llbracket N \rrbracket_x)$, where x is a linear variable in M . For the unrestricted case, a similar equivalence is required, but using replicated processes. We combine these two key insights in the following relation.

Definition 3.2 (Substitution Lifting). *For a given linear λ -term M and process P , we say that P lifts the substitutions of M , written $M \triangleleft P$, if $P \equiv (\nu x_i, u_i, y_i)(\llbracket R \rrbracket_z \mid \llbracket W_i \rrbracket_{x_i} \mid !u_i(x_i). \llbracket E_i \rrbracket_x)$ and $M = R\{W_i/x_i, E_i/y_i\}$.*

Intuitively, the relation above holds if P can be decomposed in such a way that we obtain the parallel composition of $\llbracket R \rrbracket_z$, which is the translation of a λ -term R on which no substitutions are actually carried out, and possibly many other processes that encode the λ -terms that are to be substituted into R (either linearly or in an unrestricted manner), to obtain the given term M . We now state the following completeness result for our translation of the linear λ -calculus.

Theorem 3.3 (Completeness of Translation). *If $\Gamma; \Delta \vdash M : A$ and $M \longrightarrow N$ then $\llbracket M \rrbracket_z \rightarrow^* P$ such that $N \triangleleft P$.*

As mentioned above, the processes in the image of our translation induce more non-determinism in the operational semantics of the linear λ -calculus. In particular, the process resulting from the translation of the term $\text{let } !u = M \text{ in } N$ allows for reductions to take place in both the translation of M and N , while the operational semantics of Def. 3.1 do not. However, these extra reductions do not change the observable outcome of a

λ -term, in a very precise sense. Consider the following operational semantics, consisting of the rules of of Def. 3.1 extended with an additional rule that allows for reductions under the let-binder:

Definition 3.4 (Extended Operational Semantics)

$$\frac{M \longrightarrow M'}{M \mapsto M'} \quad \frac{N \mapsto N'}{\text{let } !u = M \text{ in } N \mapsto \text{let } !u = M \text{ in } N'}$$

If we consider observables to be values (which is the standard observable in functional languages), the two operational semantics are equivalent in our typed setting. This can be made precise with the following theorem.

Theorem 3.5. $M \longrightarrow^* V$ iff $M \mapsto^* V$, where: $V ::= !M \mid \lambda x : T.M$

Proof. The left to right direction is immediate. From right to left, the idea is that we can delay all the reductions of N in the additional rule until after M reduces to a term of the form $!M$ and the substitution into N takes place.

Theorem 3.6 (Soundness of Translation). If $\llbracket M \rrbracket_z \rightarrow P$, with $\Gamma; \Delta \vdash M : A$ then there is Q and N such that $P \rightarrow^* Q$, $M \mapsto N$ and $N \triangleleft Q$

Given these two results, we can see that sharing and copying reduction arises from the typing discipline enforced by each translation, given the operational semantics for the linear λ -calculus considered above. This validates the translation from the linear λ -calculus to the π -calculus, and given that the translations from the λ to the linear λ -calculus have been shown to be sound and complete w.r.t the operational semantics in [15], these results extend to our full translations.

4 Concluding Remarks

There is a substantial body of work on the connection of functional and concurrent programming, both from the concurrency and proof theory communities. Milner [17] developed two embeddings of the λ -calculus in the π -calculus, corresponding to call-by-value and call-by-name. His focus was mainly operational, while ours is strictly driven by the logical considerations we have alluded to previously. Along these lines, Sangiorgi and Walker [19] provide a uniform presentation of encodings for call-by-value (in parallel or non parallel form), as well as call-by-need, but do not address sharing versus copying as we do here. Boudol's [5] work on the λ -calculus with multiplicities (arising from the study of Milner's call-by-name embedding), consists of a λ -calculus with inherently non-deterministic and parallel constructs for function application, drawing inspiration on many ideas from linear logic but not studying the deeper connections of the two systems. Yoshida et al. have studied sequentiality in the π -calculus [3] by introducing a typing discipline that allows for direct interpretations of call-by-value and call-by-name, presenting a full abstraction result for an encoding of PCF and connections to game semantics. This contrasts with our work which focuses on the concurrency and parallelism that can be extracted from π -calculus encodings,

instead of using the π -calculus as a tool for the study of sequential computation. Furthermore, their work does not explore connections with logic.

On the proof theory side, the connection of linear logic and concurrency has been explored in some detail, using proof net-like structures. Mazza's work on Multiport Interaction Nets [16], which are a non-deterministic extension of Interaction Nets [13], shows that these are a model of concurrent computation by encoding the full π -calculus. Faggian and Piccolo [7] perform a similar development for Ludics and the finitary linear π -calculus. A correspondence between polarised proof-nets and an IO-typed π -calculus is given in [11]. The line of work summarized above is quite different from ours, in that it is mostly concerned with semantic models for concurrency, derived from linear logic.

In work much closer to our own, Beffara [2] presents embeddings of the λ -calculus and a restricted version the λ_{μ} -calculus in a π -calculus derived from an interpretation of second-order classical linear logic. The λ -calculus translation is closely related to our copying translation. This work provides a family of call-by-value and by-name translations, in both classical and intuitionistic settings, and explores the duality of both reduction strategies by exploring the dualities of classical (linear) logic, while our work focuses on the issues of sharing and copying that can be derived from translations in the purely intuitionistic (and non-higher order) setting.

Finally, the work on Multilisp [10], which is an extension of Lisp with explicit parallelism, pioneered the concept of *futures*, which allow for the concurrent execution of function arguments and bodies. Our sharing reduction strategy captures this concept, by enabling such parallel execution, and thus validates the ideas of Multilisp in a typed setting, motivated by proof theory.

Conclusion and Future Work. We have developed two logically motivated embeddings of the simply-typed λ -calculus in a session-typed π -calculus, using standard techniques for translating natural deduction into sequent calculus and a concurrent interpretation of the latter. The logical foundation provides a clean and canonical account of each step of our translation, relying only on well-established techniques from proof theory and their interpretation as functional programming. We have shown that the two translations we consider induce a copying and a sharing reduction strategy (i.e. parallel evaluation of shared sub-expressions – no implementation of the λ -calculus in the π -calculus in this sense has been proposed before), respectively, providing a logical understanding of these concurrent evaluation strategies. Finally, the typical call-by-value and call-by-need sequential reduction strategies can be understood as two extremal traces of the sharing reduction strategy, where in the former all reductions on arguments are done before those of the function body, and in the latter the evaluation of the argument is postponed until it is needed in the function body. For future work, we plan on investigating further the issues of observational equivalence at the process level and its impact on the translations, as well as a natural generalization to dependently typed languages, using dependent session types.

Acknowledgments. Support for this research was provided by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program, under grants SFRH / BD / 33763 / 2009, INTERFACES NGN-44 / 2009, and CITI.

References

1. Barber, A., Plotkin, G.: Dual Intuitionistic Linear Logic. Technical Report LFCS-96-347. Univ. of Edinburgh (1997)
2. Beffara, E.: Functions as proofs as processes. CoRR, abs/1107.4160 (2011)
3. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the π -Calculus. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 29–45. Springer, Heidelberg (2001)
4. Boreale, M.: On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science* 195(2), 205–226 (1998)
5. Boudol, G.: The lambda-calculus with multiplicities. Technical report, INRIA (1993)
6. Caires, L., Pfenning, F.: Session Types as Intuitionistic Linear Propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010)
7. Faggian, C., Piccolo, M.: Ludics is a Model for the Finitary Linear pi-Calculus. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 148–162. Springer, Heidelberg (2007)
8. Girard, J.-Y.: Linear logic. *Theor. Comput. Sci.* 50, 1–102 (1987)
9. Gonthier, G., Abadi, M., Lévy, J.-J.: The geometry of optimal lambda reduction. In: POPL 1992, Proceedings the Nineteenth Annual ACM Symposium on Principles of Programming Languages, pp. 15–26 (1992)
10. Halstead Jr., R.H.: Multilisp: a language for concurrent symbolic computation. *ACM Trans. Program. Lang. Syst.* 7, 501–538 (1985)
11. Honda, K., Laurent, O.: An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theor. Comp. Sci.* 411, 2223–2238 (2010)
12. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
13. Lafont, Y.: Interaction nets. In: POPL 1990, pp. 95–108. ACM, New York (1990)
14. Lamping, J.: An algorithm for optimal lambda calculus reduction. In: POPL 1990, Proceedings the Seventeenth Annual ACM Symposium on Principles of Programming Languages, pp. 16–30 (1990)
15. Maraist, J., Odersky, M., Turner, D.N., Wadler, P.: Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Electr. Notes Theor. Comput. Sci.* 1, 370–392 (1995)
16. Mazza, D.: Multiport Interaction Nets and Concurrency. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 21–35. Springer, Heidelberg (2005)
17. Milner, R.: Functions as processes. *Math. Struct. in Comp. Sci.* 2(2), 119–141 (1992)
18. Sangiorgi, D.: Pi-Calculus, Internal Mobility, and Agent Passing Calculi. *Theoretical Computer Science* 167(1&2), 235–274 (1996)
19. Sangiorgi, D., Walker, D.: The π -calculus: A Theory of Mobile Processes. Cambridge University Press (2001)
20. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: PPDP 2011, pp. 161–172. ACM (2011)
21. van Bakel, S., Vigliotti, M.G.: A Logical Interpretation of the λ -Calculus into the π -Calculus. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 84–98. Springer, Heidelberg (2009)
22. Wadler, P.: A Syntax for Linear Logic. In: Brookes, S., Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1993. LNCS, vol. 802, pp. 513–529. Springer, Heidelberg (1994)
23. Wadsworth, C.: Semantics and Pragmatics of the Lambda Calculus. PhD thesis, Oxford University (1971)

Deriving Bisimulation Congruences for Conditional Reactive Systems*

Mathias Hülsbusch and Barbara König

Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany

Abstract. We consider conditional reactive systems, a general abstract framework for rewriting, in which reactive systems à la Leifer and Milner are enriched with (nested) application conditions. We study the problem of deriving labelled transitions and bisimulation congruences from a reduction semantics. That is, we synthesize interactions with the environment in order to obtain a compositional semantics. Compared to earlier work we not only address the problem of deriving information about the (minimal) context needed to obtain a full left-hand side and thus be able to perform a reduction, but also generate conditions on the remaining context.

1 Introduction

Given the reduction semantics of a process calculus, it is often hard to define a labelled transition system in such a way that the resulting bisimilarity is a congruence, in order to obtain a compositional semantics. That is, we want to ensure that subsystems can be replaced by behaviourally equivalent subsystems, without changing the overall behaviour. In [15][14] Leifer and Milner showed how to generate such labels in the general framework of reactive systems (an abstract setting for rewriting), using the notion of idem pushout squares (IPOs). This idea has been further developed by other authors [8][20][7]. The underlying idea is simply to label transitions with the minimal context required by a process/term to perform a transition. That is, using the standard example, a CCS process $a.P$ can do a move $a.P \xrightarrow{|\bar{a}.Q} P \mid Q$, with the meaning that the context $- \mid \bar{a}.Q$ is provided by the environment.

Since the existence and derivation of IPOs is a non-trivial task (due to tricky automorphism problems), Sassone and Sobociński studied the definition of IPOs in a bicategorical setting [20] (so-called groupoidal idem pushouts). Inspired by this line of work we have adapted the approach to label derivation for graph transformation systems [3][4] (borrowed contexts), for which no encompassing theory of labelled transitions and bisimulation was available until this point.

The overall approach works fine in a setting where left-hand sides are simply replaced by right-hand sides, but no extra requirements on the surrounding context are made. Such requirements or application conditions not only require the existence of a left-hand side, but ask for the presence or (more importantly) absence of certain

* Supported by the DFG Project Behaviour-GT.

components or items in the remaining system. The paper [15] introduced reactive systems together with the notion of reactive contexts, which allow to impose some limited conditions on the context, but reactive contexts are not dependent on the rule and furthermore they have to satisfy the fairly strict requirement that the decomposition of a reactive context always yields two reactive contexts.

In graph transformation there is a well-developed theory of nested application conditions [19,9]. While such conditions do not seem to be strictly necessary for programming formalisms (i.e. the setting where process calculi are often used), they are ubiquitous in specification formalisms, such as languages describing UML model transformations (see for instance [5]). On the other hand, a theory of bisimulation is extremely helpful to show behaviour preservation in model transformation, i.e., to prove that a source model is transformed into a behaviourally equivalent target model. A successful proof strategy is to show that every left-hand side of a transformation rule is bisimilar to the corresponding right-hand side and then rely on the fact that bisimilarity is a congruence. However, the presence of application conditions, especially negative application conditions which destroy monotonicity, are a severe problem, in fact the major problem we had to deal with in a case study where we compared proof techniques for showing semantics preservation in model transformation [11].

Hence we believe that it is important to study the theory of bisimulation congruences in the setting of reactive systems equipped with application conditions, so-called conditional reactive systems which we introduced in [2]. In [18] we already studied bisimulation congruences for graph transformation systems with negative application conditions. The present paper generalizes this in several respects: (i) Generalizing negative application conditions we use nested application conditions; (ii) We work in the more general framework of reactive systems instead of graph transformation systems, which are a specific instance; (iii) Instead of fixing a specific way to derive “minimal” context (via IPOs or borrowed context diagrams as in [4,18]) we define the general (and very simple) notion of representative squares, which leads to the same results (at least for saturated equivalences).

We define a notion of bisimilarity, show that it is a congruence and study it by giving an alternative characterization, which is less practical, but more intuitive. Furthermore we compare with other (different) notions of behavioural equivalence. We will here study saturated bisimilarity (as compared to IPO-bisimilarity), where a transition labelled with a minimal context can be answered by a transition with an arbitrary (possibly non-minimal) context (see [1]).

2 Conditional Reactive Systems

2.1 Reactive Systems with Conditions

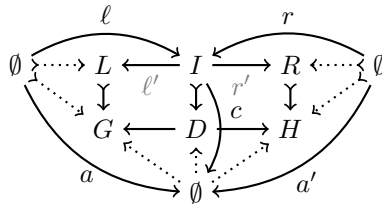
We now define the notion of reactive systems, first introduced in [15,14] for label derivation and the definition of bisimulation congruences.

Definition 1 (Reactive System Rules). Let \mathbb{C} be a category with a distinguished object 0 (not necessarily initial). A reactive system rule is a pair $R = (\ell, r)$ of arrows – called left-hand side and right-hand side respectively – with $\ell, r: 0 \rightarrow I$ for some object I . Let \mathcal{R} be a set of rules. We say that an arrow $a: 0 \rightarrow J$ reduces to $a': 0 \rightarrow J$ with the rules in \mathcal{R} (in symbols: $a \rightsquigarrow_{\mathcal{R}} a'$ or simply $a \rightsquigarrow a'$) if there exists a rule $(r, \ell) \in \mathcal{R}$ with $\ell, r: 0 \rightarrow I$ and an arrow $c: I \rightarrow J$ such that $a = \ell; c$ and $a' = r; c$.¹

In [15] it is additionally required that c is contained in a subcategory of reactive contexts, which, in our setting, will be replaced later by the requirement that c satisfies a given condition.

An important class of reactive systems can be defined over a base category \mathbb{D} which has all pushouts along monos and in which pushouts preserve monos. Then we define as $\mathbb{C} = ILC(\mathbb{D})$ the category which has as objects the objects of \mathbb{D} and as arrows cospans of the form $A \xrightarrow{f} B \xleftarrow{g} C$ (called *input-linear cospans*, because the left arrow f is a mono), where the middle object is taken up to isomorphism. Composition of cospans is performed via pushouts.

In several of the examples we will use as base category the category $\mathbb{D} = \mathbf{Graph}_{\text{fin}}$ which has finite graphs (with node and edge labels) as objects and graph morphisms as arrows. Reactive systems over $ILC(\mathbf{Graph}_{\text{fin}})$ coincide exactly with DPO graph transformation systems with injective matches (see [21]). Consider the figure below: in DPO rewriting a rule is given by a span $L \xleftarrow{\ell'} I \xrightarrow{r'} R$ of arrows in $\mathbf{Graph}_{\text{fin}}$ and a graph G can be transformed into a graph H if we can find a graph D and morphisms such that the inner two squares (drawn with gray arrows) are pushouts. (Intuitively every item of the left-hand side L not contained in I is removed from G by a pushout complement and the right-hand side R is glued to the resulting graph D .) By completing the diagram with empty graphs (corresponding to the distinguished object 0) and the dotted arrows, we obtain two commuting triangles in the cospan category (black arrows), which correspond to the conditions for reactive systems ($a = \ell; c, a' = r; c$). The objects that are rewritten in $\mathbf{Graph}_{\text{fin}}$ are the inner objects of the cospans a and a' .



In the rest of this section we will summarize definitions and results from [2]. We will first define conditions, similar to the presentation in [19,9], as tree-like structures, where nodes are annotated with quantifiers and objects and edges are annotated with arrows.

¹ For arrows $f: A \rightarrow B$ and $g: B \rightarrow C$ we use the notation $f; g$ for their composition, that is $f; g: A \rightarrow C$.

Definition 2 (Conditions). Let \mathbb{C} be a category. A condition (in \mathbb{C}) is a triple $\mathcal{A} = (A, \mathcal{Q}, S)$ where

- A is an object of \mathbb{C} (called the root object of \mathcal{A} or $\text{RO}(\mathcal{A})$),
- \mathcal{Q} is a quantifier (either \forall or \exists) and
- S is a finite set of pairs (\mathcal{A}', f) such that \mathcal{A}' is a condition and $f: A \rightarrow \text{RO}(\mathcal{A}')$ is a \mathbb{C} -arrow.

The pair (\mathcal{A}', f) will be denoted by $A \xrightarrow{f} \mathcal{A}'$. A condition \mathcal{A} can be viewed as a tree, with $\text{RO}(\mathcal{A})$ as the root and edges labelled with arrows.

Definition 3. For all conditions \mathcal{A} , all objects C and all arrows $c: \text{RO}(\mathcal{A}) \rightarrow C$ we define a satisfaction relation as follows:

$c \models (A, \forall, S)$ iff for every $(A \xrightarrow{f} \mathcal{A}') \in S$ and every arrow $\alpha: \text{RO}(\mathcal{A}') \rightarrow C$ such that $f; \alpha = c$ we have $\alpha \models \mathcal{A}'$

$c \models (A, \exists, S)$ iff for some $(A \xrightarrow{f} \mathcal{A}') \in S$ there is an arrow $\alpha: \text{RO}(\mathcal{A}') \rightarrow C$ with $f; \alpha = c$ and $\alpha \models \mathcal{A}'$

In [2] we have shown that if we instantiate \mathbb{C} with $\mathbf{Graph}_{\text{fin}}$ we are equal in expressiveness to first-order logic on graphs. If we instantiate with $ILC(\mathbf{Graph}_{\text{fin}})$ instead we are more expressive, since we are able to express the existence of isolated nodes directly in the logics (even in the presence of infinitely many edge labels).

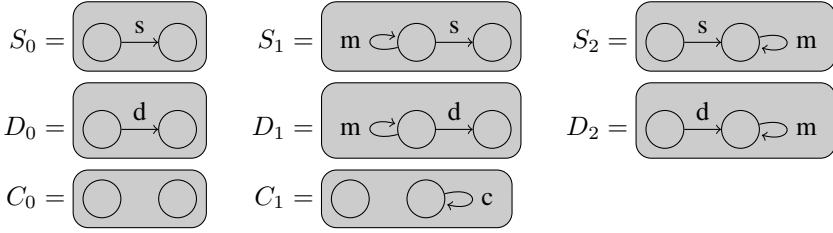
Given two conditions \mathcal{A}, \mathcal{B} with root object C we will in the following write $\mathcal{A} \models \mathcal{B}$ if for every arrow c (with source object C) $c \models \mathcal{A}$ implies $c \models \mathcal{B}$. Furthermore we write $\mathcal{A} \equiv \mathcal{B}$ whenever $\mathcal{A} \models \mathcal{B}$ and $\mathcal{B} \models \mathcal{A}$. Note that for many categories, for instance for $\mathbf{Graph}_{\text{fin}}$ and $ILC(\mathbf{Graph}_{\text{fin}})$ implication and equivalence will be undecidable, a consequence of the undecidability of implication and equivalence in first-order logic.

Now, given a reactive system over \mathbb{C} , it is natural to associate conditions with the target object of left-hand and right-hand sides and to interpret them on the contexts.

Definition 4 (Rules with application conditions). Let \mathbb{C} be a category with a distinguished object 0 . A rule with application condition is a triple (ℓ, r, \mathcal{B}) where $\ell, r: 0 \rightarrow I$ and \mathcal{B} is a condition with root object I . We say that the rule is applicable to $a: 0 \rightarrow J$ whenever $a = \ell; c$ for some $c: I \rightarrow J$ such that $c \models \mathcal{B}$. The result of the rule application is $r; c$. Again we denote by $\rightsquigarrow_{\mathcal{R}}$ (or simply \rightsquigarrow) the rewriting relation induced by a set \mathcal{R} of rules with application conditions.

Example 5. As a running example, we will introduce a simple message transportation protocol in a network of nodes, using duplex as well as simplex connections. In both cases, messages can only be transferred from node a to node b , whenever there is no message on b , waiting to be processed. If b however has a buffer (or extra capacity), it is possible to forward the message to b , even if there are already messages at b . To model this, we choose the following graph representation: Nodes in the network are (unlabelled) nodes in the graph; a simplex connection is a directed s -labelled edge; a duplex connection is a directed edge, labelled d ; a message is an m -loop at that node.

If the node has a buffer, we attach a c -loop. As category we choose $ILC(\mathbf{Graph}_{fin})$. To describe the rules, we first give the following graphs:



Using these graphs, we can now give the set of rules:

$$\begin{aligned}
 PassS &= (\emptyset \rightarrow S_1 \leftarrow S_0, \emptyset \rightarrow S_2 \leftarrow S_0, Cond_{S_0}) \\
 PassD_1 &= (\emptyset \rightarrow D_1 \leftarrow D_0, \emptyset \rightarrow D_2 \leftarrow D_0, Cond_{D_0}) \\
 PassD_2 &= (\emptyset \rightarrow D_2 \leftarrow D_0, \emptyset \rightarrow D_1 \leftarrow D_0, Cond'_{D_0})
 \end{aligned}$$

where conditions are defined as follows, where $true_A = (A, \forall, \emptyset)$:

$$\begin{aligned}
 Cond_{S_0} &= (S_0, \forall, \{((C_0, \exists, \{(true_{C_0}, C_0 \rightarrow C_1 \leftarrow C_0)\}), S_0 \rightarrow S_2 \leftarrow C_0)\}) \\
 Cond_{D_0} &= (D_0, \forall, \{((C_0, \exists, \{(true_{C_0}, C_0 \rightarrow C_1 \leftarrow C_0)\}), D_0 \rightarrow D_2 \leftarrow C_0)\}) \\
 Cond'_{D_0} &= (D_0, \forall, \{((C_0, \exists, \{(true_{C_0}, C_0 \rightarrow C_1 \leftarrow C_0)\}), D_0 \rightarrow D_1 \leftarrow \alpha - C_0)\})
 \end{aligned}$$

All graph morphisms (apart from α) are induced by the edge labels and by the positions of the nodes in the images above. The morphism α instead swaps the two nodes, in order to ensure that the buffer is at the left node of the duplex edge (see rule $PassD_2$). The conditions can be read as follows: In *all* cases, where there is a message at the target node, there must *exist* a c -loop at that node to make the rule applicable.

2.2 Representative Squares

We will now define the notion of representative squares (first introduced in [2]), which describe representative ways to close a span of arrows. Such squares are intimately related to idem pushouts [15] or borrowed context diagrams [4].

Definition 6 (Representative class of squares). A class κ of commuting squares in a category \mathbb{C} is called representative if κ satisfies the following property: for every commuting square of \mathbb{C} (such as the one consisting of f_0, f_1, g_0, g_1 on the left) there exists a square in κ (consisting of f_0, f_1, h_0, h_1) and an arrow $h: D \rightarrow E$ which makes the diagram commute (on the right).



For two arrows $f_0: A \rightarrow B, f_1: A \rightarrow C$ we denote by $\kappa(f_0, f_1)$ the set of pairs (h_0, h_1) of arrows $h_0: B \rightarrow D$ and $h_1: C \rightarrow D$ such that f_0, f_1, h_0, h_1 form a representative square in κ .

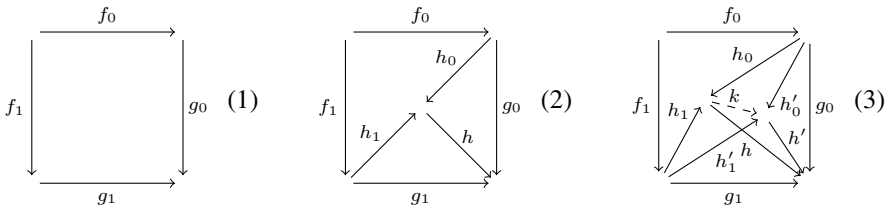
In the following, we fix a representative class κ of squares and we shall call every square in κ *representative*. Also note that the class of all squares of \mathbb{C} is representative. However, we will in the following require that for each pair a, b the set $\kappa(a, b)$ is finite, which means that the constructions described in Section 2.3 are effective since the finiteness of the transformed conditions is preserved. In the following we will discuss possible choices for the class of representative squares.

Pushouts: If we have a category where all pushouts exist, they are the most natural candidate for representative squares and we can define that every class $\kappa(a, b)$ contains only one pair of arrows: a representative pushout. Unfortunately, pushouts do not exist in many categories of interest.

Jointly epi squares: Consider the subcategory of an adhesive category [12] which consists exactly of the mono arrows. Then, the class of all squares a, b, c, d where c, d are jointly epi, is representative.

Idem pushouts: In a category where every commuting square contains an idem pushout (IPO), the IPOs are a representative class of squares. IPOs were introduced in [15] for the purpose of automatically deriving labels, specifying interactions with the environment in reactive systems, and bisimulation congruences.

IPOs are defined as follows: consider a commuting square as shown in (1) below such that $f_0; g_0 = f_1; g_1$. A *relative pushout (RPO)* for this commuting square is a triple h_0, h_1, h satisfying the following two properties: (i) *commutativity*: $f_0; h_0 = f_1; h_1$ and $h_i; h = g_i$ for $i = 0, 1$ (see (2)); (ii) *universality*: for any h'_0, h'_1, h' satisfying $f_0; h'_0 = f_1; h'_1$ and $h'_i; h' = g_i$ there exists a unique mediating arrow k such that $k; h' = h$ and $h_i; k = h'_i$ for $i = 0, 1$ (see (3)).



A commuting square as in Diagram (1) is an *idem pushout (IPO)* if the triple g_0, g_1, id is a relative pushout for the same Diagram (1).

Note that in order to obtain the congruence results of [15] more properties of IPOs than the one of Definition 6 are required.

Borrowed context squares: The category $ILC(\mathbb{D})$, where \mathbb{D} is adhesive, is of special interest for the construction of IPOs, since it integrates nicely with double-pushout (graph) rewriting as shown above. Unfortunately, this category does not have IPOs, due to automorphism problems. One solution is to switch to a bicategorical setting [21,20], another – that we are following here – is to give up the characterizations of the squares via a universal property and concentrate on the relevant properties.

The borrowed context diagrams introduced in [4] (with the extensions introduced in [21]) can be seen as GIPOs (groupoidal pushouts) in a bicategory and they are also representative squares in our sense. A commuting diagram in the cospan category is a borrowed context diagram if and only if it has the form of the diagram on the right, where the left upper square is jointly epi (j.e.), the right lower square is a pullback and the two remaining squares are pushouts: The idea behind these borrowed context diagrams is exactly to close a span of cospans in all minimal representative ways. If we assume that the right leg of all cospans is the identity, then we are in the sub-case treated above, where we consider jointly epi squares.

$$\begin{array}{ccccc}
 \emptyset & \twoheadrightarrow & L & \longrightarrow & I \\
 \downarrow & & \text{j.e.} & \downarrow & \text{PO} & \downarrow \\
 G & \twoheadrightarrow & G^+ & \longleftarrow & C \\
 \uparrow & & \text{PO} & \uparrow & \text{PB} & \uparrow \\
 J & \twoheadrightarrow & F & \longleftarrow & K
 \end{array}$$

2.3 Operations on Conditions

We continue to recapitulate concepts and results from [2]: first we define boolean operations on conditions.

Definition 7 (Boolean Operations on Conditions)

Constants: For an object A define $\text{false}_A := (A, \exists, \emptyset)$, $\text{true}_A := (A, \forall, \emptyset)$.

Negation: For a condition of the form (A, \mathcal{Q}, S) with $\mathcal{Q} \in \{\forall, \exists\}$ we define:

$$\neg(A, \forall, S) := (A, \exists, \{A \xrightarrow{f} \neg A' \mid (A \xrightarrow{f} A') \in S\})$$

$$\neg(A, \exists, S) := (A, \forall, \{A \xrightarrow{f} \neg A' \mid (A \xrightarrow{f} A') \in S\})$$

Conjunction and Disjunction: For two conditions \mathcal{A}, \mathcal{B} with root object C we define:

$$\mathcal{A} \wedge \mathcal{B} := (C, \forall, \{C \xrightarrow{\text{id}_C} \mathcal{A}, C \xrightarrow{\text{id}_C} \mathcal{B}\}) \quad \mathcal{A} \vee \mathcal{B} := (C, \exists, \{C \xrightarrow{\text{id}_C} \mathcal{A}, C \xrightarrow{\text{id}_C} \mathcal{B}\})$$

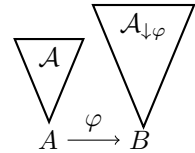
The boolean operations satisfy the usual laws of propositional logic.

One central operation is the shift of a condition along an arrow (see also [17]). Intuitively a shift corresponds to a partial evaluation, where we assume that the arrows on which the condition is to be evaluated are of the form $\varphi; c$ for a fixed φ .

Definition 8 (Shift of a Condition). Given a fixed set κ of representative squares, the shift $\mathcal{A}_{\downarrow\varphi}$ of a condition $\mathcal{A} = (A, \mathcal{Q}, S)$ along an arrow $\varphi: A \rightarrow B$ is inductively defined as follows:

$$\mathcal{A}_{\downarrow\varphi} = (B, \mathcal{Q}, \{(B \xrightarrow{g} \mathcal{A}'_{\downarrow\psi}) \mid (A \xrightarrow{f} \mathcal{A}') \in S, (\psi, g) \in \kappa(f, \varphi)\})$$

That is we perform a kind of partial evaluation on the condition. In the following we will visualize conditions by triangles and denote a shift as depicted on the right. We will now give some properties of the shift operator.



Proposition 9 (Shift [2]). Given two arrows $\varphi: A \rightarrow B$ and $c: B \rightarrow C$, and a condition \mathcal{A} with root object A , the following holds:

$$\varphi; c \models \mathcal{A} \iff c \models \mathcal{A}_{\downarrow\varphi}$$

As a consequence shift satisfies the following laws:

$$\mathcal{A}_{\downarrow\varphi; \psi} \equiv (\mathcal{A}_{\downarrow\varphi})_{\downarrow\psi} \quad (\mathcal{A} \wedge \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \wedge \mathcal{B}_{\downarrow\varphi} \quad (\mathcal{A} \vee \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \vee \mathcal{B}_{\downarrow\varphi}$$

2.4 Finiteness Assumptions

In the rest of the paper we will make the following two finiteness assumptions, which hold for many categories, for instance for our example category $ILC(\mathbf{Graph}_{\text{fin}})$:

- (i) for every pair of arrows a, b $\kappa(a, b)$ is a finite set;
- (ii) for two arrows a, c with the same domain, there are only finitely many arrows b such that $a; b = c$.

Note that for (left-linear) cospans over graphs Condition (i) can be enforced by taking borrowed context diagrams as representative squares. Furthermore Condition (ii) holds for cospans (over finite graphs), since the middle object is taken up to isomorphism.

3 Label Derivation and Saturated Bisimilarity

Our aim is now to define behavioural congruences on reactive systems. As observed by many authors before (e.g., in [15]), defining bisimulation relations on the reduction semantics introduced in Section 2 is insufficient for a compositional semantics, since in general the resulting equivalence will not be congruence. It is easy to construct an example involving two arrows a, b which both can not perform a step, whereas a can do a step when put into a suitable context c , but b can not.

Hence, it is necessary to incorporate potential interactions with the environment. We will study this first for reactive systems without conditions, i.e., we will summarize results from [15,1] with the new contribution that the IPO squares used in these papers are replaced by the conceptually simpler representative squares.

We will start by defining *context transitions*, which describe that a component a can do a step (and evolve to b) whenever the environment provides a context f . *Representative transitions* are those context transitions that are generated by representative squares.

Definition 10 (Context transitions, representative transitions). *Let \mathcal{R} be a set of reactive system rules. Let $a: 0 \rightarrow I$, $f: I \rightarrow J$, $a': 0 \rightarrow J$. We write $a \xrightarrow{f}_C a'$ whenever $a; f \rightsquigarrow a'$, i.e. if there exists a rule $(\ell, r) \in \mathcal{R}$ such that $a' = r; c$ and $a; f = \ell; c$ for some arrow c . Such transitions are called context transitions. Furthermore we write $a \xrightarrow{f}_R r; c$ when in addition $(f, c) \in \kappa(a, \ell)$. We call such transitions representative transitions.*

From the definitions it follows immediately that $a \xrightarrow{f}_R a'$ implies $a \xrightarrow{f}_C a'$. Here we are not treating the case of simulating \xrightarrow{f}_R -transitions by \xrightarrow{f}_R -transitions, which leads to a notion of equivalence analogous to IPO-bisimilarity. We take the position that it should not be observable whether a context is minimal (resp. representative) or not, leading to saturated bisimilarity defined below. Furthermore IPO-bisimilarity admits, to our knowledge, no interesting alternative characterization (as in Section 5), different from saturated bisimilarity. We state here, for completeness, that bisimilarity on representative transition does not necessarily give rise to a congruence: it would be necessary to impose some extra conditions on representative squares (such as the composition and decomposition properties of IPOs established in [15]).

Instead we concentrate on saturated semantics, requiring that every \xrightarrow{f}_C -step is matched by a \xrightarrow{f}_R -step. Under normal circumstances, this notion is impractical, since the resulting labelled transition system is infinitely branching. However, it can be easily shown that saturated semantics coincides with semi-saturated bisimilarity where every \xrightarrow{f}_R -step is matched by a \xrightarrow{f}_C -step. Since in many application areas of interest we can guarantee that $\kappa(a, b)$ is finite for every pair of arrows a, b , the transition relation \xrightarrow{f}_R is finitely branching and hence amenable to mechanization.

Definition 11 (Saturated/semi-saturated bisimilarity). *Let \mathcal{R} be a set of reactive system rules. Let R be a symmetric relation, which relates pairs of arrows a, b with source object 0 and identical target object. We say that R is a saturated bisimulation if whenever $a R b$ and $a \xrightarrow{f}_C a'$, then there exists b' such that $b \xrightarrow{f}_C b'$ and $a' R b'$. Two arrows a, b are called saturated bisimilar ($a \sim_{SAT} b$) whenever there exists a saturated bisimulation R with $a R b$.*

A relation R is a semi-saturated bisimulation if whenever $a R b$ and $a \xrightarrow{f}_R a'$, then there exists b' such that $b \xrightarrow{f}_C b'$ and $a' R b'$. Two arrows a, b are called semi-saturated bisimilar ($a \sim b$) whenever there exists a semi-saturated bisimulation R with $a R b$.

Theorem 12 ([1])

1. Saturated and semi-saturated bisimilarity coincide, i.e., for two arrows a, b we have $a \sim_{SAT} 0b \iff a \sim b$.
2. Furthermore saturated bisimilarity is a congruence, i.e., whenever we have $a, b: 0 \rightarrow I$ with $a \sim_{SAT} b$ and $c: I \rightarrow J$, then $a; c \sim_{SAT} b; c$.
3. Finally, saturated bisimilarity is the coarsest bisimulation on \rightsquigarrow that is also a congruence.

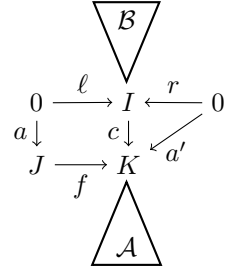
The theorem is due to [1], however observe that we here employ representative squares, different from the IPOs used in [1]. The main aim of this paper is to extend the theorem to deal with conditions and to establish an analogous (but more complex) result in the setting of conditional reactive systems, which generalizes Theorem [12]. For this we will first define a notion of labelled transitions and of (semi-)saturated bisimilarity for conditional reactive systems.

4 Bisimilarity for Conditional Reactive Systems

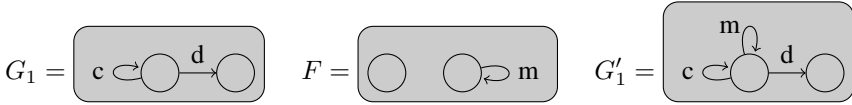
We will first define notions of context and representative transitions in the presence of (application) conditions.

Definition 13 (Context transitions, representative transitions (with application conditions)). *Let \mathcal{R} be a set of reactive system rules with conditions. Let $a: 0 \rightarrow J$, $f: J \rightarrow K$, $a': 0 \rightarrow K$ and \mathcal{A} be a condition with root object K . We write $a \xrightarrow{f, \mathcal{A}}_C a'$ whenever there exists a rule $(\ell, r, \mathcal{B}) \in \mathcal{R}$ such that $a; f = \ell; c$, $a' = r; c$ and $\mathcal{A} \models \mathcal{B}_{\downarrow c}$ for some arrow c . Furthermore we write $a \xrightarrow{f, \mathcal{A}}_R a'$ when in addition $(f, c) \in \kappa(a, \ell)$ and $\mathcal{A} = \mathcal{B}_{\downarrow c}$.*

The figure to the right visualizes this situation. Note that in the case of a context transition the square simply commutes, whereas in case of a representative transition it must be a representative square. Intuitively $a \xrightarrow{f, \mathcal{A}}_C a'$ means that a can be transformed to a' whenever the environment provides or “borrows” f and furthermore $a; f$ is put into a *passive* context (i.e. a context not participating in the reduction) satisfying \mathcal{A} . On the other hand $a \xrightarrow{f, \mathcal{A}}_R a'$ means that the context f is not arbitrary, but one of the representative contexts required for the move. In addition \mathcal{A} is the weakest possible requirement on the context. Clearly $a \xrightarrow{f, \mathcal{A}}_R a'$ implies $a \xrightarrow{f, \mathcal{A}}_C a'$.



Example 14. To illustrate one label derivation, in this case for a representative transition (using borrowed context diagrams as the class of representative squares), we need the following graphs:



We consider rule *PassD₂* (see Example 5) and the arrow $a = \emptyset \rightarrow G_1 \leftarrow C_0$ and take a borrowed context diagram where the two graphs G_1, D_2 overlap on the d -edge (there are other possible representative squares). In this case we obtain as label f (which describes the context to be borrowed) a cospan with F as the center graph (i.e., we borrow a message on the right node). Furthermore we derive the condition $\mathcal{A} = (Cond'_{D_0})_{\downarrow c} \equiv true_{C_0}$. This reflects the fact, that a message at the right node can always be transferred to the left node, since this node has unbounded capacity. In addition, the resulting cospan a' has graph G'_1 as center graph.

Definition 15 (Saturated/semi-saturated bisimilarity (with application conditions)). Let \mathcal{R} be a set of reactive system rules with conditions. Let R be a symmetric relation, which relates pairs of arrows a, b with source object 0 and identical target object. We say that R is a saturated bisimulation if whenever $a R b$ and $a \xrightarrow{f, \mathcal{A}}_C a'$, then there exist arrows b'_1, \dots, b'_n and conditions $\mathcal{A}_1, \dots, \mathcal{A}_n$ such that $b \xrightarrow{f, \mathcal{A}_i}_C b'_i$ with $a' R b'_i$ for all $i \in \{1, \dots, n\}$ and $\mathcal{A} \models \bigvee_{i=1}^n \mathcal{A}_i$. Two arrows a, b are called saturated bisimilar ($a \sim_{SAT} b$) whenever there exists a saturated bisimulation R with $a R b$.

A relation R is a semi-saturated bisimulation if in the definition above $a \xrightarrow{f, \mathcal{A}}_C a'$ is replaced by $a \xrightarrow{f, \mathcal{A}}_R a'$. Two arrows a, b are called semi-saturated bisimilar ($a \sim b$) whenever there exists a saturated bisimulation R with $a R b$.

We will motivate why several answering moves are allowed: whenever the first partner makes a move, the second partner may have the chance to simulate this move, but the used rule may depend on the context. For instance, assume that we want to show that a single A -edge is bisimilar to a single B -edge under the presence of the following rules: the A -edge can be (unconditionally) deleted, but there are two different deletion rules

for the B -edge, one that requires the presence of C edge and another that forbids the presence of a C -edge.

Observe that a move of a with condition $\mathcal{A} \equiv \text{false}$ can always be mimicked by b by doing nothing, that is, an empty set of answering moves is allowed.

Note also, that in the definition of semi-saturated bisimulation above, in the answering moves of the form $b \xrightarrow{f, \mathcal{A}_i} b'_i$ we can always assume that \mathcal{A}_i is the weakest possible condition, i.e., \mathcal{A}_i is obtained by shifting the rule condition over the context and is hence of the form $\mathcal{A} = \mathcal{B}_{\downarrow c}$ for some \mathcal{B}, c .

Furthermore it is straightforward to show that the relations \sim, \sim_{SAT} themselves are also (semi-)saturated bisimulations.

Lemma 16. *Let $a: 0 \rightarrow I, f: I \rightarrow J, a': 0 \rightarrow J$ and \mathcal{A} be a condition with root object J . In addition let d, c', f' be arrows with $d; f' = f; c'$. Then $a \xrightarrow{f, \mathcal{A}} a'$ implies $a; d \xrightarrow{f', \mathcal{A}_{\downarrow c'}} a'; c'$. As a special case (if $d = id$ and $f' = f; c'$) we obtain that $a \xrightarrow{f, \mathcal{A}} a'$ implies $a \xrightarrow{f; c', \mathcal{A}_{\downarrow c'}} a'; c'$.*

We will first show that semi-saturated bisimilarity is a congruence. It is easier to show that saturated bisimilarity is a congruence (and the two coincide anyway), but we need this result for the proof of Theorem 18.

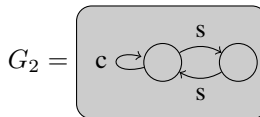
Theorem 17 (Congruence). *Semi-saturated bisimilarity is a congruence, i.e., whenever we have $a, b: 0 \rightarrow I$ with $a \sim b$ and $c: I \rightarrow J$, then $a; c \sim b; c$.*

Note that we would not obtain a congruence result if we omitted conditions from the labels: again, it is easy to think of an example with two arrows a, b where a contains a left-hand side, but can not perform the step, since the condition requires the presence of another item. On the other hand, b can not perform a reduction at all. Then both would be bisimilar if we do not take conditions into account, but by putting them into a context containing the item required by a we would obtain a pair of non-bisimilar arrows. As a next step we show that saturated and semi-saturated bisimilarity coincide.

Theorem 18 (Saturated vs. semi-saturated bisimilarity). *Saturated and semi-saturated bisimilarity coincide, i.e., for two arrows a, b we have:*

$$a \sim_{SAT} b \iff a \sim b.$$

Example 19. Using the techniques presented in this paper, we can now show that the cospans $a = \emptyset \rightarrow G_1 \leftarrow C_0$ (see Example 14) and $b = \emptyset \rightarrow G_2 \leftarrow C_0$ (for G_2 shown below) are (saturated) bisimilar.



It is also possible to prove that a duplex connection can be mimicked by two simplex connections and vice versa, if the c -loops are placed alike in both cases.

5 An Alternative Characterization

In order to characterize the equivalence that we have obtained, we will give an alternative definition as the coarsest bisimilarity for a certain transition system that is also a congruence.

Definition 20 (Environment transition). *Let \mathcal{R} be a set of reactive system rules with conditions. Let $a: 0 \rightarrow I$, $a': 0 \rightarrow I$, $d: I \rightarrow J$. We write $a \overset{d}{\rightsquigarrow} a'$ whenever there exists a rule $(\ell, r, \mathcal{B}) \in \mathcal{R}$ and an arrow c such that $a = \ell; c$, $a' = r; c$ and $c; d \models \mathcal{B}$.*

Note that $\overset{d}{\rightsquigarrow}$ -transitions are infinitely branching. Intuitively a $\overset{d}{\rightsquigarrow}$ -transition is possible if a rule can be applied under a *passive context* d that does not participate in the reduction. $a \overset{d}{\rightsquigarrow} a'$ implies $a; d \rightsquigarrow a'; d$, but the reverse does not necessarily hold, since the right-hand reduction may consume (and recreate) parts of d .

Definition 21 (Environment bisimilarity). *Let \mathcal{R} be a set of reactive system rules with conditions. Let R be a symmetric relation, which relates pairs of arrows a, b with source object 0 and identical target object. We say that R is an environment bisimulation if whenever $a \overset{d}{\rightsquigarrow} a'$, then $b \overset{d}{\rightsquigarrow} b'$ and $a' R b'$ for some arrow b' . We denote by \sim^e (environment congruence) the coarsest relation that is a congruence and an environment bisimulation.*

In order to show that saturated bisimilarity and environment congruence coincide, we first need the following lemma.

Lemma 22. *Let $a: 0 \rightarrow I$, $f: I \rightarrow J$, $a': 0 \rightarrow J$, $d: J \rightarrow K$. Let \mathcal{A} be a condition with root object J such that $a \overset{f; \mathcal{A}}{\rightarrow}_C a'$ and $d \models \mathcal{A}$. Then we have $a; f \overset{d}{\rightsquigarrow} a'$.*

In the other direction if $a; f \overset{d}{\rightsquigarrow} a'$, then there exists a condition \mathcal{A} with $a \overset{f; \mathcal{A}}{\rightarrow}_C a'$ and $d \models \mathcal{A}$. More precisely there exists a rule (ℓ, r, \mathcal{B}) and an arrow c with $a; f = \ell; c$, $a' = r; c$ and $\mathcal{A} = \mathcal{B}_{\downarrow c}$.

We will now show that the natural, but impractical, notion of environment congruence is equivalent to (semi-)saturated bisimilarity, which is more amenable to mechanization.

Theorem 23 (Saturated bisimilarity vs. environment congruence). *Saturated bisimilarity (and hence also semi-saturated bisimilarity) and environment congruence coincide, i.e., for two arrows a, b we have $a \sim_{SAT} b \iff a \sim^e b$.*

Note that the notion of environment congruence and hence also of (semi-)saturated bisimilarity is entirely independent on the notion of representative squares. This allows to choose any suitable class of representative squares in implementations or proofs.

Our new results generalize the results of Section 3 if we assume that all application conditions for rules are true, the condition \mathcal{A} in a transition will also always be true and hence (semi-)saturated bisimilarity with conditions specializes to (semi-)saturated bisimilarity without conditions. Furthermore, whenever an arrow a can do a \rightsquigarrow -step without conditions, it can do an environment transition $\overset{d}{\rightsquigarrow}$ for every (composable) arrow d . Hence environment bisimilarity is simply the coarsest relation which is a congruence and a bisimilarity.

6 Comparison

There is another equivalence to which we could naturally compare: the coarsest congruence that is a bisimulation wrt. $\overset{id}{\rightsquigarrow}$ -steps (equivalently \rightsquigarrow -steps), i.e., the bisimulation where we consider only reduction steps. Clearly, (semi-)saturated bisimilarity (\sim) is finer than this equivalence, here we show that the inclusion is strict by means of two examples. In both cases we work in the category $ILC(\mathbf{Graph}_{fin})$ and assume node-labelled graphs.

Example 24. Consider two rules: one replaces a node labelled B by a node labelled D , but only if an A -labelled node is present (rule R_1). The other (rule R_2) replaces two nodes labelled A and C by two nodes labelled A and D (i.e., here node A is deleted and recreated).

Now we consider as cospans $b = \emptyset \rightarrow G_B \leftarrow \emptyset$ the graph with two empty interfaces consisting simply of a B -labelled node, similarly we define $c = \emptyset \rightarrow G_C \leftarrow \emptyset$. Clearly b, c are not saturated bisimilar: b can perform a transition $b \xrightarrow{id, A} R$ where A requires the presence of an A -node, whereas c can not answer with context id . In other words b can reduce under an empty environment, while c can not. Both graphs differ wrt. their consumption of items provided by the environment.

On the other hand if we close the pair (b, c) wrt. all possible contexts (and take the union with the identity relation) we obtain a congruence that is also a bisimulation. It is a congruence by definition and it is a bisimilarity: if either the B -node or the C -node is replaced by a D -node, then an A -node is present in the environment, which means that the step can be mimicked by the respective partner.

The two equivalences also differ concerning the notion of an environment which may change (independently of the system rules).

Example 25. Now keep rule R_1 above, but replace R_2 by rule R_3 where a C -labelled node is replaced by an F -labelled node if an A -node is present. Furthermore imagine another rule (rule R_4), which replaces a node labelled D by a node labelled E , but only if *no* A -labelled node is present. Now use R_1, R_3 and R_4 as the only rules.

Again, consider the two cospans b, c above. They are not saturated bisimilar, since b can do two environment steps (where the first step demands the presence of an A -node and the second its absence), whereas c can do only one. This means that we take into account that some outside entity might change the environment without using the predefined reduction rules.

However, if we close the pair (b, c) wrt. all possible contexts and furthermore close the pair of graphs consisting of a single D -node and a single F -node under all contexts containing A , we obtain a bisimulation that is also a congruence. It is a bisimulation since neither b nor c can do a step (if no A -node is present), or – if they can do a step – we are sure that there exists an A in the environment. However, in this case we end up in the second part of the bisimulation, where no further moves are possible.

This means that our notion of equivalence is motivated by the following philosophy: we can observe whether an item is consumed or simply required by a rule. And second, we do not trust that the environmental context is only influenced by the reduction rules, but assume that there could be other modifications by an outside entity.

7 Conclusion

We have shown how to generate bisimulation congruences in a rule-based formalism with application conditions. We are aware of only few papers that explicitly add conditions or restrict contexts in defining behavioural equivalences. In [10] a symbolic bisimulation is studied, where bisimulation is a parametrized relation: two processes may not always be bisimilar, but only bisimilar under certain restrictions. A similar parametrization is investigated in [6], however not in the context of process algebra. Furthermore in [13] a form of context-dependent bisimulation for processes is introduced.

We believe that this paper is a first step towards a better understanding of context dependency. Several open problems still remain, for instance: is there a different way to characterize the equivalence of Section 6 (the coarsest congruence which is a bisimilarity wrt. reduction rules)?

One could also study an even coarser relation: the coarsest congruence that is contained in bisimilarity. The problem with this equivalence is that it does not naturally admit a coinductive definition. We believe that it should be possible to adapt ideas from [10] and to parametrize the bisimulation over contexts.

Considering the results of this paper we think that saturated bisimilarity (which we studied here) is more stable than the notion of IPO-bisimilarity, since it is independent on the specific choice of representative squares. On the other hand, the “right” notion of bisimilarity for process calculi is sometimes IPO-bisimilarity and it would be worthwhile studying how this notion of bisimilarity integrates into the proposed framework. On the more speculative side, conditions as presented here could be used to define barbs [16] and hence represent a means to fine-tune the equivalence.

Our main application area is graph transformation, but it should be worthwhile to study conditional reactive systems in different categories, for instance using a Lawvere theory (i.e., a category with terms as arrows).

Furthermore we plan to continue our work in [11] and to conduct further case studies in the area of the verification of model transformations. In order to obtain a practically usable method, we have to integrate up-to techniques, but we expect that this can be done without problems. Since label derivation is complex and can not easily be done manually, we however require an implementation, which we have already started to develop. In order to come to terms with the undecidability of implication we will either restrict to simpler conditions or use an approximative algorithm as in [17].

Acknowledgements. We would like to thank Vladimiro Sassone for asking an inspiring question after a talk. Furthermore we are grateful to Filippo Bonchi for valuable insights concerning the “right” notion of bisimilarity.

References

1. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS 2006, pp. 69–80. IEEE (2006)
2. Bruggink, H.J.S., Cauderlier, R., König, B., Hülsbusch, M.: Conditional reactive systems. In: Proc. of FSTTCS 2011. LIPICS, vol. 13. Schloss Dagstuhl – Leibniz Center for Informatics (2011)

3. Ehrig, H., König, B.: Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 151–166. Springer, Heidelberg (2004)
4. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *Mathematical Structures in Computer Science* 16(6), 1133–1163 (2006)
5. Engels, G., Kleppe, A., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 94–109. Springer, Heidelberg (2008)
6. Fitting, M.: Bisimulations and boolean vectors. In: *Advances in Modal Logic*, vol. 4, pp. 1–29. World Scientific Publishing (2002)
7. Di Gianantonio, P., Honsell, F., Lenisa, M.: RPO, second-order contexts, and lambda-calculus. *Logical Methods in Computer Science* 5(3) (2009)
8. Grohmann, D., Miculan, M.: Reactive Systems Over Directed Bigraphs. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 380–394. Springer, Heidelberg (2007)
9. Habel, A., Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science* 19, 245–296 (2009)
10. Hennessy, M., Lin, H.: Symbolic bisimulations. *Theoretical Computer Science* 138(2), 353–389 (1995)
11. Hülsbusch, M., König, B., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: Showing Full Semantics Preservation in Model Transformation - A Comparison of Techniques. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 183–198. Springer, Heidelberg (2010)
12. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications* 39(3) (2005)
13. Guldstrand Larsen, K.: Context-Dependent Bisimulation between Processes. PhD thesis, University of Edinburgh (1986)
14. Leifer, J.J.: Operational congruences for reactive systems. PhD thesis, University of Cambridge Computer Laboratory (September 2001)
15. Leifer, J.J., Milner, R.: Deriving Bisimulation Congruences for Reactive Systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
16. Milner, R., Sangiorgi, D.: Barbed Bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
17. Pennemann, K.-H.: Development of Correct Graph Transformation Systems. PhD thesis, Universität Oldenburg (May 2009)
18. Rangel, G., König, B., Ehrig, H.: Deriving Bisimulation Congruences in the Presence of Negative Application Conditions. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 413–427. Springer, Heidelberg (2008)
19. Rensink, A.: Representing First-Order Logic Using Graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004)
20. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: *Proc. of LICS 2005*, pp. 311–320. IEEE (2005)
21. Sobociński, P.: Deriving process congruences from reaction rules. PhD thesis, Department of Computer Science. University of Aarhus (2004)

First-Order Model Checking on Nested Pushdown Trees is Complete for Doubly Exponential Alternating Time

Alexander Kartzow

Universität Leipzig, Institut für Informatik, Johannsgasse 26, 04103 Leipzig,
Germany

Abstract. Recently we proved that first-order model checking on nested pushdown trees can be done in doubly exponential alternating time with linearly many alternations. Using the interpretation method of Compton and Henson we give a matching lower bound, i.e., we prove that first-order model checking on nested pushdown trees is complete for $\text{ATIME}(\exp_2(cn), cn)$ with respect to log-lin reductions.

Keywords: nested pushdown trees, first-order logic, $\text{ATIME}(\exp_2(cn), cn)$ lower bounds, model checking.

1 Introduction

Nested pushdown trees were first introduced by Alur et al. [1] as a representation of recursive first-order programs. Nested pushdown trees are unfoldings of pushdown graphs expanded by so-called *jump edges*. These edges connect corresponding push and pop operations. If one considers nested pushdown trees as representations of programs, each push corresponds to a function call and each pop corresponds to a function return. Thus, jump edges allow one to reason in first-order logic about pre-/post-conditions on function calls and returns. Note that in the usual representation of first-order programs by pushdown graphs, such a kind of reasoning is even not possible in monadic second-order logic. This advantage comes at a price: while monadic second-order logic is decidable on pushdown graphs [5], it is undecidable on nested pushdown trees [1]. But Alur et al. showed that a variant of the modal μ -calculus is still decidable. These results turn nested pushdown trees into an interesting class of graphs from a model-theoretic point of view. The author only knows one other natural class of graphs with these model checking properties, viz., the class of collapsible pushdown graphs. This common behaviour is explained by the fact that nested pushdown trees are uniformly first-order interpretable in collapsible pushdown graphs [4]. Nevertheless, the two classes differ when one considers first-order model checking. First-order model checking on the second level of the collapsible pushdown graph hierarchy is decidable but has nonelementary complexity [4].

In contrast, for nested pushdown trees we proved an $\text{ATIME}(\exp_2(cn), cn)$ upper bound for first-order model checking [3, 4].

The aim of this paper is to provide a matching lower bound for the first-order model checking problem on nested pushdown trees. In fact, we present a fixed nested pushdown tree with $\text{ATIME}(\exp_2(cn), cn)$ -hard first-order theory. This implies that the first-order model checking problem on nested pushdown trees is also $\text{ATIME}(\exp_2(cn), cn)$ -hard. As a byproduct of our proof we also obtain that the set of FO-sentences valid in all nested pushdown trees is $\text{ATIME}(\exp_2(cn), cn)$ -hard. But we do not know of any upper bound for this set.

Our tool is the interpretation method of Compton and Henson [2]: there is a family $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ of classes of trees such that the following holds. If there is a structure \mathfrak{A} and a sequence of interpretation $(I_n)_{n \in \mathbb{N}}$ whose formulas come from a so-called prescribed set such that I_n interprets $\mathcal{T}_3^{2^n}$ in \mathfrak{A} , then the theory of \mathfrak{A} is $\text{ATIME}(\exp_2(cn), cn)$ -hard. Moreover, the interpretation method is closed under composition in the following sense. If there is a family $(\mathcal{C}_n)_{n \in \mathbb{N}}$ of classes of structures and families of interpretations $(I_n)_{n \in \mathbb{N}}, (J_n)_{n \in \mathbb{N}}$ (using formulas of the mentioned special type) such that I_n interprets $\mathcal{T}_3^{2^n}$ in \mathcal{C}_n and J_n interprets \mathcal{C}_n in \mathfrak{A} then the theory of \mathfrak{A} is $\text{ATIME}(\exp_2(cn), cn)$ -hard.

In analogy to Compton and Henson’s $\text{ATIME}(\exp(cn), cn)$ -hardness result for the first-order theory of the binary successor tree (Example 8.3 in [2]) we prove the hardness result for model checking on nested pushdown trees in two steps. First, we provide interpretations of $\mathcal{T}_3^{2^n}$ in finite linear orders of length $\exp_2(13n)$ with one unary predicate. Using another family of interpretations we reduce the monadic second-order theories of such orders to the first-order theory of a fixed nested pushdown tree. This interpretation method yields the $\text{ATIME}(\exp_2(cn), cn)$ -hardness of the theory of this nested pushdown tree.

1.1 Related Work

It follows from the works of Volger [7] and Compton and Henson [2] that first-order model checking on (unfoldings of) pushdown graphs is $\text{ATIME}(\exp(cn), cn)$ -complete. Hence, we show that the introduction of jump edges lead to an exponential blow up in the complexity of model checking. Alur et al. [1] studied the model checking problem on nested pushdown trees for MSO and for a variant of the modal μ -calculus. The former is undecidable while the latter is EXPTIME-complete. We have shown in [4] that for first-order logic extended with the reachability predicate, the model checking is decidable but has nonelementary complexity (the lower bound already holds for (unfoldings of) pushdown graphs and even for the full infinite binary tree). Since nested pushdown trees are tree-automatic [4] it follows from [6, 4] that the extension of first-order logic by infinity quantifier \exists^∞ , modulo counting quantifiers $\exists^{n \bmod m}$ and Ramsey quantifiers Ram^n is decidable on nested pushdown trees. The model checking procedure obtained from these results has nonelementary complexity.

¹ A problem is in $\text{ATIME}(\exp_2(cn), cn)$ if there is a constant c and an alternating Turing Machine that solves the problem in times $\exp(\exp(cn))$ and cn many alternations where n is the input size.

1.2 Outline

In the next section, we fix our notation, especially concerning interpretations and nested pushdown trees. In Section 3 we recall the central result of Compton and Henson, i.e., we present the classes $\mathcal{T}_3^{2^n}$ of trees and recall how interpretation of these classes in a given structure yields lower bounds for the model checking problem. We then provide interpretations of these classes in finite linear orders of size doubly exponential in n in Section 4. In Section 5 the lower bound for model checking on nested pushdown trees is obtained by interpreting these linear orders of doubly exponential size in a fixed nested pushdown tree. Section 6 contains some concluding remarks.

2 Preliminaries

We set $\text{exp}(n) := 2^n$ and $\text{exp}_2(n) := 2^{\text{exp}(n)}$. In the following \bar{x}, \bar{y} , etc. will denote tuples of (pairwise distinct) variables $\bar{x} = x_1, x_2, \dots, x_n$, $\bar{y} = y_1, y_2, \dots, y_m$. We omit the specification of the arity of a tuple \bar{x} whenever the arity is arbitrary or is implicitly defined by the way we use the tuple.

By FO we denote *first-order logic* and by MSO we denote *monadic second-order logic*. For structures \mathfrak{A} and \mathfrak{B} we write $\mathfrak{A} \simeq \mathfrak{B}$ if \mathfrak{A} is isomorphic to \mathfrak{B} .

2.1 Interpretations

In this paper we will use the interpretation of the MSO theory of a family $(\mathcal{C}_n)_{n \in \mathbb{N}}$ of classes of structures in the MSO theory (or FO theory, respectively) of another family $(\mathcal{D}_n)_{n \in \mathbb{N}}$ in order to transfer lower bounds for the MSO model checking problem on $(\mathcal{C}_n)_{n \in \mathbb{N}}$ to the MSO (FO, respectively) model checking on $(\mathcal{D}_n)_{n \in \mathbb{N}}$. In the following we fix two (relational) signatures $\sigma = \{E_1, E_2, \dots, E_m\}$ and τ . For some σ -formula $\varphi(\bar{x}, \bar{y})$, some σ -structure \mathfrak{A} and $\bar{p} \in \mathfrak{A}$, we write $\varphi^{\mathfrak{A}}(\bar{x}, \bar{p})$ for the relation defined by φ in \mathfrak{A} with parameter \bar{p} . This means that $\varphi^{\mathfrak{A}}(\bar{x}, \bar{p}) := \{\bar{a} \in \mathfrak{A} : \mathfrak{A} \models \varphi(\bar{a}, \bar{p})\}$.

Definition 1. Let $(\mathcal{C}_n)_{n \in \mathbb{N}}$ be a family of classes of σ -structures, $(\mathcal{C}'_n)_{n \in \mathbb{N}}$ a family of classes of τ -structures and L either MSO or FO. Furthermore, let x and y be first-order variables and let \bar{x}_{E_i} be a k -tuple of first-order variables where k is the arity of E_i . Let $(I_n)_{n \in \mathbb{N}}$ be a sequence such that

$$I_n = (\delta^n(x, y), (\varphi_{E_i}^n(\bar{x}_{E_i}, y))_{E_i \in \sigma})$$

is an $(m + 1)$ -tuple of $L[\tau]$ -formulas. We call $(I_n)_{n \in \mathbb{N}}$ an L -to- L -interpretation of $(\mathcal{C}_n)_{n \in \mathbb{N}}$ in $(\mathcal{C}'_n)_{n \in \mathbb{N}}$ if for each $\mathfrak{A} \in \mathcal{C}_n$, there is a $\mathfrak{B} \in \mathcal{C}'_n$ and some $b \in \mathfrak{B}$ such that

$$\mathfrak{A} \simeq ((\delta^n)^{\mathfrak{B}}(x, b), ((\varphi_{E_i}^n)^{\mathfrak{B}}(\bar{x}_{E_i}, b))_{E_i \in \sigma}).$$

Let $(I_n)_{n \in \mathbb{N}}$ be a sequence such that

$$I_n = (\delta^n(x, y), (\varphi_{E_i}^n(\bar{x}_{E_i}, y))_{E_i \in \sigma}, \varphi_{\in}^n(x, z, y))$$

is a tuple of $FO[\tau]$ -formulas. We call $(I_n)_{n \in \mathbb{N}}$ an MSO-to-FO-interpretation of $(C_n)_{n \in \mathbb{N}}$ in $(C'_n)_{n \in \mathbb{N}}$ if for each $\mathfrak{A} \in C'_n$ there are $\mathfrak{B} \in C_n$ and $b \in \mathfrak{B}$ such that

$$\mathfrak{A} \simeq ((\delta^n)^{\mathfrak{B}}(x, b), ((\varphi^n_{E_i})^{\mathfrak{B}}(\bar{x}_{E_i}, b))_{E_i \in \sigma})$$

and $\varphi^n_{E_i}{}^{\mathfrak{B}}(x, b', b)$ ranges over all subsets of $\delta^n{}^{\mathfrak{B}}(x, b)$ as b' ranges over the universe of \mathfrak{B} .²

Remark 2. For a fixed structure \mathfrak{A} we write “ $(I_n)_{n \in \mathbb{N}}$ is an interpretation of $(C_n)_{n \in \mathbb{N}}$ in \mathfrak{A} ” as an abbreviation for “ $(I_n)_{n \in \mathbb{N}}$ is an interpretation of $(C_n)_{n \in \mathbb{N}}$ in $(D_n)_{n \in \mathbb{N}}$ where $D_n = \{\mathfrak{A}\}$ for all $n \in \mathbb{N}$ ”.

2.2 Nested Pushdown Trees

Nested pushdown trees are the unfoldings of the configuration graphs of pushdown systems with an added *jump relation* that connects every push- with the corresponding pop-operations.

Definition 3. A pushdown system is a 5-tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, (q_0, \perp))$ with a finite set of states Q , a finite set of stack symbols Σ , a transition alphabet Γ , an initial configuration $(q_0, \perp) \in Q \times \Sigma$ and a transition relation

$$\Delta \subseteq Q \times \Sigma \times \Gamma \times Q \times (\{\text{pop}, \text{id}_{\Sigma^+}\} \cup \{\text{push}_\sigma : \sigma \in \Sigma\}).$$

Let $w \in \Sigma^+$ and $\sigma \in \Sigma$. We call w a stack. A *configuration* of \mathcal{P} is a pair $(q, w) \in Q \times \Sigma^+$ of a state and a nonempty stack. We define the stack operations for all $\sigma \in \Sigma$ and $w \in \Sigma^+$ by

$$\text{push}_\sigma(w) := w\sigma \text{ and } \text{pop}(w\sigma) = w.$$

We define labelled transitions $\xrightarrow{\gamma}$ on the set of configurations of \mathcal{P} for $\gamma \in \Gamma$ as follows: $(q, w\sigma) \xrightarrow{\gamma} (p, v)$ if there is some $(q, \sigma, \gamma, p, \text{op}) \in \Delta$ such that $v = \text{op}(w\sigma)$. We use \rightarrow as abbreviation of $\bigcup_{\gamma \in \Gamma} \xrightarrow{\gamma}$.

A *run* r of \mathcal{P} is a sequence $c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} c_2 \xrightarrow{\gamma_3} \dots \xrightarrow{\gamma_n} c_n$ where the c_i are arbitrary configurations of \mathcal{P} . We denote by \preceq the prefix order on the set of runs. We call r a run from c_0 to c_n and say that the *length* of r is $\text{length}(r) := n$.

Definition 4. Let $\mathcal{P} = (Q, \Sigma, \Delta, (q_0, \perp))$ be a pushdown system. The nested pushdown tree generated by \mathcal{P} is the structure $\text{NPT}(\mathcal{P}) := (R, (\xrightarrow{\gamma})_{\gamma \in \Gamma}, \hookrightarrow)$ defined as follows:

² This condition means that $(I_n)_{n \in \mathbb{N}}$ is an FO-to-FO-interpretation of $(C_n)_{n \in \mathbb{N}}$ in $(C'_n)_{n \in \mathbb{N}}$ if we forget about the formulas $\varphi^n_{E_i}$ and $\varphi^n_{E_i}$ allows to transfer set quantification in C_n into element quantification in C'_n by replacing $\exists X$ with $\exists x\delta(x, y)$ and Xz by $\varphi^n(z, x, y)$.

- R is the set of runs starting in (q_0, \perp) ,
- $\xrightarrow{\gamma}$ connects a run r_1 with a run r_2 if r_2 extends r_1 by exactly one $\xrightarrow{\gamma}$ transition at the end, and
- the so-called jump-relation \hookrightarrow connects $r_1 \in R$ with $r_2 \in R$ if $r_1 \preceq r_2$, the final stack of r_1 and r_2 is the same word w and all stacks between have w as proper prefix.

Remark 5. Let $r_1 \preceq r_2$ be runs. Then $r_1 \hookrightarrow r_2$ holds if and only if r_2 extends r_1 by some run r that starts with some push_σ transition and ends with a pop transition that removes this σ from the stack again.

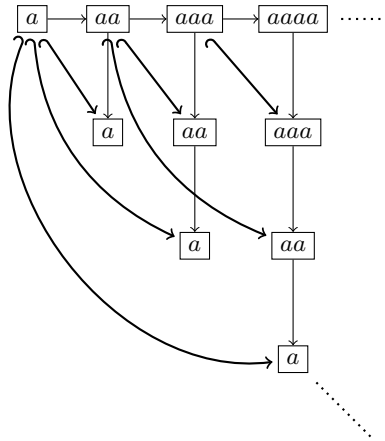


Fig. 1. Nested pushdown tree with undecidable MSO theory (the unique state and the edge labels are omitted)

Example 6. Figure 1 shows the nested pushdown tree induced by the transition relation $(q_0, a, \gamma_1, \text{push}_a, q_0), (q_0, a, \gamma_2, \text{pop}, q_0)\}$. The MSO theory of this nested pushdown tree is undecidable because the symmetric transitive closure of γ_2 -edges is the “same column relation” and the symmetric transitive closure of the jump edges is the “same diagonal relation” whence the half grid $\{(n, m) : m > n\}$ with right and downward edges is MSO interpretable in this graph. Standard arguments reduce undecidable tiling problems to the MSO theory of this half grid.

The previous example shows that the MSO theories of nested pushdown trees are undecidable in general. But the FO theories are uniformly decidable. In [3] (Theorem 2), we provided an FO model checking algorithm on nested pushdown trees. Even though we only claimed it was a 2-EXPSpace algorithm, the proof reveals the following fact.

Theorem 7. *The FO model checking problem on nested pushdown trees is in $\text{ATIME}(\exp_2(cn), n)$.*

3 The Interpretation Method

In this section, we recall some results of Compton and Henson [2] that we are going to use in the following. We first present the classes $\mathcal{T}_3^{2^n}$ of trees. Then we recall the necessary results relating interpretations of these classes to $\text{ATIME}(\exp_2(cn), cn)$ -hardness of the model checking problem.

3.1 Classes of Special Trees

A tree is a finite, prefix closed subset of \mathbb{N}^* together with the successor relation $S := \{(x, y) \in \mathbb{N}^* : \exists z \in \mathbb{N} \ y = xz\}$. For \mathfrak{T} some tree we call $|\mathfrak{T}|$ its *size*, i.e. $|\mathfrak{T}|$ is the number of elements in \mathfrak{T} . For $\mathfrak{T} = (T, S)$ some tree and $d \in T$, we denote by \mathfrak{T}_d the subtree rooted at d , i.e., $\mathfrak{T}_d = (T_d, S)$ where $T_d = \{x \in \mathbb{N}^* : dx \in T\}$. Note that for each tree $\mathfrak{T} = (T, S)$ the set $\mathbb{N} \cap T$ is the set of children of the root.

Definition 8. Let $\mathcal{T}_0^{2^n}$ be the class of the tree of depth 0, i.e., the class containing $(\{\varepsilon\}, S)$. Inductively, let $\mathcal{T}_{n+1}^{2^n}$ be the class of trees $\mathfrak{T} = (T, S)$ such that $\mathfrak{T}_d \in \mathcal{T}_n^{2^n}$ for all $d \in \mathbb{N} \cap T$ and for each $d \in \mathbb{N} \cap T$ there are at most 2^n many pairwise distinct elements $d' \in \mathbb{N} \cap T$ such that $\mathfrak{T}_{d'} \simeq \mathfrak{T}_d$, i.e., there are at most 2^n isomorphic maximal proper subtrees of \mathfrak{T} .

We will exclusively deal with $\mathcal{T}_3^{2^n}$ in this paper. The following lemma summarises some combinatorial properties of this class.

Lemma 9. $\mathcal{T}_3^{2^n}$ contains at most $(2^n + 1)^{(2^n+1)2^{n+1}}$ many trees up to isomorphism. A tree $\mathfrak{T} \in \mathcal{T}_3^{2^n}$ has size at most $2^{2^{12n}}$.

Proof. The first claim is by induction: Note that $\mathcal{T}_0^{2^n}$ consists of only one tree. Each element of $\mathcal{T}_{i+1}^{2^n}$ is determined by the number of maximal proper subtrees of each isomorphism type. Since there are at most 2^n maximal proper subtrees of the same isomorphism type, there are at most $(2^n + 1)^m$ many nonisomorphic trees in $\mathcal{T}_{i+1}^{2^n}$ where m is the number of distinct trees in $\mathcal{T}_i^{2^n}$ up to isomorphism. The first claim follows by induction.

Let $f(i)$ be the number of nonisomorphic trees in $\mathcal{T}_i^{2^n}$ and $g(i)$ the maximal number of nodes of a tree in $\mathcal{T}_i^{2^n}$. Note that the size of a tree $\mathfrak{T} \in \mathcal{T}_{i+1}^{2^n}$ is bounded by $1 + 2^n \cdot f(i) \cdot g(i)$. We conclude that

- for $\mathfrak{T} \in \mathcal{T}_1^{2^n}$, $|\mathfrak{T}| \leq 1 + 2^n \cdot 1 \cdot 1 = 2^n + 1$,
- for $\mathfrak{T} \in \mathcal{T}_2^{2^n}$, $|\mathfrak{T}| \leq 1 + 2^n \cdot (2^n + 1) \cdot (2^n + 1) \leq 2^{6n}$, and
- for $\mathfrak{T} \in \mathcal{T}_3^{2^n}$, $|\mathfrak{T}| \leq 1 + 2^n \cdot (2^n + 1)^{2^{n+1}} \cdot 2^{6n} \leq 1 + 2^{7n+(n+1) \cdot (2^n+1)} \leq 2^{2^{12n}} \quad \square$

3.2 Iterative Definitions and Prescribed Sets

The relevance of $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ for this paper stems from the fact that certain interpretations of these classes in a fixed structure \mathfrak{A} establish $\text{ATIME}(\exp_2(cn), cn)$ -hardness of the theory of \mathfrak{A} . Before we can specify the kind of interpretations that one may use for this purpose, we have to recall Compton and Henson’s

notion of *explicit* and *iterative* definitions. Let MSO^* denote the extension of MSO by explicit and iterative definitions. MSO^* is defined using the formation rules of MSO and the rules that

- for $\psi, \varphi \in \text{MSO}^*$ and P some free variable of ψ , the formula $[P = \varphi]\psi$ is in MSO^* where $[P = \varphi]$ is called an explicit definition, and
- for $\psi, \varphi \in \text{MSO}^*$ and P some free variable of ψ and φ , the formula $[P = \varphi]_n\psi$ is in MSO^* where $[P = \varphi]_n$ is called an iterative definition.

The semantics of MSO^* is defined using the semantics of MSO and the following rules³

- Let \mathfrak{A} be some structure with domain A . Let P_φ be the predicate that contain a tuple $\bar{a} \in A$ iff $\mathfrak{A} \models \varphi(\bar{a})$. Now $\mathfrak{A} \models [P = \varphi]\psi$ iff $\mathfrak{A}, P_\varphi \models \psi$, i.e. if \mathfrak{A} is a model of ψ where each occurrence of P is replaced by the relation defined by φ .
- Let \mathfrak{A} be some structure. \mathfrak{A} satisfies an iterative definition $[P = \varphi]_0\psi$ if \mathfrak{A} satisfies ψ where each occurrence of P is replaced by $\exists xx \neq x$, i.e., by a sentence which is false in every structure. \mathfrak{A} satisfies an iterative definition $[P = \varphi]_{n+1}\psi$ iff \mathfrak{A} satisfies $[P = [P = \varphi]_n\varphi]\psi$.

Analogously, we can define the extension FO^* of FO by explicit and iterative definitions. Adding explicit and iterative definitions do not increase the expressive power of MSO or FO but allow for more succinct definitions.

Definition 10. We call a set M of MSO^* -formulas a prescribed set⁴ if there is some $l \in \mathbb{N}$ such that each $\varphi \in M$ is of the form

$$[P_1 = \varphi_1]_{n_1}[P_2 = \varphi_2]_{n_2} \dots [P_k = \varphi_k]_{n_k}\psi$$

for some $k \in \mathbb{N}$ and $n_1, n_2, \dots, n_k \in \mathbb{N}$ where each φ_j is an MSO formula of size at most l whose only free set variable is P_j and ψ is an MSO formula in prenex normal form, i.e., ψ is quantifier free except for a block of quantifiers at the beginning. Fix some prescribed set M and an interpretation $I = (I_n)_{n \in \mathbb{N}}$. We say I is formed from M and – abusing notation – write $I \subseteq M$ if for each $n \in \mathbb{N}$ and for each formula φ in I_n there is an equivalent formula $\varphi' \in M$ of size linear in n (where the subscript n_i occurring in iterative definitions $[P_i = \varphi_i]_{n_i}$ are written in unary notation, i.e., a subscript n_i counts as a string of length n_i).

In the rest of this paper, the interpretations we define are always formed from some prescribed set. Since iterative definitions do not increase the expressive power of FO or MSO and since we aim at showing that the formulas used have equivalent versions in some prescribed set, we will use iterative definitions when defining MSO-to-MSO- or MSO-to-FO-interpretations.

³ In the following rules we suppress the occurrence of free variables; these are handled as usual.

⁴ Compton and Henson’s definition of prescribed sets is more general, but the special cases defined here suffice for our purpose.

3.3 Lower Bounds for Model Checking via Interpretations

One of the central results of Compton and Henson (Corollary 7.5 in [2]) is that interpretations $(I_n)_{n \in \mathbb{N}}$ formed from some prescribed set M can be used to obtain lower bounds on the model checking problem. We only state a simplified version of the more general result which is sufficient for our purposes.

Lemma 11 ([2]). *If there is some prescribed set M and $(I_n)_{n \in \mathbb{N}} \subseteq M$ is an MSO-to-FO-interpretation of $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ in a structure \mathfrak{A} , then the FO theory of \mathfrak{A} is $\text{ATIME}(\exp_2(cn), cn)$ -hard with respect to log-lin reduction⁵.*

Remark 12. In fact, Compton and Henson prove a stronger result. Under the assumptions of the lemma, \mathfrak{A} has a hereditary lower bound of $\text{ATIME}(\exp_2(cn), cn)$. This means that for T the FO-theory of \mathfrak{A} , and every subset $\Phi \subseteq T$ the sets $\text{SAT}(\Phi)$ and $\text{VAL}(\Phi)$ are $\text{ATIME}(\exp_2(cn), cn)$ -hard where $\text{SAT}(\Phi)$ denotes the set of FO sentences that are satisfiable in some model of Φ and $\text{VAL}(\Phi)$ denotes the set of FO sentences that are valid in every model of Φ .

Corollary 13. *Under the same assumptions as in the lemma, the FO model checking problem on any class \mathcal{C} such that $\mathfrak{A} \in \mathcal{C}$ is $\text{ATIME}(\exp_2(cn), cn)$ -hard.*

The use of the previous lemma is further facilitated since interpretations formed from prescribed sets are closed under composition.

Lemma 14 ([2]). *Let M and M' be prescribed sets of formulas. If $(I_n)_{n \in \mathbb{N}} \subseteq M$ and $(J_n)_{n \in \mathbb{N}} \subseteq M'$ are families of interpretations such that $(I_n)_{n \in \mathbb{N}}$ is an MSO-to-MSO-interpretation of $(\mathcal{C}_n)_{n \in \mathbb{N}}$ in $(\mathcal{D}_n)_{n \in \mathbb{N}}$ and $(J_n)_{n \in \mathbb{N}}$ is an MSO-to-FO-interpretation of $(\mathcal{D}_n)_{n \in \mathbb{N}}$ in $(\mathcal{E}_n)_{n \in \mathbb{N}}$, then there is a prescribed set M'' and an MSO-to-FO-interpretation $(K_n)_{n \in \mathbb{N}} \subseteq M''$ of $((\mathcal{C}_n)_{n \in \mathbb{N}}$ in $(\mathcal{E}_n)_{n \in \mathbb{N}}$.*

4 Interpretation of Trees in Linear Orders

Following the ideas of Compton and Henson (Examples 8.1, 8.6, and 8.8 in [2]) we first provide interpretations of the classes $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ in the classes $(\mathcal{L}_{13n})_{n \in \mathbb{N}}$ of linear orders of size exactly $2^{2^{13n}}$ with unary predicate P . We identify such a linear order \mathfrak{L} with a bitstring of the same length, where we interpret P as indicator of 1's in the bitstring. Fix a tree \mathfrak{T} . We use a string of the form $0^i 1$ in order to represent a node of depth i and encode \mathfrak{T} by traversing \mathfrak{T} in-order and code every node with the corresponding representation. The following function f does this encoding. For $k \in \mathbb{N}$ and \mathfrak{T} some tree of depth at most k , set

$$f(\mathfrak{T}, k) := 0^{k-1} 1 f(\mathfrak{T}_{i_1}, k-1) f(\mathfrak{T}_{i_2}, k-1) f(\mathfrak{T}_{i_3}, k-1) \dots f(\mathfrak{T}_{i_k}, k-1)$$

where $i_j \in \mathbb{N}$ is the j -th element of \mathbb{N} such that $i_j \in \mathfrak{T}$.

For $\mathfrak{T} \in \mathcal{T}_3^{2^n}$, its depth is bounded by 3 and \mathfrak{T} has at most $2^{2^{12n}}$ nodes. Thus, $f(\mathfrak{T}, 3)$ has length at most $4 \cdot 2^{2^{12n}} \leq 2^{2^{13n}}$. By padding 0's in the end, we obtain

⁵ A log-lin reduction is a reduction computable in logarithmic space and linear time.

bitstrings of length $2^{2^{13n}}$ that encodes \mathfrak{T} . As mentioned before, we identify this bitstring with a linear order in \mathcal{L}_{13n} .

In Example 8.1 of [2], Compton and Henson show that the classes $\mathcal{T}_3^{2^n}$ can be recovered from these encodings with MSO-to-MSO-interpretations.

Lemma 15. *There is a prescribed set M such that there is an MSO-to-MSO-interpretations $(I_n)_{n \in \mathbb{N}} \subseteq M$ of $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ in $(\mathcal{L}_{13n})_{n \in \mathbb{N}}$.*

Proof. For $Pr(x, y, z) := x < y \wedge (x < z \rightarrow y \leq z)$, set

$$\begin{aligned} \psi(x, y, Q) := & \exists x_1 \exists y_1 \forall z ((z \geq x \vee (P(x_1) \wedge Pr(x_1, x, z))) \\ & \wedge (z \geq y \vee (P(y_1) \wedge Pr(y_1, y, z)))) \\ & \vee (Pr(x_1, x, z) \wedge Pr(y_1, y, z) \wedge \neg P(x_1) \wedge \neg P(y_1) \wedge Q(x_1, y_1)) \end{aligned}$$

In ψ , $Pr(x, y, z)$ is used to express that x is the direct predecessor of y . The iterative definition $[Q = \psi]_{n+1}$ defines those tuples (a, b) where the number of consecutive 0's preceding a is equal to the number of consecutive 0's preceding b and this number is at most n . Then

$$\begin{aligned} \varphi_E := & [Q = \psi]_4 \exists x_1 \forall z' \forall z (Pr(x_1, x, z') \wedge x < y \\ & \wedge P(x) \wedge P(y) \wedge \neg P(x_1) \wedge Q(x_1, y) \wedge (x < z < y \rightarrow \neg Q(x, z))) \end{aligned}$$

says that y and the predecessor of x have the same number of consecutive preceding 0's (up to 3) and no element in between x and y has the same number of consecutive preceding 0's as x (up to 3). On a linear order that stems from the encoding $f(\mathfrak{T})$ for some $\mathfrak{T} \in \mathcal{T}_3^{2^n}$, φ_E interprets exactly the edge relation of \mathfrak{T} . Setting $I_n := (\delta(x), \varphi_E(x, y))$, $(I_n)_{n \in \mathbb{N}}$ is an MSO-to-MSO-interpretation of $(\mathcal{T}_3^{2^n})_{n \in \mathbb{N}}$ in $(\mathcal{L}_{13n})_{n \in \mathbb{N}}$. \square

5 Reduction of Linear Orders to NPT

Due to Lemmas [11], [14] and [15], it suffices to provide an MSO-to-FO-interpretation of \mathcal{L}_{13n} in some fixed nested pushdown tree in order to show $\text{ATIME}(\exp_2(cn), cn)$ -hardness of FO model checking on NPT. In the rest of this paper, we consider the fixed pushdown system

$$\begin{aligned} \mathcal{S} := & (Q, \Sigma, \Gamma, \Delta, (q_0, a)) \text{ with} \\ Q = & \{q_0, q_1\}, \Sigma = \{a\}, \\ \Gamma = & \{+_0, +_1, -_0, -_1\} \text{ and} \\ \Delta = & \{(q_i, a, +_j, \text{push}_a, q_j), (q_i, a, -_j, \text{pop}, q_j) : i, j \in \{0, 1\}\}. \end{aligned}$$

In each configuration \mathcal{S} nondeterministically changes to state q_0 or q_1 and performs a push or a pop operation. This means that the runs of \mathcal{S} are all possible runs of pushdown systems with 2 states 1 stack symbol. The nested pushdown system generated by \mathcal{S} is depicted in Figure [2].

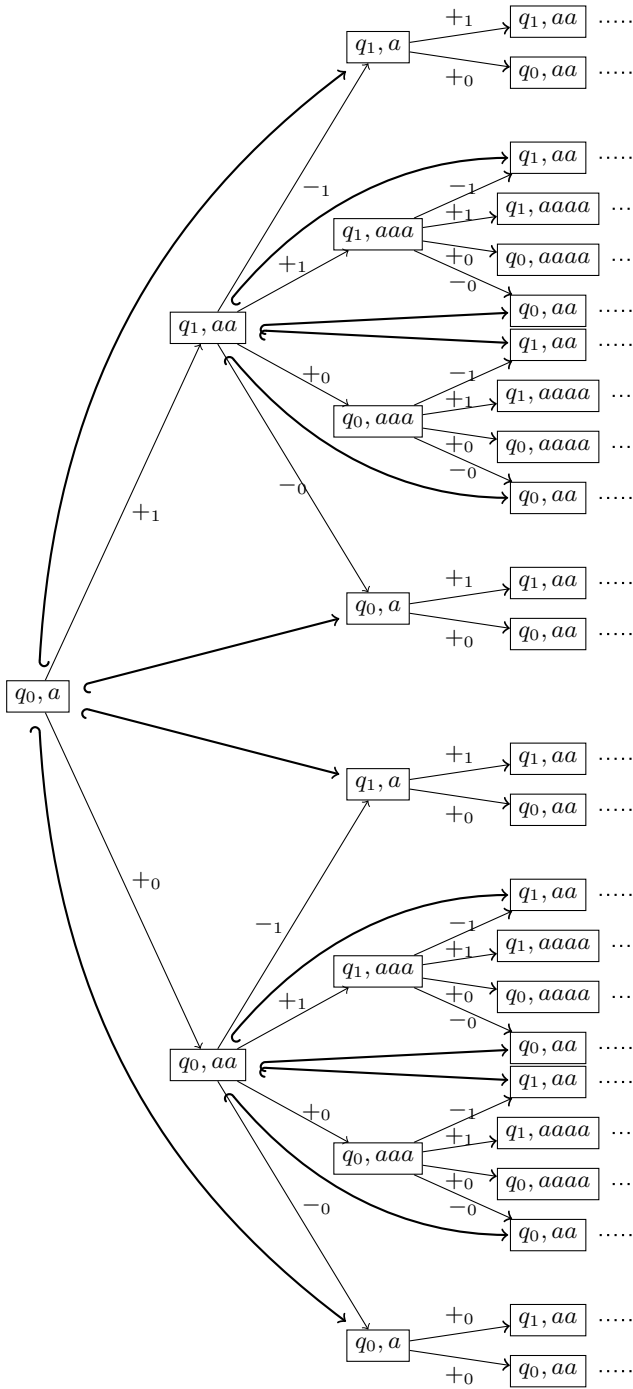


Fig. 2. Nested pushdown tree $NPT(S)$

In the following, we show that $\text{NPT}(\mathcal{S})$ contains nodes a that have $2^{2^{13n}}$ pairwise distinct ancestors each of which is connected to a by a path of length 2^{13n} whose edges consist of $\overrightarrow{\text{z}}$ and \hookrightarrow , i.e.,

$$|\{b \in \text{NPT}(\mathcal{S}) : \text{NPT}(\mathcal{S}) \models \varphi_p^{\overline{=}2^{13n}}(b, a)\}| = \exp_2(13n).$$

Such a are used to represent elements from \mathcal{L}_{13n} as follows: each of the $2^{2^{13n}}$ ancestors of a represent one element of \mathcal{L}_{13n} . The order is given by the ancestor relation which can be expressed in linear size using the formula $\varphi_p^{\leq 2^{13n}}$. Furthermore, the unary predicate P contains those nodes that are in state q_1 . This yields an FO-to-FO-interpretation of \mathcal{L}_{13n} in $\text{NPT}(\mathcal{S})$. We extend this interpretation to an MSO-to-FO-interpretation as follows. Given nodes a_1 and a_2 representing linear orders with one unary predicate P_{a_1} and P_{a_2} respectively, the formula $\varphi_e^{\overline{=}2^{13n}}(b_1, a_1, b_2, a_2)$ is satisfied if and only if there is some $j \leq 2^{2^{13n}}$ such that b_1 is the j -th element of the order induced by a_1 and b_2 is the j -th element of the order induced by a_2 . Using this fact, we can express “ a is the j -th node of the order induced by x and the j -th element of the order induced by y is in P_y ”. Thus, given a fixed $g \in \text{NPT}(\mathcal{S})$ representing a linear order in \mathcal{L}_{13n} , quantification over representations of linear orders in \mathcal{L}_{13n} is the same as quantification over monadic predicates in the order induced by g .

5.1 Short Formulas for Paths of Exponential Length

Aiming at the interpretation of \mathcal{L}_{13n} in $\text{NPT}(\mathcal{S})$, we define formulas $\varphi_p^{\leq 2^{13n}}$, $\varphi_p^{\overline{=}2^{13n}}$, $\varphi_e^{\leq 2^{13n}}$, and $\varphi_e^{\overline{=}2^{13n}}$ talking about paths in $\text{NPT}(\mathcal{S})$. $\varphi_p^{\leq 2^{13n}}(x, y)$ expresses that there is a path of length at most 2^{13n} from x to y using only **pop** transitions and jump edges. $\varphi_p^{\overline{=}2^{13n}}(x, y)$ is the variant for a path of length exactly 2^{13n} . $\varphi_p^{\leq 2^{13n}}(x_1, x_2, y_1, y_2)$ expresses that there is a path of length $k \leq 2^{13n}$ from x_1 to x_2 and a path of length k from y_1 to y_2 such that both paths only consist of jump edges and **pop** transitions and the i -th edge in one of the paths is a jump edge if and only if the i -th edge in the other is also a jump edge. Analogously, the i -th edge in one path is a **pop** if and only if the i -th edge in the other path is a **pop**.

Definition 16. Define by induction on m the following formulas:

$$\begin{aligned} \psi_1(x, y, Q) &:= \exists z(Qxz \wedge Qzy) \vee x = y \vee x \overrightarrow{\text{z}} y \vee x \overrightarrow{\text{z}} y \vee x \hookrightarrow y \\ \varphi_p^{\leq 2^m}(x, y) &:= [Q = \psi_1]_{m+1} Qxy \\ \psi_2(x, y, R) &:= \exists z \forall z' \forall z'' (Rzx \wedge Rzy) \vee \left(-Rz'z'' \wedge (x \overrightarrow{\text{z}} y \vee x \overrightarrow{\text{z}} y \vee x \hookrightarrow y) \right) \\ \varphi_p^{\overline{=}2^m}(x, y) &:= [R = \psi_2]_{m+1} Rxy \end{aligned}$$

$$\begin{aligned}
 \psi_3(x_1, x_2, y_1, y_2, S) &:= \exists z \exists z' (Sx_1 z y_1 z' \wedge S z x_2 z' y_2) \\
 &\quad \vee (x_1 = x_2 \wedge y_1 = y_2) \vee (x_1 \hookrightarrow x_2 \wedge y_1 \hookrightarrow y_2) \vee \\
 &\quad \left((x_1 \xrightarrow{0} x_2 \vee x_1 \xrightarrow{1} x_2) \wedge (y_1 \xrightarrow{0} y_2 \vee y_1 \xrightarrow{1} y_2) \right) \\
 \varphi_e^{\leq 2^m}(x_1, x_2, y_1, y_2) &:= [S = \psi_3]_{m+1} S x_1 x_2 y_1 y_2 \\
 \psi_4(x_1, x_2, y_1, y_2, T) &:= \exists z \exists z' \forall t \forall u \forall v \forall w (T x_1 z y_1 z' \wedge T z x_2 z' y_2) \\
 &\quad \vee \left(\neg T t u v w \wedge ((x_1 \hookrightarrow x_2 \wedge y_1 \hookrightarrow y_2) \vee \right. \\
 &\quad \left. ((x_1 \xrightarrow{0} x_2 \vee x_1 \xrightarrow{1} x_2) \wedge (y_1 \xrightarrow{0} y_2 \vee y_1 \xrightarrow{1} y_2)) \right) \\
 \varphi_e^{\geq 2^m}(x_1, x_2, y_1, y_2) &:= [T = \psi_4]_{m+1} T x_1 x_2 y_1 y_2
 \end{aligned}$$

5.2 Nodes with Many Ancestors

In this section, we prove that there are formulas of size linear in n that define subsets of $\text{NPT}(\mathcal{S})$ of size $\exp_2(13n)$ that can be interpreted as bitstrings of the same size. We start with an auxiliary lemma that allows to separate different ancestors of a given node of $\text{NPT}(\mathcal{S})$. The lemma says the following: Given two paths p_1, p_2 in $\text{NPT}(\mathcal{S})$ that consist of jump edges and pop transitions such that p_1 and p_2 end in the same node $c \in \text{NPT}(\mathcal{S})$ such that p_1 ends in a jump edge and p_2 ends in a pop transition, then p_1 starts in an ancestor of the first node of p_2 .

Lemma 17. *Let, $a, a', b, b', c \in \text{NPT}(\mathcal{S})$ and $m \in \mathbb{N}$. If*

$$\text{NPT}(\mathcal{S}) \models \varphi_p^{\leq 2^m}(a, b) \wedge \varphi_p^{\leq 2^m}(a', b') \wedge b' \rightarrow c \wedge b \hookrightarrow c$$

then $a \preceq b \prec a' \preceq b' \prec c$.

Proof. The nontrivial claim is that $b \prec a'$. If $x \xrightarrow{j} y$ for some $j \in \{0, 1\}$, then x has a bigger stack than y . Furthermore, if $x \hookrightarrow y$ then the stacks of x and y agree and all $x \prec z \prec y$ have bigger stacks.

Applying this observation to $b \hookrightarrow c$, we obtain that the predecessor b' of c is connected via a pop transition to c . Thus, the stacks of b and c agree while the stack of b' is bigger than that of c . Furthermore, between a' and b' all stacks are at least as big as the stack of b' . Since $b \prec c$ and its stack is smaller than that of b' , one concludes that $b \prec a'$. □

Using the previous lemma inductively, one concludes that going backward different pop and jump-edge paths lead to different nodes. One formalisation of this claim is the following corollary.

Corollary 18. *Let $a, b, c \in \text{NPT}(\mathcal{S})$ such that $\text{NPT}(\mathcal{S}) \models \varphi_p^{\leq 2^m}(a, c) \wedge \varphi_p^{\leq 2^m}(b, c)$. Either $\text{NPT}(\mathcal{S}) \models \varphi_e^{\leq 2^m}(a, c, b, c)$ or $a \neq b$.*

Definition 19. *We define the formula $\varphi_{in}^n(x) := \forall y \exists z (\varphi_p^{\leq 2^{13n}}(y, x) \rightarrow (z \hookrightarrow y))$. For each $a \in \text{NPT}(\mathcal{S})$, let $D_a^n := \{b \in \text{NPT}(\mathcal{S}) : \text{NPT}(\mathcal{S}) \models \varphi_p^{\geq 2^{13n}}(b, a)\}$.*

Since $z \hookrightarrow y$ implies that y has a direct predecessor $z' \rightarrow y$ such that the transition from z' to y is a pop-transition, $\varphi_{lin}^n(x)$ is satisfied if and only if for each sequence s of pop-transitions and jump-edges of length $2^{13n} + 1$ there is some node y connected to x via a path of form s . Due to Corollary 18, all these different sequences lead to different ancestors of x . Thus, if $\varphi_{lin}^n(x)$ holds, the paths to all elements of D_x^n form a full binary tree of depth 2^{13n} . Since D_x^n is the set of leaves of this tree, it contains exactly $\exp_2(13n)$ elements.

Corollary 20. *If $\text{NPT}(\mathcal{S}) \models \varphi_{lin}^n(a)$ then $|D_a^n| = \exp_2(13n)$.*

We will now construct elements $a \in \text{NPT}(\mathcal{S})$ that satisfy φ_{lin}^n .

Lemma 21. *For each m there is a node $a \in \text{NPT}(\mathcal{S})$ with 2^m many ancestors of distance m . Each of these ancestors is connected to a via some path of length m that only uses jump edges and pop transitions.*

Proof. The proof is by induction on m . In fact, we prove the following stronger claim: for $m \in \mathbb{N}$ and an arbitrary $a_0 \in \text{NPT}(\mathcal{S})$, we can construct a node $a \in \text{NPT}(\mathcal{S})$ with 2^m many ancestors of distance m such that each of these ancestors is a descendant of a_0 and connected to a via some path of length m that only uses \hookrightarrow - and pop-edges. Furthermore, a_0 is connected to a via a path of m \hookrightarrow -edges.

For $m = 0$ the claim holds trivially by setting $a := a_0$.

Now assume that for some $m \in \mathbb{N}$ the claim holds. Let $a_0 \in \text{NPT}(\mathcal{S})$. Let a_1 be a node in $\text{NPT}(\mathcal{S})$ satisfying the claim with respect to m and a_0 . Let a_2 be the unique node such that $a_1 \xrightarrow{+} a_2$. Let a_3 be a node in $\text{NPT}(\mathcal{S})$ satisfying the claim with respect to m and a_2 . Let a be the unique node such that $a_3 \xrightarrow{-} a$.

Note that a_1 and a_3 are the ancestors of a of distance 1. Each of these has 2^m ancestors of distance m . By Lemma 17 these are disjoint whence a has $2 \cdot 2^m = 2^{m+1}$ ancestors at distance $m + 1$. Moreover a_0 is connected to a_1 via a \hookrightarrow path of length m and $a_1 \hookrightarrow a$. Thus, a satisfies the claim. \square

Remark 22. Note that the construction in the previous proof does not rely on the use of the transition $\xrightarrow{+}$ and $\xrightarrow{-}$. In each construction step, we can arbitrarily replace $\xrightarrow{+}$ by $\xrightarrow{+ \circ}$ and $\xrightarrow{-}$ by $\xrightarrow{- \circ}$. Hence, the state of each node occurring in the construction can be chosen independently.

Corollary 23. *For a fixed string $b \in \{0, 1\}^{\exp_2(13n)}$, there is some $a \in \text{NPT}(\mathcal{S})$ such that $\text{NPT}(\mathcal{S}) \models \varphi_{lin}^n(a)$ and the i -th element of D_a^n (w.r.t. \prec) is in state q_1 if and only if the i -th bit of b is 1.*

5.3 Interpretation of Order and Monadic Quantification

We fix an element a satisfying $\varphi_{lin}^n(a)$. We can interpret every \hookrightarrow -edge as 0 and every \rightarrow -edge as 1. Using this convention each path p of length 2^{13n} from some ancestor b to a can be interpreted as the 2^n -bit number \hat{p} induced by its transitions. By induction on Lemma 17, we obtain that b is the \hat{p}_b -th element of

D_a^n with respect to \prec for all $b \in D_a^n$ and for p_b the unique path from b to a of length 2^{13n} .

We next present a formula of size linear in n that defines \prec on D_a^n . Afterwards we will show that monadic quantification in linear orders in \mathcal{L}_{13n} can be reduced to first-order quantification in $\text{NPT}(\mathcal{S})$.

Recall that $b \prec b'$ holds for $b, b' \in D_a^n$ if and only if for p the minimal path from b to a and p' the minimal path from b' to a (both of length 2^{13n}) have a common suffix and at the maximal position where p and p' differ, p consists of a jump edge. Note that this implies that p' contains at this position a \overrightarrow{j} -edge for some $j \in \{0, 1\}$. Let $I_n = (\delta^n, \varphi_{<}^n, \varphi_p^n)$ be given by

$$\begin{aligned} \delta^n(x, y) &:= \varphi_{lin}^n(y) \wedge \varphi_p^{\overleftarrow{2^{13n}}}(x, y), \\ \varphi_{<}^n(x_1, x_2, y) &:= \exists z \exists z_1 \exists z_2 \delta^n(x_1, y) \wedge \delta^n(x_2, y) \wedge z_1 \hookrightarrow z \wedge z_2 \rightarrow z \\ &\quad \wedge \varphi_p^{\leq 2^{13n}}(z, y) \wedge \varphi_p^{\leq 2^{13n}}(x_1, z_1) \wedge \varphi_p^{\leq 2^{13n}}(x_2, z_2) \text{ and} \\ \varphi_p^n(x, y) &:= \exists z (z \overleftarrow{j} x \vee z \overrightarrow{j} x) \wedge \delta^n(x, y) \end{aligned}$$

Note that for every $a \in \text{NPT}(\mathcal{S})$ with $\text{NPT}(\mathcal{S}) \models \varphi_{lin}^n(y)$, $\delta^{n\text{NPT}(\mathcal{S})}(x, a)$ is a set of size $\exp_2(13n)$ that is linearly ordered by $\varphi_{<}^{n\text{NPT}(\mathcal{S})}(x_1, x_2, a)$. Furthermore, $\varphi_p^{n\text{NPT}(\mathcal{S})}(x, a)$ selects the subset of nodes of $\delta^{n\text{NPT}(\mathcal{S})}(x, a)$ which represent runs that end in state q_1 . Due to remark 22, for each $\mathfrak{L} \in \mathcal{L}_{13n}$ there is some $a \in \text{NPT}(\mathcal{S})$ such that I_n interprets \mathfrak{L} in $\text{NPT}(\mathcal{S})$. Thus, $(I_n)_{n \in \mathbb{N}}$ is an FO-to-FO-interpretation of $(\mathcal{L}_{13n})_{n \in \mathbb{N}}$ in $\text{NPT}(\mathcal{S})$.

We extend this interpretation to an MSO-to-FO-interpretation. Given some $\mathfrak{L} \in \mathcal{L}_{13n}$ we can identify its domain with the set $\{1, 2, \dots, \exp_2(13n)\}$ such that its order coincides with the order of the natural numbers on this set. Given some $a \in \text{NPT}(\mathcal{S})$ such that $\text{NPT}(\mathcal{S}) \models \varphi_{lin}^n(a)$, we identify the linear order \mathfrak{L}_a obtained by I_n' with parameter a with the set $\{n \in \mathfrak{L}_a : \mathfrak{L}_a \models P_n\}$. Since all subsets of $\{1, 2, \dots, \exp_2(13n)\}$ appear as predicates of orders in \mathcal{L}_{13n} , quantification over subsets of $\{1, 2, \dots, \exp_2(13n)\}$ can be reduced to quantification over elements satisfying φ_{lin}^n . We only need to construct a formula $\varphi_{=}^n(b, a, b', a')$ which expresses that b is the j -th element of D_a^n iff b' is the j -th element of $D_{a'}^n$. Note that this is the case if and only if the minimal path from b to a consists of the same transitions (in the same order) as the path from b' to a' . Thus, we may set

$$\begin{aligned} \varphi_{=}^n(x_1, x_2, y_1, y_2) &:= \varphi_{lin}^n(x_2) \wedge \varphi_{lin}^n(y_2) \wedge \varphi_e^{\overleftarrow{2^{13n}}}(x_1, x_2, y_1, y_2), \\ \varphi_{\neq}^n(x, z, y) &:= \exists z' (\varphi_{=}^n(x, y, z', z) \wedge \varphi_p^n(z', z)) \text{ and} \\ I_n &:= (\delta^n(x, y), \varphi_{<}^n(x_1, x_2, y), \varphi_p^n(x, y), \varphi_{\neq}^n(x, z, y)). \end{aligned}$$

Theorem 24. *There is a prescribed set M such that there is an MSO-to-FO-interpretation $(I_n)_{n \in \mathbb{N}} \subseteq M$ of $(\mathcal{L}_{13n})_{n \in \mathbb{N}}$ in $\text{NPT}(\mathcal{S})$.*

Proof. Note that all formulas occurring in I_n are in prenex normal form. Moreover their size is linear in n . By moving iterative definitions to the front, we obtain an interpretation as desired. \square

Using the result of Compton and Henson (Lemma 11), we immediately get the following corollary (the second part uses Remark 12).

Corollary 25. *The FO theory of $\text{NPT}(\mathcal{S})$ is $\text{ATIME}(\exp_2(cn), cn)$ -hard. Thus, FO model checking on the class of all NPT is $\text{ATIME}(\exp_2(cn), cn)$ -complete with respect to log-lin reductions. Moreover, the set of FO sentences valid in every nested pushdown tree, i.e., the theory of nested pushdown trees is $\text{ATIME}(\exp_2(cn), cn)$ -hard.*

6 Conclusions

We have studied the complexity of first-order model checking on the class of nested pushdown trees. We obtained a matching lower bound resulting in the fact that the first-order model checking is $\text{ATIME}(\exp_2(cn), cn)$ -complete. This bound even holds for a fixed nested pushdown tree. Thus, also the expression complexity of first order model checking is exactly in $\text{ATIME}(\exp_2(cn), cn)$. The exact structure complexity of first-order model checking remains open. We have given an $\text{ATIME}(\exp(cn), cn)$ -algorithm 3 but we do not have any lower bounds.

Another open question concerns decidability of model checking for other fragments of monadic second order logic on nested pushdown trees. For instance, is first-order logic extended by the transitive closure operator decidable? Is the extension of first-order logic by regular reachability decidable? We know 4 that the extension of first-order logic by the reachability predicate is decidable and has non-elementary complexity. We have started to investigate the extension by transitive closure operators and we believe that it is undecidable if we allow sufficiently many nestings of the transitive closure operator.

References

1. Alur, R., Chaudhuri, S., Madhusudan, P.: Languages of Nested Trees. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 329–342. Springer, Heidelberg (2006)
2. Compton, K.J., Henson, C.W.: A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic* 48(1), 1–79 (1990)
3. Kartzow, A.: FO Model Checking on Nested Pushdown Trees. In: Kráľovič, R., Nawiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 451–463. Springer, Heidelberg (2009)
4. Kartzow, A.: First-Order Model Checking On Generalisations of Pushdown Graphs. PhD thesis, TU Darmstadt Fachbereich Mathematik (July 2011)
5. Muller, D.E., Schupp, P.E.: The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.* 37, 51–75 (1985)
6. Rubin, S.: Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic* 14(2), 169–209 (2008)
7. Volger, H.: Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first order theories. *Theor. Comput. Sci.* 23, 333–337 (1983)

Model Checking Languages of Data Words^{*}

Benedikt Bollig¹, Aiswarya Cyriac¹, Paul Gastin¹, and K. Narayan Kumar²

¹ LSV, ENS Cachan, CNRS & INRIA, France
{bollig,cyriac,gastin}@lsv.ens-cachan.fr

² Chennai Mathematical Institute, India
kumar@cmi.ac.in

Abstract. We consider the model-checking problem for data multi-pushdown automata (DMPA). DMPA generate data words, i.e., strings enriched with values from an infinite domain. The latter can be used to represent an unbounded number of process identifiers so that DMPA are suitable to model concurrent programs with dynamic process creation. To specify properties of data words, we use monadic second-order (MSO) logic, which comes with a predicate to test two word positions for data equality. While satisfiability for MSO logic is undecidable (even for weaker fragments such as first-order logic), our main result states that one can decide if all words generated by a DMPA satisfy a given formula from the full MSO logic.

1 Introduction

In recent years, there has been an increasing interest in data words and data trees, i.e., structures over an infinite alphabet. Data trees may serve as a model of XML documents where the data part refers to attribute values or text contents [3]. Data words, on the other hand, are suitable to model the behavior of concurrent programs where an unbounded number of processes communicate via message passing [4, 5].

Naturally, a variety of formalisms have been considered to specify sets of data words in the context of verification. A considerable amount of work has gone into the study of temporal and monadic second-order (MSO) logic, mainly focusing on the satisfiability problem [2, 8, 9, 19]. MSO logic over data words allows us to check data values of two word positions for equality. However, the logic is so complex that only severely restricted fragments preserve its decidability. A remarkable result due to Bojańczyk et al. states that satisfiability is decidable for first-order logic when it is restricted to two variables [2], albeit of very high complexity, as it is equivalent to reachability for Petri nets. Elementary upper bounds were only obtained by restricting the logic further [8, 19]. Anyway, decidability crucially relies on the fact that there is only one data value per position, which is clearly not sufficient to model executions of concurrent message-passing programs. Indeed, the lack of expressiveness and extensibility of those logics limits their use for verification.

^{*} Supported by LIA InForMel, ARCUS, and DIGITEO LoCoReP.

In this paper, we consider the model-checking problem, which has not received as much attention in the context of data words as satisfiability, and we adopt the orthogonal approach of restricting the domain of data words instead of pruning the logic. More precisely, we introduce data multi-pushdown automata (DMPA), which may, for example, represent the behavior of a concurrent program. Requirements specifications for such languages can then be written in the full MSO logic. Our main result is that the model-checking problem is decidable: Do all data words accepted by a DMPA satisfy a given MSO formula?

Like automata over finite alphabets, a DMPA uses standard building blocks such as states and stack symbols. Moreover, it has (finitely many) registers, which can store concrete data values in a run. Unlike a simple pushdown automaton, a DMPA is equipped with several stacks and can define non-context free behaviors. However, while multi-pushdown automata over a finite alphabet are often used for the verification of concurrent recursive programs [15, 16], modeling recursion is not our primary goal. Rather, in the context of an unbounded number of processes, context-sensitive rewriting is necessary to describe distributed protocols, as they typically operate in several phases. For example, DMPA are able to model a token-based leader election protocol where the number of processes is unknown. Though such a protocol can be implemented locally in terms of finite-state processes, their global, observable behavior is not context-free.

Our decidability proof relies on the following idea: A tree-like structure in terms of (multiply) nested words over a *finite* alphabet is built on top of a data word and is used to recover word positions that carry the same data value. Nested words naturally appear as runs of DMPA. To preserve decidability of MSO model checking, as we deal with several stacks, we have to impose a bound on the number of switches from one stack to another. Model checking DMPA can then be reduced to satisfiability of MSO logic over nested words with a bounded number of phases, which is decidable due to [15].

At first glance, DMPA produce data words, which are linearly ordered and of course suitable to describe sequential behaviors. One important aspect of MSO logic, however, is that it can easily define causal dependencies between events that go beyond the linear order induced by a data word. Our approach is, therefore, not restricted to sequential systems, but allows us to model complex dynamic concurrent programs and protocols from mobile computing. Our hope is that this will help to bring data words closer to applications in verification.

Related Work. A wide range of automata over data words have been introduced in the literature [2, 6, 12, 13, 14, 20]. For all of them, MSO model checking is undecidable. Moreover, none of them is suited to represent distributed protocols: they either run on one-dimensional data words, process a data word in several passes, or do not support concurrency. An automaton model that captures the interplay of communicating processes is due to [4]. Its modeling power, however, comes at the price of an undecidable emptiness problem.

Model checking of counter machines against freeze LTL was considered in [10, 11]. That setting is quite different from ours, as formulas are interpreted over runs, which contain counter evaluations as data values. Moreover, the temporal

logic, which can be embedded into MSO logic, has to be restricted further to obtain decidability results.

Our approach of introducing DMPA as a model of programs and using unrestricted MSO as requirements-specification language is partly inspired by [17]. There, Leucker et al. consider dynamic message sequence graphs as a model of dynamic communicating systems where an unbounded number of processes communicate through message exchange. No link with data words was established, though, and rules are context-free so that a leader election protocol cannot be described. Thus, we provide a more general, but conceptually simple, framework with a generic proof of decidability of MSO model checking.

Outline. In Section 2 we introduce data words and DMPA. Section 3 presents MSO logic to specify properties over data words. In Section 4 we establish decidability of the model-checking problem. We conclude in Section 5.

2 Data Words and Data Multi-pushdown Automata

By $\mathbb{N} = \{0, 1, 2, \dots\}$, we denote the set of natural numbers. For $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, \dots, n\}$. A *ranked alphabet* is a non-empty set Σ where every letter $a \in \Sigma$ has an arity, denoted $\text{arity}_\Sigma(a) \in \mathbb{N}$. We sometimes write $\text{arity}(a)$ instead of $\text{arity}_\Sigma(a)$ when Σ is clear from the context. For any set D , we let $\Sigma_D = \{a(d_1, \dots, d_m) \mid a \in \Sigma, m = \text{arity}(a), \text{ and } d_1, \dots, d_m \in D\}$.

Henceforth, we fix a finite ranked alphabet Σ (of *labels*) and an infinite set D (of *data values*). The elements of Σ_D are called *actions*. A *data word* is a sequence of actions, i.e., an element from Σ_D^* . Given a data word $w = w_1 \dots w_n$ of length n , we denote by $\text{dom}(w)$ its domain $\{1, \dots, n\}$, i.e., its set of positions. For $i \in \text{dom}(w)$ with $w_i = a(d_1, \dots, d_m)$, we let $\text{label}(i)$ refer to a and $\text{data}_k(i)$ to d_k , for all $k \in \{1, \dots, m\}$. Moreover, we set $\text{arity}(i) = m$. For example, if $D = \mathbb{N}$ and $\Sigma = \{a, b\}$ with $\text{arity}(a) = 1$ and $\text{arity}(b) = 2$, then $a(4) b(7, 9) a(6) b(10, 7)$ is a data word from Σ_D^* . We have $\text{label}(3) = a$ and $\text{data}_2(4) = 7$.

To represent systems whose executions are data words, we use data multi-pushdown automata (DMPA). Basically, a DMPA is a multi-pushdown automaton with $\mathfrak{h} \geq 1$ stacks over some finite alphabet. In addition, it has $\mathfrak{k} \geq 0$ global registers, $R = \{r_1, \dots, r_{\mathfrak{k}}\}$, which can store data values. Data values can also be stored on stacks along with a stack symbol from a finite ranked alphabet \mathcal{Z} . To refer to these data values, we use *parameters* from the infinite supply $P = \{p_1, p_2, \dots\}$. A transition of a DMPA depends on the current state of the automaton, the data values that are stored in the registers, and the top symbol of the stack chosen by the transition as well as its associated data values. The data values that are stored along with $A \in \mathcal{Z}$ can be accessed by means of the parameters $P_A = \{p_1, \dots, p_{\text{arity}(A)}\} \subseteq P$. Now, a transition is controlled by a guard that allows us to compare data values stored in registers with data values from the target stack. A *guard* (wrt. A) is generated by the grammar $\Phi ::= \text{true} \mid \pi_1 = \pi_2 \mid \Phi \wedge \Phi \mid \neg \Phi$ where $\pi_1, \pi_2 \in R \cup P_A$. For example, guard $r_1 = r_2 \wedge \neg(r_1 = p_3)$ requires that the contents of register r_1 equals the data

value held in r_2 , but is different from the third data value stored on top of the target stack. If the guard is satisfied, the automaton outputs one or several actions that may use the data values represented by $R \cup P_A$. They may also use *fresh* data values, and we will use the parameters $Q = \{q_1, q_2, \dots\}$ as placeholders for them. Finally, the transition updates the current state, the register contents, and the stacks. Register and stack updates are allowed to use stored data values as well as fresh ones. More precisely, an *update* (wrt. A) is a tuple $\text{upd} = (\pi_1, \dots, \pi_k, u_1, \dots, u_h)$. For $i \in [k]$, $\pi_i \in R \cup P_A \cup Q$ determines the new data value stored in r_i . For example, if $\pi_i = p_1$, then r_i obtains the first value stored on top of the the target stack; if $\pi_i = r_i$, then r_i is left unchanged; if $\pi_i = q_j$, then r_i will get some fresh data value. For $t \in [h]$, u_t is a string over $\mathcal{Z}_{R \cup P_A \cup Q}$, which, instantiated with data values, is pushed onto stack t . Again, this string may use data values stored in registers (R) or the target stack (P_A), as well as fresh data values (Q). Let us formally define DMPA.

Definition 1 (data multi-pushdown automaton). Let $k \geq 0$ and $h \geq 1$. A $(k$ -register, h -stack) data multi-pushdown automaton (DMPA) over (Σ, D) is a 6-tuple $\mathcal{A} = (S, \mathcal{Z}, s_0, Z, F, \Delta)$ where S is a finite set of states, \mathcal{Z} is a finite ranked alphabet of stack symbols, $s_0 \in S$ is the initial state, $Z \in \mathcal{Z}$ is the start symbol with $\text{arity}(Z) = 0$, and $F \subseteq S$ is the set of final states. Moreover, Δ is a finite set of transitions. A transition δ is of the form

$$t:A, s \xrightarrow{\Phi, u, \text{upd}} s'$$

where $s, s' \in S$ are states, $t \in [h]$ is a stack, $A \in \mathcal{Z}$, Φ is a guard wrt. A , $u \in (\Sigma_{R \cup P_A \cup Q})^*$, and upd is an update wrt. A . We let $\Pi_\delta = R \cup P_A \cup Q_\delta$ with Q_δ the set of parameters from Q occurring in u or upd .

We let $\text{Conf}_\mathcal{A} := S \times D^k \times 2^D \times (\mathcal{Z}_D^*)^h$ denote the set of configurations of \mathcal{A} . Configuration $\gamma = [s, \mathbf{r}, U, w_1, \dots, w_h]$ with $\mathbf{r} = (d_1, \dots, d_k)$ says that the current state is s , the content of register r_i is d_i , the data values from U have already been used, and the stack contents are w_1, \dots, w_h where we assume that the topmost symbol is written last. Now, consider a transition $\delta = t:A, s \xrightarrow{\Phi, u, \text{upd}} s'$ with $\text{upd} = (\pi_1, \dots, \pi_k, u_1, \dots, u_h)$, which we call a t -transition since it pops A from stack t . It is enabled at γ if $w_t = w'_t A(d'_1, \dots, d'_m)$ and $\sigma \models \Phi$ where $\sigma : R \cup P_A \rightarrow D$ is the interpretation defined by $\sigma(r_i) = d_i$ for $i \in [k]$ and $\sigma(p_j) = d'_j$ for $j \in [m]$. In this case, for any extension of σ to Π_δ (still denoted σ) assigning to parameters in Q_δ pairwise distinct fresh values from $D \setminus U$, there is a concrete transition $\gamma \xrightarrow{\sigma(u)}_{\sigma, \delta} \gamma'$ where $\sigma(u)$ is the data word obtained from u by replacing any parameter $\pi \in \Pi_\delta$ occurring in u by $\sigma(\pi) \in D$, and

$$\gamma' = [s', (\sigma(\pi_1), \dots, \sigma(\pi_k)), U \cup \sigma(Q_\delta), w_1 \sigma(u_1), \dots, w'_t \sigma(u_t), \dots, w_h \sigma(u_h)] .$$

A configuration of the form $[s_0, (d_1, \dots, d_k), \{d_1, \dots, d_k\}, Z, \varepsilon, \dots, \varepsilon]$ with the data values d_1, \dots, d_k pairwise distinct is called *initial*, and a configuration $[s, \mathbf{r}, U, w_1, \dots, w_h]$ such that $s \in F$ is called *final*. A *run* of \mathcal{A} on $w \in \Sigma_D^*$

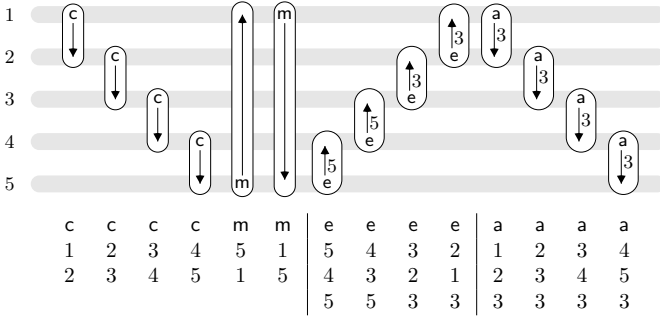


Fig. 1. A data word generated by the leader election protocol

is a sequence $\gamma_0 \xrightarrow{w_1}_{\sigma_1, \delta_1} \gamma_1 \xrightarrow{w_2}_{\sigma_2, \delta_2} \dots \xrightarrow{w_n}_{\sigma_n, \delta_n} \gamma_n$ such that $w = w_1 \dots w_n$ and γ_0 is initial. The run is *accepting* if γ_n is final. We let $L(\mathcal{A}) := \{w \in \Sigma_D^* \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}$ be the *language* of \mathcal{A} . Note that $L(\mathcal{A})$ is closed under permutation of data values.

It is easy to see that DMPA have an undecidable emptiness problem, even when we assume only two stacks as well as labels and stack symbols with arity 0. Therefore, we will restrict the number of *phases*, a notion that goes back to La Torre et al. who introduced it for multi-stack pushdown automata [15]. In one phase, one can only pop from one particular stack. Formally, for $\ell \geq 1$, a run $\gamma_0 \xrightarrow{w_1}_{\sigma_1, \delta_1} \gamma_1 \xrightarrow{w_2}_{\sigma_2, \delta_2} \dots \xrightarrow{w_n}_{\sigma_n, \delta_n} \gamma_n$ is an ℓ -*phase run* if the sequence $\delta_1 \dots \delta_n$ can be split into ℓ blocks, each block using only t -transitions for some $t \in [\text{h}]$. By $L_\ell(\mathcal{A})$, we then denote the restriction of $L(\mathcal{A})$ to data words that are accepted by ℓ -phase runs.

Remark 2. A DMPA that does not use its stacks (i.e., it never replaces the start symbol Z) corresponds to a restriction of fresh-register automata [20]. The restriction consists in allowing the automaton to read only data values that are either fresh or stored in the registers. In the terminology of [20], *local*-freshness transitions are discarded, while *global*-freshness transitions are permitted.

Example 3. We will specify a 1-register 2-stack DMPA that models the communication flow of a token-based leader election protocol. One possible behavior of the protocol is captured by the data word from Figure 1. The underlying alphabet of labels is $\Sigma = \{c, m, e, a\}$. The labels c and m have arity 2, and the labels e and a have arity 3. Data values from $D = \mathbb{N}$ will be used to model process identifiers (pids).

In the figure, a root process with pid 1 initiates a cascade of process spawns. It first executes action $c(1, 2)$, which creates a new process with pid 2. Process 2 then executes $c(2, 3)$ to create a new process 3, and so on. The number of processes created is not fixed a priori. The creation phase is followed by a message exchange between the very last process and the root, whereupon the former initiates the election phase. In the election phase, a process d non-deterministically chooses either the pid received from $d + 1$ or its own identity, and forwards it to

	s	stack	Φ	action	upd			s'
lep_1	s_0	$1:Z()$	$true$	ε	q_1	$Z()C(q_1)$	$Y()$	s_1
lep_2	s_1	$1:C(p_1)$	$true$	$c(p_1, q_1)$	r_1	$X(p_1)E(q_1, p_1)C(q_1)$	ε	s_1
lep_3	s_1	$1:C(p_1)$	$true$	$m(p_1, r_1)m(r_1, p_1)$	p_1	ε	ε	s_2
lep_4	s_2	$1:E(p_1, p_2)$	$true$	$e(p_1, p_2, r_1)$	r_1	ε	$A(p_2, p_1)$	s_2
lep_5	s_2	$1:X(p_1)$	$true$	ε	r_1	ε	ε	s_2
lep_6	s_2	$1:X(p_1)$	$true$	ε	p_1	ε	ε	s_2
lep_7	s_2	$1:Z()$	$true$	ε	r_1	ε	ε	s_3
lep_8	s_3	$2:A(p_1, p_2)$	$true$	$a(p_1, p_2, r_1)$	r_1	ε	ε	s_3
lep_9	s_3	$2:Y()$	$true$	ε	r_1	ε	ε	s_4

Fig. 2. A DMPA for the leader election protocol

$d - 1$ by executing $e(d, d - 1, \ell)$. In the following announcement phase, the pid ℓ of the elected leader is forwarded to all the processes, by executing actions of the form $a(d, d + 1, \ell)$. The figure depicted on top of the data word illustrates the creation, election, and announcement phases and the processes involved in their actions. Recall that data values have no meaning and may only be checked for equality, and we could have assumed any possible permutation of pids.

Figure 2 depicts a 1-register 2-stack DMPA $\mathcal{A}_{lep} = (S, \mathcal{Z}, s_0, Z, F, \Delta)$ for the leader election protocol. Hereby, $S = \{s_0, \dots, s_4\}$, $F = \{s_4\}$, and $\mathcal{Z} = \{Z, Y, X, C, E, A\}$ where $arity(Z) = arity(Y) = 0$, $arity(X) = arity(C) = 1$, and $arity(E) = arity(A) = 2$. Moreover, Δ contains 9 transitions, $\text{lep}_1, \dots, \text{lep}_9$.

A run of \mathcal{A}_{lep} involving four processes is given in Figure 3 (we omit the renamings involved in transitions). The transitions $\text{lep}_1, \text{lep}_2, \text{lep}_3$ put up the creation phase, represented by the upper part of the figure. For every action $c(d, d + 1)$ that is produced, $X(d)E(d + 1, d)$ is written onto the first stack to be used in the election phase. Simultaneously, the topmost stack symbol $C(d + 1)$ stores the process $d + 1$ that has to perform the following action. During the creation phase, the register stores the identity 1 so that it can later execute $m(4, 1)m(1, 4)$. The election phase is performed by transitions lep_4 to lep_6 . Here, the register stores the current leader ℓ which is sent to the next process in lep_4 with action $e(d + 1, d, \ell)$. Moreover, $A(d, d + 1)$ is written onto the second stack to prepare the announcement phase. Then, process d chooses either to preserve the current leader in lep_5 or to switch the leader to itself in lep_6 . Transition lep_7 triggers the announcement phase which is performed by lep_8 where the final leader ℓ stored in the register is sent to all processes with $a(d, d + 1, \ell)$. Note that, lep_8 causes the only control change, from the first to the second stack. We actually have $L(\mathcal{A}_{lep}) = L_2(\mathcal{A}_{lep})$. \square

	$[s_0 \ 0 \ \{0\}$	$Z()$	$\varepsilon]$
$\xrightarrow{\varepsilon}$	lep_1	$[s_1 \ 1 \ \{0, 1\}$	$Z()C(1) \quad Y()$
$\xrightarrow{c(1,2)}$	lep_2	$[s_1 \ 1 \ \{0, 1, 2\}$	$Z()X(1)E(2, 1)C(2) \quad Y()$
$\xrightarrow{c(2,3)}$	lep_2	$[s_1 \ 1 \ \{0, \dots, 3\}$	$Z()X(1)E(2, 1)X(2)E(3, 2)C(3) \quad Y()$
$\xrightarrow{c(3,4)}$	lep_2	$[s_1 \ 1 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1)X(2)E(3, 2)X(3)E(4, 3)C(4) \quad Y()$
$\xrightarrow{m(4,1)m(1,4)}$	lep_3	$[s_2 \ 4 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1)X(2)E(3, 2)X(3)E(4, 3) \quad Y()$
$\xrightarrow{e(4,3,4)}$	lep_4	$[s_2 \ 4 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1)X(2)E(3, 2)X(3) \quad Y()A(3, 4)$
$\xrightarrow{\varepsilon}$	lep_5	$[s_2 \ 4 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1)X(2)E(3, 2) \quad Y()A(3, 4)$
$\xrightarrow{e(3,2,4)}$	lep_4	$[s_2 \ 4 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1)X(2) \quad Y()A(3, 4)A(2, 3)$
$\xrightarrow{\varepsilon}$	lep_6	$[s_2 \ 2 \ \{0, \dots, 4\}$	$Z()X(1)E(2, 1) \quad Y()A(3, 4)A(2, 3)$
$\xrightarrow{e(2,1,2)}$	lep_4	$[s_2 \ 2 \ \{0, \dots, 4\}$	$Z()X(1) \quad Y()A(3, 4)A(2, 3)A(1, 2)$
$\xrightarrow{\varepsilon}$	lep_5	$[s_2 \ 2 \ \{0, \dots, 4\}$	$Z() \quad Y()A(3, 4)A(2, 3)A(1, 2)$
$\xrightarrow{\varepsilon}$	lep_7	$[s_3 \ 2 \ \{0, \dots, 4\}$	$\varepsilon \quad Y()A(3, 4)A(2, 3)A(1, 2)$
$\xrightarrow{a(1,2,2)}$	lep_8	$[s_3 \ 2 \ \{0, \dots, 4\}$	$\varepsilon \quad Y()A(3, 4)A(2, 3)$
$\xrightarrow{a(2,3,2)}$	lep_8	$[s_3 \ 2 \ \{0, \dots, 4\}$	$\varepsilon \quad Y()A(3, 4)$
$\xrightarrow{a(3,4,2)}$	lep_8	$[s_3 \ 2 \ \{0, \dots, 4\}$	$\varepsilon \quad Y()$
$\xrightarrow{\varepsilon}$	lep_9	$[s_4 \ 2 \ \{0, \dots, 4\}$	$\varepsilon \quad \varepsilon]$

Fig. 3. A run of the leader election protocol

3 Monadic Second-Order Logic

While DMPA serve as system models, we use monadic second-order logic to specify properties of data words. We assume countably infinite supplies of first-order and second-order variables. We let x, y, \dots denote first-order variables, which vary over word positions, and we use X, Y, \dots to denote second-order variables, which vary over sets of positions.

Definition 4 (MSO logic over data words). *The class $\text{MSO}_{\text{d-word}}(\Sigma, D)$ of monadic second-order (MSO) formulas over data words is given by the following grammar, where a ranges over Σ , and $1 \leq k, l \leq \max(\text{arity}(\Sigma))$:*

$$\varphi ::= a(x) \mid d_k(x) = d_l(y) \mid x \leq y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

Formula $a(x)$ holds in a data word $w \in \Sigma_D^*$ if $\text{label}(i) = a$ when x is interpreted as $i \in \text{dom}(w)$. Formula $d_k(x) = d_l(y)$ is satisfied wrt. interpretations i and j of x and y , respectively, if $k \leq \text{arity}(i)$, $l \leq \text{arity}(j)$, and $\text{data}_k(i) = \text{data}_l(j)$. Formula $x \leq y$, the boolean connectives, and quantifiers are self-explanatory. We also use the usual abbreviations $x < y, \forall x\varphi, \varphi \rightarrow \psi \dots$

For a data word w and a formula $\varphi(x_1, \dots, x_n, X_1, \dots, X_m)$ with free variables in $\{x_1, \dots, x_n, X_1, \dots, X_m\}$, we write $w, i_1, \dots, i_n, I_1, \dots, I_m \models \varphi$ if φ evaluates to true when interpreting the first-order variables by $i_1, \dots, i_n \in \text{dom}(w)$ and the second-order variables by $I_1, \dots, I_m \subseteq \text{dom}(w)$, respectively.

If φ is a sentence, i.e., it does not have any free variable, then we set $L(\varphi)$ to be the set of data words w such that $w \models \varphi$.

Example 5. We define a property satisfied by the 1-register 2-stack DMPA \mathcal{A}_{lep} from Example 3 modeling the leader election protocol. To express that every new process will eventually receive an announcement containing a unique leader pid, we write $\varphi = \exists z \forall x (c(x) \rightarrow \exists y (a(y) \wedge x \leq y \wedge d_2(x) = d_2(y) \wedge d_1(z) = d_3(y)))$. We have $L(\mathcal{A}_{lep}) = L_2(\mathcal{A}_{lep}) \subseteq L(\varphi)$. \square

Example 6. This example will show that MSO logic can be used to abstract from the linear order of a data word to model partially ordered behaviors.

We model concurrent programs where processes can fork other processes and exchange messages via send and receive primitives. Unlike in the leader election protocol, we will model sends and receives separately, which allows us a finer treatment when we formulate MSO properties. Again, processes have unique pids, which are modeled as data values. We let $D = \mathbb{N}$ be the pids and $\Sigma = \{s, f, !, ?\}$ be the labels. Label s takes one pid d , and $s(d) \in \Sigma_D$ indicates that d has just started its execution. Labels f , $!$, and $?$ each take two arguments, one for the executing (i.e., forking, sending, receiving) process, and one for the communication partner (i.e., the new, receiving, sending process, respectively). In particular, $f(c, d)$ is matched by $s(d)$, and $!(c, d)$ is matched by $?(d, c)$. Figure 4 shows two data words, w_1 and w_2 . The graphs above them illustrate their interpretation as the execution of a concurrent program, connecting a fork with a corresponding start action and a send with a matching receive. This connection will, in the following, be formalized in terms of MSO logic.

Rather than the linear order of the data word, we are interested in the causal dependencies in the underlying concurrent execution modeled by this data word, and these can be captured via MSO formulas. Let $x <_{\text{proc}} y$ be a shorthand for $x < y \wedge d_1(x) = d_1(y)$, which denotes that there is a process that executes first x and later y (we may say that a word position is “executed”, as it is considered as a system event). In Figure 4, the relation induced by $x <_{\text{proc}} y$ is given in terms of the transitive closure of the horizontal edges. For example, $w_1, 1, 2 \models x <_{\text{proc}} y$ and $w_1, 1, 6 \models x <_{\text{proc}} y$. Now, consider formula $x <_{\text{m}} y$, which stands for $!(x) \wedge ?(y) \wedge x < y \wedge d_1(x) = d_2(y) \wedge d_2(x) = d_1(y)$. We assume a bound 1 on the channel capacities. To say that x and y form a message, we let $x <_{\text{msg}} y$ abbreviate $x <_{\text{m}} y \wedge \neg \exists z (x < z < y \wedge (x <_{\text{m}} z \vee z <_{\text{m}} y))$. For example, we have $w_1, 6, 7 \models x <_{\text{msg}} y$. Let us relate a fork position with the first position executed by the new process: $x <_{\text{fork}} y$ stands for $f(x) \wedge s(y) \wedge d_2(x) = d_1(y)$. For example, $w_1, 4, 5 \models x <_{\text{fork}} y$. To define causal dependencies between positions of a data word, we let $<_{\text{causal}}$ denote the transitive closure of the relation $<_{\text{proc}} \cup <_{\text{msg}} \cup <_{\text{fork}}$. It corresponds to the transitive closure of the edge relation depicted in Figure 4. Note that the transitive closure of an MSO definable binary relation is indeed MSO definable [7]. For example, $w_1, 1, 7 \models x <_{\text{causal}} y$, but $w_1, 4, 6 \not\models x <_{\text{causal}} y$ and $w_1, 6, 4 \not\models x <_{\text{causal}} y$.

Like in the leader election protocol, we assume a system architecture that allows us to pass pids along messages or process forks. When we consider concrete implementations of such concurrent programs, it is crucial that a process sending a message to another knows the pid of the receiving process. We will determine an MSO formula $\varphi_{\text{realizable}}$ that checks a system for such consistency (*realizability* in

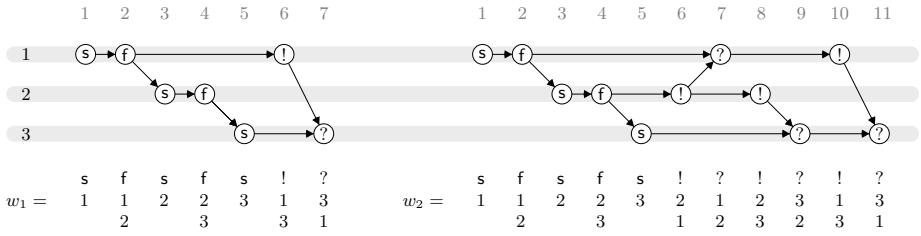


Fig. 4. Two data words

the terminology of [5]). It uses a formula $knows(x, y)$, which holds if the process executing x , right before performing the action, knows the pid of the process executing y . Regarding Figure 4, we would like to have $w_1, 6, 7 \not\models knows(x, y)$, as there is no way to communicate pid 3 to the process with pid 1. On the other hand, we will have $w_2, 10, 11 \models knows(x, y)$ as pid 3 can be communicated to process 1 along the message from 2 (which spawned 3) to 1. We first describe a formula $x <_{flow} y$, which intuitively says that there is some flow of information from position x to position y . In other words, pids can be passed from the process executing x to the process executing y . We set $<_{flow}$ to be the transitive closure of $<_{causal} \cup <_{fork}^{-1}$. For example, $w_1, 3, 6 \models x <_{flow} y$, but $w_1, 4, 6 \not\models x <_{flow} y$. Now, we set $knows(x, y)$ to be $\exists y' (y' <_{proc} y \wedge flow(y', x))$. Finally, we consider our system to be realizable if it satisfies $\varphi_{realizable} := \forall x \forall y (x <_{msg} y \rightarrow knows(x, y))$. We have $w_1 \not\models \varphi_{realizable}$ but $w_2 \models \varphi_{realizable}$. \square

4 Model Checking DMPA

We now present our main result, decidability of the model-checking problem for (bounded control change) DMPA wrt. MSO logic:

Theorem 7. *For a DMPA \mathcal{A} over (Σ, D) , a natural number $\ell \geq 1$, and a sentence $\varphi \in MSO_{d-word}(\Sigma, D)$, one can decide if $L_\ell(\mathcal{A}) \subseteq L(\varphi)$.*

The rest of this section is devoted to the proof of Theorem 7, which we outline in the following. First, we represent a run ρ of a DMPA as a (multiply) nested data word. The nested data word associated with ρ is the concatenation of the instantiations of transitions used in ρ . In addition, it has nesting edges from a pushed stack symbol to the position where it is popped. There is a precise correspondence between (ℓ -phase) runs and (certain ℓ -phase) nested data words. Moreover, the data word generated by ρ will be exactly the word projection (without nesting edges) of its nested word onto the alphabet Σ_D . Next, we look at abstract nested words, which, instead of data values, contain the parameters used in the run. We, therefore, deal with nested words over a finite alphabet. A nested data word and the abstract version corresponding to a run are depicted in Figure 5. The trick is now that we can, using the nesting edges, define

MSO formulas over abstract nested words that recover equality of data values in the concrete version. As the set of abstract nested words that correspond to accepting runs of the DMPA is also definable in MSO logic, we reduce, in this way, the model-checking problem for a DMPA to a satisfiability problem over abstract nested words. Satisfiability of MSO formulas over ℓ -phase nested words is decidable due to [15] so that the theorem follows.

Nested Words. Let $\mathfrak{h} \geq 1$. An \mathfrak{h} -stack alphabet is a (possibly infinite) alphabet Γ together with mappings $stack : \Gamma \rightarrow \{0, 1, \dots, \mathfrak{h}\}$ and $type : \Gamma \rightarrow \{\text{push}, \text{pop}, \text{int}\}$ such that, for all $a \in \Gamma$, we have $type(a) = \text{int}$ iff $stack(a) = 0$. Given $w = a_1 \dots a_n \in \Gamma^*$ and $i \in \text{dom}(w)$, we let $stack(i) = stack(a_i)$ and $type(i) = type(a_i)$. For $t \in [\mathfrak{h}]$, we call $w \in \Gamma^*$ *t*-well-nested if it can be generated by the context-free grammar $A ::= aAb \mid AA \mid \varepsilon \mid c$ where $a, b, c \in \Gamma$ are such that $stack(a) = stack(b) = t \neq stack(c)$, $type(a) = \text{push}$, and $type(b) = \text{pop}$.

A nested word over Γ is a pair $W = (w, \curvearrowright)$ where $w \in \Gamma^*$ and $\curvearrowright \subseteq \text{dom}(w) \times \text{dom}(w)$ is the binary *matching relation*, which is uniquely determined as follows: for all $i, j \in \text{dom}(w)$, $i \curvearrowright j$ iff $i < j$ and there is $t \in [\mathfrak{h}]$ such that $stack(i) = stack(j) = t$, $type(i) = \text{push}$, $type(j) = \text{pop}$, and $a_{i+1} \dots a_{j-1}$ is *t*-well-nested. Note that there might be push or pop positions that are not matched wrt. \curvearrowright . The set of nested words over Γ is denoted by $Nested(\Gamma)$.

Let $\ell \geq 1$. A nested word (w, \curvearrowright) is an ℓ -phase nested word if w can be written as $w_1 \dots w_\ell$ with $w_i \in \Gamma^*$ where, for all $i \in \{1, \dots, \ell\}$, there is $t \in [\mathfrak{h}]$ such that, for each letter $a \in \Gamma$ that occurs in w_i , $type(a) = \text{pop}$ implies $stack(a) = t$.

The class $\text{MSO}_{\text{nw}}(\Gamma)$ of MSO formulas over nested words is given by the following grammar, where a ranges over Γ :

$$\varphi ::= a(x) \mid x \curvearrowright y \mid x \leq y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

The atomic predicates are interpreted over a nested word $W = (a_1 \dots a_n, \curvearrowright)$ as follows: $W, i \models a(x)$ if $a_i = a$, $W, i, j \models x \curvearrowright y$ if $i \curvearrowright j$, and $W, i, j \models x \leq y$ if $i \leq j$. The other connectives are as expected.

Theorem 8 (La Torre et al. [15]). *Given a finite \mathfrak{h} -stack alphabet Γ , $\ell \geq 1$, and a sentence $\varphi \in \text{MSO}_{\text{nw}}(\Gamma)$, one can decide if there is an ℓ -phase nested word W over Γ such that $W \models \varphi$.*

Nested Data Words. Next, we define nested words carrying data values. Let Γ be a finite *ranked* \mathfrak{h} -stack alphabet, i.e., every letter $a \in \Gamma$ has some *arity* $_{\Gamma}(a) \in \mathbb{N}$. We can interpret Γ_D as an infinite \mathfrak{h} -stack alphabet in the obvious manner. A *nested data word* over (Γ, D) is a nested word over Γ_D . Notions from data words such as $\text{dom}(w)$ and $data_k(i)$ can be transferred to nested data words $W = (w, \curvearrowright)$ by applying them to the w -component.

The set $\text{MSO}_{\text{d-nw}}(\Gamma, D)$ of MSO formulas over nested data words is given by the following grammar, where a ranges over Γ , and $1 \leq k, l \leq \max(\text{arity}(\Gamma))$:

$$\varphi ::= a(x) \mid d_k(x) = d_l(y) \mid x \curvearrowright y \mid x \leq y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

We omit the definition of the semantics, which is as expected.

Suppose $\Sigma \subseteq \Gamma$. Given a nested data word W over (Γ, D) , we denote by $\text{Proj}_\Sigma(W)$ the data word from Σ_D^* obtained by restricting (or projecting) W to Σ_D and discarding \curvearrowright . Using a simple relativization, we obtain:

Proposition 9. *Let $\Sigma \subseteq \Gamma$ and $\varphi \in \text{MSO}_{\text{d-word}}(\Sigma, D)$ be a sentence. We can effectively construct a sentence $\tilde{\varphi} \in \text{MSO}_{\text{d-nw}}(\Gamma, D)$ such that, for all $W \in \text{Nested}(\Gamma_D)$, we have $W \models \tilde{\varphi}$ iff $\text{Proj}_\Sigma(W) \models \varphi$.*

Parse Words. Let $\ell \geq 1$ and $\mathcal{A} = (S, \mathcal{Z}, s_0, Z, F, \Delta)$ be a (\mathbb{k} -register, \mathbb{h} -stack) DMPA over (Σ, D) . Without loss of generality, we assume that there is a mapping $\text{stack} : \mathcal{Z} \rightarrow [\mathbb{h}]$ such that each stack symbol $A \in \mathcal{Z}$ is written on/removed from $\text{stack}(A)$ only. We assume $\text{stack}(Z) = 1$. We define a finite ranked \mathbb{h} -stack alphabet $\Gamma = \Sigma \uplus \mathcal{Z} \uplus \overline{\mathcal{Z}} \uplus S \uplus \overline{S}$ where $\overline{\mathcal{Z}} = \{\overline{A} \mid A \in \mathcal{Z}\}$ and $\overline{S} = \{\overline{s} \mid s \in S\}$ contain a marked copy of every letter from \mathcal{Z} and S , respectively. We retain the arities defined by the alphabets Σ and \mathcal{Z} . We let $\text{arity}_\Gamma(\overline{s}) = \text{arity}_\Gamma(s) = \mathbb{k}$ for all $s \in S$ and $\text{arity}_\Gamma(\overline{A}) = \text{arity}_\Gamma(A)$ for all $A \in \mathcal{Z}$. Moreover, $\text{type}(a) = \text{int}$ and $\text{stack}(a) = 0$ for all $a \in \Sigma \cup S \cup \overline{S}$. Finally, $\text{stack}(\overline{A}) = \text{stack}(A)$, $\text{type}(A) = \text{push}$, and $\text{type}(\overline{A}) = \text{pop}$ for all $A \in \mathcal{Z}$.

Let $\mathcal{O} \subset \Pi$ denote the finite set of parameters occurring in \mathcal{A} . Recall that, by $\Gamma_{\mathcal{O}}$, we denote the (finite) \mathbb{h} -stack alphabet $\{a(\pi_1, \dots, \pi_m) \mid a \in \Gamma, m = \text{arity}_\Gamma(a), \text{ and } \pi_1, \dots, \pi_m \in \mathcal{O}\}$. For $b = a(\pi_1, \dots, \pi_m) \in \Gamma_{\mathcal{O}}$ and $1 \leq k \leq m$, we denote by $\text{par}_k(b)$, its k -th parameter π_k .

We are now ready to define the (abstract and concrete) parse words of \mathcal{A} . Consider a transition $\delta = t:A, s \xrightarrow{\overline{\varphi}, u, \text{upd}} s'$ with $\text{upd} = (\pi_1, \dots, \pi_k, u_1, \dots, u_{\mathbb{h}})$. Note that $t = \text{stack}(A)$ by assumption. Let $m = \text{arity}_\Gamma(A)$. We define the string of δ as $\text{string}(\delta) := \overline{s}(r_1, \dots, r_k)\overline{A}(p_1, \dots, p_m)uu_1 \dots u_{\mathbb{h}}s'(\pi_1, \dots, \pi_k) \in \Gamma_{\mathcal{O}}^*$. For instance, for the DMPA \mathcal{A}_{lep} from Example 3, we have

$$\begin{aligned} \text{string}(\text{lep}_1) &= \overline{s}_0(r_1)\overline{Z}()C(q_1)Y()s_1(q_1) \\ \text{string}(\text{lep}_2) &= \overline{s}_1(r_1)\overline{C}(p_1)c(p_1, q_1)X(p_1)E(q_1, p_1)C(q_1)s_1(r_1) \end{aligned}$$

For an interpretation $\sigma : \mathcal{O} \rightarrow D$, we define similarly the string of the concrete transition $\sigma(\delta)$ with data values $\sigma(\pi)$ substituted for parameters $\pi \in \mathcal{O}$. It is denoted $\text{string}(\sigma(\delta))$. Note that the string of a transition does not consider guards. Guards are taken into account later, in Proposition 14.

Consider a run ρ of \mathcal{A} of the form

$$\gamma_0 \xrightarrow{w_1} \sigma_1, \delta_1 \quad \gamma_1 \xrightarrow{w_2} \sigma_2, \delta_2 \quad \dots \quad \xrightarrow{w_n} \sigma_n, \delta_n \quad \gamma_n$$

The nested word (w, \curvearrowright) with $w = Z()s_0(r_1, \dots, r_k)\text{string}(\delta_1) \dots \text{string}(\delta_n) \in \Gamma_{\mathcal{O}}^*$ is the *abstract parse word* of ρ and denoted apw_ρ . The nested data word (w', \curvearrowright) with $w' = Z()s_0(\sigma_1(r_1), \dots, \sigma_1(r_k))\text{string}(\sigma_1(\delta_1)) \dots \text{string}(\sigma_n(\delta_n)) \in \Gamma_D^*$ is the *concrete parse word* of ρ denoted pw_ρ . Notice that $\text{dom}(\text{pw}_\rho) = \text{dom}(\text{apw}_\rho)$. Moreover, ρ is ℓ -phase iff pw_ρ is ℓ -phase iff apw_ρ is ℓ -phase.

Figure 5 illustrates an abstract parse word and a concrete parse word of the run $\text{lep}_1\text{lep}_2\text{lep}_2\text{lep}_3\text{lep}_4\text{lep}_6\text{lep}_4\text{lep}_5\text{lep}_7\text{lep}_8\text{lep}_8\text{lep}_9$ of DMPA \mathcal{A}_{lep} from Example 3. The curved lines depict the nesting relation \curvearrowright (straight lines for stack 1, dotted lines for stack 2).

	Z	s ₀	s ₀	Z	Z	C	Y	s ₁	s ₁	C	c	X	E	C	s ₁	s ₁	C	c	X	E	C	s ₁	s ₁	C	m	m	s ₂	
abstract	1	r ₁	r ₁					q ₁	q ₁	r ₁	p ₁	p ₁	p ₁	q ₁	q ₁	r ₁	r ₁	p ₁	p ₁	p ₁	q ₁	q ₁	r ₁	r ₁	p ₁	p ₁	r ₁	p ₁
	2													q ₁	p ₁													
	3																											
concrete	1	0	0					1	1	1	1	1	2	2	1	1	2	2	2	3	3	1	1	3	3	1	3	3
	2													2	1													
	3																											
Trans:				lep ₁						lep ₂					lep ₂					lep ₃								
Block:		0		1						2					3					4								

s ₂	E	e	A	s ₂	s ₂	X	s ₂	s ₂	E	e	A	s ₂	s ₂	X	s ₂	s ₂	Z	s ₃	s ₃	A	a	s ₃	s ₃	A	a	s ₃	s ₂	Y	s ₄	
r ₁	p ₁	p ₁	p ₂	r ₁	r ₁	p ₁	p ₁	r ₁	r ₁	p ₁	p ₂	r ₁	r ₁	p ₁	r ₁	r ₁	r ₁	r ₁	r ₁	r ₁	p ₁	p ₁	r ₁	r ₁	p ₁	p ₁	r ₁	r ₁	r ₁	
	p ₂	p ₂	p ₁																		p ₂	p ₂								
			r ₁																											
3	3	3	2	3	3	2	2	2	2	1	2	2	1	2	2	2	2	2	2	2	1	1	2	2	2	2	2	2	2	2
	2	2	3							1	1	2									2	2			3	3				
			3								2											2								
	lep ₄			lep ₆			lep ₄			lep ₅			lep ₇			lep ₈			lep ₈			lep ₉								
	5			6			7			8			9			10			11			12								

Fig. 5. An abstract and a concrete parse word (split over two lines)

The data word $\sigma(u)$ generated by a concrete transition $\sigma(\delta)$ is precisely the Σ -projection of $string(\sigma(\delta))$. Hence, the data word generated by a run ρ is $Proj_{\Sigma}(pw_{\rho})$. The data words accepted by \mathcal{A} with ℓ -phase runs are the Σ -projections of the concrete parse words of these runs:

Proposition 10. *We have $L_{\ell}(\mathcal{A}) = \{Proj_{\Sigma}(pw_{\rho}) \in \Sigma_D^* \mid \rho \text{ is an } \ell\text{-phase accepting run of } \mathcal{A}\}$.*

An abstract parse word is an abstraction of several concrete parse words. Our aim is to recover from an abstract parse word all data equalities that hold in the concrete parse word. To do so, we will define formulas $\varphi_{k,l}(x, y) \in MSO_{nw}(I_{\mathcal{O}})$ for all $1 \leq k, l \leq \max(arity(\Gamma))$, with free variables x and y . Intuitively, $\varphi_{k,l}(x, y)$ will hold in an abstract parse word iff $d_k(x) = d_l(y)$ holds in any corresponding concrete parse word.

We first give some definitions and macros. Block 0 of an abstract parse word consists of the first two positions, which are labelled $Z()$ and $s_0(r_1, \dots, r_k)$ respectively. Then, we find a concatenation of blocks of the form $string(\delta) = \bar{s}(r_1, \dots, r_k)\bar{A}(p_1, \dots, p_m)vs'(\pi_1, \dots, \pi_k)$ for some transition δ (see Figure 5). For any position x , we denote by $Block(x)$ the block of x . We use $Block(x) = Block(y)$ to state that x and y belong to the same block, which can be expressed by the following first-order formula: $x, y \leq 2 \vee \exists x', y' (x' \leq x, y \leq y' \wedge \bigvee_{\bar{s} \in \bar{S}_{\mathcal{O}}} \bar{s}(x') \wedge$

$\bigvee_{s \in S_{\mathcal{O}}} s(y') \wedge \forall z ((z < y' \wedge \bigvee_{\bar{s} \in \bar{S}_{\mathcal{O}}} \bar{s}(z) \rightarrow z \leq x')$). Moreover, we will use the macro $\text{Block}(x) \leq \text{Block}(y) := x \leq y \vee \text{Block}(x) = \text{Block}(y)$.

Let par_k denote the k -th parameter of position x of the abstract parse word. The macro $\text{par}_k(x) = \text{par}_l(y)$ says that x and y carry the same parameter at indices k and l , respectively. It is the disjunction of formulas $b(x) \wedge b'(y)$ where $b, b' \in \Gamma_{\mathcal{O}}$ are such that $\text{par}_k(b) = \text{par}_l(b')$. We also let $\text{existspar}_k(x)$ be the disjunction of formulas $b(x)$ where $b \in \Gamma_{\mathcal{O}}$ is such that $\text{arity}(b) \geq k$.

Let us see how to propagate a data value to a *later block*. Clearly, we have $d_k(x) = d_l(y)$ in the concrete parse word if in the abstract parse word the formula

$$\psi_{k,l}(x, y) := \left(\begin{array}{l} \text{Block}(x) = \text{Block}(y) \wedge \text{par}_k(x) = \text{par}_l(y) \\ \vee \text{Block}(x) \neq \text{Block}(y) \wedge x + 1 = y \wedge k = l \leq \mathbb{k} \\ \vee x \curvearrowright y \wedge \bigvee_{a=A(\dots) \in \mathcal{Z}_{\mathcal{O}}} (a(x) \wedge k = l \leq \text{arity}(A)) \end{array} \right)$$

holds. Note that $\psi_{k,l}(x, y)$ implies $\text{Block}(x) \leq \text{Block}(y)$ and that $\psi_{k,k}(x, x)$ is equivalent to $\text{existspar}_k(x)$. Data equality is also ensured if we can reach y from x using a sequence of $\psi_{i,j}$ steps. It is well-known that such a “transitive closure” can be defined in MSO: we let $m = \max(\text{arity}(\Gamma))$ and define $x \rightsquigarrow_l y$ by

$$\text{existspar}_k(x) \wedge \forall X_1, \dots, X_m \left(x \in X_k \wedge \bigwedge_{1 \leq i, j \leq m} \forall z_1, z_2 (z_1 \in X_i \wedge \psi_{i,j}(z_1, z_2)) \rightarrow z_2 \in X_j \right) \rightarrow y \in X_l$$

so that we have $\text{apw}_{\rho}, i, j \models x \rightsquigarrow_l y$ iff for some $n > 0$, there are sequences $i = i_0, i_1, \dots, i_n = j$ and $k = k_0, k_1, \dots, k_n = l$ such that $\text{apw}_{\rho}, i_p, i_{p+1} \models \psi_{k_p, k_{p+1}}(x, y)$ for all $0 \leq p < n$. Therefore, $x \rightsquigarrow_l y$ in the abstract parse word implies $d_k(x) = d_l(y)$ in the concrete parse word.

For the general case, we will prove that $d_k(x) = d_l(y)$ in the concrete parse word iff there exists a position z such that the i -th value of z was propagated to x as its k -th value and to y as its l -th value. So we define

$$\varphi_{k,l}(x, y) := \exists z \bigvee_i z \rightsquigarrow_k x \wedge z \rightsquigarrow_l y$$

Example 11. We will see how the formulas defined above retrieve data equality on the abstract parse word from Figure 5. Let us check whether the data at the third component of e in Block 5 is same as the data at the second component of a in Block 11, i.e., whether r_1 in Block 5 and p_2 of Block 11 hold the same value. First, p_2 in Block 11 equals p_1 in Block 5, which equals q_1 in Block 3, which is fresh. Similarly, r_1 in Block 5 is the same as p_1 in Block 4, which equals q_1 in Block 3, which is fresh. Hence, from q_1 of Block 3, we can reach both r_1 in Block 5 and p_2 in Block 11 by \rightsquigarrow , thus concluding they hold the same data value. \square

We can prove that the formulas $\varphi_{k,l}(x, y)$ defined above are indeed correct.

Proposition 12. *For all runs ρ of \mathcal{A} and all positions $i, j \in \text{dom}(\text{pw}_{\rho})$, we have $\text{pw}_{\rho}, i, j \models d_k(x) = d_l(y)$ iff $\text{apw}_{\rho}, i, j \models \varphi_{k,l}(x, y)$.*

Corollary 13. *For every sentence $\xi \in \text{MSO}_{\text{d-nw}}(\Gamma, D)$, we can effectively construct a sentence $\widehat{\xi} \in \text{MSO}_{\text{nw}}(\Gamma_{\mathcal{O}})$ such that, for all runs ρ of \mathcal{A} , we have $\text{pw}_{\rho} \models \xi$ iff $\text{apw}_{\rho} \models \widehat{\xi}$.*

We obtain $\widehat{\xi}$ by replacing every occurrence of $d_k(x) = d_l(y)$ in ξ with $\varphi_{k,l}(x, y)$, and every occurrence of $a(x)$ with the disjunction of formulas $b(x)$ where $b = a(\pi_1, \dots, \pi_m) \in \Gamma_{\mathcal{O}}$. The result then follows from Proposition 12.

The last proposition needed for our proof says that the set of abstract parse words of ℓ -phase accepting runs of \mathcal{A} is MSO definable. The MSO formula has to make sure that there is a suitable assignment of blocks to guards such that the guards are satisfied. Satisfaction is checked using the formulas $\varphi_{k,l}(x, y)$.

Proposition 14. *There is $\psi \in \text{MSO}_{\text{nw}}(\Gamma_{\mathcal{O}})$ such that $L(\psi) = \{\text{apw}_{\rho} \mid \rho \text{ is an } \ell\text{-phase accepting run of } \mathcal{A}\}$.*

Proof (of Theorem 7). Let $\varphi \in \text{MSO}_{\text{d-word}}(\Sigma, D)$. We consider the formula $\widehat{\varphi} \in \text{MSO}_{\text{nw}}(\Gamma_{\mathcal{O}})$ obtained from Proposition 9 and Corollary 13. We show that $L_{\ell}(\mathcal{A}) \subseteq L(\varphi)$ iff the formula $\psi \rightarrow \widehat{\varphi}$ is valid over $\text{Nested}(\Gamma_{\mathcal{O}})$ where ψ is from Proposition 14. This validity is decidable due to Theorem 8.

\Rightarrow : Assume $L_{\ell}(\mathcal{A}) \subseteq L(\varphi)$. Let $W \in \text{Nested}(\Gamma_{\mathcal{O}})$ be such that $W \models \psi$. By Proposition 14, there is an ℓ -phase accepting run ρ of \mathcal{A} such that $W = \text{apw}_{\rho}$. By Proposition 10, we get $\text{Proj}_{\Sigma}(\text{pw}_{\rho}) \in L_{\ell}(\mathcal{A})$. Hence $\text{Proj}_{\Sigma}(\text{pw}_{\rho}) \models \varphi$ and, by Proposition 9, we get $\text{pw}_{\rho} \models \widehat{\varphi}$. Finally, Corollary 13 implies $W = \text{apw}_{\rho} \models \widehat{\varphi}$.

\Leftarrow : Assume that $\psi \rightarrow \widehat{\varphi}$ is valid. Let $w \in L_{\ell}(\mathcal{A})$. By Proposition 10, there is an ℓ -phase accepting run ρ of \mathcal{A} such that $w = \text{Proj}_{\Sigma}(\text{pw}_{\rho})$. By Proposition 14, we have $\text{apw}_{\rho} \models \psi$ and since $\psi \rightarrow \widehat{\varphi}$ is valid we get $\text{apw}_{\rho} \models \widehat{\varphi}$. We obtain $\text{pw}_{\rho} \models \widehat{\varphi}$ by Corollary 13 and, finally, $w = \text{Proj}_{\Sigma}(\text{pw}_{\rho}) \models \varphi$ by Proposition 9. \square

5 Conclusion

In this paper, we introduced DMPA and showed that their model-checking problem is decidable wrt. the full MSO logic over data words. Note that this contributes to the area of parametrized verification [1], as model checking can prove that a property holds for any number of processes.

An important next step is to bridge the gap between DMPA specifications and concrete implementations, for example in terms of automata with pid-passing capabilities [4, 5]. Recall that the leader election protocol requires a context-sensitive specification when we define its behavior globally. However, it can be implemented as a finite-state system when we assume several local copies of processes that can send and receive messages as well as process identities. It remains to identify classes of DMPA that can be implemented in this way.

Recall that our main result relies on Theorem 8, whose proof [15, 18] essentially shows that ℓ -phase nested words have bounded tree width. It would be worthwhile to study if one can reduce our model-checking problem to a satisfiability problem for MSO logic over some class of graphs of bounded tree width or bounded clique width.

References

1. Abdulla, P.A.: Forcing Monotonicity in Parameterized Verification: From Multisets to Words. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 1–15. Springer, Heidelberg (2010)
2. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. *ACM Trans. Comput. Log.* 12(4), 27 (2011)
3. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and applications to XML reasoning. *J. ACM* 56(3) (2009)
4. Bollig, B.: An automaton Over Data Words that Captures EMSO Logic. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011 – Concurrency Theory. LNCS, vol. 6901, pp. 171–186. Springer, Heidelberg (2011)
5. Bollig, B., Hélouët, L.: Realizability of Dynamic MSC Languages. In: Ablayev, F.M., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 48–59. Springer, Heidelberg (2010)
6. Cheng, E.Y.C., Kaminski, M.: Context-free languages over infinite alphabets. *Acta Inf.* 35(3), 245–267 (1998)
7. Courcelle, B.: Graph rewriting: an algebraic and logic approach. In: *Handbook of Theoretical Computer Science*, vol. B, pp. 193–242. MIT Press (1990)
8. David, C., Libkin, L., Tan, T.: On the Satisfiability of Two-Variable Logic Over Data Words. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 248–262. Springer, Heidelberg (2010)
9. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic* 10(3) (2009)
10. Demri, S., Lazić, R., Sangnier, A.: Model Checking Freeze LTL Over One-Counter Automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 490–504. Springer, Heidelberg (2008)
11. Demri, S., Sangnier, A.: When Model-Checking Freeze LTL over Counter Machines Becomes Decidable. In: Ong, C.-H.L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 176–190. Springer, Heidelberg (2010)
12. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable Automata over Infinite Alphabets. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010)
13. Kaminski, M., Francez, N.: Finite-memory automata. *Theoretical Computer Science* 134(2), 329–363 (1994)
14. Kaminski, M., Zeitlin, D.: Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.* 21(5), 741–760 (2010)
15. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: *LICS 2007*, pp. 161–170. IEEE Computer Society Press (2007)
16. La Torre, S., Madhusudan, P., Parlato, G.: Context-Bounded Analysis of Concurrent Queue Systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
17. Leucker, M., Madhusudan, P., Mukhopadhyay, S.: Dynamic Message Sequence Charts. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 253–264. Springer, Heidelberg (2002)
18. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Ball, T., Sagiv, M. (eds.) POPL 2011, pp. 283–294. ACM (2011)
19. Niewerth, M., Schwentick, T.: Two-variable logic and key constraints on data words. In: Milo, T. (ed.) ICDT 2011, pp. 138–149. ACM (2011)
20. Tzevelekos, N.: Fresh-register automata. In: Ball, T., Sagiv, M. (eds.) POPL 2011, pp. 295–306. ACM (2011)

Branching-Time Model Checking of Parametric One-Counter Automata

Stefan Göller¹, Christoph Haase², Joël Ouaknine², and James Worrell²

¹ Institut für Informatik, Universität Bremen, Germany

² Department of Computer Science, University of Oxford, UK

Abstract. We study the computational complexity of model checking EF logic and modal logic on parametric one-counter automata (POCA). A POCA is a one-counter automaton whose counter updates are either integer values encoded in binary or integer-valued parameters. Given a formula and a configuration of a POCA, the model-checking problem asks whether the formula is true in this configuration for all possible valuations of the parameters. We show that this problem is undecidable for EF logic via reduction from Hilbert’s tenth problem, however for modal logic we prove PSPACE-completeness. Obtaining the PSPACE upper bound involves analysing systems of linear Diophantine inequalities of exponential size that admit solutions of polynomial size. Finally, we show that model checking EF logic on POCA without parameters is PSPACE-complete.

1 Introduction

Counter automata, a fundamental and widely-studied model of computation, consist of a finite-state controller which manipulates a finite set of counters ranging over the naturals. A classic result by Minsky states that Turing completeness can already be obtained when restricting to two counters [17]. Due to this fact, research has subsequently focused on studying restricted classes of counter automata and related formalisms. Among others, we note the use of restrictions to a single counter (one-counter automata or *OCA*, for short), restrictions on the underlying structure of the controller (such as flatness [5,15]), restrictions on the kinds of allowable tests on the counters, and on the types of computations considered (such as reversal-boundedness [10,11]). Counter automata are also closely related to Petri nets and pushdown automata. In recent years, motivated by complexity-theoretic considerations on the one hand and practical applications on the other, researchers have investigated decision problems for counter automata with additional primitive operations on counters, such as additive updates encoded in *binary* [11,15] or even in *parametric* form, *i.e.*, updates whose precise values depend on a finite set of parameters [3,12]. We refer to such counter automata as *succinct* and *parametric* respectively, the former being a subclass of the latter. Natural applications of such counter automata include the modeling of resource-bounded processes, numeric data types, programs with lists, recursive or multi-threaded programs, and XML query evaluation; see, *e.g.*, [4,11,10,11].

The two most prominent decision problems for counter automata are *reachability* and *model checking*. Reachability asks whether there is path between two configurations in the potentially infinite transition system generated by a counter automaton. For counter

automata with parameters, this problem generalises to asking whether there exists a valuation of the parameters such that reachability holds between two configurations in the concrete transition system induced through the valuation. Model checking is the problem of deciding whether a formula given in some temporal logic holds in a configuration of the transition system induced by a counter automaton, and when parameters are present whether the formula holds in a configuration in *all* transition systems induced by all possible valuations. Due to Minsky's result, the restriction to a *single* counter is a natural way to potentially obtain decidability for reachability and model checking problems. Consequently, in this paper we restrict our attention to this class of counter automata, and in particular investigate model checking problems for *succinct one-counter automata (SOCA)* and *parametric one-counter automata (POCA)*.

State of the art. Reachability is known to be NL-complete for OCA and has recently been shown to be NP-complete for SOCA and decidable for POCA [9]. The complexity of model-checking problems for various temporal logics including LTL, CTL and fragments thereof has been studied for OCA, SOCA and POCA in a number of recent works [20,8,7,6,22]. When comparing OCA with SOCA, an exponential complexity jump for the model checking problem may arise: both CTL and μ -calculus model checking on OCA are PSPACE-complete [20,7], whereas for SOCA these problems are EXSPACE-complete [20,6]. However, this jump is not inherent, since for example model checking LTL is PSPACE-complete for both OCA and SOCA. When parameters come into play, model checking LTL on POCA is NEXP-complete and becomes undecidable for CTL [6]. In [8], model checking the fragment EF of CTL on OCA, which can be seen as an extension of modal logic with a reachability predicate, is shown to be complete for P^{NP} . Despite its relatively limited expressiveness, EF is a useful specification language, and in particular bisimilarity checking of arbitrary systems against finite systems is polynomial-time reducible to EF model checking [13].

Our contribution. In this paper, we investigate the decidability and complexity of EF and modal logic (ML) model checking on transition systems generated by SOCA and POCA. As mentioned above, CTL model checking of POCA is undecidable [6], which is shown by reduction from the reachability problem for two-counter automata. In [6], we conjectured that EF model checking on POCA could be decidable, which is not unreasonable for two reasons. First, the undecidability proof for CTL on POCA in [6] heavily relies on the use of the *until* operator. Second, reachability for POCA is decidable [9], which is shown via a translation into the quantifier-free fragment of Presburger arithmetic with divisibility. Since there exist extensions of the latter theory that allow for universal quantification, see *e.g.* [2], and since EF primarily allows for reasoning about reachability relations, it seemed plausible that an instance of an EF model-checking problem on POCA could be translated into a sentence in such an extended theory. Nevertheless, we show in this paper that model checking EF logic on POCA is undecidable via a different reduction, namely from Hilbert's tenth problem, which Matiyasevich showed to be undecidable [16]. On the positive side, we establish tight complexity bounds for model checking POCA and SOCA against large fragments of EF. First, by dropping the reachability modality and thus restricting EF to ML, we show that the model-checking problem for POCA becomes PSPACE-complete. Obtaining the PSPACE upper bound involves a careful analysis of the size of the solution sets

Table 1. Complexity of model checking EF, ML and CTL/modal μ -calculus on OCA, SOCA and POCA

	OCA	SOCA	POCA
CTL, μ -cal.	PSPACE-complete [7,20]	EXPSPACE-complete [6,20]	Π_1^0 -complete [6]
EF	P^{NP} -complete [7,8]	PSPACE-complete	Π_1^0-complete
ML	P-complete [14]		

of certain systems of linear Diophantine inequalities of potentially exponential size. Second, when no parameters are present, we show that EF model checking for SOCA is PSPACE-complete. The main technical challenge is to develop an “exponential periodicity property” that characterizes those counter values at which an EF formula holds. Our results are summarized in **bold font** in Table 1, which also summarizes known results from the literature.

Structure of this paper. We introduce basic definitions and notations in Section 2 and present results on model checking POCA in Section 3, Section 4 deals with model checking SOCA before we conclude in Section 5. Due to space limitations, details of some proofs are deferred to a full version of this paper.

2 Preliminaries

Throughout this paper, we denote by $\mathbb{N} = \{0, 1, \dots\}$ the *non-negative integers* and by \mathbb{Z} the *integers*. We define $[i, j] \stackrel{\text{def}}{=} \{i, i + 1, \dots, j\}$ and introduce $[i]$ as an abbreviation for $[1, i]$. For any $n \in \mathbb{N}$, we denote by $\lg n$ the smallest $i \in \mathbb{N}$ such that $n \leq 2^i$. Given a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we write $f(n) = \text{poly}(n)$ (resp. $f(n) = \text{exp}(n)$) if there is some polynomial $p(n)$ such that $f(n) \leq p(n)$ (resp. $f(n) \leq 2^{p(n)}$) for each $n \in \mathbb{N}$.

The Branching-Time Logic EF: Formulas of EF over a finite set \mathbb{P} of *atomic propositions* are inductively defined by the following grammar, where p ranges over \mathbb{P} :

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \text{EX}\varphi \mid \text{EF}\varphi.$$

We define the standard Boolean abbreviations $\varphi_1 \vee \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$ and $\varphi_1 \leftrightarrow \varphi_2 \stackrel{\text{def}}{=} \varphi_1 \rightarrow \varphi_2 \wedge \varphi_2 \rightarrow \varphi_1$. Moreover, we define the additional modalities $\text{AX}\varphi \stackrel{\text{def}}{=} \neg\text{EX}\neg\varphi$ and $\text{AG}\varphi \stackrel{\text{def}}{=} \neg\text{EF}\neg\varphi$. *Modal Logic* (ML) is obtained from EF by disallowing the EF operator. An EF formula φ is in *negation normal form* if all negation symbols occur only in front of atomic propositions. The *size* $|\varphi|$ of EF formulas φ is defined as usual.

The semantics of an EF formula is given in terms of transition systems. A *transition system* T is a tuple $T = (S, \mathbb{P}, \lambda, \longrightarrow)$, where S is the set of *states*, \mathbb{P} is a finite set of *atomic propositions*, $\lambda : S \rightarrow 2^{\mathbb{P}}$ is the *state-labeling function* and $\longrightarrow \subseteq S \times S$ is the *transition relation*. We use infix notation for \longrightarrow and write $s \longrightarrow s'$ whenever $(s, s') \in \longrightarrow$. An *s-s' path* ϱ in a transition system T is a finite sequence of states $\varrho : s_1 \cdots s_n$ such that $s = s_1$, $s' = s_n$ and $s_i \longrightarrow s_{i+1}$ for all $i \in [n - 1]$, and we write $\varrho : s \longrightarrow^* s'$ to express that ϱ is an *s-s' path*. Table 2 presents the semantics

Table 2. Semantics of EF

$$\begin{aligned}
(T, s) \models p &\iff p \in \lambda(s) & (T, s) \models \varphi_1 \wedge \varphi_2 &\iff (T, s) \models \varphi_1 \text{ and } (T, s) \models \varphi_2 \\
(T, s) \models \neg\varphi &\iff (T, s) \not\models \varphi & (T, s) \models \text{EX}\varphi &\iff \exists s' \in S.(T, s') \models \varphi \text{ and } s \longrightarrow s' \\
&& (T, s) \models \text{EF}\varphi &\iff \exists s' \in S.(T, s') \models \varphi \text{ and } s \longrightarrow^* s'
\end{aligned}$$

of EF formulas. Given an EF formula φ , a transition system T and a state $s \in S$, the satisfaction relation $(T, s) \models \varphi$ is defined by induction on the structure of φ , and we say φ holds at s in T if $(T, s) \models \varphi$.

Parametric One-Counter Automata: Let $X = \{x_1, \dots, x_n\}$ denote a finite set of *parameters*, and let $\text{Op} \stackrel{\text{def}}{=} \{\text{add}(z), \text{add}(x) : z \in \mathbb{Z}, x \in X\} \cup \{\text{zero}\}$ be a set of *operations*. A *parametric one-counter automaton (POCA)* is a tuple $\mathcal{A} = (Q, X, \mathbb{P}, \lambda, \Delta)$, where Q is a finite set of *control locations*, \mathbb{P} is a finite set of *atomic propositions*, $\lambda : Q \rightarrow 2^{\mathbb{P}}$ is the *location-labeling function*, and $\Delta \subseteq Q \times \text{Op} \times Q$ is the *transition relation*. A *succinct one-counter automaton (SOCA)* is a POCA with $X = \emptyset$. We write $q \xrightarrow{\text{op}} q'$ whenever $(q, \text{op}, q') \in \Delta$. By $n_{\max}(\mathcal{A})$ we denote the largest absolute value of all integers occurring in the operations of \mathcal{A} . The *size* $|\mathcal{A}|$ of a POCA \mathcal{A} is defined as $|\mathcal{A}| \stackrel{\text{def}}{=} |\Delta| + \lg n_{\max}(\mathcal{A})$. A *valuation* $\nu : X \rightarrow \mathbb{Z}$ is a function assigning an integer to each parameter. Given a POCA \mathcal{A} , a valuation induces a SOCA \mathcal{A}^ν which is obtained by replacing each transition $q \xrightarrow{\text{add}(x_i)} q'$ with $q \xrightarrow{\text{add}(\nu(x_i))} q'$. For a SOCA \mathcal{A} , we denote by $T(\mathcal{A}) \stackrel{\text{def}}{=} (S_{\mathcal{A}}, \mathbb{P}, \lambda_{\mathcal{A}}, \longrightarrow_{\mathcal{A}})$ the *transition system induced by \mathcal{A}* , where $S_{\mathcal{A}} \stackrel{\text{def}}{=} Q \times \mathbb{N}$, $\lambda_{\mathcal{A}} \stackrel{\text{def}}{=} (q, n) \mapsto \lambda(q)$, and $(q, n) \longrightarrow_{\mathcal{A}} (q', n')$ if, and only if, either $q \xrightarrow{\text{add}(z)} q'$ and $n' = n + z$, or $q \xrightarrow{\text{zero}} q' \in \Delta$ and $n = n' = 0$. For convenience, we write $q(n)$ instead of (q, n) for states in $S_{\mathcal{A}}$. Given two states $q(n)$ and $q'(n')$, *reachability* is to decide whether there exists a $q(n)$ - $q'(n')$ path in $T(\mathcal{A})$.

Proposition 1 ([9]). *Reachability in SOCA is NP-complete.*

The *model-checking problem* for POCA, and thus for SOCA, is defined as follows:

ML/EF MODEL CHECKING ON POCA

INPUT: A POCA $\mathcal{A} = (Q, X, \mathbb{P}, \lambda, \Delta)$, $q \in Q$ and an ML/EF formula φ .

QUESTION: Does $(T(\mathcal{A}^\nu), q(0)) \models \varphi$ hold for each assignment $\nu : X \rightarrow \mathbb{Z}$?

We note that deciding whether $(T(\mathcal{A}^\nu), q(0)) \models \varphi$ holds for each assignment ν is the complement of deciding if $(T(\mathcal{A}^\nu), q(0)) \models \neg\varphi$ holds for some assignment ν .

We close this section with an example of a model-checking problem. Figure [1] depicts a SOCA \mathcal{A}_i with $i \in [0, m]$ for some $m \in \mathbb{N}$. Starting in state $q_i(n)$ with $n \in [0, 2^{m+1} - 1]$, it is easily verified that the state $q_z(0)$, which is labeled with p_i , is reachable from $q_i(n)$ if, and only if, the coefficient of 2^i in the binary expansion of n is 1, which is the case if, and only if, $(T(\mathcal{A}), q_i(n)) \models \text{EF}p_i$ or alternatively $(T(\mathcal{A}), q_i(n)) \models \text{EX}^{m+2}p_i$. Here, EX^{m+2} is an abbreviation for the $m + 2$ -fold application of the EX operator.

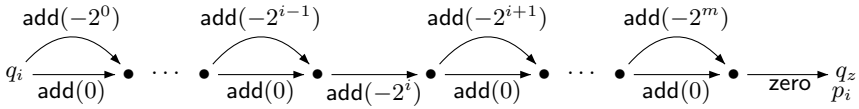


Fig. 1. SOCA \mathcal{A}_i used for testing a bit of a number $n \in [2^{m+1} - 1]$

3 Model Checking POCA

In this section, we prove that model checking EF on POCA is undecidable (Section 3.1). We show that for ML model checking on POCA is decidable and in PSPACE (Section 3.2).

3.1 Model Checking EF on POCA

We now consider model checking EF on POCA and show that this problem is Π_1^0 -complete. With EF being a notational fragment of CTL, membership in Π_1^0 follows from the fact that CTL model checking on POCA is Π_1^0 -complete [6]. Thus, we concentrate in this section on a matching Π_1^0 -lower bound by giving a reduction from Hilbert’s Tenth Problem to the complement of the model checking problem.

HILBERT’S TENTH PROBLEM (HTP)

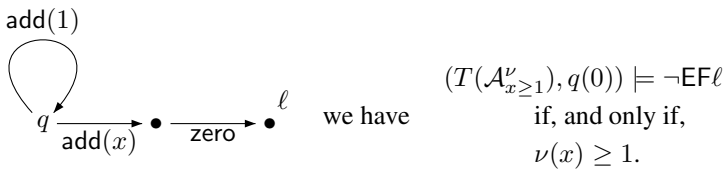
INPUT: A polynomial p with coefficients ranging over the integers.

QUESTION: Do there exist $a_1, \dots, a_n \in \mathbb{Z}$ such that $p(a_1, \dots, a_n) = 0$?

HTP was shown to be Σ_1^0 -complete by Matiyasevich [16]. Note that HTP remains Σ_1^0 -hard if we restrict the a_i to range over \mathbb{N} : A Diophantine equation $p(x_1, x_2, \dots, x_n) = 0$ is solvable in the integers if, and only if, one of the 2^n equations $p(\pm x_1, \dots, \pm x_n) = 0$ has a solution in the naturals. Replacing every unknown with the sum of squares of four unknowns gives, by Lagrange’s Theorem, the reduction in the other direction.

Moreover, we may assume with no loss of generality that $a_i > 0$ for each $i \in [n]$. If some a_i were to be zero in a solution, we can obtain a new polynomial p' in $n - 1$ variables by replacing a_i with 0 in p .

Let us fix some polynomial p with coefficients ranging over \mathbb{Z} . We will subsequently show how we can compute from p a POCA \mathcal{A}_p with a control state q_p and an EF formula φ_p such that p has a solution over the naturals if, and only if, $(T(\mathcal{A}_p^\nu), q_p(0)) \models \varphi_p$ for some valuation ν of the parameters of \mathcal{A} . Recall that the valuation of the parameters of \mathcal{A}_p ranges over \mathbb{Z} . However, we can easily ensure with a simple EF formula that a parameter x is positive. For the following SOCA $\mathcal{A}_{x \geq 1}$.

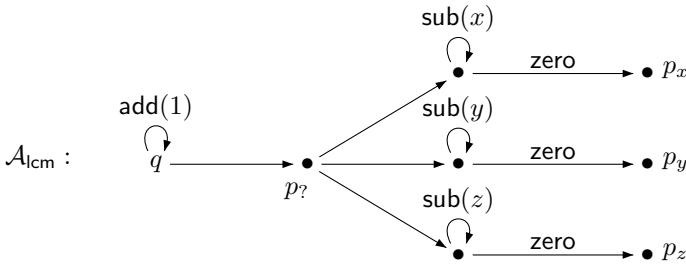


More challenging than testing if a parameter is positive when reducing from HTP is that we need to be able to express a multiplication relation over the parameters in the POCA.

In order to do that, we employ a trick that became popular by the work of Robinson [18] which allows us to define multiplication in terms of the least common multiple. In fact given $x, y \in \mathbb{N}$, we have

$$\begin{aligned} & \text{lcm}(x + y, x + y + 1) - \text{lcm}(x, x + 1) - \text{lcm}(y, y + 1) \\ &= (x^2 + x + 2xy + y^2 + y) - (x^2 + x) - (y^2 + y) = 2xy \end{aligned}$$

We note that addition and subtraction of the parameters can easily be realized by introducing additional *slack parameters* in the POCA. Thus, we can enhance our POCA by transitions of the kind $\text{sub}(x)$, meaning that $\nu(x)$ is subtracted from the counter, provided the counter is at least $\nu(x)$. We now demonstrate that for parameters x, y, z of some POCA that each assume positive values, which we can check as seen above, we can “express” in EF that $z = \text{lcm}(x, y)$. Consider the following POCA \mathcal{A}_{lcm} , where unlabeled transitions are assumed to be labeled with “add(0)”:



The idea is to express that for all $n \in \mathbb{N}$, we have that both x and y divide n if, and only if, z divides n . We note that for each $\nu : \{x, y, z\} \rightarrow \mathbb{Z}$ with $\nu(x), \nu(y), \nu(z) \geq 1$ we have that $(T(\mathcal{A}_{\text{lcm}}^\nu), q(0)) \models \text{AG}(p? \rightarrow ((\text{EF}p_x \wedge \text{EF}p_y) \leftrightarrow \text{EF}p_z))$ if, and only if, $\nu(z) = \text{lcm}(\nu(x), \nu(y))$.

Thus, by introducing a sufficient number of slack variables, we can express multiplication, addition and subtraction, which allows us to solve HTP for any arbitrary polynomial. Thus, we obtain the following theorem.

Theorem 2. *Model checking EF logic on POCA is Π_1^0 -complete.*

We note that by [16] there exists a *fixed universal* polynomial $p_u(n, k, x_1, \dots, x_m)$ such that for each recursively enumerable set $S \subseteq \mathbb{N}$, there is some $k_0 \in \mathbb{N}$ such that $S = \{n \in \mathbb{N} \mid \exists n_1, \dots, n_m \in \mathbb{N} : p_u(n, k_0, n_1, \dots, n_m) = 0\}$. This allows us to strengthen our result insofar as there exists a *fixed* EF formula φ and a *fixed* POCA $\mathcal{A} = (Q, X, \mathbb{P}, \lambda, \Delta)$ with a transition $q \xrightarrow{\text{add}(y)} q' \in \Delta$ and a control state $q_0 \in Q$ such that it is Π_1^0 -complete to decide for a given $n \in \mathbb{N}$ whether by replacing y with n , $(T(\mathcal{A}^\nu), q_0(0)) \models \varphi$ holds for all $\nu : X \rightarrow \mathbb{Z}$.

3.2 Model Checking ML on POCA

This section will be devoted to proving a PSPACE upper bound for model checking ML on POCA. Let us fix some POCA $\mathcal{A} = (Q, X, \mathbb{P}, \lambda, \Delta)$ with $X = \{x_1, \dots, x_\ell\}$, some

control state $q_0 \in Q$ and some ML formula α . *Provided* that ML model checking of SOCA is in PSPACE (we show that even model checking EF on SOCA is in PSPACE in Section 4.2), in order to obtain a PSPACE upper bound, it is sufficient to show that if $(T(\mathcal{A}^\nu), q_0(0)) \models \alpha$ holds for some $\nu : X \rightarrow \mathbb{Z}$ then there is some $\mu : X \rightarrow \mathbb{Z}$ such that $(T(\mathcal{A}^\mu), q(0)) \models \alpha$ and $|\mu(x)|$ can be represented with polynomially many bits in $|\mathcal{A}| + |\alpha|$ for each $x \in X$, since such an assignment can be guessed in PSPACE.

For each $q \in Q$ and each subformula φ of α , let us define $\mathcal{M}(q, \varphi) \subseteq \mathbb{Z}^\ell \times \mathbb{N} \subseteq \mathbb{Z}^{\ell+1}$ as follows:

$$\mathcal{M}(q, \varphi) \stackrel{\text{def}}{=} \{(z_1, \dots, z_\ell, n) \mid (T(\mathcal{A}^\nu), q(n)) \models \varphi \text{ and } \nu(x_i) = z_i, i \in [1, \ell]\}.$$

Before we proceed with the proof of the upper bound, we need to introduce some additional notation. For an integer matrix $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$, we denote by $\|A\| = \max_i \{\sum_j |a_{ij}|\}$ the norm of A . For an integer vector $\mathbf{b} = (b_i)$, we denote by $\|\mathbf{b}\| = \sum_i |b_i|$ the norm of \mathbf{b} . A system of linear Diophantine inequalities (SLDI) is a system of the form $\mathcal{S} = (A\mathbf{x} \geq \mathbf{b})$, where $A \in \mathbb{Z}^{m \times n}$ is an $m \times n$ matrix, $\mathbf{b} \in \mathbb{Z}^m$ is an m -vector and \mathbf{x} is an n -vector of indeterminates all ranging over the integers. By $\text{Sol}(\mathcal{S})$, we denote the set of integer solutions to the SLDI $\mathcal{S} = (A\mathbf{x} \geq \mathbf{b})$. Finally, we define $\|\mathcal{S}\|_{\text{mat}} \stackrel{\text{def}}{=} \|A\|$ and $\|\mathcal{S}\|_{\text{vec}} \stackrel{\text{def}}{=} \|\mathbf{b}\|$.

Recall that x_1, \dots, x_ℓ are the parameters of \mathcal{A} . Our overall goal is to express $\mathcal{M}(q, \varphi)$ by a union of solutions to SLDIs, each of the form

$$\mathcal{S} = (A\mathbf{x} \geq \mathbf{b}), \quad \text{where } A \in \mathbb{Z}^{m \times (\ell+1)} \text{ and } \mathbf{b} \in \mathbb{Z}^m \text{ for some } m \geq 1.$$

In the remainder of this section, we will assume for any $(A\mathbf{x} \geq \mathbf{b})$ that A is some $m \times (\ell + 1)$ matrix and \mathbf{b} is some m -vector for some $m \geq 1$. The intuition is that the i^{th} component of \mathbf{x} with $i \in [\ell]$ is going to correspond to the parameter x_i of \mathcal{A} and the $(\ell + 1)^{\text{th}}$ component of \mathbf{x} is going to correspond to the counter value where the ML formula is evaluated. In case $A = (a_{ij})$ we define $\|A\|_{\ell+1} \stackrel{\text{def}}{=} \max\{|a_{i(\ell+1)}| : i \in [m]\}$ and lift this definition to $\|\mathcal{S}\|_{\ell+1} \stackrel{\text{def}}{=} \|A\|_{\ell+1}$.

In order to prove that small valuations $\nu : X \rightarrow \mathbb{Z}$ suffice for α , we are now going to prove that for each $q \in Q$ and each subformula φ of α , we have

$$\mathcal{M}(q, \alpha) = \bigcup_{i \in I} \text{Sol}(\mathcal{S}_i)$$

for some index set I with $\|\mathcal{S}_i\|_{\text{mat}} = \text{poly}(|\varphi|)$ and $\|\mathcal{S}_i\|_{\text{vec}} = \text{poly}(|\varphi|) \cdot \exp(|\mathcal{A}|)$ for each $i \in I$. Once this fact has been established, we will show that each SLDI \mathcal{S}_i admits solutions that can be represented using polynomially many bits in $|\mathcal{A}| + |\alpha|$, thus establishing the desired upper bound on necessary valuations of the parameters of \mathcal{A} .

We require some additional notation that, together with the subsequent lemma, will be useful for proving the existence of sets of SLDIs of “small” size for each $\mathcal{M}(q, \varphi)$. Let $H \subseteq \mathbb{Z}^{\ell+1}$. We define $H - x_k \stackrel{\text{def}}{=} \{(z_1, \dots, z_\ell, z_{\ell+1} - z_k) \in \mathbb{Z}^{\ell+1} \mid (z_1, \dots, z_{\ell+1}) \in H\}$ for each $k \in [\ell]$ and $H - z \stackrel{\text{def}}{=} \{(z_1, \dots, z_\ell, z_{\ell+1} - z) \in \mathbb{Z}^{\ell+1} \mid (z_1, \dots, z_{\ell+1}) \in H\}$ for each $z \in \mathbb{Z}$. The following lemma states that solutions to SLDIs are closed under the operations $-x_k$ and $-z$ and gives bounds on the blow-up of the introduced norms.

We remark that we do not require an effective variant of this lemma to establish our PSPACE upper bound.

Lemma 3. *Let $\mathcal{S} = (A\mathbf{x} \geq \mathbf{b})$ be an SLDI with $A = (a_{ij}) \in \mathbb{Z}^{m \times (\ell+1)}$. Then the following holds:*

- (1) *For each $k \in [\ell]$ there is some SLDI \mathcal{S}' with $\text{Sol}(\mathcal{S}') = \text{Sol}(\mathcal{S}) - x_k$, $\|\mathcal{S}'\|_{\text{mat}} \leq \|\mathcal{S}\|_{\text{mat}} + \|\mathcal{S}\|_{\ell+1}$, $\|\mathcal{S}'\|_{\ell+1} = \|\mathcal{S}\|_{\ell+1}$, and $\|\mathcal{S}'\|_{\text{vec}} = \|\mathcal{S}\|_{\text{vec}}$.*
- (2) *For each $z \in \mathbb{Z}$, there is some SLDI \mathcal{S}' with $\text{Sol}(\mathcal{S}') = \text{Sol}(\mathcal{S}) - z$, $\|\mathcal{S}'\|_{\text{mat}} = \|\mathcal{S}\|_{\text{mat}}$, $\|\mathcal{S}'\|_{\ell+1} = \|\mathcal{S}\|_{\ell+1}$, and $\|\mathcal{S}'\|_{\text{vec}} \leq \|\mathcal{S}\|_{\text{vec}} + \|\mathcal{S}\|_{\ell+1} \cdot |z|$.*

Proof. Let us assume $\mathbf{b} = (b_i)$. For Point (1), let $k \in [1, \ell]$. For each $(z_1, \dots, z_{\ell+1}) \in \mathbb{Z}^{\ell+1}$ we have

$$\begin{aligned} & (z_1, \dots, z_{\ell+1}) \in \text{Sol}(\mathcal{S}) - x_k \\ \iff & (z_1, \dots, z_{\ell}, z_{\ell+1} + z_k) \in \text{Sol}(\mathcal{S}) \\ \iff & \forall i \in [1, m] : \left(\sum_{j \in [1, \ell]} a_{ij} \cdot z_j + a_{i(\ell+1)}(z_{\ell+1} + z_k) \geq b_i \right) \\ \iff & \forall i \in [1, m] : \left((a_{ik} + a_{i(\ell+1)})z_k + \sum_{\substack{j \in [1, \ell+1], \\ j \neq k}} a_{ij} \cdot z_j \geq b_i \right). \end{aligned}$$

We can thus define the matrix $A' = (a'_{ij})$, where $a'_{ij} = a_{ij}$ if $j \neq k$ and $a_{ij} = a_{ij} + a_{i(\ell+1)}$ if $j = k$, for each $i \in [1, m]$. We put $\mathcal{S}' = (A'\mathbf{x} \geq \mathbf{b})$ and we just proved $\text{Sol}(\mathcal{S}') = \text{Sol}(\mathcal{S}) - x_k$. Moreover, it holds $\|\mathcal{S}'\|_{\text{mat}} = \|A'\| \leq \|A\| + \|A\|_{\ell+1} = \|\mathcal{S}\|_{\text{mat}} + \|\mathcal{S}\|_{\ell+1}$, $\|\mathcal{S}'\|_{\ell+1} = \|A\|_{\ell+1} = \|\mathcal{S}\|_{\ell+1}$, and $\|\mathcal{S}'\|_{\text{vec}} = \|\mathbf{b}\| = \|\mathcal{S}\|_{\text{vec}}$.

Point (2) is shown analogously. \square

We are now ready to prove the desired lemma.

Lemma 4. *For every $q \in Q$ and every subformula φ of α in negation normal form, we have $\mathcal{M}(q, \varphi) = \bigcup_{i \in I} \text{Sol}(\mathcal{S}_i)$, where I is some index set and each \mathcal{S}_i is some SLDI with $\|\mathcal{S}_i\|_{\text{mat}} \leq |\varphi|$, $\|\mathcal{S}_i\|_{\ell+1} \leq 1$, $\|\mathcal{S}_i\|_{\text{vec}} \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\varphi|$.*

Proof. We prove the lemma by structural induction on φ .

Case $\varphi = p$ for some $p \in \mathbb{P}$ (the case $\varphi = \neg p$ is dual).

First, let us assume $p \in \lambda(q)$. Then $\mathcal{M}(q, \varphi) = \mathbb{Z}^{\ell} \times \mathbb{N}$, which can be described by the solutions to the single SLDI $\mathcal{S} \stackrel{\text{def}}{=} (A\mathbf{x} \geq \mathbf{b})$ with $\mathbf{b} \stackrel{\text{def}}{=} \mathbf{0}$ and $A \stackrel{\text{def}}{=} (a_{ij}) \in \mathbb{Z}^{1 \times (\ell+1)}$ with $a_{1j} \stackrel{\text{def}}{=} 0$ for each $j \in [1, \ell]$ and $a_{1(\ell+1)} \stackrel{\text{def}}{=} 1$. Note that $\|\mathcal{S}\|_{\text{mat}} = \|A\| = 1 = |\varphi|$, $\|\mathcal{S}\|_{\ell+1} = \|A\|_{\ell+1} = 1$, and $\|\mathcal{S}\|_{\text{vec}} = \|\mathbf{b}\| = 0 \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\varphi|$.

In case $p \notin \lambda(q)$, we have $\mathcal{M}(q, \varphi) = \emptyset$, which we express as the solutions of the SLDI $\mathcal{S} = (A\mathbf{x} \geq \mathbf{b})$, where A is $1 \times (\ell + 1)$ zero matrix and $\mathbf{b} \stackrel{\text{def}}{=} \mathbf{1}$. We have $\|\mathcal{S}\|_{\text{mat}} = \|A\| = 0 \leq 1 = |\varphi|$, $\|\mathcal{S}\|_{\ell+1} = \|A\|_{\ell+1} = 0 \leq 1$, and $\|\mathcal{S}\|_{\text{vec}} = \|\mathbf{b}\| = 0 \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\varphi|$.

Case $\varphi = \psi \vee \psi'$: By the induction hypothesis we have $\mathcal{M}(q, \psi) = \bigcup_{i \in I} \text{Sol}(\mathcal{S}_i)$ for some index set I and for SLDI \mathcal{S}_i , for each $i \in I$ and $\mathcal{M}(q, \psi') = \bigcup_{i \in I'} \text{Sol}(\mathcal{S}'_i)$ for some index set I' and for SLDI \mathcal{S}'_i , for each $i \in I'$. Obviously we can write $\mathcal{M}(q, \varphi)$ as $\bigcup_{i \in I} \text{Sol}(\mathcal{S}_i) \cup \bigcup_{i \in I'} \text{Sol}(\mathcal{S}'_i)$ and the bounds on the norms easily carry over from induction hypothesis.

Case $\varphi = \psi \wedge \psi'$: By induction the hypothesis we have $\mathcal{M}(q, \psi) = \bigcup_{i \in I} \text{Sol}(\mathcal{S}_i)$ for some index set I and for SLDIs \mathcal{S}_i , for each $i \in I$ and $\mathcal{M}(q, \psi') = \bigcup_{i \in I'} \text{Sol}(\mathcal{S}'_i)$ for some index set I' and for SLDIs \mathcal{S}'_i , for each $i \in I'$. Let us assume $\mathcal{S}_i = (A_i \mathbf{x} \geq \mathbf{b}_i)$ for each $i \in I$ and $\mathcal{S}'_i = (A'_i \mathbf{x} \geq \mathbf{b}'_i)$ for each $i \in I'$. We define the matrix $A_{ii'} \stackrel{\text{def}}{=} \begin{pmatrix} A_i \\ A'_i \end{pmatrix}$ and the vector $b_{ii'} \stackrel{\text{def}}{=} \begin{pmatrix} b_i \\ b'_i \end{pmatrix}$ for each $i \in I$ and each $i' \in I'$. Obviously, we have $\mathcal{M}(q, \varphi) = \mathcal{M}(q, \psi) \cap \mathcal{M}(q, \psi') = \bigcup_{i \in I, i' \in I'} \text{Sol}(A_{ii'} \mathbf{x} \geq b_{ii'})$. Again, the bounds on the norms immediately carry over from induction hypothesis.

Case $\varphi = \text{AX}\psi$: By the induction hypothesis, we have $\mathcal{M}(q', \psi) = \bigcup_{i \in I_{q'}} \text{Sol}(\mathcal{S}_{i,q'})$ for some SLDIs $\mathcal{S}_{i,q'}$ for each $q' \in Q$. Let us assume that $\mathcal{S}_{i,q'} = (A_{i,q'} \mathbf{x} \geq \mathbf{b}_{i,q'})$ for each $i \in I_{q'}$ and each $q' \in Q$. Before giving the translation, we need to introduce some auxiliary SLDIs $\mathcal{S}_{\circ z}$ and $\mathcal{S}_{\circ x_k}$ for each $z \in \mathbb{Z}$, each $k \in [\ell]$ and each $\circ \in \{<, >, \leq, \geq\}$ such that

$$\begin{aligned} \text{Sol}(\mathcal{S}_{\circ z}) &= \{(z_1, \dots, z_{\ell+1}) \in \mathbb{Z}^{\ell+1} \mid z_{\ell+1} \circ z\} \text{ and} \\ \text{Sol}(\mathcal{S}_{\circ x_k}) &= \{(z_1, \dots, z_{\ell+1}) \in \mathbb{Z}^{\ell+1} \mid z_{\ell+1} \circ z_k\}. \end{aligned}$$

For $z \in \mathbb{Z}$, we only give $\mathcal{S}_{\circ z}$ for $\circ = "<"$, the remaining cases for \circ can be defined analogously. We put $\mathcal{S}_{< z} \stackrel{\text{def}}{=} (A \mathbf{x} \geq \mathbf{b})$, where $A \stackrel{\text{def}}{=} (a_{1j}) \in \mathbb{Z}^{1 \times (\ell+1)}$ with $a_{1j} \stackrel{\text{def}}{=} 0$ if $j \in [\ell]$ and $a_{1(\ell+1)} \stackrel{\text{def}}{=} -1$, and finally $\mathbf{b} \stackrel{\text{def}}{=} (-z + 1)$ since over the integers we have $z_{\ell+1} < z$ if, and only if, $z_{\ell+1} \leq z - 1$ if, and only if, $-z_{\ell+1} \geq -z + 1$. Observe that $\|\mathcal{S}_{\circ z}\|_{\text{mat}} \leq 1$, $\|\mathcal{S}_{\circ z}\|_{\ell+1} \leq 1$, and $\|\mathcal{S}_{\circ z}\|_{\text{vec}} \leq |z| + 1$ for each $\circ \in \{<, >, \leq, \geq\}$.

Likewise, we define $\mathcal{S}_{\circ x_k}$ for $\circ = "<"$, the other cases for \circ can be dealt with analogously. The reader easily verifies that one can define $\mathcal{S}_{< x_i} \stackrel{\text{def}}{=} (C \mathbf{x} \geq \mathbf{d})$ with $C \stackrel{\text{def}}{=} (c_{1j}) \in \mathbb{Z}^{1 \times (\ell+1)}$ with $c_{1j} \stackrel{\text{def}}{=} 1$ if $j = i$, $c_{1j} \stackrel{\text{def}}{=} -1$ if $j = \ell + 1$, and $c_{1j} \stackrel{\text{def}}{=} 0$ otherwise. Moreover, we put $\mathbf{d} \stackrel{\text{def}}{=} (1)$. Observe that $\|\mathcal{S}_{\circ x_k}\|_{\text{mat}} \leq 1$, $\|\mathcal{S}_{\circ x_k}\|_{\ell+1} \leq 1$, and $\|\mathcal{S}_{\circ x_k}\|_{\text{vec}} \leq 1$ for each $\circ \in \{<, >, \leq, \geq\}$. We now define

$$\mathcal{M}(q, \varphi) \stackrel{\text{def}}{=} \text{Sol}(\mathcal{S}_{\geq 0}) \cap \bigcap_{\substack{q \stackrel{\text{add}(y)}{\in} \Delta \\ y \in \mathbb{Z} \cup X}} \left(\text{Sol}(\mathcal{S}_{< y}) \cup \bigcup_{i \in I_{q'}} (\text{Sol}(\mathcal{S}_{i,q'}) - y) \right).$$

In the same fashion as for disjunction and conjunction, we can express the right-hand side of the latter equality as a union of SLDIs. Note that in this modification process the number of rows of the matrix may change, but *neither* do the norms of the matrices *nor* the norms of the vectors of the systems. The reader easily verifies that the $\|\cdot\|_{\text{mat}}$, $\|\cdot\|_{\ell+1}$, and $\|\cdot\|_{\text{vec}}$ norms of each auxiliary SLDI satisfy the bounds required by the lemma. Hence, in order to bound the norms of the SLDI that occur in the final union, it

suffices to bound the norms of each SLDI \mathcal{S} such that $\text{Sol}(\mathcal{S}) = \text{Sol}(\mathcal{S}_{i,q'}) - y$ for some $q' \in Q$, some $i \in I_{q'}$ and some $q \xrightarrow{\text{add}(y)} q' \in \Delta$, where $y \in \mathbb{Z} \cup X$. To this end, we apply Lemma 3 by distinguishing between $y \in \mathbb{Z}$ and $y \in X$.

If $y = x_k$ for some $k \in [\ell]$, i.e. $y \in X$, we obtain the following bounds by Point (1) of Lemma 3:

- $\|\mathcal{S}\|_{\text{mat}} \stackrel{\text{Lemma 3(1)}}{\leq} \|A_{i,q'}\| + \|A_{i,q'}\|_{\ell+1} \stackrel{\text{IH}}{\leq} |\psi| + 1 = |\varphi|,$
- $\|\mathcal{S}\|_{\ell+1} \stackrel{\text{Lemma 3(1)}}{=} \|A_{i,q'}\|_{\ell+1} \stackrel{\text{IH}}{\leq} 1,$ and
- $\|\mathcal{S}\|_{\text{vec}} \stackrel{\text{Lemma 3(1)}}{=} \|\mathbf{b}_{i,q'}\| \stackrel{\text{IH}}{\leq} (n_{\max}(\mathcal{A}) + 1) \cdot |\psi| \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\varphi|$

In case $y \in \mathbb{Z}$, we obtain the following by Point (2) of Lemma 3:

- $\|\mathcal{S}\|_{\text{mat}} \stackrel{\text{Lemma 3(2)}}{=} \|A_{i,q'}\| \stackrel{\text{IH}}{\leq} |\psi| \leq |\varphi|,$
- $\|\mathcal{S}\|_{\ell+1} \stackrel{\text{Lemma 3(2)}}{=} \|A_{i,q'}\|_{\ell+1} \stackrel{\text{IH}}{\leq} 1,$ and
- $\|\mathcal{S}\|_{\text{vec}} \stackrel{\text{Lemma 3(2)}}{\leq} \|\mathbf{b}_{i,q'}\| + \|A_{i,q'}\|_{\ell+1} \cdot |y| \stackrel{\text{IH}}{\leq} (n_{\max}(\mathcal{A}) + 1) \cdot |\psi| + 1 \cdot n_{\max}(\mathcal{A}) \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\varphi|$

Case $\varphi = \text{EX}\psi$. By induction hypothesis, we have $\mathcal{M}(q', \psi) = \bigcup_{i \in I_{q'}} \text{Sol}(\mathcal{S}_{i,q'})$ for some SLDI $\mathcal{S}_{i,q'}$ for each $q' \in Q$. Let us assume that $\mathcal{S}_{i,q'} = (A_{i,q'} \mathbf{x} \geq \mathbf{b}_{i,q'})$ for each $i \in I_{q'}$ and each $q' \in Q$. We define

$$\mathcal{M}(q, \varphi) \stackrel{\text{def}}{=} \text{Sol}(\mathcal{S}_{\geq 0}) \cap \left(\bigcup_{q \xrightarrow{\text{add}(y)} q' \in \Delta} \bigcup_{i \in I_{q'}} (\text{Sol}(\mathcal{S}_{i,q'}) - y) \right).$$

The analysis of the sizes of the norms can be proven analogously as for the case $\varphi = \text{AX}\psi$. \square

The following lemma from [19] states that solvable SLDI have small solutions whose norm is independent on the number of rows of the SLDI.

Lemma 5 ([19], p. 239). *Each solvable SLDI $A\mathbf{x} \geq \mathbf{b}$ has a solution of norm at most $\text{poly}(\|A\| + \|\mathbf{b}\|)$.*

Let us return to our original formula α . By Lemma 4 there exists some SLDI \mathcal{S}_i such that $\mathcal{M}(q_0, \alpha) = \text{Sol}(\mathcal{S}_i)$, and where $\|\mathcal{S}_i\|_{\text{mat}} \leq |\alpha|$ and $\|\mathcal{S}_i\|_{\text{vec}} \leq (n_{\max}(\mathcal{A}) + 1) \cdot |\alpha|$. Since we are interested if $(T(\mathcal{A}^\nu), q_0(0)) \models \alpha$ for some $\nu : X \rightarrow \mathbb{Z}$, think of adding to each matrix that occurs in \mathcal{S}_i two more rows expressing that $x_{\ell+1} = 0$. Let us call the resulting SLDI \mathcal{S}'_i . By Lemma 5 we know that if \mathcal{S}'_i is solvable, then \mathcal{S}'_i has a solution of norm at most $\text{poly}(n_{\max}(\mathcal{A}) + |\alpha|)$. In other words, if $(T(\mathcal{A}^\nu), q_0(0)) \models \alpha$ for some $\nu : X \rightarrow \mathbb{Z}$, then $(T(\mathcal{A}^\mu), q_0(0)) \models \alpha$ already holds for some $\mu : X \rightarrow \mathbb{Z}$ and $\mu(x)$ is polynomially bounded in $|\mathcal{A}| + |\alpha|$ for each $x \in X$.

Hence, we obtain the following theorem.

Theorem 6. *ML model checking for POCA is in PSPACE.*

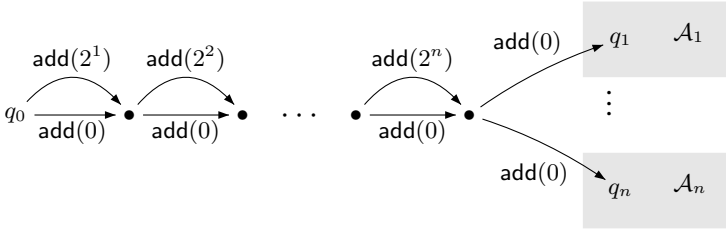


Fig. 2. SOCA \mathcal{A} constructed for simulating the QBF formula α

4 Model Checking SOCA

In this section we prove that model checking ML on SOCA is PSPACE-hard (Section 4.1) and that model checking EF on SOCA is in PSPACE (Section 4.2).

4.1 Model Checking ML on SOCA

PSPACE-hardness of ML model checking on SOCA follows from a straight-forward reduction from QBF.

Proposition 7. *Model checking ML on SOCA is PSPACE-hard.*

Proof. We give a reduction from QBF. Let $\alpha = \exists x_1 \forall x_2 \dots \exists x_n \beta(x_1, \dots, x_n)$ be an instance of QBF. Without loss of generality, we can assume that β is in 3-CNF, i.e., of the form $\beta = \bigwedge_{i \in [m]} \beta_i$, where each clause β_i consists of three literals, so $\beta_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$. We construct in polynomial time a SOCA $\mathcal{A} = (Q, \mathbb{P}, \lambda, \Delta)$ and an ML formula φ such that for some $q_0 \in Q$ we have that α is valid if, and only if, $(T(\mathcal{A}), q_0(0)) \models \varphi$. We define $\mathbb{P} \stackrel{\text{def}}{=} \{p_i \mid i \in [n]\}$. The states and transitions of \mathcal{A} are given in Figure 2 where the SOCA \mathcal{A}_i is taken from Figure 1. Finally, we define φ to be the ML formula that is obtained by replacing each $\exists x_i$ from α with EX, each $\forall x_i$ with AX, and each literal ℓ_{i_j} with $\text{EX}^{n+2} p_{i_j}$ if $\ell_{i_j} = x_{i_j}$ and $\neg \text{EX}^{n+2} p_{i_j}$ if $\ell_{i_j} = \overline{x_{i_j}}$. It is easily verified that α is valid if, and only if, $(T(\mathcal{A}), q_0(0)) \models \varphi$. \square

4.2 Model Checking EF on SOCA

In this section, we are going to show that EF model checking on SOCA is in PSPACE, and hence PSPACE-complete by Proposition 7. To this end, let us fix some SOCA $\mathcal{A} = (Q, \mathbb{P}, \lambda, \delta)$. Our result is based on the following lemma, which expresses periodicity properties of reachability relations in \mathcal{A} .

Lemma 8. *There are naturals $\tau, \varepsilon, \delta = \exp(|\mathcal{A}|)$ with $\varepsilon \geq n_{\max}(\mathcal{A})$ such that for each $n, n', m, m' > \tau$ with $n \equiv n' \pmod{\delta}$ and $m \equiv m' \pmod{\delta}$ the following statements hold for each $q, q' \in Q$:*

- (1) If $m + \varepsilon < n$ and $m' + \varepsilon < n'$, then $q(n) \rightarrow_{\mathcal{A}}^* q'(m)$ if, and only if, $q(n') \rightarrow_{\mathcal{A}}^* q'(m')$.
- (2) If $m > n + \varepsilon$ and $m' > n' + \varepsilon$, then $q(n) \rightarrow_{\mathcal{A}}^* q'(m)$ if, and only if, $q(n') \rightarrow_{\mathcal{A}}^* q'(m')$.

Section 4.3 will be devoted to sketching a proof of Lemma 8. Assume the constants τ , ε and δ from Lemma 8 to be fixed for the rest of this section. Let us define $\mathcal{M}(q, \varphi) = \{n \in \mathbb{N} : (T(\mathcal{A}), q(n)) \models \varphi\}$ for each control state $q \in Q$ and each EF formula φ over \mathbb{P} . For the PSPACE upper bound, we will show that $\mathcal{M}(q, \varphi)$ is ultimately periodic with period δ .

Lemma 9. *If $n \equiv n' \pmod{\delta}$, then $n \in \mathcal{M}(q, \varphi)$ if, and only if, $n' \in \mathcal{M}(q, \varphi)$, for each control state $q \in Q$, each EF formula φ over \mathbb{P} and each $n, n' > \tau + |\varphi| \cdot \varepsilon + \delta$.*

Proof. Without loss of generality assume $n' > n$. We show $(T(\mathcal{A}), q(n)) \models \varphi$ if, and only if, $(T(\mathcal{A}), q(n + \delta)) \models \varphi$ by induction on $|\varphi|$, from which the statement will follow. We only consider the most interesting cases $\varphi = \text{EX}\varphi'$ and $\varphi = \text{EF}\varphi'$, the other cases are easy.

If $\varphi = \text{EX}\varphi'$, we have $(T(\mathcal{A}), q(n)) \models \varphi$ if, and only if, there is some $q' \in Q$ and $z \in \mathbb{Z}$ such that $q \xrightarrow{\text{add}(z)} q' \in \Delta$ and $(T(\mathcal{A}), q'(n+z)) \models \varphi'$. Since $n+z > \tau + |\varphi'| \cdot \varepsilon + \delta$, the induction hypothesis yields $(T(\mathcal{A}), q'(n+z)) \models \varphi'$ if, and only if, $(T(\mathcal{A}), q'(n+z+\delta)) \models \varphi'$. Hence $(T(\mathcal{A}), q(n)) \models \text{EX}\varphi'$ if, and only if, $(T(\mathcal{A}), q(n+\delta)) \models \text{EX}\varphi'$.

If $\varphi = \text{EF}\varphi'$, we have $(T(\mathcal{A}), q(n)) \models \varphi$ if, and only if, there are $q' \in Q$, $m \in \mathbb{N}$ and ϱ such that $\varrho : q(n) \rightarrow_{\mathcal{A}}^* q'(m)$ and $(T(\mathcal{A}), q(m)) \models \varphi'$. Suppose $m > \tau + |\varphi'| \cdot \varepsilon + \delta$ and no counter value less than δ occurs along ϱ , so in particular there is no zero test along ϱ . The induction hypothesis yields $(T(\mathcal{A}), q(m+\delta)) \models \varphi'$, and by shifting ϱ by δ the existence of a path $\varrho' : q(n+\delta) \rightarrow_{\mathcal{A}}^* q(m+\delta)$ follows, hence $(T(\mathcal{A}), q(n+\delta)) \models \text{EF}\varphi'$. Otherwise, if $m \leq \tau + |\varphi'| \cdot \varepsilon + \delta$ or a counter value less than δ occurs along ϱ , Lemma 8. Point (1) guarantees that $q(n) \rightarrow_{\mathcal{A}}^* q'(m)$ if, and only if, $q(n+\delta) \rightarrow_{\mathcal{A}}^* q'(m)$, which again allows us to conclude that $(T(\mathcal{A}), q(n)) \models \text{EF}\varphi'$. The direction $(T(\mathcal{A}), q(n)) \models \varphi$ implies $(T(\mathcal{A}), q(n+\delta)) \models \varphi$ follows analogously. \square

Theorem 10. *EF model checking of SOCA is PSPACE-complete.*

Proof. PSPACE-hardness has already been established in Section 4.1. For the upper bound, Algorithm 1 is an alternating algorithm that decides $(T(\mathcal{A}), q(n)) \models \varphi$ in PSPACE. For brevity, the cases $\varphi = \text{AX}\varphi'$ and $\varphi' = \text{AG}\varphi'$ have been left out, they are defined complementary to their EX respectively EF counterparts. We only sketch correctness of the case $\varphi = \text{EF}\varphi'$ by induction on $|\varphi|$, all other cases are obviously correct. Let $m = \max\{n + \varepsilon + \delta, \tau + |\varphi'| \cdot \varepsilon + \delta\}$. Suppose $(T(\mathcal{A}), q(n)) \models \text{EF}\varphi'$, there is some $q'(n')$ such that $q(n) \rightarrow_{\mathcal{A}}^* q'(n')$ and $(T(\mathcal{A}), q'(n')) \models \varphi'$. If $n' > m$, Lemma 9 guarantees that there is $n'' \in [0, m]$ such that $(T(\mathcal{A}), q'(n'')) \models \varphi'$, and Lemma 8. Point (2) yields $q(n) \rightarrow_{\mathcal{A}}^* q'(n'')$, which by Proposition 1 can be checked in NP. By the induction hypothesis, Algorithm 1 returns *true* on input $q'(n'')$ and φ' , which concludes the correctness proof. \square

Algorithm 1. Fragment of the EF SOCA model checking algorithm

Input: EF formula φ , configuration $q(n)$ of \mathcal{A}

case $\varphi = p$: **return** $p \in \lambda(q)$

case $\varphi = \neg p$: **return** $p \notin \lambda(q)$

case $\varphi = \varphi_1 \wedge \varphi_2$: **return** $(T(\mathcal{A}), q(n)) \models \varphi_1$ and $(T(\mathcal{A}), q(n)) \models \varphi_2$

case $\varphi = \varphi_1 \vee \varphi_2$: **return** $(T(\mathcal{A}), q(n)) \models \varphi_1$ or $(T(\mathcal{A}), q(n)) \models \varphi_2$

case $\varphi = \text{EX}\varphi'$: **existential move**:

choose $q \xrightarrow{\text{op}} q' \in \Delta$

case $\text{op} = \text{add}(z)$: **return** $(T(\mathcal{A}), q'(n+z)) \models \varphi'$

case $\text{op} = \text{zero}$ and $n = 0$: **return** $(T(\mathcal{A}), q'(0)) \models \varphi'$

case $\varphi = \text{EF}\varphi'$: **existential move**:

choose $q'(m)$ such that $q(n) \xrightarrow{*}_{\mathcal{A}} q'(m)$ and $m \in [0, \max\{n+\varepsilon+\delta, \tau+|\varphi'|\cdot\varepsilon+\delta\}]$

return $(T(\mathcal{A}), q'(m)) \models \varphi'$

4.3 Proof Sketch of Lemma 8

In this section, we give a proof sketch of Lemma 8 which was left open in the previous section. The technical details are deferred to a full version of this paper.

On a technical level, it is helpful to view SOCA as *weighted graphs*, an approach also used in [9]. Given a SOCA \mathcal{A} , its corresponding weighted graph $G_{\mathcal{A}}$ is obtained by removing all zero-labeled edges from \mathcal{A} , and for every edge labeled with $\text{add}(z)$, $G_{\mathcal{A}}$ has an edge labeled with z . Thus, we can assign any path π in $G_{\mathcal{A}}$ a *weight* $w(\pi)$ and a *drop* $d(\pi)$, which is the smallest weight of all prefixes of π . This allows us to relate runs in $T(\mathcal{A})$ with paths in $G_{\mathcal{A}}$: there is a zero-test free run $q(n) \xrightarrow{*}_{\mathcal{A}} q'(n')$ if, and only if, there is a path π from q to q' in $G_{\mathcal{A}}$ with $w(\pi) = n' - n$ and $d(\pi) \geq -n$.

Let us fix a SOCA \mathcal{A} and its corresponding graph G . In order to prove the periodicity properties expressed in Lemma 8 we will use cycles in G in order to construct paths whose weight is periodic for some period δ . For a start, let us concentrate on *cycles* in G with *negative weight*. Given a strongly connected component (SCC) S in G , we define $\text{gcd } S$ as greatest common divisor of the set of all weights of all loop-free cycles in S . Note that $\text{gcd } S = \exp(|\mathcal{A}|)$. It is easy to check that $\text{gcd } S$ divides the weight of every cycle that runs through S , so $\text{gcd } S$ could potentially serve as a period. However, if the weights of all cycles in S have the same sign, we cannot necessarily construct a cycle whose weight is an arbitrary multiple of $\text{gcd } S$. For example, let $\{5, 7\}$ be the set of all weights of simple cycles in some SCC S with $S = \{q\}$ for some $q \in Q$. We have $\text{gcd } S = 1$, however there is no cycle π in S with, say, $w(\pi) = 23$. This obstacle is related to the *Frobenius problem*, which is stated as follows [21]: given $x_1 < \dots < x_n \in \mathbb{N}$ such that $\text{gcd}\{x_1, \dots, x_n\} = 1$, what is the *largest* $g \in \mathbb{N}$ such that g cannot be represented as non-negative integer linear combination of the x_i . It is shown in [21] that $g < x_n^2$. Thus in our example, this fact guarantees that there is a q -cycle π with $w(\pi) = m$ for every $m \geq 49$. The preceding observations allow us to conclude that once a certain threshold is crossed, we have periodicity of weights of cycles in an SCC.

Lemma 11. *There exists a local threshold $\gamma \in \mathbb{N}$ such that $\gamma = \exp(|\mathcal{A}|)$ and for all $w, w' < -\gamma$ and $q \in Q$ such that $w \equiv w' \pmod{\text{gcd } S}$ for some SCC S such that*

$q \in S$, whenever there exists a q -cycle π with $w(\pi) = w$ then there exists q -cycle π' with $w(\pi') = w'$ and $d(\pi') \geq w(\pi') - \gamma$.

Proving this lemma involves some tedious analysis of paths in G , but it is not too complicated. Note that the drop of π' does not get too large. We can now generalise Lemma 11 to arbitrary paths, and we define the *global period* δ as the least common multiple of $\gcd S$ of all SCCs in G . It is easily checked that $\delta = \exp(|\mathcal{A}|)$. Now consider an arbitrary q - q' path π in G with negative weight. If we find a q'' -cycle π' along π with $w(\pi') < -\gamma$, we can invoke Lemma 11 in order to obtain a q'' -cycle π'' with $w(\pi') \equiv w(\pi'') \pmod{\delta}$. Thus, by using a counting argument on the number of control locations of \mathcal{A} , we can define a *global threshold* $\varepsilon = \exp(|\mathcal{A}|)$ that guarantees the existence of such a cycle. This allows us to state a variant of Lemma 11 for arbitrary paths:

Lemma 12. *For all $w, w' \in \mathbb{Z}$ such that $w, w' < -\varepsilon$ and $w \equiv w' \pmod{\delta}$, whenever there exists a q - q' path π with $w(\pi) = w$ then there exists a q - q' path π' with $w(\pi') = w'$ and $d(\pi') \geq w(\pi') - \gamma$.*

We can now “re-import” the observations made for paths in weighted graphs to paths in $T(\mathcal{A})$ and sketch how to prove Lemma 8. To this end, we define $\tau \stackrel{\text{def}}{=} 2\varepsilon$. Regarding Point 1 of the lemma, we have that $\min\{n, n'\} - \min\{m, m'\} > \varepsilon$. Lemma 12 thus guarantees the existence of a path π with $w(\pi) = n - m$ if, and only if, there is a path π' with $w(\pi') = n' - m'$. Since $d(\pi) \geq w(\pi) - \tau$ and $m > \tau$, the existence of a run $q(n) \xrightarrow{*}_{\mathcal{A}} q'(m)$ is guaranteed. The same argument yields a run $q(n') \xrightarrow{*}_{\mathcal{A}} q'(m')$. Finally regarding Point 2, by using a symmetry argument, we can get a similar statement as in Lemma 12 for paths with positive weight that exceed ε . The existence of the desired runs then follows from an argument similar to Point 1.

5 Conclusion

We have strengthened our results from [6] and have proved that model checking the CTL fragment EF on POCA is undecidable via reduction from Hilbert’s tenth problem. We showed that, when dropping the reachability modality, we regain decidability: Model checking ML on POCA is PSPACE-complete, which was proved by showing the existence of small solutions for a class of systems of linear Diophantine inequalities whose matrix norm is small. We showed that it is also PSPACE-complete to model check EF on SOCA by establishing an exponential periodicity property. It is interesting to mention that, in contrast to CTL, one can avoid an exponential complexity jump for EF and ML when model checking SOCA. More precisely, model checking EF (respectively ML) is P^{NP} -complete (respectively P-complete) on OCA, whereas it is PSPACE-complete for SOCA.

References

1. Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., Vojnar, T.: Programs with Lists are Counter Automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 517–531. Springer, Heidelberg (2006)

2. Bozga, M., Iosif, R.: On Decidability within the Arithmetic of Addition and Divisibility. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 425–439. Springer, Heidelberg (2005)
3. Bozga, M., Iosif, R., Lakhnech, Y.: Flat Parametric Counter Automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 577–588. Springer, Heidelberg (2006)
4. Chitic, C., Rosu, D.: On validation of xml streams using finite state machines. In: Proc. of WebDB, pp. 85–90. ACM, New York (2004)
5. Comon, H., Jurski, Y.: Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, Springer, Heidelberg (1998)
6. Göller, S., Haase, C., Ouaknine, J., Worrell, J.: Model Checking Succinct and Parametric One-Counter Automata. In: Abramsky, S., Gavaille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 575–586. Springer, Heidelberg (2010)
7. Göller, S., Lohrey, M.: Branching-time Model Checking of One-counter Processes. In: Proc. of STACS. LIPIcs, vol. 5, pp. 405–416. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010)
8. Göller, S., Mayr, R., To, A.W.: On the Computational Complexity of Verifying One-Counter Processes. In: Proc. of LICS, pp. 235–244. IEEE Computer Society (2009)
9. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in Succinct and Parametric One-Counter Automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009)
10. Hague, M., Lin, A.W.: Model Checking Recursive Programs with Numeric Data Types. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 743–759. Springer, Heidelberg (2011)
11. Ibarra, O.H., Dang, Z.: On the solvability of a class of diophantine equations and applications. *Theor. Comput. Sci.* 352(1), 342–346 (2006)
12. Ibarra, O.H., Jiang, T., Tr an, N., Wang, H.: New decidability results concerning two-way counter machines and applications. In: Lingas, A., Carlsson, S., Karlsson, R. (eds.) ICALP 1993. LNCS, vol. 700, Springer, Heidelberg (1993)
13. Jan car, P., Ku cera, A., Mayr, R.: Deciding bisimulation-like equivalences with finite-state processes. *Theor. Comput. Sci.* 258(1-2), 409–433 (2001)
14. Lange, M.: Model checking propositional dynamic logic with all extras. *J. Applied Logic* 4(1), 39–49 (2006)
15. Leroux, J., Sutre, G.: Flat Counter Automata Almost Everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)
16. Matiyasevich, Y.: Enumerable sets are Diophantine. *Soviet Math. Dokl.* 11, 354–357 (1970)
17. Minsky, M.L.: Recursive unsolvability of Post’s problem of “tag” and other topics in theory of Turing machines. *Annals of Mathematics. Second Series* 74, 437–455 (1961)
18. Robinson, J.: Definability and Decision Problems in Arithmetic. *J. Symbolic Logic* 14(2), 98–114 (1949)
19. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., New York (1986)
20. Serre, O.: Parity Games Played on Transition Graphs of One-Counter Processes. In: Aceto, L., Ing lfsd ttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 337–351. Springer, Heidelberg (2006)
21. Shallit, J.: The Frobenius Problem and its Generalizations. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 72–83. Springer, Heidelberg (2008)
22. To, A.W.: Model Checking FO(R) over One-Counter Processes and beyond. In: Gr adel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 485–499. Springer, Heidelberg (2009)

Synthesizing Probabilistic Composers^{*}

Sumit Nain and Moshe Y. Vardi

Department of Computer Science,
Rice University, Houston, TX 77005, USA
{nain,vardi}@cs.rice.edu

Abstract. Synthesis from components is the automated construction of a composite system from a library of reusable components such that the system satisfies the given specification. This is in contrast to classical synthesis, where systems are always “constructed from scratch”. In the *control-flow* model of composition, exactly one component is in control at a given time and control is switched to another when the component reaches an exit state. The composition can then be described implicitly by a transducer, called a *composer*, which *statically* determines how the system transitions between components.

Recently, Lustig, Nain and Vardi have shown that control-flow synthesis of deterministic composers from libraries of probabilistic components is decidable. In this work, we consider the more general case of probabilistic composers. We show that probabilistic composers are more expressive than deterministic composers, and that the synthesis problem still remains decidable.

Keywords: synthesis, temporal logic, probabilistic components.

1 Introduction

Hardware and software systems are rarely built from scratch. Almost every non-trivial system is based on existing components. A typical component might be used in the design of multiple systems. Examples of such components include function libraries, web APIs, and ASICs. The construction of systems from reusable components is an area of active research. Some examples of important work on the subject can be found in Sifakis’ work on component-based construction [11], and de Alfaro and Henzinger’s work on “interface-based design” [7]. Furthermore, other situations, such as web-service orchestration [3], can be viewed as the construction of systems from libraries of reusable components.

Synthesis is the automated construction of a system from its specification. In contrast to model checking, which involves verifying that a system satisfies the given specification, synthesis aims to automatically construct the required system from its formal specification. The modern approach to temporal synthesis was initiated by Pnueli and Rosner who introduced linear temporal logic

^{*} Work supported in part by NSF grants CCF-0728882 and CNS 1049862, and BSF grant 9800096. The authors are grateful to Orna Kupferman for suggesting the study of probabilistic composers.

(LTL) synthesis [10]. In LTL synthesis, the specification is given in LTL and the system constructed is a finite-state transducer modeling a reactive system. In this setting it is always assumed that the system is “constructed from scratch” rather than “composed” from existing components. Recently, Lustig and Vardi [9] introduced the study of synthesis from reusable components. The use of components abstracts much of the detailed behavior of a sub-system, and allows one to write specifications that mention only the aspects of sub-systems relevant for the synthesis of the system at large.

A major concern in the study of synthesis from reusable components is the choice of a mathematical model for the components and their composition. The exact nature of the reusable components in a software library may differ. One finds in the literature many different types of components; for example, function libraries (for procedural programming languages) or object libraries (for object-oriented programming languages). Indeed, there is no single “right” model encompassing all possible facets of the problem. The problem of synthesis from reusable components is a general problem to which there are as many facets as there are models for components and types of composition [11].

Our model is based on the *control-flow composition* model of [9]. In this model, a component is just a *transducer*, i.e., a finite-state machine with outputs. Transducers constitute a canonical model for reactive components, abstracting away internal architecture and focusing on modeling input/output behavior. In contrast to [9], we allow components to be probabilistic, i.e., the transducers have a probabilistic transition function. The use of probabilistic transducers is a common approach to modeling systems where there is probabilistic uncertainty about the results of input actions. Intuitively, we aim at constructing a reliable system from unreliable components. There is a rich literature about verification and analysis of such systems, cf. [5,6,12,13], as well about synthesis in the face of probabilistic uncertainty [1]. The introduction of probability requires us to use a probabilistic notion of correctness; here we choose the *qualitative* criterion that the specification be satisfied with probability 1, leaving the study of *quantitative criteria* to future work.

In control-flow composition, control is held by a single component at every point in time. When the current component reaches an exit state, the execution passes to the start state of another component. The flow of control between components can itself be modeled by a supervisory transducer called a *composer*. Given the name of the current component and exit state, the composer gives the name of the next component in control. In essence, the composer describes how the composite system is put together and can be viewed as an implicit description of the composite system. The goal of the synthesis problem is then to find a suitable composer such that the resulting system satisfies the specification.

Here we consider two kinds of specification formalism: *embedded parity* and *deterministic parity automaton* (DPW). An embedded parity specification is given by associating a natural number called a priority to each state of each component in the given set of components. The specification is satisfied if the run of the system satisfies the parity condition with probability 1. A DPW

specification is satisfied if the input-output behavior of the system is accepted by the DPW with probability 1.

In previous work [8], the synthesis problem for probabilistic components was shown to be decidable for both DPW and embedded parity specifications. However, while the components there were probabilistic, the composer, and thus the control-flow of the composite system, was still deterministic. Since the composer is itself a transducer that describes the flow of control between components in a composition, it is natural to consider the more general case of allowing the composer to also be a probabilistic transducer. In this case, not just the behavior of individual components, but also the flow of control between components is probabilistic. Does this allow the composite system to satisfy more specifications? That is, we would like to know whether probabilistic composers are more *expressive* than deterministic composers. And if so, how do we synthesize them? These questions are the focus of this paper.

We have two main goals: to investigate the expressiveness of probabilistic composers and to solve the synthesis problem for probabilistic composers. We show that expressive power depends on the type of specification. For embedded parity specifications, allowing the composer to be probabilistic gives no advantage. In particular, if a suitable probabilistic composer exists then a suitable deterministic composer also exists. In contrast, for the more general case of a DPW specification, we find that probabilistic composers are more expressive. We give an instance of the DPW synthesis problem such that there exists a suitable probabilistic composer that solves it, but no suitable deterministic composer exists. We note that expressiveness is not just a theoretical concern. The fact that probabilistic composers are more expressive means that some systems can only be constructed using a probabilistic composition. As a result the DPW synthesis problem for probabilistic composers becomes important in its own right and not just as an extension of the deterministic case. It is interesting that probabilistic composers only gain expressive power in the presence of specifications with memory (DPW specifications). We view this as another example of a memory vs. randomness trade-off that is well-known in the game-theoretic literature [4].

In [8], the DPW synthesis problem for deterministic composers is solved by reducing it to the embedded parity version. But for probabilistic composers, a similar approach does not work because their expressive power differs for embedded parity and DPW specifications. Instead, we solve the DPW synthesis problem for probabilistic composers by a reduction to deterministic composers. The key insight is that it is possible to equip the library with an additional component with a specific structure, called M_{rand} , such that the probabilistic choices made by a composer can be simulated by the probabilistic transitions within M_{rand} . The idea is that whenever a probabilistic composer C makes a probabilistic transition, the equivalent deterministic composer C' can instead call an instance of M_{rand} to simulate the moves of C . The result is that the DPW synthesis problem for probabilistic composers is decidable.

Finally, the embedded-parity version of our synthesis problem can also be viewed as a partial information stochastic game between two players, the

composer C, which chooses components, and the environment E, which chooses paths through the components chosen by C. The partial information arises because our model of composition is static, which means the components chosen by the composer cannot depend on the inputs selected by the environment. In contrast to standard parity games, partial information stochastic games are known to be undecidable even for co-Büchi objectives (and thus for parity objectives) [2]. Thus, in the framework of games, our result can be viewed as a rare positive result for partial-information stochastic games. We explain this game-theoretic view of the problem in more detail in Section 7.

This paper is self-contained, except for certain proofs that have been omitted to save space; a longer version is posted on the authors' home pages.

2 Preliminaries

A *deterministic transducer* is a tuple $B = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $L : Q \rightarrow \Sigma_O$ is an output function labeling states with output letters, and $\delta : Q \times \Sigma_I \rightarrow Q$ is a transition function.

A *strongly connected component* of a directed graph $G = (V, E)$ is a subset U of V , such that for all $u, v \in U$, u is reachable from v . We can define a natural partial order on the set of maximal strongly connected components of G as follows: $U_1 \leq U_2$ if there exists $u_1 \in U_1$ and $u_2 \in U_2$ such that u_1 is reachable from u_2 . An *ergodic set* of G is a minimal element of this partial order.

A probability distribution on a finite set X is a function $\mu : X \rightarrow [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$. The *support* of μ , denoted $supp(\mu)$, is the set $\{x \in X : \mu(x) > 0\}$. $Dist(X)$ denotes the set of all probability distributions on set X . A *probabilistic transducer*, is a tuple $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, F, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : (Q - F) \times \Sigma_I \rightarrow Dist(Q)$ is a probabilistic transition function, $F \subseteq Q$ is a set of exit states, and $L : Q \rightarrow \Sigma_O$ is an output function labeling states with output letters. Note that there are no transitions out of an exit state. If F is empty, we say \mathcal{T} is a probabilistic transducer without exits.

Given a probabilistic transducer $M = (\Sigma_I, \Sigma_O, Q, q_0, \delta, F, L)$, a *strategy* for M is a function $f : Q^* \rightarrow Dist(\Sigma_I)$ that probabilistically chooses an input for each sequence of states. A strategy is *memoryless* if the choice depends only on the last state in the sequence. A memoryless strategy is a function $g : Q \rightarrow Dist(\Sigma_I)$. A strategy is *pure* if the choice is deterministic. A pure strategy is a function $h : Q^* \rightarrow \Sigma_I$, and a memoryless and pure strategy is a function $h : Q \rightarrow \Sigma_I$.

A strategy f along with a probabilistic transducer M , with set of states Q , induces a probability distribution on Q^ω , denoted μ_f . By standard measure theoretic arguments, it suffices to define μ_f for the cylinders of Q^ω , which are sets of the form $\beta \cdot Q^\omega$, where $\beta \in Q^*$. First we extend δ to exit states as follows: for $a \in \Sigma_I$, $q \in F$, $q' \in Q$, $\delta(q, a)(q) = 1$ and $\delta(q, a)(q') = 0$ when $q' \neq q$. Then we define $\mu_f(q_0 \cdot Q^\omega) = 1$, and for $\beta \in Q^*$, $q, q' \in Q$, $\mu_f(\beta q q' \cdot Q^\omega) = \mu_f(\beta q) (\sum_{a \in \Sigma_I} f(\beta q)(a) \times \delta(q, a)(q'))$. These conditions say that there is a unique

start state, and the probability of visiting a state q' , after visiting βq , is the same as the probability of the strategy picking a particular letter multiplied by the probability that the transducer transitions from q to q' on that input letter, summed over all input letters.

Let M be a probabilistic transducer, Q be its set of states, and f be a memory-less strategy for M . We define the graph induced by f on Q , denoted by $G_{M,f}$, as the directed graph (Q, E) , where $(q_1, q_2) \in E$ if $\sum_{a \in \Sigma_I} f(q_1)(a) \delta(q_1, a)(q_2) > 0$. That is, there is an edge from q_1 to q_2 if the transducer can transition from the state q_1 to the state q_2 on an input letter that the strategy chooses with positive probability. Given $q_1, q_2 \in Q$, we say that q_2 is reachable from q_1 if there is a path from q_1 to q_2 in $G_{M,f}$. We say a state is ergodic if it belongs to some ergodic set of $G_{M,f}$. An ergodic set is reachable if there is a path from the start state to some state in the ergodic set. A state q of M is *reachable under f* , if there is a path in $G_{M,f}$ from q_0 to q .

A *library* is a set of probabilistic transducers that share the same input and output alphabets. Each transducer in the library is called a *component*. Given a finite set of directions D , we say a library \mathcal{L} has width D , if each component in the library has exactly $|D|$ exit states. Since we can always add dummy unreachable exit states to any component, we assume, w.l.o.g., that all libraries have an associated width, usually denoted D . In the context of a particular component, we often refer to elements of D as exits, and subsets of D as sets of exits.

An *index function* for a transducer is a function that assigns a natural number, called a priority index, to each state of the transducer. An index function for a library is a function that assigns a priority to every state of every component in the library. Given an index function α , and a set of states X , we denote by $\alpha(X)$ the highest priority assigned by α in X .

3 Control-Flow Composition

Let \mathcal{L} be a library with width D . We first informally describe our notion of probabilistic control-flow composition of components from \mathcal{L} . Each library component can be used multiple times in a composition, and we treat these occurrences as distinct *component instances*. Thus, the size of a composition, a priori, is not bounded. The component instances in a composition take turns interacting with the environment, and at each point in time, exactly one component instance is active. When the active component instance reaches an exit state, control is transferred probabilistically to the start state of some other component instance.

Given a set of component instances to be composed, a control flow composition can be defined by giving, for each exit state of each component instance, the probability distribution that determines the probability of transitioning from that exit state to another component instance. This information can be represented naturally by a probabilistic transducer, called a (probabilistic) *composer*, whose set of states corresponds to the set of component instances and whose input alphabet corresponds to the set of exits D . Then, for each component

instance and exit, the transition function of the composer gives the probability distribution that determines which component instance will be in control next.

Formally, a *composer* over a library \mathcal{L} with width D is a probabilistic transducer $C = (D, \mathcal{L}, \mathcal{M}, M_0, \Delta, \lambda)$. Here \mathcal{M} is an arbitrary finite set of states. In particular, there is no a priori bound on the size of \mathcal{M} . Each $M_i \in \mathcal{M}$ is the name of an instance of a component from \mathcal{L} and $\lambda(M_i) \in \mathcal{L}$ is the type of M_i . The transition function $\Delta : \mathcal{M} \times D \rightarrow Dist(\mathcal{M})$ gives a distribution on the set of instance names. We say a composer is a *deterministic composer* if its transition function is deterministic.

Note that while each component name M_i is distinct, the corresponding component instances $\lambda(M_i)$ need not be distinct. Each composer defines a unique composition over components from \mathcal{L} . The current state of the composer corresponds to the component that is in control. The transition function Δ describes how to transfer control between components: $\Delta(M, i)(M') = p$ denotes that when the composition is in the i th exit state of component instance $\lambda(M)$ it moves to the start state of component instance $\lambda(M')$ with probability p . A composer can be viewed as an implicit representation of a composition. We give the explicit definition of composition below.

Definition 1 (Control-flow Composition). *Let $C = (D, \mathcal{L}, \mathcal{M}, M_0, \Delta, \lambda)$ be a composer over library \mathcal{L} with width D , such that $\mathcal{M} = \{M_0, \dots, M_n\}$, $\lambda(M_i) = (\Sigma_I, \Sigma_O, Q_i, q_0^i, \delta_i, F_i, L_i)$ and $F_i = \{q_x^i : x \in D\}$. The composition defined by C , denoted T_C , is a probabilistic transducer $\langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \emptyset, L \rangle$, where $Q = \bigcup_{i=0}^n (Q_i \times \{i\})$, $q_0 = \langle q_0^0, 0 \rangle$, $L(\langle q, i \rangle) = L_i(q)$, and the transition function δ is defined as follows: For $\sigma \in \Sigma_I$, $\langle q, i \rangle \in Q$ and $\langle q', j \rangle \in Q$,*

1. *If $q \in Q_i \setminus F_i$, then*

$$\delta(\langle q, i \rangle, \sigma)(\langle q', j \rangle) = \begin{cases} \delta_i(q, \sigma)(q') & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

2. *If $q = q_x^i \in F_i$, then $\delta(\langle q, i \rangle, \sigma)(\langle q', j \rangle) = \Delta(M_i, x)(M_k)$.*

When the composition is in a state $\langle q, i \rangle$ corresponding to a non-exit state q of component instance $\lambda(M_i)$, it behaves like $\lambda(M_i)$. When the composition is in a state $\langle q_f, i \rangle$ corresponding to an exit state q_f of component instance $\lambda(M_i)$, the control is transferred to the start state of another component instance as determined by the transition function of the composer. Thus, at each point in time, only one component is active and interacting with the environment. Note that our notion of composition is *static*, where the components called are determined before run time, rather than *dynamic*, where the components called are determined during run time.

Definition 2. *Given a library \mathcal{L} with width D , an exit control relation is a set $R \subseteq D \times \mathcal{L}$. We say that a composer $C = (D, \mathcal{L}, \mathcal{M}, M_0, \Delta, \lambda)$ over \mathcal{L} is compatible with R , if the following holds: for all $M, M' \in \mathcal{M}$ and $i \in D$, if $\Delta(M, i) = M'$ then $(i, M') \in R$. Thus, each element of R can be viewed as a constraint on how the composer is allowed to connect components.*

4 Defining the Synthesis and Realizability Problems

The goal of synthesis from components is to find a composer C over library \mathcal{L} such that \mathcal{T}_C , the composition defined by C , satisfies the given specification, which is typically some ω -regular property. In general, a specification property is usually defined as a subset of $(\Sigma \times Q)^\omega$ where Σ is the input alphabet and Q is the set of states of the system. However, here we assume, without loss of generalization, that the states of the transducers ‘remember’ the input, so defining a property as a subset of Q^ω is sufficient. In this paper, we focus on two different but related formalisms for describing ω -regular properties:

- *embedded parity specification*: This is a simple specification given by an *index function* α , which assigns a *priority* to each state of the system. We say a run of the system satisfies the *parity condition* if the highest priority visited infinitely often is even. Then the ω -regular property defined by the specification is just the set of all runs that satisfies the parity condition.
- *deterministic parity word automata* (DPW): This is a more powerful formalism that can express all ω -regular specifications. Given a DPW A , the ω -regular property defined by A is simply the language of A .

While the two formalisms differ in power, they ultimately both use the parity condition to check whether a run of the system is accepting or not. As we see in Section 5, the contrast between the relatively weak embedded parity specification and the more general DPW specification is a useful tool to study the expressiveness of probabilistic composers. In particular, we note that a DPW specification has memory while an embedded parity specification is memoryless. As we see in the next section, this difference illuminates the issue of expressiveness of probabilistic composers.

Now, in order to formalize the synthesis problem, we need to define an appropriate notion of correctness for a reactive probabilistic system w.r.t. a given specification. We assume the presence of an adversarial environment that controls the input, while the system, a probabilistic transducer, controls the output. We require that the executions of the system satisfy the specification with probability 1 irrespective of the inputs selected by the environment. Our notion of correctness is *qualitative* [12].

Definition 3. *Let M be a probabilistic transducer, Q be the state space of M , and $P \subseteq Q^\omega$ be an ω -regular property. Let f be a strategy for M . Then f is winning for the environment if $\mu_f(P) < 1$. We say that M satisfies P if there exists no winning strategy for the environment. If M satisfies P , and the property P is given by index function α or DPW A , we say M satisfies α or, respectively, M satisfies A .*

Let \mathcal{L} be a library, α be an index function, and A be a DPW. Let C be a composer over \mathcal{L} . We say C satisfies α or C satisfies A , if \mathcal{T}_C satisfies α or, respectively, \mathcal{T}_C satisfies A .

The two types of specifications give rise to two related synthesis problems.

Definition 4. *The embedded parity realizability problem for probabilistic composers is: Given a library \mathcal{L} with width D , an exit control relation R for \mathcal{L} , and*

an index function α for \mathcal{L} , decide whether there exists a probabilistic composer C over \mathcal{L} , such that \mathcal{T}_C satisfies α and C is compatible with R . If such a composer exists we say \mathcal{L} realizes α under R . The embedded parity synthesis problem for probabilistic composers is to find such a composer C if it exists.

Definition 5. *The DPW realizability problem for probabilistic composers is: Given a library \mathcal{L} and a DPW specification A , decide whether there exists a probabilistic composer C over \mathcal{L} , such that \mathcal{T}_C satisfies A . If such a composer exists, we say that \mathcal{L} realizes A . The DPW synthesis problem for probabilistic composers is to find such a composer C if it exists.*

We can obtain weaker versions of these problems by restricting ourselves to deterministic composers. The resulting problems are then known to be decidable:

Theorem 1. [8] *The embedded parity realizability and synthesis problems for deterministic composers are decidable.* \square

Theorem 2. [8] *The DPW realizability and synthesis problems for deterministic composers are decidable.* \square

We observe that while one might hope to solve the more general case of probabilistic composers by using the methods of [8], there are two main difficulties with that approach. First, in [8], the DPW version of the problem is solved by reducing it to the embedded parity version. However, as we show in the next section, while deterministic and probabilistic composers have the same expressive power for embedded parity specifications, probabilistic composers are strictly more expressive than deterministic composers for DPW specifications. Thus it is not possible to reduce the DPW realizability of probabilistic composers to the embedded parity version. Second, the automata theoretic techniques used in [8] crucially depend on the fact that every deterministic composer can be represented as a regular D -tree (a tree with constant branching degree D) where D is the width of the library. This is because when the control-flow is deterministic, there is exactly one successor component for each exit of a component, and so the number of outgoing edges from each component is always the same as the number of exits. When the composer is probabilistic, the branching degree of its unfolding can be as large as the number of its states, which is not a priori bounded.

5 The Expressive Power of Probabilistic Composers

Given a specification formalism, it is natural to ask the following question: Do we gain any additional power for solving the synthesis problem by allowing composers to be probabilistic? That is, are there instances of the synthesis problem (i.e. a library and a specification) such that some probabilistic composer satisfies the specification, but no deterministic composer does? If the answer is yes, then we say that probabilistic composers are more expressive than deterministic composers for that class of specifications. Otherwise, we say they are equally expressive.

5.1 Embedded Parity Specifications

We first consider the issue of expressiveness for embedded parity specifications. Our main result here is that deterministic and probabilistic composers are both equally expressive for embedded parity specifications. To prove this, we need to show that, for every library \mathcal{L} , exit control relation R and index function α , if there is a probabilistic composer over \mathcal{L} that satisfies α under R , then we can also find a deterministic composer over \mathcal{L} that satisfies α under R . We first recall the following useful characterization of winning strategies.

Theorem 3. [8] *Let M be a probabilistic transducer and α be an index function.*

1. *If there exists a winning strategy for the environment then there exists a pure and memoryless winning strategy.*
2. *Let f be a memoryless strategy for M . Then f is winning for the environment iff $G_{M,f}$ has a reachable ergodic set whose highest priority is odd.* \square

The key idea behind our approach is that instead of directly comparing probabilistic composers to deterministic composers, we should instead compare a probabilistic composer C to a composer C' which is ‘less probabilistic’ than it. We formalize what it means for one composer to be less probabilistic than another, by introducing a partial order, denoted \leq_{prob} , on the set of all composers.

Definition 6. *Let $\text{COMP}(\mathcal{L})$ be the set of all probabilistic composers over \mathcal{L} . We define the partial order \leq_{prob} on $\text{COMP}(\mathcal{L})$: for $C_1 = (D, \mathcal{L}, \mathcal{M}_1, M_0^1, \Delta_1, \lambda_1)$ and $C_2 = (D, \mathcal{L}, \mathcal{M}_2, M_0^2, \Delta_2, \lambda_2)$, we have $C_1 \leq_{\text{prob}} C_2$ if*

- $\mathcal{M}_1 = \mathcal{M}_2$, $M_0^1 = M_0^2$, and $\lambda_1 = \lambda_2$
- $\forall i \in D, M \in \mathcal{M}_1$, $\text{supp}(\Delta_1(M, i)) \subseteq \text{supp}(\Delta_2(M, i))$

If $C_1 \leq_{\text{prob}} C_2$ and $C_2 \leq_{\text{prob}} C_1$, we say that C_1 and C_2 are qualitatively equivalent. We denote by $<_{\text{prob}}$ the strict partial order corresponding to \leq_{prob} .

Thus $C_1 <_{\text{prob}} C_2$ if they have the same set of states and output functions and the set of possible transitions of C_1 is a proper subset of the set of possible transitions of C_2 . We note that $<_{\text{prob}}$ is a well-founded relation, and deterministic composers are the minimal elements of the relation. The well-foundedness of $<_{\text{prob}}$ is crucial because it allows the use of induction.

Next we show that if all the composers obtained by removing transitions from a composer C , fail to satisfy α under R , then C itself also fails to satisfy α under R . The intuition here is that the behavior of C can be determined by looking at the behavior of composers that make fewer probabilistic choices than C but have the same underlying structure.

Lemma 1. *Let \mathcal{L} be a library with width D , R be an exit control relation and α be an index function for \mathcal{L} , and let $C \in \text{COMP}(\mathcal{L})$ be such that C is not deterministic. Suppose that for all $C' \in \text{COMP}(\mathcal{L})$ and $C' <_{\text{prob}} C$, we have that C' does not satisfy α under R . Then C also does not satisfy α under R .*

Proof. Let $C = (D, \mathcal{L}, \mathcal{M}, M_0, \Delta, \lambda) \in \text{COMP}(\mathcal{L})$. If C is not compatible with R , then C trivially does not satisfy α under R . So we assume that C is compatible with R . We first arbitrarily choose $M_a \in \mathcal{M}$ and $i_0 \in D$ such that

$|supp(\Delta(M_a, i_0))| > 1$. That is, we require that the transition out of state M_a on input i_0 is not deterministic. Since it is given that C is not a deterministic composer, there is at least one such state and input pair. Consider M_a and i_0 to be fixed for the rest of this proof.

Let $supp(\Delta(M_a, i_0)) = \{M_1, \dots, M_k\} \subseteq \mathcal{M}$. For $1 \leq j \leq k$, we define probabilistic composers $C_j = (D, \mathcal{L}, \mathcal{M}, M_0, \Delta_j, \lambda)$, where $\Delta_j(M_a, i_0) = M_j$ and for all $M \in \mathcal{M}$, $i \in D$, $\Delta_j(M, i) = \Delta(M, i)$ when $i \neq i_0$ or $M_a \neq M$. Thus each C_j is deterministic at state M_a and behaves exactly like C at other states. Further, for each possible choice of next state available to C at state M_a on input i_0 , there is exactly one of the C_j 's that makes that choice deterministically. By construction, for all $1 \leq j \leq k$, we have $C_j \in COMP(\mathcal{L})$, $C_j < C$, and C_j is compatible with R . So, by the assumption in the theorem statement, C_j does not satisfy α for all $1 \leq j \leq k$. Then, by Definition 3 and Theorem 3, for each $1 \leq j \leq k$, there exists memoryless strategy f_j for \mathcal{T}_{C_j} that is winning for the environment. Note that each f_j is also a memoryless strategy for \mathcal{T}_C .

Let Q be the set of states of \mathcal{T}_C , q_0 be the start state of $\lambda(M_0)$ and q be the exit state of $\lambda(M_a)$ in direction i_0 . For $1 \leq j \leq k$, let $G_j = G_{\mathcal{T}_C, f_j}$ and $G'_j = G_{\mathcal{T}_{C_j}, f_j}$. Now, by construction, we have $\Delta_j(M, i) = \Delta(M, i)$ for $i \neq i_0$ or $M \neq M_a$. That is, C and C_j differ in their choices only when the composition is in exit state q of component $\lambda(M_a)$. Further, G_j and G'_j have the same set of vertices Q and G'_j is a subgraph of G_j . Thus all edges that lie in G_j but not in G'_j must have q as their source. Let X_j be a reachable ergodic set of G'_j such that $\alpha(X_j)$ is odd. Such an X_j must exist because f_j is winning for the environment for \mathcal{T}_{C_j} . Then, since G'_j is a subgraph of G_j , X_j is also reachable and strongly connected in G_j . Also q is the source of all the edges that lie in G_j but not in G'_j . So if q does not lie in X_j , then no edges can leave X_j in G_j and X_j is also a reachable ergodic set of G_j . In this case, f_j is also a winning strategy for the environment for \mathcal{T}_C . Note that this argument does not depend on the particular value of j . For the rest of the proof we can assume that $q \in X_j$ for all $1 \leq j \leq k$.

Let $X = \bigcup_{j=1}^k X_j$ and $x \in X$ be such that $\alpha(x) = \alpha(X)$. Since, by definition, $\alpha(X_j)$ is odd for all $1 \leq j \leq k$, therefore $\alpha(x)$ is also odd. We assume, without loss of generalization, that $x \in X_1$ and define a memoryless strategy f for \mathcal{T}_C such that: $f(x) = f_1(x)$ for $x \in (Q - X) \cup X_1$, and $f(x) = f_j(x)$ for $x \in X_j - \bigcup_{i=1}^{j-1} X_i$. Let $G = G_{\mathcal{T}_C, f}$. Since f takes the same values as f_1 on X_1 , then X_1 must be strongly connected in G . We first show that no edges in G leave X . Since X_1 is an ergodic set of G_1 , therefore q is the source of all the edges that leave X_1 in G . By construction of the C_j , each of these edges goes to some X_j . Thus no edges from X_1 leave X in G . Similarly, any edge leaving a vertex $x \in X_j - \bigcup_{i=1}^{j-1} X_i$ must stay in X_j , because f and f_j agree on those vertices and $q \notin X_j - \bigcup_{i=1}^{j-1} X_i$. Thus there are no edges leaving X in G . Thus X must contain at least one ergodic set of G . Let Y be this ergodic set of G .

We next show that X_1 is reachable in G from every vertex in X . Clearly X_1 is reachable in G from X_1 . Assume that X_1 is reachable in G from every vertex in $X_m - \bigcup_{i=1}^{m-1} X_i$, for all $1 < m \leq j$, for some $j < k$. Let $X' = X_{j+1} - \bigcup_{i=1}^j X_i$ and let $x \in X'$. We claim that there is a path in G that starts from x and leaves

X' . If there is no such path, then some $Y' \subseteq X'$ must be an ergodic set of G . Since f and f_{j+1} agree on vertices in X' , if Y' is an ergodic set of G , then Y' is also an ergodic set of G_{j+1} . But this contradicts the fact that X_{j+1} is an ergodic set of G'_{j+1} , and so is strongly connected in G_{j+1} . Thus there is a path in G that starts from x and leaves X' . Further, any outgoing edge from a vertex in X' cannot leave X_{j+1} , because f and f_{j+1} agree on X' . Thus there is a path from x to some vertex y in $X_{j+1} - X' = \bigcup_{i=1}^j X_i$. Now, by the inductive hypothesis, X_1 is reachable in G from y . Thus X_1 is also reachable in G from x . Since x was chosen arbitrarily, X_1 is reachable in G from every vertex in $X_{j+1} - \bigcup_{i=1}^j X_i$. By induction, X_1 is reachable in G from every vertex in X . In particular, X_1 is reachable in G from the ergodic set $Y \subseteq X$ of G . Thus, X must be contained in Y . Then we have $\alpha(x) \leq \alpha(X_1) \leq \alpha(Y) \leq \alpha(X) = \alpha(x)$. Thus $\alpha(Y)$ is also odd.

Finally, all that remains is to show that Y is reachable from the start state q_0 . Since f_1 is winning for the environment for \mathcal{T}_{C_1} , we know that X_1 is reachable in G_1 from the start state q_0 . Since $X_1 \subseteq X$, X is also reachable in G_1 from q_0 . Consider the shortest path in G_1 that starts from q_0 and reaches X . Then all vertices, except the last one, in this path are in $Q - X$. Since f and f_1 agree on vertices in $Q - X$, this path is also present in G . Then X is reachable from q_0 in G and so Y is reachable from q_0 in G . Thus f is a pure and memoryless winning strategy for the environment. □

Finally, we show that, for embedded parity specifications, probabilistic composers are not more expressive than deterministic composers.

Theorem 4. *Let \mathcal{L} be a library with width D and α be an index function for \mathcal{L} . There is a probabilistic composer $C \in \text{COMP}(\mathcal{L})$ that satisfies α if and only if there is a deterministic composer $C' \in \text{COMP}(\mathcal{L})$ that satisfies α .*

Proof. Follows immediately from Lemma [11](#) using transfinite induction on the well-founded strict partial order $<_{\text{prob}}$. □

As a consequence of Theorem [4](#), together with Theorem [11](#), we obtain:

Theorem 5. *The embedded parity synthesis problem for probabilistic composers is decidable.* □

5.2 DPW Specifications

While embedded parity specifications are memoryless, DPW specifications have an associated memory (the state of the DPW). This difference turns out to be crucial in determining the expressive power of probabilistic composers. As we see below, the ability to make random transitions allows a probabilistic composer to successfully deal with memoryful specifications where deterministic composers fail. We find that, in contrast to the embedded parity case, there are instances of the DPW realizability problem where a suitable probabilistic composer exists but no suitable deterministic composer exists. Thus, probabilistic composers are strictly more expressive than deterministic composers for DPW specifications.

We describe a suitable problem instance, consisting of a library \mathcal{L} and DPW A . Let $\Sigma = \{a, b, c, b', c'\}$ and A be a DPW that accepts a word over Σ iff it contains at least one occurrence of bab' or cac' . The language of A is $\Sigma^*(bab' + cac')\Sigma^\omega$. Consider the library $\mathcal{L} = \{M_1, M_2, M_3\}$ consisting of the three components M_1 , M_2 and M_3 , which are shown in Fig. 1 (the figure depicts a composition built using a single instance of each component). Each component in the library has a single exit state. The input alphabet of each component is $\{0, 1\}$ and the output alphabet is Σ . The components M_2 and M_3 each consist of a single state, which serves as both the start and exit state. As a result, they have no internal transitions. They are only distinguished by the output in their single state. M_2 outputs b' and M_3 outputs c' . The component M_1 has four states and its transition function is such that every run from its start state to its exit state always deterministically produces an output of either aba or aca , depending solely on the input selected by the environment in the start state.

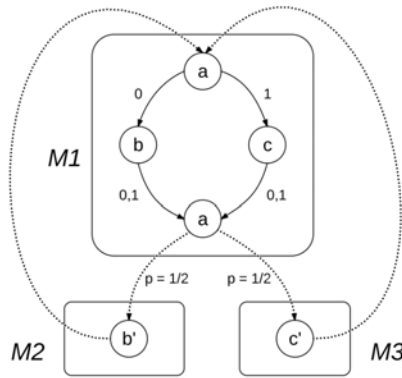


Fig. 1. A composition with probabilistic control-flow that satisfies $\Sigma^*(bab' + cac')\Sigma^\omega$

Now consider a composition built from components in \mathcal{L} that is defined by a deterministic composer. Since each component in \mathcal{L} has exactly one exit, if the composer is deterministic, then the composition can be viewed as a linear sequence of components, i.e., a pipeline. We claim that in this situation, the environment can always prevent bab' or cac' from occurring anywhere in the output. This is because bab' can only be output if an instance of M_2 occurs immediately after an instance of M_1 in the pipeline, but in any such case, the environment can always force that particular instance of M_1 to output aca instead of aba . Similarly, the environment can always prevent cac' from being output. The result is that no deterministic composer over \mathcal{L} can satisfy A .

Theorem 6. *Let \mathcal{L} and A be as defined above. Then there does not exist a deterministic composer over \mathcal{L} that satisfies A . \square*

In contrast to the deterministic case, there is a probabilistic composer over \mathcal{L} that satisfies A . Intuitively, the composer needs to overcome the fact that the

environment has complete control over the output of M_1 . It can do this by probabilistically connecting each instance of M_1 to instances of both M_2 and M_3 . Then the control that the environment has over the output of M_1 becomes irrelevant. One such composition is shown in Fig. 1, where control is transferred from the single exit state of M_1 to either M_2 or M_3 with equal probability.

Theorem 7. *Let \mathcal{L} and A be as defined above. Then there exists a probabilistic composer over \mathcal{L} that satisfies A . □*

6 Synthesizing Probabilistic Composers

In the previous section, we saw that probabilistic composers are more expressive than deterministic composers for DPW specifications, but both have the same expressive power for embedded parity specifications. This unfortunately rules out following the approach of [8] to solve the DPW synthesis problem for probabilistic composers by reducing it to the embedded parity version. Instead, we show that the DPW synthesis problem for probabilistic composers can be reduced to the DPW synthesis problem for deterministic composers. Since, by Theorem 2, the latter problem is decidable, this suffices to solve the probabilistic version too.

The key idea behind our reduction is that probabilistic choices made by a composer can be simulated with the help of a component with a specific structure. Consider a component, called M_{rand} , which ignores the environment’s input and transitions uniformly at random from its start state to each of its two exit states. Now suppose that a composer C over \mathcal{L} probabilistically calls two different components, say M_1 and M_2 . Then this behavior can be simulated by a deterministic composer, say C' , that first calls M_{rand} , and then calls M_1 and M_2 from the two exits of M_{rand} . In this way, we can replace a probabilistic composer over \mathcal{L} by a deterministic composer over the larger library $\mathcal{L} \cup \{M_{\text{rand}}\}$. We first formally define the special component M_{rand} .

Definition 7. *Let Σ_I and Σ_O be the input and output alphabets of every component in \mathcal{L} . Let b be a fresh output symbol not contained in Σ_O . We define the probabilistic transducer $M_{\text{rand}} = \{\Sigma_I, \{b\}, Q, q_0, \delta, F, L\}$, where $F = \{q_1, q_2, \dots, q_{|D|}\}$, $Q = \{q_0\} \cup F$, $L(q) = b$ for all $q \in Q$, and $\delta(q_0, a)(q) = 1/|D|$ for all $a \in \Sigma_I$ and $q \in F$.*

So M_{rand} has $|D|$ final states and $|D| + 1$ total states, it outputs b in every state, and it transitions with uniform probability from the start state to a final state irrespective of the input. Note that M_{rand} is not a component in \mathcal{L} since $b \notin \Sigma_O$ by construction. If we add M_{rand} to \mathcal{L} to obtain a larger library, we also have to translate DPW specifications for \mathcal{L} into specifications for the larger library. The idea is to modify the DPW to ignore the output of M_{rand} .

Definition 8. *Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$ be a DPW specification for \mathcal{L} . We define the DPW $A^b = (\Sigma_O \cup \{b\}, Q_A \times \{0, 1\}, (s_0, 0), \delta_A^b, \alpha_A^b)$, where $\alpha_A^b(q, 0) = \alpha_A(q)$ and $\alpha_A^b(q, 1) = 1$, and δ_A^b is defined as follows:*

- For $a \in \Sigma_O$, $i \in \{0, 1\}$ and $s \in Q_A$, $\delta_A^b((s, i), a) = (\delta_A(s), 0)$
- For $i \in \{0, 1\}$ and $s \in Q_A$, $\delta_A^b((s, i), b) = (s, 1)$

Thus A^b ignores b and behaves like A on other inputs. A^b accepts a word w iff A accepts w' where w' is the result of removing all occurrences of b in w . Note that A^b is a DPW specification for the library $\mathcal{L}' = \mathcal{L} \cup \{M_{rand}\}$.

We now reduce the problem of finding a probabilistic composer over \mathcal{L} that satisfies A to the problem of finding a deterministic composer over \mathcal{L}' that satisfies A^b . We give a mapping that transforms a deterministic composer C over \mathcal{L}' to a probabilistic composer $prob(C)$ over \mathcal{L} . The intuition behind the mapping is that C uses multiple instances of M_{rand} to simulate the probabilistic choices made by $prob(C)$, such that C and $prob(C)$ have the same behaviour.

Definition 9. Let $\mathcal{L}' = \mathcal{L} \cup \{M_{rand}\}$ and $C = (D, \mathcal{L}', \mathcal{M}, M_1, \Delta, \lambda)$ be a deterministic composer over \mathcal{L}' . Let $\mathcal{M} = \mathcal{M}_0 \cup M_{rand}$ where for all $M \in \mathcal{M}_0$, $\lambda(M) \neq M_{rand}$ and for all $M \in M_{rand}$, $\lambda(M) = M_{rand}$. Then $prob(C) = (D, \mathcal{L}, \mathcal{M}_0, M_1, \Delta', \lambda)$ is the probabilistic composer over \mathcal{L} , whose probabilistic transition function Δ' is defined as follows: For all $M \in \mathcal{M}_0$ and $i \in D$,

- $\Delta'(M, i)$ is a uniform distribution on its support
- For all $M' \in \mathcal{M}_0$, $\Delta'(M, i)(M') > 0$ if there is a finite run of C that starts in M , ends in M' , and visits only states in M_{rand} .

Note that the mapping $prob$ is not reversible, and given a probabilistic composer C over \mathcal{L} , there might not be a deterministic composer C' over \mathcal{L}' such that $C = prob(C')$. However, if we partition the set of all composers over \mathcal{L} by qualitative equivalence (see Defn. 6), then we obtain a reversible mapping. We use this reversible mapping to show that the synthesis of probabilistic composers is reducible to the synthesis of deterministic composers over a larger library.

Lemma 2. Let C' be a deterministic composer over $\mathcal{L} \cup \{M_{rand}\}$ and let C be a probabilistic composer over \mathcal{L} such that C is qualitatively equivalent to $prob(C')$. Then C satisfies A iff C' satisfies A^b . □

Theorem 8. The DPW synthesis problem for probabilistic composers is decidable. □

7 Discussion

In the framework of parity games, the embedded parity version of our synthesis problem can be viewed as a 2-player stochastic game with partial information; that is, one player cannot see the moves of the other player in full. Informally, the game is the following: We are given a library \mathcal{L} of n components each with D exits with index function α . The two players are the composer C and the environment E . Player C chooses components and player E chooses paths through the components chosen by C . However, C cannot see the moves E makes inside a component. At the start C chooses a component M from \mathcal{L} . The turn passes

to E , who chooses a sequence of inputs, inducing a path in M from its start state to some exit x of M . The turn then passes to C , which must choose some component M' in \mathcal{L} and pass the turn to E . As C cannot see the moves made by E inside M , C cannot base its choice on the run of E in M , but only on the exit induced by the inputs selected by E and previous moves made by C . So C must choose the same next component M' for different runs that reach exit x of M . In general, different runs will visit different priorities inside M . This is a two-player stochastic parity game where one of the players does not have full information. If C has a winning strategy that requires a finite amount of memory, then we can use such a strategy to obtain a suitable finite composer that satisfies the index function α , thus solving the embedded parity problem. If C has no winning strategy or if every winning strategy requires infinite memory, then α is not realizable from the library \mathcal{L} .

In contrast to standard parity games, partial information 2-player stochastic parity games are known to be undecidable in general [2]. Thus, when viewed in the framework of games, our result is a rare positive result for partial-information stochastic games. Since the general problem is undecidable, the best result one can hope for is to show that some restricted but useful class of partial information parity games is decidable. Our result on the embedded parity synthesis problem can be viewed as just such a result.

References

1. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Proc. TCS 2004, pp. 493–506. Kluwer (2004)
2. Baier, C., Bertrand, N., Größer, M.: On Decision Problems for Probabilistic Büchi Automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 287–301. Springer, Heidelberg (2008)
3. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of E -services That Export Their Behavior. In: Orłowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
4. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Trading memory for randomness. In: QEST 2004, pp. 206–217. IEEE Computer Society (2004)
5. Courcoubetis, C., Yannakakis, M.: Markov Decision Processes and Regular Events. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 336–349. Springer, Heidelberg (1990)
6. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. *Journal of the ACM* 42, 857–907 (1995)
7. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: *Engineering Theories of Software-Intensive Systems*, pp. 83–104. Springer, Heidelberg (2005)
8. Lustig, Y., Nain, S., Vardi, M.Y.: Synthesis from probabilistic components. In: Proc. CSL 2011. LIPICs, vol. 12, pp. 412–427 (2011)
9. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)
10. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. POPL 1989, pp. 179–190 (1989)

11. Sifakis, J.: A framework for component-based construction extended abstract. In: Proc. SEFM 2005, pp. 293–300. IEEE Computer Society (2005)
12. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. 26th FOCS 1985, pp. 327–338 (1985)
13. Vardi, M.Y.: Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 265–276. Springer, Heidelberg (1999)

On the Complexity of Computing Probabilistic Bisimilarity

Di Chen¹, Franck van Breugel^{2,*}, and James Worrell^{1,**}

¹ Department of Computer Science, University of Oxford, UK

² Department of Computer Science and Engineering, York University, Canada

Abstract. Probabilistic bisimilarity is a fundamental notion of equivalence on labelled Markov chains. It has a natural generalisation to a probabilistic bisimilarity pseudometric, whose definition involves the Kantorovich metric on probability distributions. The pseudometric has discounted and undiscounted variants, according to whether one discounts the future in observing discrepancies between states.

This paper is concerned with the complexity of computing probabilistic bisimilarity and the probabilistic bisimilarity pseudometric on labelled Markov chains. We show that the problem of computing probabilistic bisimilarity is **P**-hard by reduction from the monotone circuit value problem. We also show that the discounted pseudometric is rational and can be computed exactly in polynomial time using the network simplex algorithm and the continued fraction algorithm. In the undiscounted case we show that the pseudometric is again rational and can be computed exactly in polynomial time using the ellipsoid algorithm. Finally, using the notion of couplings on Markov chains, we show that the pseudometric can be used to compute bounds on the variational distance of trace distributions, which is **NP**-hard to compute directly.

1 Introduction

Probabilistic bisimilarity is a notion of equivalence for probabilistic labelled transition systems, introduced by Larsen and Skou [21]. It is based on Park and Milner’s classical notion of bisimilarity for (non-deterministic) labelled transition systems [23]. A very similar and widely used concept on Markov chains, called *lumpability*, can be found as far back as the classical text of Kemeny and Snell [20]. A system and its probabilistic bisimilarity quotient can be considered indistinguishable, and quotienting by probabilistic bisimilarity is a widely used compression technique in verification and performance analysis [18,19].

The first part of this paper concerns the complexity of computing probabilistic bisimilarity. It is known that this can be done in polynomial time, e.g., by partition refinement [2,11,32]. Our first result shows that probabilistic bisimilarity is **P**-hard, and therefore **P**-complete. As a consequence probabilistic bisimilarity

* Supported by NSERC and the Leverhulme Trust.

** Supported by the EPSRC.

is not in \mathbf{NC} unless $\mathbf{P} = \mathbf{NC}$. (Recall that \mathbf{NC} is a subclass of \mathbf{P} comprising problems that can be solved in polylogarithmic time using PRAMs of polynomial size [16]. Informally such problems are considered to be efficiently parallelisable.) By contrast, language equivalence of probabilistic automata is in \mathbf{NC} [31], as are related equivalence problems such as tree isomorphism [16].

For (non-deterministic) labelled transition systems it is known that computing bisimilarity is \mathbf{P} -complete [4,26]. However the proof in the probabilistic case requires a different construction than in *op. cit.*

For probabilistic systems it is natural to generalise from bivalent notions of equivalence, such as probabilistic bisimilarity or language equivalence [30], to quantitative measures of similarity. As well as being more informative, such measures are more meaningful in the presence of rounding errors in computation and modelling (see, for example, [15]).

In the second part of this paper we consider a *probabilistic bisimilarity pseudometric* on labelled Markov chains. This generalises the notion of probabilistic bisimilarity by assigning a *similarity distance* to pairs of states of a labelled Markov chain. The smaller the distance, the more alike the states, with states at zero distance if and only if they are probabilistic bisimilar. This pseudometric was first introduced in [12] and, together with closely related notions, has subsequently been studied in the context of systems biology [28], games [9], planning [10] and security [8], among others. The definition of the pseudometric is based on the classical Kantorovich metric on probability distributions. The pseudometric has discounted versions, which discount the future in observing discrepancies between states.

We show that for labelled Markov chains with rational transition probabilities the discounted probabilistic bisimilarity pseudometric is rational and can be computed exactly by a polynomial-time algorithm. In particular, we show that the distances can be approximated by using the network simplex algorithm repeatedly and the exact distances can be obtained from the approximated ones by means of the continued fraction algorithm. In the undiscounted case we also obtain a polynomial-time algorithm to exactly compute the pseudometric, this time using the heavier machinery of the ellipsoid algorithm. In combination with our lower bound on computing probabilistic bisimilarity we conclude that computing the pseudometric is \mathbf{P} -complete. These results go beyond previous work which only showed how to approximate the pseudometric up to some desired level of precision [7]. In the undiscounted case it was only known how to approximate the pseudometric using polynomial space [6]. We use the notion of couplings of Markov chains to show that the pseudometric is an upper bound on the variational distance between the trace distributions generated by states of the Markov chain, which is \mathbf{NP} -hard to compute directly [22].

Fu [14] shows that the complexity of approximating a bisimilarity pseudometric on probabilistic automata, which generalise labelled Markov chains, lies in the intersection of \mathbf{NP} and \mathbf{coNP} . Even more general than probabilistic automata are stochastic games. A generalisation of the bisimilarity pseudometric

from labelled Markov chains to stochastic games has been shown to be as hard as the sum-of-square-roots problem [9], a problem not known even to be in NP.

2 Probabilistic Bisimilarity

In this section we introduce labelled Markov chains and probabilistic bisimilarity, and we show that computing probabilistic bisimilarity is P-hard.

A *labelled Markov chain* is a tuple $\mathcal{M} = (S, \Sigma, \pi, \ell)$ consisting of a finite set of *states* S , a finite set of *labels* Σ , a rational *transition matrix* π such that $\sum_{t \in S} \pi_{s,t} = 1$ for all $s \in S$, and a *labelling function* $\ell : S \rightarrow \Sigma$.

A *probabilistic bisimulation* on \mathcal{M} is an equivalence relation $R \subseteq S \times S$ such that if $s R t$ then $\ell(s) = \ell(t)$ and $\sum_{u \in E} \pi_{s,u} = \sum_{u \in E} \pi_{t,u}$ for each R -equivalence class E , i.e., related states have the same label and the same probability to transition into any given equivalence class. It is a standard result that there is a largest probabilistic bisimulation on \mathcal{M} and that this relation is an equivalence relation (see, e.g., [25, Section 7.6]). The maximum probabilistic bisimulation is called *probabilistic bisimilarity* and is denoted \sim . From now on, we mostly refer to probabilistic bisimilarity as simply bisimilarity.

We are interested in the problem of computing bisimilarity \sim on \mathcal{M} . The decision version of the problem asks whether $s \sim t$ for two designated states $s, t \in S$.

The above formulation of the bisimilarity problem is convenient for our hardness proof, however variations, such as replacing state labels with labels on transitions, can easily be accommodated. It is also not difficult to reduce the problem above to the restricted case in which the set of labels has two elements.

For a state s , let $\text{succ}(s) = \{u : \pi_{s,u} > 0\}$. We say that a transition matrix π is *uniform* if for all $s \in S$ and $u, v \in \text{succ}(s)$, $\pi_{s,u} = \pi_{s,v}$. That is, the transition probability out of each state is a uniform distribution over its support.

Lemma 1 (Matching Lemma). *Assume that $|\text{succ}(s)| = |\text{succ}(t)|$ and π is uniform. Then $s \sim t$ if and only if $\ell(s) = \ell(t)$ and there exists a bijection $f : \text{succ}(s) \rightarrow \text{succ}(t)$ with $u \sim f(u)$ for each $u \in \text{succ}(s)$.*

Proof. Suppose that $s \sim t$. Since \sim is a bisimulation, $\ell(s) = \ell(t)$ and for each \sim -equivalence class E ,

$$\sum_{x \in E} \pi_{s,x} = \sum_{x \in E} \pi_{t,x}. \tag{1}$$

Since $|\text{succ}(s)| = |\text{succ}(t)|$ and π is uniform, $|E \cap \text{succ}(s)| = |E \cap \text{succ}(t)|$ for each \sim -equivalence class E . Hence there exists a bijection $f : \text{succ}(s) \rightarrow \text{succ}(t)$ with $u \sim f(u)$ for all $u \in \text{succ}(s)$.

Conversely, assume that $\ell(s) = \ell(t)$ and suppose that f is a bijection as above. To conclude that $s \sim t$ we prove that the smallest equivalence relation containing $\sim \cup \{(s, t)\}$, which we denote by R , is a bisimulation.

Since \sim is a bisimulation and $\ell(s) = \ell(t)$, R only relates states with the same label. Moreover, since every R -equivalence class is a union of \sim -equivalence classes, it suffices to show (1) for \sim -equivalences classes only.

Assume uRv . We distinguish three cases. First, let $u = s$ and $v = t$. Because of the existence of the bijection f , we have that $|E \cap \text{succ}(s)| = |E \cap \text{succ}(t)|$ for each \sim -equivalence class E . Because π is uniform, (II) holds for each \sim -equivalence class E . Second, let $u \sim s$ and $v \sim t$. Recall that \sim is a bisimulation. Hence, for each \sim -equivalence class E ,

$$\sum_{x \in E} \pi_{u,x} = \sum_{x \in E} \pi_{s,x} = \sum_{x \in E} \pi_{t,x} = \sum_{x \in E} \pi_{v,x},$$

where we use $u \sim s$, the previous case, and $v \sim t$. The third and final case, $u \sim v$, follows immediately from the fact that \sim is a bisimulation. \square

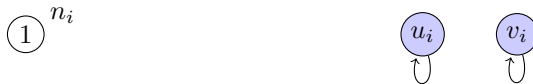
Theorem 2. *Deciding probabilistic bisimilarity is P-hard.*

Proof. We reduce from the MONOTONE CIRCUIT VALUE problem which is P-hard [16, Theorem 6.2.2]. Recall that a monotone circuit is a finite directed acyclic graph in which nodes have in-degree either two or zero. Nodes with in-degree two are labelled \wedge or \vee ; nodes with in-degree zero, called input nodes, are labelled either true (1) or false (0). There is a distinguished output node with out-degree zero. The MONOTONE CIRCUIT VALUE problem is to compute the output of a given monotone circuit.

Given a circuit C , we define a Markov chain $\mathcal{M}(C)$ with a uniform transition matrix. For each node n_i of C and its incoming edges, we include a gadget consisting of states and their outgoing transitions in $\mathcal{M}(C)$. Note that the transitions of the Markov chain go in the opposite direction of the edges of the circuit. Each gadget contains states u_i and v_i . We will prove that $u_i \sim v_i$ if and only if n_i evaluates to true. We define the labelling function ℓ such that states have the same label if and only if they belong to the same gadget and the gadget does not represent an input node that is labelled false. In the diagrams below, states have the same label if and only if they have the same index and the same colour.

We describe $\mathcal{M}(C)$ by giving gadgets for each input node, *and*-gate and *or*-gate of C .

The gadget for input node labelled true is shown below.

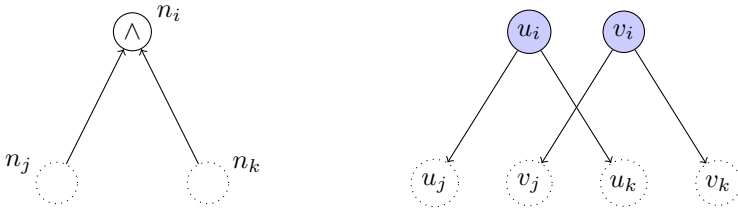


The gadget for input node labelled false is shown below.



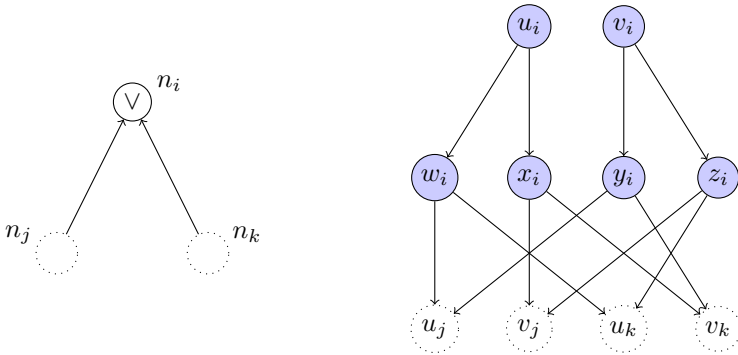
Note that u_i and v_i have the same label if and only if n_i is labelled true and therefore $u_i \sim v_i$ if and only if n_i is labelled true.

The gadget for an *and*-gate is shown below.



Note that u_j, v_j and u_k, v_k are states of the gadgets corresponding to the nodes n_j and n_k . The correctness of this gadget amounts to showing that $u_i \sim v_i$ if and only if both $u_j \sim v_j$ and $u_k \sim v_k$. This follows immediately from the Matching Lemma and the fact that the definition of ℓ precludes $u_j \sim v_k$ and $v_j \sim u_k$ in case n_j and n_k are different nodes. If n_j and n_k are one and the same node, the *and*-gate can be removed from the circuit.

The gadget for an *or*-gate is shown below.



The correctness of this gadget amounts to showing that $u_i \sim v_i$ if and only if $u_j \sim v_j$ or $u_k \sim v_k$.

$$\begin{aligned}
 u_i \sim v_i &\text{ iff } (w_i \sim y_i \wedge x_i \sim z_i) \vee (w_i \sim z_i \wedge x_i \sim y_i) \quad [\text{Matching Lemma}] \\
 &\text{ ff } u_j \sim v_j \vee u_k \sim v_k \quad [\text{Matching Lemma}]
 \end{aligned}$$

In the last step we use again the fact that the definition of ℓ precludes $u_j \sim v_k$ and $v_j \sim u_k$ in case n_j and n_k are different nodes. If n_j and n_k are one and the same node, the *or*-gate can be removed from the circuit.

This completes the description of the gadgets. The Markov chain $\mathcal{M}(C)$ is obtained by composing the gadgets for each node of C . The transduction of a circuit to a Markov chain is done gate by gate. To produce the output gadget corresponding to each circuit gate one only needs to store the indices of the gate and its two inputs, and the states of the output gadget. Thus the reduction can be done in deterministic logarithmic space. \square

The proofs of **P**-hardness of ordinary bisimilarity for labelled transition systems by Balcázar, Gabarró and Sántha [4] and Sawa and Jančar [26] are also by reduction from MONOTONE CIRCUIT VALUE. However in the probabilistic case

disjunction cannot be translated directly as in the non-deterministic case. Interestingly, a formally identical gadget to the above disjunction gadget appears in Toran’s proof of **DET**-hardness of graph isomorphism [29]. However **DET** is a subclass of **P** and the graph isomorphism problem is not known to be **P**-hard.

3 The Bisimilarity Pseudometric

In this section we recall the definition of a bisimilarity pseudometric on labelled Markov chains. We first give a logical characterisation, due to Desharnais, Gupta, Jagadeesan and Panangaden [12], based on a real-valued semantics for Larsen and Skou’s probabilistic modal logic [21]. This characterisation illustrates the sense in which states that are close in the pseudometric satisfy similar behavioural properties. In the next section we give a more abstract fixed-point characterisation of the pseudometric, which will be used in our algorithms.

The logic \mathcal{L} is defined by the grammar

$$\varphi ::= \sigma \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \ominus q \mid \diamond\varphi \tag{2}$$

where $\sigma \in \Sigma$ and $q \in [0, 1]$ is rational.

We consider a real-valued semantics of \mathcal{L} , which is parameterised by a *discount factor* $c \in (0, 1]$. The smaller the value of c , the more the future is discounted, with $c = 1$ being the *undiscounted* case. Given a labelled Markov chain $\mathcal{M} = (S, \Sigma, \pi, \ell)$, the interpretation of a formula φ is a function $\llbracket \varphi \rrbracket : S \rightarrow [0, 1]$ defined by the following clauses:

$$\begin{aligned} \llbracket \sigma \rrbracket(s) &= \begin{cases} 1 & \text{if } \ell(s) = \sigma \\ 0 & \text{otherwise} \end{cases} \\ \llbracket \varphi \vee \psi \rrbracket(s) &= \max(\llbracket \varphi \rrbracket(s), \llbracket \psi \rrbracket(s)) \\ \llbracket \neg\varphi \rrbracket(s) &= 1 - \llbracket \varphi \rrbracket(s) \\ \llbracket \varphi \ominus q \rrbracket(s) &= \max(\llbracket \varphi \rrbracket(s) - q, 0) \\ \llbracket \diamond\varphi \rrbracket(s) &= c \cdot \sum_{t \in S} \pi_{s,t} \cdot \llbracket \varphi \rrbracket(t) \end{aligned}$$

A *pseudometric* is a relaxation of the notion of an ordinary metric in which different states can have distance zero. Formally a (1-bounded) pseudometric on a set S is a map $d : S \times S \rightarrow [0, 1]$ such that for all $s, t, u \in S$, $d(s, s) = 0$, $d(s, t) = d(t, s)$ and $d(s, u) \leq d(s, t) + d(t, u)$.

Given a discount factor $c \in (0, 1]$ the function $d_c : S \times S \rightarrow [0, 1]$ assigns a distance to every pair of states of a labelled Markov chain according to the following definition:

$$d_c(s, t) = \sup_{\varphi \in \mathcal{L}} |\llbracket \varphi \rrbracket(s) - \llbracket \varphi \rrbracket(t)|. \tag{3}$$

It is straightforward that, with this definition, d_c is a pseudometric. The following theorem justifies our description of d_c as a bisimilarity pseudometric.

Theorem 3. [25, Section 8.2] $d_c(s, t) = 0$ if and only if $s \sim t$.

In [9], Chatterjee, de Alfaro, Majumdar and Raman enriched the logic \mathcal{L} by the addition of fixed-point operators, yielding a *quantitative μ -calculus* \mathcal{L}_μ which can express reachability and ω -regular specifications. For example, the \mathcal{L}_μ -formula $\mu x.(\sigma \vee \Diamond x)$ represents the probability to reach a σ -labelled state, while $\nu y. \mu x.((\sigma \wedge y) \vee \Diamond x)$ represents the probability to infinitely often visit a σ -labelled state. It is shown in [9] that $d_c(s, t) = \sup_{\varphi \in \mathcal{L}_\mu} |[\varphi](s) - [\varphi](t)|$ for any pair of states $s, t \in S$; thus d_c can equivalently be defined in terms of the more powerful logic \mathcal{L}_μ .

4 Matchings, Couplings and the Kantorovich Metric

In this section we give a fixed-point characterisation of the probabilistic bisimilarity pseudometric. Based on this we relate the pseudometric to the classical notion of *couplings* on Markov chains.

Say that a probability distribution ω on $S \times S$ is a *matching* of probability distributions μ, ν on S if

$$\begin{aligned} \sum_{v \in S} \omega(u, v) &= \mu(u) && \text{for all } u \in S \\ \sum_{u \in S} \omega(u, v) &= \nu(v) && \text{for all } v \in S. \end{aligned}$$

In other words, ω is a joint probability distribution whose marginals are μ and ν .

Suppose that (S, d) is a finite metric space. The *Kantorovich metric* d_K on the set of probability distributions on S is defined by

$$d_K(\mu, \nu) = \min_{\omega \in \Omega_{\mu, \nu}} \sum_{u, v \in S} d(u, v) \cdot \omega(u, v),$$

where $\Omega_{\mu, \nu}$ is the set of matchings of μ and ν .

Informally we can characterise the bisimilarity pseudometric $d_c(s, t)$ as the distance between the distributions $\pi_{s, -}$ and $\pi_{t, -}$ in the Kantorovich metric over (S, d_c) . This characterisation is recursive, and accordingly we will show that d_c is a fixed point of a functional Δ_c based on the Kantorovich metric.

Define $\Delta_c : [0, 1]^{S \times S} \rightarrow [0, 1]^{S \times S}$ as follows. If $\ell(s) \neq \ell(t)$ then $\Delta_c(d)(s, t) = 1$ and if $\ell(s) = \ell(t)$ then

$$\Delta_c(d)(s, t) = c \cdot \min_{\omega \in \Omega_{s, t}} \sum_{u, v \in S} d(u, v) \cdot \omega(u, v), \tag{4}$$

where $\Omega_{s, t}$ is the set of matchings of $\pi_{s, -}$ and $\pi_{t, -}$.

The set $[0, 1]^{S \times S}$ is a complete lattice in the pointwise order. It is shown in [5, Proposition 38] that Δ_c is a monotone selfmap on $[0, 1]^{S \times S}$ and thus, by Tarski’s fixed point theorem, has a least fixed point. Since the least element of $[0, 1]^{S \times S}$ is a pseudometric, Δ_c maps a pseudometric to a pseudometric, and the least upper-bound of a set of pseudometrics is a pseudometric, we can conclude that the least fixed point of Δ_c is a pseudometric as well. This turns out to be the pseudometric d_c .

Theorem 4. [6, Theorem 4.6] d_c is the least fixed point of Δ_c .

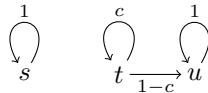
Remark 5. In the relational setting it is traditional to view bisimilarity as a greatest fixed point. Intuitively the situation is opposite in the pseudometric setting because the bottom element of $[0, 1]$ represents relatedness.

Theorem 6. If $c < 1$ then d_c is the unique fixed point of Δ_c .

Proof sketch. We can show that Δ_c is c -Lipschitz. From Banach’s fixed point theorem we can conclude that the fixed point is unique. \square

However, Δ_1 need not have a unique fixed point. For example, consider the labelled Markov chain with a single state. Then Δ_1 is the identity mapping.

Example 7. Consider the Markov chain below, where $\ell(s) = \ell(t) \neq \ell(u)$:



For $c < 1$, $d_c(s, t) = \frac{c-c^2}{1-c^2}$. The pseudometric $\mathbf{0}$ assigns to every pair of states distance zero. For all $n \in \mathbb{N}$, $\Delta_c^n(\mathbf{0})(s, t) \leq \frac{c-c^{2n+1}}{1+c}$. This shows that the fixed point may not be reached by a finite number of iterations of Δ_c .

For each $s, t \in S$, let $\omega_{(s,t),(-,-)}$ be a matching of $\pi_{s,-}$ and $\pi_{t,-}$. Then the Markov chain $\mathcal{C} = (S \times S, \omega)$ is a coupling (see, e.g., [24, Chapter 11] for a discussion of couplings). Such a coupling can be seen as two copies of \mathcal{M} running synchronously, although not necessarily independently. Couplings are typically used to give upper bounds on convergence to stationary distributions. Here we use them to a slightly different end. Given a coupling \mathcal{C} , as above, define the *discrepancy* of a state $(s, t) \in S \times S$, denoted $d_{\mathcal{C}}(s, t)$, to be the probability that a trajectory of \mathcal{C} starting in state (s, t) reaches a state (u, v) with $\ell(u) \neq \ell(v)$.

Formally, given a coupling \mathcal{C} , we define $\Gamma_{\mathcal{C}} : [0, 1]^{S \times S} \rightarrow [0, 1]^{S \times S}$ as follows. If $\ell(s) \neq \ell(t)$ then $\Gamma_{\mathcal{C}}(d)(s, t) = 1$ and if $\ell(s) = \ell(t)$ then

$$\Gamma_{\mathcal{C}}(d)(s, t) = \sum_{u,v \in S} d(u, v) \cdot \omega_{(s,t),(u,v)}.$$

We leave it to the reader to check that $\Gamma_{\mathcal{C}}$ is a monotone selfmap on $[0, 1]^{S \times S}$. By Tarski’s fixed point theorem, $\Gamma_{\mathcal{C}}$ has a least fixed point, which we denote by $d_{\mathcal{C}}$. As we will show next, $d_{\mathcal{C}}$ is closely related to our bisimilarity pseudometric d_1 .

Theorem 8. $d_1 = \min\{d_{\mathcal{C}} : \mathcal{C} \text{ is a coupling}\}$.

As a consequence of the above theorem, the bisimilarity pseudometric d_1 corresponds to the minimal coupling. Next, we will show that two states have discrepancy zero in some coupling if and only if they are bisimilar.

Proposition 9. $d_C(s, t) = 0$ for some coupling \mathcal{C} if and only if $s \sim t$.

Proof. From Theorem 8 we can conclude that $d_C(s, t) = 0$ for some coupling \mathcal{C} if and only if $d_1(s, t) = 0$. By Theorem 3 this gives us the desired result. \square

The following simple lemma shows that the discrepancy can be used to bound the variational distance between trace distributions. This can be seen as a quantitative version of the folklore that bisimilar states satisfy the same linear-time properties. In the lemma we use $\text{Pr}_{\mathcal{M},s}(A)$ to denote the probability that a run of the labelled Markov chain \mathcal{M} started in state s is in the set A . For a formal definition of $\text{Pr}_{\mathcal{M},s}(A)$ and a definition of measurable subset of the set Σ^ω of infinite sequences over Σ , we refer the reader to, e.g., [3, Chapter 10].

Lemma 10 (Coupling Lemma). *Let \mathcal{C} be a coupling of the labelled Markov chain $\mathcal{M} = (S, \Sigma, \pi, \ell)$. Then for any measurable set $A \subseteq \Sigma^\omega$ and $s, t \in S$,*

$$|\text{Pr}_{\mathcal{M},s}(A) - \text{Pr}_{\mathcal{M},t}(A)| \leq d_C(s, t).$$

As a consequence of the Coupling Lemma and Theorem 8, our bisimilarity pseudometric is an upper bound for the variational distance between trace distributions.

Corollary 11. *For any measurable set $A \subseteq \Sigma^\omega$ and $s, t \in S$,*

$$|\text{Pr}_{\mathcal{M},s}(A) - \text{Pr}_{\mathcal{M},t}(A)| \leq d_1(s, t).$$

Whereas the variational distance between trace distributions is NP-hard to compute, as shown by Lyngsø and Pedersen in [22], we will show that our bisimilarity pseudometric can be computed in polynomial time.

5 Algorithms for Bisimilarity Pseudometrics

5.1 The Discounted Case

Let $c < 1$ be a fixed rational discount factor. Given a labelled Markov chain \mathcal{M} , we show that d_c is rational and can be computed exactly in time polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$.¹

In Theorem 4 we have characterised d_c as the least fixed point of Δ_c . While the stipulation that d_c be the least fixed point is essential in the undiscounted case, it is redundant in the discounted case. In the latter case, Δ_c has a unique fixed point (see Theorem 6). As a consequence, d_c is also the greatest fixed point of Δ_c for $c < 1$. Thus, by Tarski’s fixed point theorem, we have

$$d_c = \bigsqcup \{ d \in \mathbb{R}^{S \times S} : d \leq \Delta_c(d) \wedge 0 \leq d \leq 1 \}. \tag{5}$$

¹ We denote by $\text{size}(X)$ the size of the representation of an object X . We represent rational numbers as quotients of integers written in binary. For example, the size of a rational number is the sum of the bit lengths of its numerator and denominator and the size of a matrix is the sum of the sizes of its entries.

This simple change in perspective is fruitful because the characterisation (5) directly yields a translation of the problem of computing d_c to the following linear program:

$$\begin{aligned}
 &\text{maximise } \sum_{s,t \in S} d_{s,t} \\
 &\text{such that } d_{s,t} \leq c \cdot \sum_{u,v \in S} d_{u,v} \cdot \omega(u,v) & \omega \in \Omega_{s,t}, \ell(s) \neq \ell(t) \\
 & \quad d_{s,t} = 1 & \ell(s) = \ell(t) \\
 & \quad 0 \leq d_{s,t} \leq 1
 \end{aligned} \tag{6}$$

As we will see, the linear program (6) can be solved in polynomial time using the ellipsoid algorithm. We pursue this option in the undiscounted setting below. However, here we do not require such powerful techniques. Instead we just use the linear programming formulation to observe that the fixed point of Δ_c is rational and bounded in size by a polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$. We then approximate the fixed point by repeating the network simplex algorithm, obtaining the exact solution by rounding by means of the continued fraction algorithm.

Recall that the set of matchings $\Omega_{s,t}$ is a polytope in $\mathbb{R}^{S \times S}$ defined by the following constraints:

$$\sum_{v \in S} \omega(u,v) = \pi_{s,u} \text{ and } \sum_{u \in S} \omega(u,v) = \pi_{t,v} \text{ and } \omega(u,v) \geq 0 \tag{7}$$

In general, $\Omega_{s,t}$ is infinite and therefore the set of constraints in (6) is infinite also. However, for each fixed d the linear function mapping a matching ω to $c \cdot \sum_{u,v} d(u,v) \cdot \omega(u,v)$ achieves its minimum on $\Omega_{s,t}$ at some vertex. Thus, writing $V(\Omega_{s,t})$ for the (finite) set of vertices of $\Omega_{s,t}$, we can replace $\Omega_{s,t}$ with $V(\Omega_{s,t})$ in (6), obtaining a linear program with the same feasible region. We denote the polytope defined by the set of constraints of this linear program by D . To prove that the distances are rational, we first observe the following.

Proposition 12. *Each $\omega \in V(\Omega_{s,t})$ is rational of size polynomial in $\text{size}(\mathcal{M})$.*

Proof sketch. Since a vertex of $\Omega_{s,t}$ is by definition an intersection of hyperplanes given by the (in)equalities defining $\Omega_{s,t}$ and the coefficients of the (in)equalities are rationals bounded in size by $\text{size}(\mathcal{M})$, we can conclude that each $\omega \in V(\Omega_{s,t})$ is rational of size polynomial in $\text{size}(\mathcal{M})$. □

Proposition 13. *d_c is rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$.*

Proof sketch. Along a similar line of reasoning as used in the proof of Proposition 12, we can conclude that the vertices of polytope D are rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$.

Since the function mapping any d of the polytope D to $\sum_{s,t \in S} d_{s,t}$ is linear, it attains its maximum, d_c , at some vertex of D , which, as we have just shown, is rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$. □

Note that the proofs of Proposition 12 and 13 are also valid for $c = 1$ and, hence, d_1 is rational as well. Having established that d_c is rational, we now give a simple iterative algorithm to approximate d_c starting from the pseudometric $\mathbf{0}$.

Proposition 14. *For all $n \in \mathbb{N}$, $\Delta_c^n(\mathbf{0})$ is rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$ and can be computed in time polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$.*

Proof sketch. We prove this property by induction on n . Obviously, the property holds for $n = 0$. Let $n > 0$. Obviously, the property holds when $\ell(s) \neq \ell(t)$. Otherwise, $\Delta_c^n(\mathbf{0})(s, t) = c \cdot \min_{\omega \in \Omega_{s,t}} \sum_{u,v \in S} \Delta_c^{n-1}(\mathbf{0})(u, v) \cdot \omega(u, v)$. The above minimum is attained at a vertex of $\Omega_{s,t}$. As we have seen in Proposition 12, these vertices are rationals of size polynomial in $\text{size}(\mathcal{M})$. Furthermore, by induction, $\Delta_c^{n-1}(\mathbf{0})$ is rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$. Hence, $\Delta_c^n(\mathbf{0})(s, t)$ is a rational of size polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$. Computing $\Delta_c(d)(s, t)$ is a minimum-cost flow problem for which there are versions of the network simplex algorithm that are strongly polynomial time 11. □

To get ϵ -close to d_c , we need to iterate $\lceil \log_c(\epsilon) \rceil$ times.

Proposition 15. *For all $\epsilon > 0$, $\|\Delta_c^{\lceil \log_c(\epsilon) \rceil}(\mathbf{0}) - d_c\| \leq \epsilon$.*

From Proposition 14 and 15 we can conclude that we can approximate d_c in time polynomial in $\text{size}(\mathcal{M})$, $\text{size}(c)$ and $\log_2(\frac{1}{\epsilon})$. Once we have iterated close enough to d_c , we can use the continued fraction algorithm (see, e.g. 17, Section 5.1) to obtain d_c .

Theorem 16. *The pseudometric d_c can be computed in time polynomial in $\text{size}(\mathcal{M})$ and $\text{size}(c)$.*

Proof sketch. This follows now immediately from the observation made by Etesami and Yannakakis 13, page 2540] that for problems whose solutions are rational, of size polynomial in the input size, if we can solve the approximation problem in polynomial time, then we can also solve the exact computation problem in polynomial time by using the continued fraction algorithm. □

5.2 The Undiscounted Case

Throughout this section we refer to the undiscounted bisimilarity pseudometric as d , rather than d_1 and likewise use Δ instead of Δ_1 .

In the previous section we gave a reduction of the problem of computing d_c to linear programming by characterising d_c as the greatest fixed point of Δ_c for $c < 1$. However, recall from Section 4 that d is not in general the greatest fixed point of Δ . Nevertheless we can recover a greatest-fixed-point characterisation of d by separately handling the set of bisimilar states, i.e. the states at distance zero. This will allow us to use linear programming to compute d .

As a first step we define $\Delta' : [0, 1]^{S \times S} \rightarrow [0, 1]^{S \times S}$ by

$$\Delta'(d)(s, t) = \begin{cases} \Delta(d)(s, t) & \text{if } s \not\sim t \\ 0 & \text{if } s \sim t. \end{cases}$$

Proposition 17. *Δ' has a unique fixed point.*

Proof sketch. Since Δ is monotone, we can easily deduce that Δ' is monotone as well. According to Tarski's fixed point theorem, Δ' has a least and a greatest fixed point. Hence, it is sufficient to prove that if $d \leq d'$ are both fixed points of Δ' then $d = d'$.

To this end, let $m = \max\{d'(s, t) - d(s, t) : s, t \in S\}$ and let M be the set of pairs $\{(s, t) \in S \times S : d'(s, t) - d(s, t) = m\}$ which maximise the discrepancy between d' and d . We will show that $m = 0$, which implies that $d = d'$. We distinguish two cases.

Assume that $(s, t) \in M$ such that $\ell(s) \neq \ell(t)$. Then

$$d'(s, t) - d(s, t) = \Delta'(d')(s, t) - \Delta'(d)(s, t) = 1 - 1 = 0$$

and, hence, $m = 0$.

Otherwise, for all $(s, t) \in M$ we have that $\ell(s) = \ell(t)$. In this case, we claim that $M \subseteq \sim$. From this claim it follows that for all $(s, t) \in M$,

$$d'(s, t) - d(s, t) = \Delta'(d')(s, t) - \Delta'(d)(s, t) = 0 - 0 = 0$$

and, hence, $m = 0$. It just remains to prove the claim.

By Proposition 9 it suffices to define a coupling \mathcal{C} such that $d_{\mathcal{C}}(s, t) = 0$ for all $(s, t) \in M$. To define \mathcal{C} we must specify a matching $\omega \in \Omega_{s,t}$ for each pair of states $(s, t) \in S \times S$. For $(s, t) \notin M$ any matching will do. For $(s, t) \in M$ we show that we can choose a matching $\omega \in \Omega_{s,t}$ whose support is contained in M . Then in the coupling \mathcal{C} no pair in $(s, t) \in M$ can reach a pair (u, v) with $\ell(u) \neq \ell(v)$, that is, $d_{\mathcal{C}}(s, t) = 0$.

Let $(s, t) \in M$. Suppose $\Delta'(d)(s, t) = \sum_{u,v \in S} d(u, v) \cdot \omega(u, v)$, where $\omega \in \Omega_{s,t}$. Then

$$\begin{aligned} m &= d'(s, t) - d(s, t) \\ &= \Delta'(d')(s, t) - \Delta'(d)(s, t) \\ &= \left(\min_{\omega' \in \Omega_{s,t}} \sum_{u,v \in S} d'(u, v) \cdot \omega'(u, v) \right) - \sum_{u,v \in S} d(u, v) \cdot \omega(u, v) \\ &\leq \sum_{u,v \in S} d'(u, v) \cdot \omega(u, v) - \sum_{u,v \in S} d(u, v) \cdot \omega(u, v) \\ &= \sum_{u,v \in S} (d'(u, v) - d(u, v)) \cdot \omega(u, v). \end{aligned}$$

Since $d'(u, v) - d(u, v) \leq m$ and $\sum_{u,v \in S} \omega(u, v) = 1$, we can conclude from $\sum_{u,v \in S} (d'(u, v) - d(u, v)) \cdot \omega(u, v) \geq m$ that $d'(u, v) - d(u, v) = m$ whenever $\omega(u, v) > 0$. □

Corollary 18. *d is the unique fixed point of Δ' .*

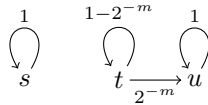
Proof. It is enough to prove that \mathbf{d} is a fixed point of Δ' . On the one hand, suppose that $s \sim t$. Then $\mathbf{d}(s, t) = 0 = \Delta'(d)(s, t)$ by Theorem 3. On the other hand, suppose that $s \not\sim t$. Then $\mathbf{d}(s, t) = \Delta(\mathbf{d})(s, t) = \Delta'(\mathbf{d})(s, t)$. □

Corollary 18 implies that d is the greatest fixed point of Δ' . Thus, following the development in Section 5.1, we can compute d as the solution to the following linear program:

$$\begin{aligned}
 &\text{maximise } \sum_{s,t \in S} d_{s,t} \\
 &\text{such that } d_{s,t} = 0 && s \sim t \\
 & && \ell(s) \neq \ell(t) \\
 & d_{s,t} = 1 && \omega \in V(\Omega_{s,t}), s \not\sim t, \ell(s) = \ell(t) \\
 & d_{s,t} \leq \sum_{u,v \in S} d_{u,v} \cdot \omega(u,v)
 \end{aligned} \tag{8}$$

Unfortunately we cannot solve (8) using the iterative method adopted in the discounted case. The reason is that it may require exponentially many iterations of Δ' to achieve a sufficiently close approximation to d .

Example 19. Consider the Markov chain below, where $\ell(s) = \ell(t) \neq \ell(u)$:



Then $d(s, t) = 1$ and $(\Delta')^n(\mathbf{0})(s, t) \leq n \cdot 2^{-m}$ for all $n \in \mathbb{N}$. This shows that it may require exponentially many iterations in $\text{size}(\mathcal{M})$ to approximate the fixed point of Δ' .

Instead we use the ellipsoid algorithm (see, e.g. [27, Chapter 14]) to solve the linear program (8). According to Proposition 12 (which also holds for $c = 1$), the coefficients of the constraints of the linear program (8) are rational of size polynomial in $\text{size}(\mathcal{M})$. By, e.g. [27, Corollary 14.1a], to conclude that the linear program (8) can be solved in time polynomial in $\text{size}(\mathcal{M})$, it suffices to show that there exists a polynomial time separation algorithm. In our setting, given a $d \in \mathbb{R}^{S \times S}$ rational of size polynomial in $\text{size}(\mathcal{M})$, the separation algorithm has to decide whether d satisfies the constraints of (8) or not, and, in the latter case, find in time polynomial in $\text{size}(\mathcal{M})$ a separating hyperplane, i.e., an $\alpha \in \mathbb{Q}^{S \times S}$ such that

$$\sum_{u,v \in S} d(u,v) \cdot \alpha(u,v) < \sum_{u,v \in S} d'(u,v) \cdot \alpha(u,v) \tag{9}$$

for all $d' \in \mathbb{R}^{S \times S}$ that satisfy the constraints of (8).

Let $d \in \mathbb{R}^{S \times S}$ be rational of size polynomial in $\text{size}(\mathcal{M})$. For each pair of states $s, t \in S$, we consider the following linear program:

$$\begin{aligned}
 &\text{minimise } \sum_{u,v \in S} d(u,v) \cdot \omega_{u,v} \\
 &\text{such that } \sum_{v \in S} \omega_{u,v} = \pi_{s,u} \text{ and } \sum_{u \in S} \omega_{u,v} = \pi_{t,v} \text{ and } \omega_{u,v} \geq 0
 \end{aligned} \tag{10}$$

This linear program is a minimum-cost flow problem for which there are versions of the network simplex algorithm that can compute an $(\omega_{u,v})_{u,v \in S}$, which satisfies the constraints of (10) and minimizes the objective function, and that are strongly polynomial time [1].

Note that d satisfies the constraints of (10) if and only if $d(s, t)$ is smaller than or equal to the optimal value of (10) for each pair of states $s, t \in S$. Otherwise, there exists a pair of states $s, t \in S$ such that $d(s, t)$ is greater than the optimal

value of (10). Let $\omega \in V(\Omega_{s,t})$ be a vertex that realizes the optimal value of (10). As we have seen in Proposition 12, ω is rational of size polynomial in $\text{size}(\mathcal{M})$.

It remains to define an α that satisfies (9). We define α in terms of ω as follows:

$$\alpha(u, v) = \begin{cases} \omega(u, v) - 1 & \text{if } (u, v) = (s, t) \\ \omega(u, v) & \text{otherwise.} \end{cases}$$

Proposition 20. *Assume that d does not satisfy the constraints of (10). Then for all $d' \in \mathbb{R}^{S \times S}$ that satisfy the constraints of (10), we have (9).*

Proof. Since d does not satisfy the constraints of (10), there exists a pair of states $s, t \in S$ such that $d(s, t) > \sum_{u,v \in S} d(u, v) \cdot \omega(u, v)$. Hence,

$$\sum_{u,v \in S} d(u, v) \cdot \alpha(u, v) < 0. \quad (11)$$

Assume that $d' \in \mathbb{R}^{S \times S}$ satisfies the constraints of (10). Then we have that $d'(s, t) \leq \sum_{u,v \in S} d'(u, v) \cdot \omega(u, v)$. Hence,

$$\sum_{u,v \in S} d'(u, v) \cdot \alpha(u, v) \geq 0. \quad (12)$$

From (11) and (12) we can immediately conclude (9). \square

6 Conclusion

The linear program (6) shows *inter alia* that the problem of computing probabilistic bisimilarity can naturally be reduced to linear programming. It would be interesting to relate the resulting procedure for computing probabilistic bisimilarity to the classical partition refinement algorithm.

The problem of computing probabilistic bisimilarity bears some similarity to the graph isomorphism problem. While the latter is not known to be in **P**, for certain restricted graph classes, such as graphs of bounded degree or with bounded colour classes, it is in **DET** (a subclass of **P**). By contrast, deciding probabilistic bisimilarity is **P**-hard already for labelled Markov chains with branching degree at most two, in which at most four states share the same label.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows – theory, algorithms and applications (1993)
2. Baier, C.: Polynomial Time Algorithms for Testing Probabilistic Bisimulation and Simulation. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 50–61. Springer, Heidelberg (1996)
3. Baier, C., Katoen, J.-P.: Principles of Model Checking (2008)
4. Balcázar, J.L., Gabarró, J., Sántha, M.: Deciding bisimilarity is P-complete. FAC 4(6A), 638–648 (1992)
5. van Breugel, F., Hermida, C., Makkai, M., Worrell, J.: Recursively defined metric spaces without contraction. TCS 380(1-2), 171–197 (2007)

6. van Breugel, F., Sharma, B., Worrell, J.: Approximating a behavioural pseudometric without discount for probabilistic systems. *LMCS* 4(2:2) (2008)
7. van Breugel, F., Worrell, J.: Approximating and computing behavioural distances in probabilistic transition systems. *TCS* 360(1-3), 373–385 (2006)
8. Cai, X., Gu, Y.: Measuring Anonymity. In: Bao, F., Li, H., Wang, G. (eds.) *ISPEC 2009*. LNCS, vol. 5451, pp. 183–194. Springer, Heidelberg (2009)
9. Chatterjee, K., de Alfaro, L., Majumdar, R., Raman, V.: Algorithms for game metrics. In: *FSTTCS*, pp. 107–118 (2008)
10. Comanici, G., Precup, D.: Basis function discovery using spectral clustering and bisimulation metrics. In: *AAAI*, pp. 325–330 (2011)
11. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. *IPL* 87(6), 309–315 (2003)
12. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled Markov processes. *TCS* 318(3), 323–354 (2004)
13. Etessami, K., Yannakakis, M.: On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing* 39(6), 2531–2597 (2010)
14. Fu, H.: The complexity of deciding a behavioural pseudometric on probabilistic automata. Technical Report AIB-2011-26, RWTH Aachen (2011)
15. Giacalone, A., Jou, C.-C., Smolka, S.A.: Algebraic reasoning for probabilistic concurrent systems. In: *PROCOMET*, pp. 443–458 (1990)
16. Greenlaw, R., James Hoover, H., Ruzzo, W.L.: *Limits to Parallel Computation: P-Completeness Theory* (1995)
17. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization* (1988)
18. Hillston, J.: *A Compositional Approach to Performance Modelling* (1996)
19. Katoen, J.-P., Kemna, T., Zapreev, I.S., Jansen, D.N.: Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking. In: Grumberg, O., Huth, M. (eds.) *TACAS 2007*. LNCS, vol. 4424, pp. 87–101. Springer, Heidelberg (2007)
20. Kemeny, J.G., Laurie Snell, J.: *Finite Markov Chains* (1960)
21. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *I&C* 94(1), 1–28 (1991)
22. Lyngsø, R.B., Pedersen, C.N.S.: The consensus string problem and the complexity of comparing hidden Markov models. *JCSS* 65(3), 545–569 (2002)
23. Milner, R.: *Communication and Concurrency* (1989)
24. Mitzenmacher, M., Upfal, E.: *Probability and Computing* (2005)
25. Panangaden, P.: *Labelled Markov Processes* (2009)
26. Sawa, Z., Jančar, P.: Behavioural equivalences on finite-state systems are PTIME-hard. *Computers and Artificial Intelligence* 24(5), 513–528 (2005)
27. Schrijver, A.: *Theory of Linear and Integer Programming* (1986)
28. Thorsley, D., Klavins, E.: Approximating stochastic biochemical processes with Wasserstein pseudometrics. *IET Systems Biology* 4(3), 193–211 (2010)
29. Torán, J.: On the hardness of graph isomorphism. *SIAM Journal on Computing* 33(5), 1093–1108 (2004)
30. Tzeng, W.-G.: A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing* 21(2), 216–227 (1992)
31. Tzeng, W.-G.: On path equivalence of nondeterministic finite automata. *IPL* 58(1), 43–46 (1996)
32. Valmari, A., Franceschinis, G.: Simple $O(m \log n)$ Time Markov Chain Lumping. In: Esparza, J., Majumdar, R. (eds.) *TACAS 2010*. LNCS, vol. 6015, pp. 38–52. Springer, Heidelberg (2010)

Probabilistic Transition System Specification: Congruence and Full Abstraction of Bisimulation[★]

Pedro Rubén D'Argenio and Matias David Lee

FaMAF, Universidad Nacional de Córdoba – CONICET
{dargenio,lee}@famaf.unc.edu.ar

Abstract. We present a format for the specification of probabilistic transition systems that guarantees that bisimulation equivalence is a congruence for any operator defined in this format. In this sense, the format is somehow comparable to the *ntyft/ntyxt* format in a non-probabilistic setting. We also study the modular construction of probabilistic transition systems specifications and prove that some standard conservative extension theorems also hold in our setting. Finally, we show that the trace congruence for image-finite processes induced by our format is precisely bisimulation on probabilistic systems.

1 Introduction

Plotkin's approach to operational semantics [21] is the standard way to give semantics to specification and programming language in terms of transition systems. It has been formalized with an algebraic flavor as *Transition Systems Specifications (TSS)* [8, 9, 12, 13, 20, etc.]. Basically, a TSS contains a signature, a set of actions or labels, and a set of rules. The signature defines the terms in the language. The set of actions represents all possible activities that a process (i.e., a term over the signature) can perform. The rules define how a process should behave (i.e., perform certain activities) in terms of the behavior of its subprocesses, that is, the rules define compositionally the transition system associated to each term of the language. A particular focus of these formalizations was to provide a meta-theory that ensures a diversity of semantic properties by simple inspection on the form of the rules. Thus, there are results on congruences and full abstraction, conservative extension, security, etc. (see, e.g., [1, 2, 20] for overviews).

In this paper we focus on congruence and full abstraction. A congruence theorem guarantees that whenever the rules of a TSS are in a particular format, then a designated equivalence relation is preserved by every context in the signature of such TSS. Thus, for instance, strong bisimulation equivalence [19] is a congruence on any TSS in the *ntyft/ntyxt* format [12]. Full abstraction is somewhat a dual result: an equivalence relation is fully abstract with respect to a particular format if it is the largest relation s.t. no context definable in the format can exhibit different behavior when applied to two equivalent processes. For example, strong bisimilarity is fully abstract w.r.t. the *ntyft/ntyxt* format [12] but not w.r.t. the *tyft/tyxt* format [13] or the GSOS format [8].

The introduction of probabilistic process algebras [4, 14, 25, etc.] motivated the need for a theory of structural operational semantics to define *probabilistic* transition systems. A few results have appeared in this direction [6, 7, 16, 17] and, to our knowledge,

[★] Partially supported by Project ANPCYT PAE-PICT 02272 and SeCyT-UNC.

only these works present congruence theorems for (probabilistic) bisimilarity [18], but no full abstraction result. All previously mentioned studies consider transitions in the form of a quadruple denoted by $t \xrightarrow{a,q} t'$, where t and t' are terms in the language, a is an action or label, and $q \in (0, 1]$ is a probability value. A transition of that form denotes that term t can perform an action a and with probability q continue with the execution of t' . Moreover, it is required that $\pi_{t,a}$, defined by $\pi_{t,a}(t') = \sum_{t \xrightarrow{a,q} t'} q$, is a probability distribution. (This interpretation corresponds to the reactive view, it varies under the generative view [25].) This notation introduces several problems. The first one is that the transition relation cannot be treated as a set because two different derivations may yield the same quadruple. This requires artifacts like multisets or bookkeeping indexes. The second one is that formats need to be defined jointly on a set of rules rather than a single rule to ensure that $\pi_{t,a}$ is a probability distribution. (Notice that $\pi_{t,a}$ depends on a set of transitions which are obtained using different rules.)

Rather than following this approach, we directly represent transitions as a triple $t \xrightarrow{a} \pi_{t,a}$. Thus, a single triple contains the complete information of the probabilistic jump. Moreover, this representation also allows for non-determinism in the sense that if $t \xrightarrow{a} \pi$ and $t \xrightarrow{a} \pi'$ not necessarily $\pi = \pi'$ as requested by reactive systems. Hence, our *probabilistic transition system specifications (PTSS)* define objects very much like Segala's probabilistic automata [22]. So, each probabilistic transition $t \xrightarrow{a} \pi$ is obtained by a single derivation in our PTSSs, and hence formats focus on single rules (as it is the case for non-probabilistic TSSs). This significantly eases the inspection of the format. In addition, a byproduct of this choice is that the proof strategies for the majority of the lemmas and theorems of this paper are much the same as those for their non-probabilistic relatives. We observe that this way of representing transitions in rules for process algebra has already appeared in [5], it is also used in the Segala-GSOS format [7] and it is pretty much related to bialgebraic approaches to SOS [7, 15].

In this paper we introduce PTSS with negative and quantitative premises which also allow for lookahead. We use stratification [9, 12] as means to define probabilistic transition systems and prove the existence and uniqueness of models for stratifiable PTSSs (Sec. 3). We also propose a format, which we call *nt μ fv/nt μ xv*, that is very much like the *ntyft/ntyxt* format in non-probabilistic TSS and show that bisimilarity is a congruence for any operation defined under this format (Sec. 4). Besides, we give a definition for the modular construction of PTSSs and give sufficient conditions to ensure that one PTSS conservatively extends another (Sec. 5). We finally show that bisimilarity is fully abstract with respect to the *nt μ fv/nt μ xv* format, that is, it is the coarsest congruence w.r.t. any operator defined in *nt μ fv/nt μ xv* PTSSs that is included in trace equivalence (Sec. 6).

2 Preliminaries

We assume the presence of an infinite set of (term) variables \mathcal{V} and we let $x, y, z, x', x_0, x_1, \dots$ range over \mathcal{V} . A *signature* is a structure $\Sigma = (F, r)$, where (i) F is a set of *function names* disjoint with \mathcal{V} , and (ii) $r : F \rightarrow \mathbb{N}_0$ is a *rank function* which gives the arity of a function name; if $f \in F$ and $r(f) = 0$ then f is called a *constant name*. Let $W \subseteq \mathcal{V}$ be a set of variables. The set of Σ -terms over W , notation $T(\Sigma, W)$ is the

least set satisfying: (i) $W \subseteq T(\mathcal{L}, W)$, and (ii) if $f \in F$ and $t_1, \dots, t_{r(f)} \in T(\mathcal{L}, W)$, then $f(t_1, \dots, t_{r(f)}) \in T(\mathcal{L}, W)$. $T(\mathcal{L}, \emptyset)$ is abbreviated as $T(\mathcal{L})$; the elements of $T(\mathcal{L})$ are called *closed terms*. $T(\mathcal{L}, \mathcal{V})$ is abbreviated as $\mathbb{T}(\mathcal{L})$; the elements of $\mathbb{T}(\mathcal{L})$ are called *open terms*. $\text{Var}(t) \subseteq \mathcal{V}$ is the set of variables in the open term t .

Since our aim is to deal with languages that describe probabilistic behavior, apart from signatures, variables, and terms, we also need to introduce probability distributions on terms and variables to run on these distributions. Let $\Delta(T(\mathcal{L}))$ denote the set of all (discrete) probability distributions on $T(\mathcal{L})$. We let $\pi, \pi', \pi_0, \pi_1, \dots$ range over $\Delta(T(\mathcal{L}))$. As usual, for $\pi \in \Delta(T(\mathcal{L}))$ and $T \subseteq T(\mathcal{L})$, we define $\pi(T) = \sum_{t \in T} \pi(t)$. For $t \in T(\mathcal{L})$, let δ_t denote the Dirac distribution, that is, $\delta_t(t') = 1$ if $(t=t')$ then 1 else 0. Moreover, the product measure $\prod_{i=1}^n \pi_i$ is defined by $(\prod_{i=1}^n \pi_i)(t_1, \dots, t_n) = \prod_{i=1}^n \pi_i(t_i)$. In particular, if $n = 0$, $(\prod_{j \in \emptyset} \pi_j) = \delta_\emptyset$ is the distribution that assigns probability 1 to the 0-ary tuple. Let $g : T(\mathcal{L})^n \rightarrow T(\mathcal{L})$ and recall that $g^{-1}(t') = \{\vec{t} \in T(\mathcal{L})^n \mid g(\vec{t}) = t'\}$. Then $(\prod_{i=1}^n \pi_i) \circ g^{-1}$ is a well defined probability distribution on closed terms. In particular, if $g : T(\mathcal{L})^0 \rightarrow T(\mathcal{L})$ and $g(\emptyset) = t$, then $(\prod_{j \in \emptyset} \pi_j) \circ g^{-1} = \delta_\emptyset \circ g^{-1} = \delta_t$.

A *distribution variable* is a variable that takes values on $\Delta(T(\mathcal{L}))$. Let \mathcal{M} be an infinite set of distribution variables and let $\mu, \mu', \mu_0, \mu_1, \dots$ range over \mathcal{M} . For a term variable $x \in \mathcal{V}$ we let δ_x be an *instantiable Dirac distribution*. That is, δ_x is a symbol that takes value δ_t whenever variable x takes value t . Let $\mathcal{D} = \{\delta_x : x \in \mathcal{V}\}$ be the set of instantiable Dirac distributions according to the variable set \mathcal{V} .

A substitution is a mapping that assigns terms to variables. In our case we need to extend this notion to probabilistic variables and instantiable Dirac distributions. A (*closed*) *substitution* ρ is a mapping in $(\mathcal{V} \cup \mathcal{M}) \rightarrow (T(\mathcal{L}) \cup \Delta(T(\mathcal{L})))$ such that $\rho(x) \in T(\mathcal{L})$ whenever $x \in \mathcal{V}$, and $\rho(\mu) \in \Delta(T(\mathcal{L}))$ whenever $\mu \in \mathcal{M}$. A substitution ρ extends to open terms and sets as usual and to instantiable Dirac distributions by $\rho(\delta_x) = \delta_{\rho(x)}$.

Example 1. We introduce the signature of a probabilistic process algebra that includes many of the most representative operators. We assume the existence of a set \mathcal{L} of action labels. Then, our signature (which is the base of our running example) contains: two constants, $\mathbf{0}$ (stop process) and ε (skip process); a family of n -ary probabilistic prefix operators $a.([p_1]_-\oplus \dots \oplus [p_n]_-)$ with $a \in \mathcal{L}, n \geq 1, p_1, \dots, p_n \in (0, 1]$ s.t. $\sum_{i=1}^n p_i = 1$ (we usually write $a. \sum_{i=1}^n [p_i]_t_i$ for given terms t_1, \dots, t_n); binary operators $_- + _-$ (alternative composition or sum), $_- ; _-$ (sequential composition), and, for each $B \subseteq \mathcal{L}$, $_-\parallel_B _-$ (parallel composition); and a unary operator $\mathbf{U}(_-)$ that we call *unreach*. The intended meaning of $a. \sum_{i=1}^n [p_i]_t_i$ is that this term can perform action a and move to term t_i with probability p_i . The *unreach* operation $\mathbf{U}(t)$ can perform an action a and stop if there is a probabilistic execution (or *scheduler*) from t in which action a is never performed (or properly speaking, it is not performed with probability 1). Finally, $t \parallel_B t'$ is a CSP-like parallel composition where actions in B are forced to synchronize and all other actions should be performed independently. The rest of the operators have the usual meaning. \square

3 Probabilistic Transition System Specifications

A (probabilistic) transition relation prescribes what possible activity can be performed by a term in a signature. Such activity is described by the label of the action and a

probability distribution on terms that indicates the probability to reach a particular new term. We will follow the probabilistic automata style of probabilistic transitions [22] which are a generalization of the so called reactive model [18]. So, let Σ be a signature and A be a set of labels. A *transition relation* is a set $\rightarrow \subseteq PTr(\Sigma, A)$, where $PTr(\Sigma, A) = T(\Sigma) \times A \times \Delta(T(\Sigma))$. We denote $(t, a, \pi) \in \rightarrow$ by $t \xrightarrow{a} \pi$.

Transition relations are usually defined by means of structured operational semantics in Plotkin's style [21]. Algebraic characterizations of this style were provided in [9, 12, 13] where the term *transition system specification* was used and which we adopt in our paper. In fact, based on these works, we define *probabilistic transition system specifications*.

Definition 2. A probabilistic transition system specification (PTSS) is a triple $P = (\Sigma, A, R)$ where $\Sigma = (F, r)$ is a signature, A is a set of labels, and R is a set of rules of the form:

$$\frac{\{t_k \xrightarrow{a_k} \mu_k : k \in K\} \cup \{t_l \xrightarrow{b_l} : l \in L\} \cup \{\mu_j(W_j) \geq_j q_j : j \in J\}}{t \xrightarrow{a} \sum_{i \in I} p_i (\prod_{n_i \in N_i} \nu_{n_i}) \circ g_i^{-1}}$$

where K, L, J are index sets, I is a denumerable index set, each N_i is a finite index set, $t, t_k, t_l \in T(\Sigma)$, $a, a_k, b_l \in A$, $\mu_k, \mu_j \in \mathcal{M}$, $W_j \subseteq \mathcal{V}$, $\geq_j \in \{>, \geq, <, \leq\}$, $p_i, q_j \in [0, 1]$ with $\sum_{i \in I} p_i = 1$, each g_i is a function s.t. $g_i : T(\Sigma)^{N_i} \rightarrow T(\Sigma)$, and $\nu_{n_i} \in \mathcal{M} \cup \mathcal{D}$.

An expression of the form $t \xrightarrow{a} \pi$, $(t \xrightarrow{a} \pi, \pi(T) \geq p)$ is a *positive literal* (*negative literal*, *quantitative literal*, resp.). For any rule $r \in R$, literals above the line are called *premises*, notation $\text{prem}(r)$; the literal below the line is called *conclusion*, notation $\text{conc}(r)$. We denote with $\text{pprem}(r)$ ($\text{nprem}(r)$, $\text{qprem}(r)$) the set of positive (negative, quantitative, resp.) literals of the rule r . A rule r is called *positive* if there are no negative premises, i.e., $\text{nprem}(r) = \emptyset$. A PTSS is called *positive* if it has only positive rules. A rule r without premises is called *axiom*. In general, we allow that the sets of positive, negative, and quantitative premises are infinite.

Substitutions provide instances to the rules of a PTSS that, together with some appropriate machinery, allow us to define probabilistic transition relations. Given a substitution ρ , it extends to literals as follows: $\rho(t \xrightarrow{a} \mu) = \rho(t) \xrightarrow{a} \rho(\mu)$, $\rho(t \xrightarrow{a} \pi) = \rho(t) \xrightarrow{a} \pi$, $\rho(\mu(W) \geq p) = \rho(\mu)(\rho(W)) \geq p$, and $\rho(t \xrightarrow{a} \sum_{i \in I} p_i (\prod_{n_i \in N_i} \nu_{n_i}) \circ g_i^{-1}) = \rho(t) \xrightarrow{a} \sum_{i \in I} p_i (\prod_{n_i \in N_i} \rho(\nu_{n_i})) \circ g_i^{-1}$. Then, the notion of substitution extends to rules as expected. We say that r' is a (closed) instance of a rule r if there is a (closed) substitution ρ so that $r' = \rho(r)$. We say that ρ is a *proper substitution of r* if for all quantitative premise $\rho(\mu(W)) \geq p$ of r it holds that $\rho(\mu(w)) > 0$ for all $w \in W$. Thus, if ρ is proper, all terms in $\rho(W)$ are in the support set of $\rho(\mu)$. Proper substitutions avoid the introduction of spurious terms. This is of particular importance for the conservative extension theorem (Theorem 14).

Example 3. The rules for the process algebra of Example 1 are defined in Table 1. We consider the set of actions $A = \mathcal{L} \cup \{\sqrt{\cdot}\}$ where $\sqrt{\cdot} \notin \mathcal{L}$. In the table we use the following shorthand notations for the target of the conclusion which we also adopt along the paper. We omit the summation if I is a singleton and, if $g(\cdot) = t$, we write

Table 1. Rules for our probabilistic process algebra ($Y \subseteq \mathcal{V}$ is a countably infinite set)

$$\begin{array}{c}
 \varepsilon \xrightarrow{\checkmark} \delta_0 \quad a. \sum_{i=1}^n [p_i]x_i \xrightarrow{a} \sum_{i=1}^n p_i \delta_{x_i} \quad \frac{x \xrightarrow{a} \mu}{x+y \xrightarrow{a} \mu} \quad \frac{y \xrightarrow{a} \mu}{x+y \xrightarrow{a} \mu} \\
 \\
 \frac{x \xrightarrow{a} \mu}{x; y \xrightarrow{a} \mu; \delta_y} \quad a \neq \checkmark \quad \frac{x \xrightarrow{\checkmark} \mu \quad y \xrightarrow{a} \mu'}{x; y \xrightarrow{a} \mu'} \quad \frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \mu'}{x \parallel_B y \xrightarrow{a} \mu \parallel_B \mu'} \quad a \in B \setminus \{\checkmark\} \\
 \\
 \frac{x \xrightarrow{a} \mu}{x \parallel_B y \xrightarrow{a} \mu \parallel_B \delta_y} \quad a \notin B \cup \{\checkmark\} \quad \frac{y \xrightarrow{a} \mu}{x \parallel_B y \xrightarrow{a} \delta_x \parallel_B \mu} \quad a \notin B \cup \{\checkmark\} \quad \frac{x \xrightarrow{\checkmark} \mu \quad y \xrightarrow{\checkmark} \mu'}{x \parallel_B y \xrightarrow{\checkmark} \delta_0} \\
 \\
 \frac{x \xrightarrow{a} \mu}{U(x) \xrightarrow{a} \delta_0} \quad \frac{x \xrightarrow{b} \mu \quad \mu(Y) \geq 1 \quad \{U(y) \xrightarrow{a} \mu'_y \mid y \in Y\}}{U(x) \xrightarrow{a} \delta_0} \quad b \neq a, x \notin Y
 \end{array}$$

δ_i instead of $(\prod_{n_i \in \emptyset} \nu_{n_i}) \circ g^{-1}$. Thus, in the rules of ε and $U(x)$, we write δ_0 instead of $\sum_{i \in \{1\}} 1(\prod_{n_i \in \emptyset} \nu_{n_i}) \circ g_0^{-1}$ with $g_0(\cdot) = \mathbf{0}$. If $g = id$ is the identity function, we only write μ instead of $\mu \circ id^{-1}$ as it is the case in the conclusion of rules for $+$. Finally, for an n -ary operator f , we write $f(\nu_1, \dots, \nu_n)$ instead of $(\nu_1 \times \dots \times \nu_n) \circ f^{-1}$. For instance, in the first rule of the sequential composition, we write $\mu; \delta_y$ instead of $(\mu \times \delta_y) \circ (;)^{-1}$.

We give some examples of closed instances of rules to understand the notation in the target of the conclusion. Take the closed instance $a. \sum_{i=1}^3 [p_i]t_i \xrightarrow{a} \sum_{i=1}^3 p_i \delta_{t_i}$ of the rule of the probabilistic prefix operator and assume that $t_1 \neq t_2 = t_3$. Then, $(\sum_{i=1}^3 p_i \delta_{t_i})(t_1) = p_1$ which is what we expect. Moreover $(\sum_{i=1}^3 p_i \delta_{t_i})(t_2) = (p_2 + p_3)$ which is also what we expect, since we need $(\sum_{i=1}^3 p_i \delta_{t_i})(\{t_1, t_2, t_3\}) = 1$ (and $\{t_1, t_2, t_3\} = \{t_1, t_2\}!$).

Now, take the same term $a. \sum_{i=1}^3 [p_i]t_i$ and the closed instance of the first rule of sequential composition $\frac{a. \sum_{i=1}^3 [p_i]t_i \xrightarrow{a} \pi}{(a. \sum_{i=1}^3 [p_i]t_i); \varepsilon \xrightarrow{a} \pi; \delta_\varepsilon}$ with $\pi = \sum_{i=1}^3 p_i \delta_{t_i}$. Notice that $(\pi; \delta_\varepsilon)(t_2; \varepsilon) = (\pi \times \delta_\varepsilon)(\{(t_2, \varepsilon)\}) = (p_2 + p_3)$. Instead, for example, $(\pi; \delta_\varepsilon)(t_2; \mathbf{0}) = (\pi \times \delta_\varepsilon)(\{(t_2, \mathbf{0})\}) = \pi(t_2)\delta_\varepsilon(\mathbf{0}) = 0$, and $(\pi; \delta_\varepsilon)(t_2 + \varepsilon) = (\pi \times \delta_\varepsilon)(\{(\cdot)^{-1}(t_2 + \varepsilon)\}) = (\pi \times \delta_\varepsilon)(\emptyset) = 0$. \square

As has already been argued many times (see, e.g., [9, 12, 24]), transition system specifications with negative premises do not uniquely define a transition relation and different reasonable techniques may lead to incomparable choices. In any case, we expect that a transition relation associated to a PTSS P (i) respects the rules of P , that is, whenever the premises of a closed instance of a rule of P belong to the transition relation, so does its conclusion; and (ii) it does not include more transitions than those explicitly justified, i.e., a transition is defined only whenever there is a closed rule whose premises are in the transition relation. The first notion corresponds to that of model, and the second one to that of supported transition.

Before formally defining these notions we introduce some notation. Given a transition relation $\rightarrow \subseteq PTr(\Sigma, A)$, a positive literal $t \xrightarrow{a} \pi$ holds in \rightarrow , notation $\rightarrow \models t \xrightarrow{a} \pi$, if $(t, a, \pi) \in \rightarrow$. A negative literal $t \not\xrightarrow{a} \pi$ holds in \rightarrow , notation $\rightarrow \models t \not\xrightarrow{a} \pi$, if there is no $\pi \in \Delta(T(\Sigma))$ s.t. $(t, a, \pi) \in \rightarrow$. A quantitative literal $\pi(T) \geq p$ holds in \rightarrow , notation

$\rightarrow \models \pi(T) \geq p$ precisely when $\pi(T) \geq p$. Notice that the satisfaction of a quantitative literal does not depend on the transition relation. We nonetheless use this last notation as it turns out to be convenient. Given a set of literals H , we write $\rightarrow \models H$ if $\forall \phi \in H : \rightarrow \models \phi$.

Definition 4. Let $P = (\Sigma, A, R)$ be a PTSS. Let $\rightarrow \subseteq \text{Ptr}(\Sigma, A)$ be a probabilistic transition system. Then \rightarrow is a supported model of P if it satisfies that: $\psi \in \rightarrow$ iff there is a rule $\frac{H}{\chi} \in R$ and a proper substitution ρ s.t. $\rho(\chi) = \psi$ and $\rightarrow \models \rho(H)$.

Notice that the form of the target of the conclusion of a rule guarantees that if $\psi = t \xrightarrow{a} \pi$ then π is indeed a probability distribution (and hence, $\pi(T(\Sigma)) = 1$).

We have already pointed out that PTSSs with negative premises do not uniquely define a transition relation. In fact, a PTSS may have more than one supported model. For instance, the PTSS with the single constant f , set of labels $\{a, b\}$ and the two rules $\frac{f \xrightarrow{a} \mu}{f \xrightarrow{a} \delta_f}$ and $\frac{f \xrightarrow{b} \nu}{f \xrightarrow{b} \delta_f}$, has two supported models: $\{f \xrightarrow{a} \delta_f\}$ and $\{f \xrightarrow{b} \delta_f\}$. We will not dwell on this problem which has been studied at length in [9] and [24] in a non-probabilistic setting. We will only focus on the stratification method [12] which has been widely used to give meaning to TSS with negative premises. A stratification defines an order on closed positive literals that ensures that, in the stratified PTSS, the validity of a transition does not depend on the negation of the same transition.

Definition 5. Let $P = (\Sigma, A, R)$ be a PTSS. A function $S : \text{Ptr}(\Sigma, A) \rightarrow \alpha$, where α is an ordinal, is called stratification of P (and P is said to be stratified) if for every rule

$$r = \frac{\{t_k \xrightarrow{a_k} \mu_k : k \in K\} \cup \{t_l \xrightarrow{b_l} \nu : l \in L\} \cup \{\mu_j(W_j) \geq q_j : j \in J\}}{t \xrightarrow{a} \sum_i p_i (\prod_{n_i \in N_i} \nu_{n_i}) \circ g_i^{-1}}$$

and substitution $\rho : (\mathcal{V} \cup \mathcal{M}) \rightarrow (T(\Sigma) \cup \Delta(T(\Sigma)))$ it holds that: (i) for all $k \in K$, $S(\rho(t_k \xrightarrow{a_k} \mu_k)) \leq S(\text{conc}(r))$, and (ii) for all $l \in L$ and $\mu \in \mathcal{M}$, $S(\rho(t_l \xrightarrow{b_l} \mu)) < S(\text{conc}(r))$. Each set $S_\beta = \{\phi \mid S(\phi) = \beta\}$, with $\beta < \alpha$, is called stratum. If for all $k \in K$, $S(\rho(t_k \xrightarrow{a_k} \mu_k)) < S(\text{conc}(r))$, then the stratification is said to be strict.

A transition relation is constructed stratum by stratum in an increasing manner by transfinite recursion. If it has been decided whether a transition in a stratum $S_{\beta'}$, with $\beta' < \beta$, is valid or not, we already know the validity of the negative premise occurring in the premises of a transition φ in stratum S_β (since all positive instances of the negative premises are in strictly lesser strata) and hence we can determine the validity of φ .

Definition 6. Let $P = (\Sigma, A, R)$ be a PTSS with stratification $S : \text{Ptr}(\Sigma, A) \rightarrow \alpha$ for some ordinal α . For all rule r , let $D(r)$ be the smallest regular cardinal greater than $\lceil \text{pprem}(r) \rceil$, and let $D(P)$ be the smallest regular cardinal such that $D(P) \geq D(r)$ for all $r \in R$. The transition relation $\rightarrow_{P,S}$ associated with P (and based on S) is defined by $\rightarrow_{P,S} = \bigcup_{\beta < \alpha} \rightarrow_{P_\beta}$, where each $\rightarrow_{P_\beta} = \bigcup_{j \leq D(P)} \rightarrow_{P_{\beta,j}}$ and each $\rightarrow_{P_{\beta,j}}$ is defined by

$$\rightarrow_{P_{\beta,j}} = \left\{ \psi \mid S(\psi) = \beta \text{ and } \exists r \in R \text{ and proper substitution } \rho \text{ s.t. } \psi = \text{conc}(\rho(r)), \right. \\ \left. \begin{aligned} & (\bigcup_{\gamma < \beta} \rightarrow_{P_\gamma}) \cup (\bigcup_{j' < j} \rightarrow_{P_{\beta,j'}}) \models \text{qprem}(\rho(r)) \cup \text{pprem}(\rho(r)) \text{ and} \\ & (\bigcup_{\gamma < \beta} \rightarrow_{P_\gamma}) \models \text{nprem}(\rho(r)) \end{aligned} \right\}$$

The PTSS with the only two rules $\frac{f \xrightarrow{a} \mu}{f \xrightarrow{a} \delta_f}$ and $\frac{f \xrightarrow{a} \mu}{f \xrightarrow{b} \delta_f}$ (given before) can be stratified by a function S such that $S(f \xrightarrow{a} \delta_f) = 0$ and $S(f \xrightarrow{b} \delta_f) = 1$. Using S , the model associated with the PTSS is $\{f \xrightarrow{b} \delta_f\}$. More interestingly, a stratification for our running example is given by $S(t \xrightarrow{a} \mu) = \zeta(t)$ where $\zeta(\mathbf{0}) = \zeta(\varepsilon) = \zeta(a, \sum_{i=1}^n [p_i] t_i) = 0$, $\zeta(t_1 + t_2) = \zeta(t_1; t_2) = \zeta(t_1 \parallel_B t_2) = \max\{\zeta(t_1), \zeta(t_2)\}$, and $\zeta(\mathbf{U}(t)) = \zeta(t) + 1$. Notice that this stratification is *not* strict.

The existence of a stratification guarantees the existence of a supported model. In fact, such model is the one in Def. 6 (Theorem 7) and it is the only possible one defined via stratification (Theorem 8). Moreover, if it is defined using a strict stratification, the supported model is unique (Theorem 9).

The proofs of the following theorems follow closely the proofs of their non-probabilistic counterparts in [12] (Theorem 2.15, Lemma 2.16 and Theorem 2.18, respectively). The only actual difference lies on the quantitative premises, which do not pose any particular problem since their validity only depends on the substitution. For the next theorems, let $P = (\Sigma, A, R)$ be a PTSS with stratification S .

Theorem 7. *The transition relation $\rightarrow_{P,S}$ is a supported model of P .*

Theorem 8. *If S' is another stratification for P , $\rightarrow_{P,S} = \rightarrow_{P,S'}$.*

Theorem 9. *If S is strict, then, $\rightarrow_{P,S}$ is the only supported model of P .*

4 The $nt\mu fv/nt\mu xv$ Format and the Congruence Theorem

In this section we present one of the main results of our paper: we introduce a general format that ensures that bisimulation equivalence is a congruence for any operator defined in this format. The importance of the theorem is that congruence of bisimilarity is guaranteed by mere inspection of the rules. We first define the notion of bisimulation on probabilistic transition system [18]. We use a more modern (but equivalent) definition.

Given a relation $R \subseteq T(\Sigma) \times T(\Sigma)$, a set $Q \subseteq T(\Sigma)$ is R -closed if for all $t \in Q$ and $t' \in T(\Sigma)$, $t R t'$ implies $t' \in Q$ (i.e. $R(Q) \subseteq Q$). If a set Q is R -closed we write $R\text{-closed}(Q)$. It is easy to verify that if two relation $R, R' \subseteq T(\Sigma) \times T(\Sigma)$ are such that $R' \subseteq R$, then for all set $Q \subseteq T(\Sigma)$, $R\text{-closed}(Q)$ implies $R'\text{-closed}(Q)$.

Definition 10. *A relation $R \subseteq T(\Sigma) \times T(\Sigma)$ is a bisimulation if R is symmetric and for all $t, t' \in T(\Sigma)$, $\pi \in \Delta(T(\Sigma))$, $a \in A$,*

$$t R t' \text{ and } t \xrightarrow{a} \pi \text{ imply that there exists } \pi' \in \Delta(T(\Sigma)) \text{ s.t. } t' \xrightarrow{a} \pi' \text{ and } \pi R \pi',$$

where $\pi R \pi'$ if and only if $\forall Q \subseteq T(\Sigma) : R\text{-closed}(Q) \Rightarrow \pi(Q) = \pi'(Q)$. We define the relation \sim as the smallest relation that includes all other bisimulation. It is known that \sim is itself a bisimulation relation and an equivalence relation.

Before introducing the $nt\mu fv/nt\mu xv$ format, we give a first approach to extend the $ntyft/ntyxt$ format with probabilities that considers a very restrictive form of quantitative premise. It can also be seen as a generalization of Segala-GSOS format [7] with

terms in the premises as well as lookahead. This first approach considers rules of the form

$$\frac{\{t_m \xrightarrow{a_m} \mu_m : m \in M\} \cup \{t_n \xrightarrow{b_n} \nu_n : n \in N\} \cup \{\mu_l(z_l) > 0 : l \in L\}}{f(x_1, \dots, x_{r(f)}) \xrightarrow{a} \sum_{i \in I} p_i (\prod_{n_i \in N_i} \nu_{n_i}) \circ g_i^{-1}} \quad (\text{F})$$

where M , N , and L are index sets, μ_m, z_l, x_k ($1 \leq k \leq r(f)$) are all different variables, $f \in F$, $t_m, t_n \in \mathbb{T}(\Sigma)$, and p_i and g_i are like in Def. 2. Notice that all rules in Table 1 respond to this format except for the last one which has a quantitative premise comparing to a number different from 0. (It can be proved that bisimilarity is a congruence for any operator defined in format (F).)

In the following we present several counterexamples justifying the restrictions imposed by format in eq. (F). We consider a signature with a unary operator f and three constants b , c and d , together with a label a . We will also consider axioms $c \xrightarrow{a} \delta_c$ and $d \xrightarrow{a} (0.5 \cdot \delta_c + 0.5 \cdot \delta_d)$, and no rule associated to constant b . (We write π_d for $(0.5 \cdot \delta_c + 0.5 \cdot \delta_d)$). Notice that $c \sim d$. In the following we concentrate in rules for f .

The need that the source of the conclusion of a rule has a particular format has already been shown by several counterexamples in [12, 13] for the *tyft/tyxt* format. We adapt an example from [12] to motivate the need. Consider the axiom $f(b) \xrightarrow{a} \delta_{f(b)}$. Then $f(f(b)) \sim b$ since none of them perform any action. But $f(f(f(b)))$ and $f(b)$ are not bisimilar since $f(b)$ can perform an action but $f(f(f(b)))$ cannot. Similarly, the requirement that all variables μ_m, z_l, x_k are different is inherited from the *tyft/tyxt* format. Examples from [13] should be easily adaptable to our setting.

The next example shows that the target of a positive premise cannot be a distribution on a particular (shape of) term. Consider rule $\frac{x \xrightarrow{a} \delta_c}{f(x) \xrightarrow{a} \delta_c}$. Then, despite that $c \sim d$, $f(c)$ and $f(d)$ are not bisimilar since $d \xrightarrow{a} \delta_c$ is *not* a valid transition in the (unique) supported model. A similar effect has rule $\frac{x \xrightarrow{a} \mu \quad \mu(d) > 0}{f(x) \xrightarrow{a} \delta_c}$, which shows that quantitative literals cannot enquire over arbitrary terms: note that $f(c)$ and $f(d)$ are not bisimilar since $c \xrightarrow{a} \delta_c$ and $\delta_c(d) = 0$.

Allowing for a quantitative literal that compares with a value different from 0 is also problematic. Consider rule $\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \mu' \quad \mu(y) \geq 1}{f(x) \xrightarrow{a} \delta_c}$. Again $f(c)$ and $f(d)$ are not bisimilar since $d \xrightarrow{a} \pi_d$, and there is no single term t in which $\pi_d(t) \geq 1$.

This example suggest that quantitative premises should have the form $\mu(Y) > p$ or $\mu(Y) \geq p$ where Y is a set of variables. So the previous rule could be recast as $\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \mu' \quad \mu(\{y, z\}) \geq 1}{f(x) \xrightarrow{a} \delta_c}$. However, the same problem repeats if we introduce a new constant e with $e \xrightarrow{a} (0.4 \cdot \delta_c + 0.3 \cdot \delta_d + 0.3 \cdot \delta_e)$. In fact, it turns out that Y needs to be *infinite* (consider the case in which a new infinite set of constants $\{e_n\}_{n \in \mathbb{N}_0}$ is defined with $e_n \xrightarrow{a} (\sum_{i \in \mathbb{N}_0} \frac{1}{2^{i+1}} \cdot \delta_{e_i})$). Moreover, it is necessary that all terms that substitutes some variable in Y have symmetric behavior. Notice that the term substituting z is not required to perform action a , which was not the originally intended behavior. Moreover, symmetry is also necessary for the congruence result as we will see later.

After the previous considerations, we extend format (F) with quantitative premises of the form $\mu(Y) > p$ or $\mu(Y) \geq p$. We call this format *ntufv/ntuxv* following the nomenclature of [12, 13]. Later we give more examples justifying our restrictions.

Let $\{Y_l\}_L$ be a family of sets of variables with the same cardinality. Given a tuple \vec{y} , the l -th element of \vec{y} is denoted by $\vec{y}(l)$. Fix a set $\text{Diag}\{Y_l\}_L \subseteq \prod_{l \in L} Y_l$ so that:

- (i) for all $l \in L$, $\pi_l(\text{Diag}\{Y_l\}_L) = Y_l$ (here, π_l indicates the l -th projection); and
- (ii) for all $\vec{y}, \vec{y}' \in \text{Diag}\{Y_l\}_L$, $(\exists l \in L : \vec{y}(l) = \vec{y}'(l)) \Rightarrow \vec{y} = \vec{y}'$.

Notice that if each set $Y_l = \{y_l^0, y_l^1, y_l^2, \dots\}$, a possible definition for $\text{Diag}\{Y_l\}_L$ may be $\text{Diag}\{Y_l\}_L = \{(y_0^0, y_1^0, \dots, y_L^0), (y_0^1, y_1^1, \dots, y_L^1), (y_0^2, y_1^2, \dots, y_L^2), \dots\}$.

Definition 11 (ntufv/ntuxv). Let $P = (\Sigma, A, R)$ be a stratifiable PTSS. A rule $r \in R$ is in *ntufv* format if it has the form

$$\frac{\bigcup_{m \in M} \{t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}} : \vec{z} \in \mathcal{Z}\} \cup \bigcup_{n \in N} \{t_n(\vec{z}) \xrightarrow{b_{n_i}} : \vec{z} \in \mathcal{Z}\} \cup \{\mu_l^{\vec{z}}(Y_l) \triangleright_l q_l : l \in L, \vec{z} \in \mathcal{Z}\}}{f(x_1, \dots, x_{r(f)}) \xrightarrow{a} \sum_{i \in I} p_i(\prod_{n_i \in N_i} v_{n_i}) \circ g_i^{-1}}$$

with $\triangleright_l \in \{>, \geq\}$, for all $l \in L$, satisfying the following conditions:

1. Each set Y_l should be at least countably infinite, for all $l \in L$, and the cardinality of L should be strictly smaller than that of the Y_l 's.
2. $\mathcal{Z} = \text{Diag}\{Y_l\}_L \times \prod_{w \in W} \{w\}$, with $W \subseteq \mathcal{V} \setminus \bigcup_{l \in L} Y_l$.
3. All variables $\mu_m^{\vec{z}}$, with $m \in M$ and $\vec{z} \in \mathcal{Z}$, are different.
4. For all $\vec{z}, \vec{z}' \in \mathcal{Z}$, $m \in M$, if $\mu_m^{\vec{z}} = v_{n_i}$ and $\mu_m^{\vec{z}'} = v_{n_h}$ for some $n_i \in N_i$, $n_h \in N_h$, $i, h \in I$, then $\vec{z} = \vec{z}'$.
5. For all $l \in L$, $Y_l \cap \{x_1, \dots, x_{r(f)}\} = \emptyset$, and $Y_l \cap Y_{l'} = \emptyset$ for all $l' \in L$, $l \neq l'$.
6. All variables $x_1, \dots, x_{r(f)}$ are different.
7. $f \in F$ and for all $m \in M$ and $n \in N$, $t_m, t_n \in \mathbb{T}(\Sigma)$. In all cases, if $t \in \mathbb{T}(\Sigma)$ and $\text{Var}(t) \subseteq \{w_1, \dots, w_H\}$, $t(w'_1, \dots, w'_H)$ is the same term as t where each occurrence of variable w_h (if it appears in t) has been replaced by variable w'_h , for $1 \leq h \leq H$.

A rule $r \in R$ is in *ntuxv* format if its form is like before but with the conclusion having instead the form $x \xrightarrow{a} \sum_{i \in I} p_i(\prod_{n_i \in N_i} v_{n_i}) \circ g_i^{-1}$. It satisfies the same conditions as above only that $x \notin Y_l$ for all $l \in L$ instead of $Y_l \cap \{x_1, \dots, x_{r(f)}\} = \emptyset$ in item 5.

P is in *ntufv* (resp. *ntuxv*) format if all its rules are in *ntufv* (resp. *ntuxv*) format. P is in *ntufv/ntuxv* format if all its rules are either in *ntufv* format or *ntuxv* format.

We define notation $t_m(\vec{Z}_m) \xrightarrow{a_m} \mu_m$ as an abbreviation for $\{t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}} : \vec{z} \in \mathcal{Z}\}$ where $\vec{Z}_m = \text{Diag}\{Y_l\}_{L'} \times \prod_{w \in W'} \{w\}$ with $L' \subseteq L$ and $W' \subseteq W$, where the number of variables of t_m is exactly the dimension of \vec{Z}_m (i.e. $|\text{Var}(t_m)| = |L'| + |W'|$). Similarly, we define $t_n(\vec{Z}_n) \xrightarrow{b_{n_i}}$ as an abbreviation for $\{t_n(\vec{z}) \xrightarrow{b_{n_i}} : \vec{z} \in \mathcal{Z}\}$, and $\mu_l(Y_l) \triangleright_l q_l$ for the set $\{\mu_l^{\vec{z}}(Y_l) \triangleright_l q_l : \vec{z} \in \mathcal{Z}\}$. Thus, rule $\frac{y \xrightarrow{a} \mu}{x+y \xrightarrow{a} \mu}$ is the notational rewriting of rule $\frac{\{y \xrightarrow{a} \mu_i | i \geq 0\}}{x+y \xrightarrow{a} \mu_0}$

and rule $\frac{x \xrightarrow{b} \mu \quad \mu(Y) \geq 1 \quad \{U(y) \xrightarrow{a} \mu'_i | y \in Y\}}{U(x) \xrightarrow{a} \delta_0} \quad b \neq a$ can be rewritten to $\frac{x \xrightarrow{b} \mu \quad \mu(Y) \geq 1 \quad U(Y) \xrightarrow{a} \mu'}{U(x) \xrightarrow{a} \delta_0} \quad b \neq a$. In fact, notice that all rules of our running example (see Table 1) are in *ntufv* format.

Restrictions [3](#), [5](#), [6](#) and [7](#) are basically the same requirements present in the format of eq. (F). Hence, all examples given before also apply to the $nt\mu fv/nt\mu xv$ format. Besides, notice that rule $\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \mu' \quad \mu(y) \geq 1}{f(x) \xrightarrow{a} \delta_c}$ given before is not in $nt\mu fv/nt\mu xv$ format, but

the intended behavior can be encoded as the $nt\mu fv$ rule $\frac{x \xrightarrow{a} \mu \quad Y \xrightarrow{a} \mu' \quad \mu(Y) \geq 1}{f(x) \xrightarrow{a} \delta_c}$.

The next example shows that quantitative literals cannot check for upper bounds (or equality). Consider the rule $\frac{x \xrightarrow{a} \mu \quad Y \xrightarrow{a} \mu' \quad \mu(Y) \leq 0.5}{f(x) \xrightarrow{a} \delta_c}$ with c and d defined as before. $f(c)$ and $f(d)$ are not bisimilar because $f(d) \xrightarrow{a} \delta_c$ by taking the substitution ρ such that $\rho(y) = c$ for all $y \in Y$, but $f(c) \not\xrightarrow{a}$ since there is no set of terms T such that *properly* substituted in Y (i.e., such that $\delta_c(t) > 0$ for all $t \in T$), $\delta_c(T) \leq 0.5$.

Finally, if symmetry of behavior on variables in Y_l were not enforced, it would also be possible to distinguish distributions that are equivalent. Consider now a signature with constants c, d , and $\{n, n' \mid n \in \mathbb{N}_0\}$, unary operator f and rules $n \xrightarrow{n} \delta_n, n' \xrightarrow{n} \delta_n, c \xrightarrow{a} \pi$, and $d \xrightarrow{a} \pi'$ with $\pi = \sum_{i \in \mathbb{N}_0} \frac{1}{2^{i+1}} \cdot \delta_n$ and $\pi' = \sum_{i \in \mathbb{N}_0} (\frac{1}{2^{i+2}} \cdot \delta_n + \frac{1}{2^{i+2}} \cdot \delta_{n'})$, and $\frac{x \xrightarrow{a} \mu \quad \{y_k \xrightarrow{k} \mu_k \mid k \in \mathbb{N}_0\} \quad \mu(\{y_k\}_{k \in \mathbb{N}_0}) \geq 1}{f(x) \xrightarrow{b} \mu}$. Notice that $c \sim d$; nonetheless, $f(c) \xrightarrow{b} \delta_c$ but $f(d) \not\xrightarrow{b}$

since $d \xrightarrow{a} \pi'$ but there is no way to match both n and n' to two different variables y_{k_1} and y_{k_2} (for all $n \in \mathbb{N}_0$), and hence $\pi'(\rho(\{y_k\}_{k \in \mathbb{N}_0})) = 0.5$ for any substitution ρ satisfying the positive premises. We finally mention that conditions [11](#) and [4](#) in Def. [11](#) are more technical and their justification only becomes apparent in the proof of Theorem [12](#).

The strategy of proof for the congruence theorem follows the lines of the proof of Theorem 4.14 in [\[12\]](#) though some considerable rework is needed to manipulate quantitative premises. Notice, however that we do not require well-foundedness.

Theorem 12. *Let P be a stratifiable PTSS in $nt\mu fv/nt\mu xv$ format. Then \sim is a congruence relation.*

5 Modular Properties

Often, one wants to extend a language with new operations and behaviors. This is naturally done by adding new functions and rules to the original PTSS. In other words, given two PTSSs P^0 and P^1 , one wants to combine them in a new PTSS $P^0 \oplus P^1$, where we generally assume that P^0 is the original PTSS and P^1 is the extension. A desired property is that the extension does not alter the behavior of the terms in the original language. That is, one expects that for every old term $t \in T(\Sigma^0)$, the set of outgoing transitions defined by P^0 is exactly the same that those defined by $P^0 \oplus P^1$. In this case we say that $P^0 \oplus P^1$ is a *conservative extension* of P^0 .

Definition 13. *Let $\Sigma^0 = (F^0, r^0)$ and $\Sigma^1 = (F^1, r^1)$ be two signatures s.t. $f \in F^0 \cap F^1 \Rightarrow r^0(f) = r^1(f)$. The sum of Σ^0 and Σ^1 , notation $\Sigma^0 \oplus \Sigma^1$, is the new signature $(F^0 \cup F^1, r)$ where $r(f) =$ if $f \in F^0$ then $r^0(f)$ else $r^1(f)$ for all $f \in F^0 \cup F^1$.*

Given two PTSS $P^0 = (\Sigma^0, A^0, R^0)$ and $P^1 = (\Sigma^1, A^1, R^1)$ s.t. $\Sigma = \Sigma^0 \oplus \Sigma^1$ is defined, the sum of P^0 and P^1 , notation $P^0 \oplus P^1$, is the PTSS $P^0 \oplus P^1 = (\Sigma^0 \oplus \Sigma^1, A^0 \cup A^1, R^0 \cup R^1)$. We say that $P^0 \oplus P^1$ is a conservative extension of P^0 and that P^1 can be added

conservatively to P^0 if $P^0 \oplus P^1$ is stratifiable and for all $t \in T(\Sigma^0)$, $a \in A^0 \cup A^1$ and $\mu \in \mathcal{A}(T(\Sigma^0 \cup \Sigma^1))$ it holds $t \xrightarrow{a} \mu \in \rightarrow_{P^0 \oplus P^1} \Leftrightarrow t \xrightarrow{a} \mu \in \rightarrow_{P^0}$

Basically, a rule is well-founded if there is no circular dependency of variables in its set of premises. We adapt the definition of well-founded from [13] to our setting. Besides, we also require that distribution variables in the premises appear always bound.

Let W be a set containing positive and quantitative literals. The *variable dependency graph* of W is a directed graph with (i) set of nodes $\cup\{\text{Var}(\psi) : \psi \in W\}$, and (ii) edges $\{(x, \mu) : x \in \text{Var}(t), (t \xrightarrow{a} \mu) \in W\} \cup \{(x, \mu) : x \in X, (\mu(X) \geq p) \in W\}$. W is *well-founded* if any backward chain of edges in the variable dependency graph is finite and every distribution variable has a predecessor. A rule is *well-founded* if the set of all its premises is well-founded. A PTSS is *well-founded* if all its rules are well-founded. A rule r is called *pure* if it is well-founded and does not contain free variables. A PTSS P is called *pure* if all of its rules are pure.

Theorem 14 gives sufficient conditions to ensure that a PTSS can be extended conservatively and its similar to Theorem 4.8 in [10]. Theorem 15 gives sufficient conditions to ensure that the sum PTSS $P^0 \oplus P^1$ is stratifiable, knowing that the original PTSSs P^0 and P^1 are also stratifiable. Its proof follows closely that of Theorem 5.8 in [12].

Theorem 14. *Let $P^0 = (\Sigma^0, A^0, R^0)$ be a PTSS in pure $\text{nt}\mu\text{fv}/\text{nt}\mu\text{xv}$ format and let $P^1 = (\Sigma^1, A^1, R^1)$ be a PTSS such that for all rule $r \in R^1$ with $\text{conc}(r) = t \xrightarrow{a} \mu$, $t \notin \mathbb{T}(\Sigma_0)$. Let $P = P^0 \oplus P^1$ be defined and stratifiable. Then P^1 can be added conservatively to P^0 .*

Theorem 15. *Let $\Sigma^0 = (F^0, r^0)$ and $\Sigma^1 = (F^0, r^1)$ be two signatures with constants $a^0 \in F^0$ and $a^1 \in F^1$, such that $\Sigma^0 \oplus \Sigma^1$ is defined. Let $P^0 = (\Sigma^0, A^0, R^0)$ and $P^1 = (\Sigma^1, A^1, R^1)$ be two stratifiable PTSS. If for all substitutions ρ_0 and ρ_1 and rules $r_0 \in R^0$ and $r_1 \in R^1$, it holds that $\rho_0(\psi) \neq \rho_1(\phi)$ with $\phi = \text{conc}(r_1)$ and $\psi \in \text{pprem}(r_0)$ or $\psi = t \xrightarrow{a} \mu$ with $t \xrightarrow{a} \in \text{nprem}(r_0)$, then $P^0 \oplus P^1$ is also stratifiable.*

6 Tracing Bisimulation

Two terms are (*possibilistic*) trace equivalent if they can perform the same sequences of actions with some positive probability (but not necessarily the same). In this section we show that the trace congruence induced by the $\text{nt}\mu\text{fv}/\text{nt}\mu\text{xv}$ format is exactly a “finitary” version of the bisimulation equivalence. This relation, which we called *bounded bisimilarity*, agrees with \sim on image finite probabilistic transition systems. (\rightarrow_P is image-finite iff for all $t \in T(\Sigma)$ and $a \in A$, the set $\{\mu \mid t \xrightarrow{a}_P \mu\}$ is finite.)

Definition 16. *Let $P = (\Sigma, A, R)$ be a stratifiable PTSS with associated relation \rightarrow_P . Given $t \in T(\Sigma)$, a sequence $a_1 \dots a_n \in A^*$ is a *trace* from t iff there are terms $t_0, \dots, t_n \in T(\Sigma)$ and distributions π_1, \dots, π_n s.t. $t_0 = t$, $t_i \xrightarrow{a_{i+1}} \pi_{i+1}$ and $\pi_{i+1}(t_{i+1}) > 0$ for $0 \leq i < n$. Let $\text{Tr}(t)$ be the set of all traces from t . Two terms $t, t' \in T(\Sigma)$ are trace equivalent with respect to P , notation $t \equiv_P^T t'$, iff $\text{Tr}(t) = \text{Tr}(t')$.*

We say that $C[x_1, \dots, x_n]$ is a *context* if $C[x_1, \dots, x_n]$ is an open term in which at most the distinct variables x_1, \dots, x_n appear. As usual, $C[t_1, \dots, t_n]$ denotes the term obtained by replacing all occurrences of variables x_i by t_i .

Definition 17. Let $P = (\Sigma, A, R)$ be a stratifiable PTSS in $nt\mu fv/nt\mu xv$ format. Two terms $t, t' \in T(\Sigma)$ are trace congruent with respect to $nt\mu fv/nt\mu xv$, notation $t \equiv_{nt\mu fv/nt\mu xv}^T t'$, iff for all PTSS $P' = (\Sigma', A', R')$ in $nt\mu fv/nt\mu xv$ format which can be added conservatively to P and for every context $C[x]$ it holds that $C[t] \equiv_{P \oplus P'}^T C[t']$.

Let $P = (\Sigma, A, R)$ be a stratifiable PTSS with associated relation \rightarrow_P . The relations $\simeq_P^n \subseteq T(\Sigma) \times T(\Sigma)$ for $n \in \mathbb{N}$ are inductively defined by:

$$\begin{aligned} \simeq_P^0 &= T(\Sigma) \times T(\Sigma) \\ \simeq_P^{n+1} &= \{(t, t') \mid (t \xrightarrow{a} \pi \Rightarrow \exists \pi' : t' \xrightarrow{a} \pi' \wedge \pi \simeq_P^n \pi') \wedge (t' \xrightarrow{a} \pi' \Rightarrow \exists \pi : t \xrightarrow{a} \pi \wedge \pi \simeq_P^n \pi')\} \end{aligned}$$

Given $t, t' \in T(\Sigma)$, t and t' are n -bounded bisimilar iff $t \simeq_P^n t'$. We say that t and t' are bounded bisimilar, notation $t \simeq_P t'$, if $t \simeq_P^n t'$ for all $n \in \mathbb{N}$.

Bounded bisimilarity and bisimulation equivalence agree on image-finite probabilistic transition systems [5, Lemma 3.5.8]. That is, if \rightarrow_P is image-finite, then $\sim = \simeq_P$.

We now define the *bisimulation tester*, that is, a PTSS P_T that can be added conservatively to another PTSS and introduce contexts that are able to distinguish non-bisimilar terms. More precisely, P_T introduces two family of functions, binary functions B_n , $(k+1)$ -ary functions Pr_n^k ($n, k \in \mathbb{N}$), and a trivial constant \perp . Their intended meaning is as follows. $B_n(t, u)$ can detect whether t and u are n -bounded bisimilar by showing transition $B_n(t, u) \xrightarrow{yes} \delta_\perp$. Otherwise, $B_n(t, u) \xrightarrow{no} \delta_\perp$. In this way, two non-bisimilar terms t and u can be distinguished by the context $B_n(t, _)$ for some appropriate n . Pr_n^k is used as an auxiliary operator to test the measures of k (not necessarily different) $(n-1)$ -bounded bisimulation equivalence classes. More precisely, $Pr_n^k(t, u_1, \dots, u_k) \xrightarrow{(a, q_1, \dots, q_k)} \delta_\perp$ if there is a transition $t \xrightarrow{a} \pi$ such that $\pi([u_1]_{\simeq^{n-1}}) \geq q_1, \dots, \pi([u_k]_{\simeq^{n-1}}) \geq q_k$, where q_1, \dots, q_k are some rational numbers.

Definition 18. Let $P = (\Sigma, A, R)$ be a PTSS. The bisimulation tester of P is a PTSS $P_T = (\Sigma_T, A_T, R_T)$ where $\Sigma \subseteq \Sigma_T$ and Σ_T contains binary functions B_n and functions Pr_n^k with arity $k+1$, $n \in \mathbb{N}$ and a constant \perp , $A_T = A \cup (\bigcup_{i>0} (A \times \mathbb{Q}^i)) \cup \{yes, no\}$, and R contains the following rules (for all $n, k > 0$, $a \in A$ and $q \in \mathbb{Q}$):

$$\begin{aligned} (1) \quad & B_0(x, y) \xrightarrow{yes} \delta_\perp \quad \frac{Pr_n^k(x, z_1, \dots, z_k) \xrightarrow{(a, q_1, \dots, q_k)} \mu \quad Pr_n^k(y, z_1, \dots, z_k) \xrightarrow{(a, q_1, \dots, q_k)} \delta_\perp}{B_n(x, y) \xrightarrow{no} \delta_\perp} \quad (3) \\ (2) \quad & \frac{x \xrightarrow{a} \mu \quad \{B_{n-1}(z_i, Z_i) \xrightarrow{yes} \mu_i, \mu(Z_i) \geq q_i\}_{i=1}^k}{Pr_n^k(x, z_1, \dots, z_k) \xrightarrow{(a, q_1, \dots, q_k)} \delta_\perp} \quad \frac{B_n(x, y) \not\xrightarrow{no} \delta_\perp \quad B_n(y, x) \not\xrightarrow{no} \delta_\perp}{B_n(x, y) \xrightarrow{yes} \delta_\perp} \quad (4) \end{aligned}$$

The idea behind functions Pr_n^k explained above becomes apparent in rule (2). Besides, notice that distinction between two non n -bounded bisimilar terms is revealed by rule (3) where the negative premise indicates that it is not able to find an a -transition for y that measures more than q_i in each equivalence class $[z_i]_{\simeq^{n-1}}$ (in the appropriate instance of z_i) while the positive premise is able to do it for x .

Observe that P_T is in $nt\mu fv$ format but is *not* pure. Though this is not necessary, it is quite convenient in our case: the non-pure rule (3) allows for instances of arbitrary terms (and hence arbitrary $(n-1)$ -bounded bisimulation equivalence classes) which is

in the core of the definition of the n -bisimulations. Nevertheless, the fact that $P_{\mathcal{T}}$ is not pure is not a problem to ensure that it extends conservatively a given PTSS in a well behaved manner using Theorems [14] and [15].

It is not too difficult to find a stratification for $P_{\mathcal{T}}$ (it can be obtained in a similar manner to [12, Lemma 6.8]). The following lemma is the core of Theorem [20] below.

Lemma 19. *Let $P = (\Sigma, A, R)$ be a stratifiable PTSS in pure $nt\mu f\nu/nt\mu x\nu$ format containing at least one constant in its signature. Moreover, yes, no $\notin A$ and Σ does not contain function names B_n and P_n^k for all $n, k \in \mathbb{N}$. Then, $B_n(t, t') \xrightarrow{\text{yes}} \delta_{\perp} \in \rightarrow_{P \oplus P_{\mathcal{T}}} \Leftrightarrow t \simeq_P^n t'$, for all $t, t' \in T(\Sigma)$.*

Theorem [20] states that bisimulation equivalence is *fully abstract* with respect to the $nt\mu f\nu/nt\mu x\nu$ format and trace equivalence. That is, it states that bisimulation equivalence is the coarsest congruence with respect to any operator whose semantics is defined through $nt\mu f\nu/nt\mu x\nu$ rules and that is included in trace equivalence. Its proof is a direct consequence of Theorem [12, Lemma 19] and [3, Lemma 3.5.8].

Theorem 20. *Let $P = (\Sigma, A, R)$ be a stratifiable PTSS in pure $nt\mu f\nu/nt\mu x\nu$ format containing at least one constant in Σ . Moreover, \rightarrow_P is image-finite, yes, no $\notin A$ and Σ does not contain function names B_n and P_n^k for all $n, k \in \mathbb{N}$. Then, for all $t, t' \in T(\Sigma)$, $t \equiv_{nt\mu f\nu/nt\mu x\nu}^T t' \Leftrightarrow t \simeq_P t' \Leftrightarrow t \sim t'$*

7 Concluding Remarks

Related Work. SOS for probabilistic systems have received relatively little attention. To our knowledge, only [6, 7, 16, 17] study rule formats to specify probabilistic transition systems, and in [7, 15] they are embedded in general bialgebraic frameworks.

Both RTSS format [17] and PGSOS format [6, 7] consider transitions with the form $t \xrightarrow{a, q} t'$ as already explained in the introduction. They allow for the specification of only reactive probabilistic systems (i.e. they should satisfy that if $t \xrightarrow{a} \pi$ and $t \xrightarrow{a} \pi'$, then $\pi = \pi'$). Moreover, these formats are very much like GSOS [8] in the sense that premises are of the form $x_i \xrightarrow{a_i, q_i} y_i$ or $x_i \xrightarrow{b_i}$ where each x_i is a variable appearing on the term $f(\vec{x})$ at the source of the conclusion. Moreover, q_i needs to be a variable, so there is no possibility of testing for a particular probability value. In addition, RTSS allows for a restricted form of lookahead: only one step ahead from variable y_i can be tested and moreover probabilities should be appropriately combined in the conclusion of the rule. We remark that both RTSS and PGSOS formats can be encoded in the $nt\mu f\nu/nt\mu x\nu$ format. Segala-GSOS format [7] allows for rules like in eq. (F), with the restriction that terms t_m and t_n can only be any of the variables x_k . Therefore, lookahead is not permitted. Clearly this format can also be encoded in the $nt\mu f\nu/nt\mu x\nu$ format.

Bialgebras present an abstract categorical framework to study structured operational semantics and, in this setting, general congruence theorems have been presented [15, 23]. They introduce the so called *abstract GSOS* and *abstract safe ntree* [15, 23]. In fact, Segala-GSOS is derived as an instance of abstract GSOS [7]. In a recent and yet unpublished work, we showed that the $nt\mu f\nu/nt\mu x\nu$ format reduces to a form of *probabilistic ntree* format, just like the $ntyf\nu/ntyxt$ format reduces to *ntree* format [11]. As in the

non-probabilistic case, negative premises are not reducible to the form $x \xrightarrow{a}$ and retain the form $t \xrightarrow{a}$ with t being an arbitrary term. Precisely because of this, our format (like the $ntyft/ntyxt$ format) cannot be instanced as an abstract safe tree. Moreover, it is also not fully clear to us how to encode quantitative premises in the bialgebraic framework.

Notice that none of the previously mentioned formats can encode the bisimulation tester of Def. 18 since it needs lookahead, negative premises of the form $f(\vec{x}) \xrightarrow{a}$, and quantitative premises testing against any possible probability value and none of the previous formats allow for all these simultaneously. In fact, to the authors knowledge no full abstraction result for rule formats has been presented before for PTSS. However, related to this result, we should remark that testers for bisimulation of *deterministic* probabilistic transition systems were already introduced in [18].

We also remark that the $nt\mu fv/nt\mu xv$ format should be considered as a probabilistic extension of the $tyft/tyxt$ and $ntyft/ntyxt$ formats [12, 13]. These formats can be encoded in $nt\mu fv/nt\mu xv$ format if non-probabilistic transitions $t \xrightarrow{a} t'$ are considered as a probabilistic transition in the usual way, i.e., as $t \xrightarrow{a} \delta_{\perp}$. Finally, we observe that there is a rule format for generative probabilistic systems [16, 17] which is not covered by our format since it is very different in nature to the model we use.

Conclusion. In this article we have introduced PTSSs and the $nt\mu fv/nt\mu xv$ format for rules that specify probabilistic transition systems. We proved that bisimilarity is a congruence for all operators definable in this format and that it is also the least congruence relation preserved by all such operators included in possibilistic trace equivalence. We have also presented several standard theorems that ensure definability and uniqueness of models and conservative extensions, among others.

We highlight the introduction of our quantitative premises which, in combination with lookahead, permits the constructions of powerful operators. An example is the tester of Def. 18. Another one, more interesting, is a deadlock measuring operator dk where $dk(t) \xrightarrow{q} \nu$ iff t reaches a deadlock state with probability larger or equal to q in any possible resolution of nondeterminism. The rules are as follows

$$\begin{array}{c}
 \frac{\{x \xrightarrow{a} \mid a \in A\}}{dk(x) \xrightarrow{1} \delta_{\perp}} \quad \frac{\{B_n(x, y) \xrightarrow{yes} \mu_n \mid n \in \mathbb{N}_0\}}{B(x, y) \xrightarrow{yes} \delta_{\perp}} \\
 \frac{x \xrightarrow{a} \mu \quad \left\{ dk(z_i) \xrightarrow{p_i} \mu_i, \quad \mu(Z_i) \geq q_i, \quad B(z_i, Z_i) \xrightarrow{yes} \mu'_i, \quad B(z_i, z_j) \xrightarrow{yes} \right\}_{\substack{i, j \in I \\ i \neq j}}}{dk(x) \xrightarrow{\sum_{i \in I} q_i p_i} \delta_{\perp}} \quad \begin{array}{l} I \text{ is a countable} \\ \text{index set and} \\ \sum_{i \in I} q_i \leq 1 \end{array}
 \end{array}$$

The last rule appropriately collect the probabilities by looking ahead on disjoint (non-bisimilar) terms (notice the use of the bisimulation tester). Operation dk is somehow related to the zero process of [3] that allows for detection of inevitable deadlock.

We remark that the congruence theorem also holds for PTSs with subprobability distributions (i.e. distributions such that $\pi(T(\Sigma)) < 1$). However, we do not know whether the full abstraction result remains valid in this setting: our tester would fail to distinguish c from d where $c \xrightarrow{a} (0.5 \cdot \delta_c + 0.5 \cdot \delta_{\perp})$, $c \xrightarrow{a} (0.5 \cdot \delta_c)$, and $d \xrightarrow{a} (0.5 \cdot \delta_c + 0.5 \cdot \delta_{\perp})$.

Acknowledgement. We would like to thank the anonymous referees whose suggestions let us improve the presentation of our paper.

References

1. Aceto, L., Fokkink, W., Verhoef, C.: Conservative extension in structural operational semantics. In: *Current Trends in Theor. Comput. Sci.*, pp. 504–524. World Scientific (2001)
2. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: *Handbook of Process Algebra*, pp. 197–292. Elsevier (2001)
3. Baeten, J.C.M., Bergstra, J.A.: Process Algebra with a Zero Object. In: Baeten, J.C.M., Klop, J.W. (eds.) *CONCUR 1990. LNCS*, vol. 458, pp. 83–98. Springer, Heidelberg (1990)
4. Baeten, J.C.M., Bergstra, J.A., Smolka, S.A.: Axiomatizing probabilistic processes: ACP with generative probabilities. *Inf. Comput.* 121(2), 234–255 (1995)
5. Baier, C.: On Algorithmics Verification Methods for Probabilistic Systems. Habilitation thesis, University of Mannheim (1999)
6. Bartels, F.: GSOS for probabilistic transition systems. *Electr. Notes Theor. Comput. Sci.* 65(1) (2002)
7. Bartels, F.: On Generalised Coinduction and Probabilistic Specification Formats. PhD thesis, Vrije Universiteit (2004)
8. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. *J. ACM* 42(1), 232–268 (1995)
9. Bol, R., Groote, J.F.: The meaning of negative premises in transition system specifications. *J. ACM* 43(5), 863–914 (1996)
10. D'Argenio, P.R., Verhoef, C.: A general conservative extension theorem in process algebras with inequalities. *Theor. Comput. Sci.* 177(2), 351–380 (1997)
11. Fokkink, W., van Glabbeek, R.J.: Ntyft/ntyxt rules reduce to ntree rules. *Inf. Comput.* 126(1) (1996)
12. Groote, J.F.: Transition system specifications with negative premises. *Theor. Comput. Sci.* 118(2), 263–299 (1993)
13. Groote, J.F., Vaandrager, F.: Structured operational semantics and bisimulation as a congruence. *Inf. Comput.* 100(2), 202–260 (1992)
14. Jonsson, B., Larsen, K.G., Yi, W.: Probabilistic extensions of process algebras. In: *Handbook of Process Algebra*, pp. 685–710. Elsevier (2001)
15. Klin, B.: Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.* 412(38), 5043–5069 (2011)
16. Lanotte, R., Tini, S.: Probabilistic Congruence for Semistochastic Generative Processes. In: Sassone, V. (ed.) *FOSSACS 2005. LNCS*, vol. 3441, pp. 63–78. Springer, Heidelberg (2005)
17. Lanotte, R., Tini, S.: Probabilistic bisimulation as a congruence. *ACM Trans. Comput. Log.* 10(2) (2009)
18. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* 94(1), 1–28 (1991)
19. Milner, R.: *Communication and Concurrency*. Prentice-Hall, Inc. (1989)
20. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. *Theor. Comput. Sci.* 373(3), 238–272 (2007)
21. Plotkin, G.: A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University (1981); reprinted in *J. Log. Algebr. Program.* 60-61, 17–139 (2004)
22. Segala, R.: *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis. MIT (1995)
23. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: *LICS*, pp. 280–291 (1997)
24. van Glabbeek, R.J.: The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.* 60-61, 229–258 (2004)
25. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.* 121(1), 59–80 (1995)

On the Complexity of the Equivalence Problem for Probabilistic Automata^{*}

Stefan Kiefer¹, Andrzej S. Murawski², Joël Ouaknine¹,
Björn Wachter¹, and James Worrell¹

¹ Department of Computer Science, University of Oxford, UK

² Department of Computer Science, University of Leicester, UK

Abstract. Deciding equivalence of probabilistic automata is a key problem for establishing various behavioural and anonymity properties of probabilistic systems. In recent experiments a randomised equivalence test based on polynomial identity testing outperformed deterministic algorithms. In this paper we show that polynomial identity testing yields efficient algorithms for various generalisations of the equivalence problem. First, we provide a randomized **NC** procedure that also outputs a counterexample trace in case of inequivalence. Second, we consider equivalence of probabilistic cost automata. In these automata transitions are labelled with integer costs and each word is associated with a distribution on costs, corresponding to the cumulative costs of the accepting runs on that word. Two automata are equivalent if they induce the same cost distributions on each input word. We show that equivalence can be checked in randomised polynomial time. Finally we show that the equivalence problem for probabilistic visibly pushdown automata is logspace equivalent to the problem of whether a polynomial represented by an arithmetic circuit is identically zero.

1 Introduction

Probabilistic automata were introduced by Michael Rabin [21] as an extension of deterministic finite automata. Nowadays probabilistic automata, together with associated notions of refinement and equivalence, are widely used in automated verification and learning. Two probabilistic automata are said to be equivalent if each word is accepted with the same probability by both automata. Checking two probabilistic automata for equivalence has been shown crucial for efficiently establishing various behavioural and anonymity properties of probabilistic systems, and is the key algorithmic problem underlying the APEX tool [19,17,13].

It was shown by Tzeng [28] that equivalence for probabilistic automata is decidable in polynomial time. By contrast, the natural analog of language inclusion, that one automaton accepts each word with probability at least as great as another automaton, is undecidable [6] even for automata of fixed dimension [4].

^{*} Research supported by EPSRC grant EP/G069158. The first author is supported by a postdoctoral fellowship of the German Academic Exchange Service (DAAD).

It has been pointed out in [8] that the equivalence problem for probabilistic automata can also be solved by reducing it to the minimisation problem for weighted automata and applying an algorithm of Schützenberger [24].

In [13] we suggested a new *randomised* algorithm which is based on *polynomial identity testing*. In our experiments [13] the randomised algorithm compared well with the Schützenberger-Tzeng procedure on a collection of benchmarks. In this paper we further explore the connection between polynomial identity testing and the equivalence problem of probabilistic automata. We show that polynomial identity testing yields efficient algorithms for various generalisations of the equivalence problem.

In Section 3 we give a new randomised **NC** algorithm for deciding equivalence of probabilistic automata. Recall that **NC** is the subclass of **P** containing those problems that can be solved in polylogarithmic parallel time [11] (see also Section 2). Tzeng [29] considers the path equivalence problem for nondeterministic automata which asks, given nondeterministic automata \mathcal{A} and \mathcal{B} , whether each word has the same number of accepting paths in \mathcal{A} as in \mathcal{B} . He gives a deterministic **NC** algorithm for deciding path equivalence which can be straightforwardly adapted to yield an **NC** algorithm for equivalence of probabilistic automata. Our new randomised algorithm has the same parallel time complexity as Tzeng’s algorithm, but it also outputs a word on which the automata differ in case of inequivalence, which Tzeng’s algorithm cannot. Our algorithm is based on the *Isolating Lemma*, which was used in [18] to compute perfect matchings in randomised **NC**. The randomised algorithm in [13], which relies on the Schwartz-Zippel lemma, can also output a counterexample, exploiting the self-reducibility of the equivalence problem—however it does not seem possible to use this algorithm to compute counterexamples in **NC**. Whether there is a deterministic **NC** algorithm that outputs counterexamples in case of inequivalence remains open.

In Section 4 we consider equivalence of probabilistic automata with one or more cost structures. Costs (or rewards, which can be considered as negative costs) are omnipresent in probabilistic modelling for capturing quantitative effects of probabilistic computations, such as consumption of time, (de-)allocation of memory, energy usage, financial gains, etc. We model each cost structure as an integer-valued *counter*, and annotate the transitions with counter changes.

In nondeterministic cost automata [2,15] the cost of a word is the minimum of the costs of all accepting runs on that word. In probabilistic cost automata we instead associate a probability distribution over costs with each input word, representing the probability that a run over that word has a given cost. Whereas equivalence for nondeterministic cost automata is undecidable [2,15], we show that equivalence of probabilistic cost automata is decidable in randomised polynomial time (and in deterministic polynomial time if the number of counters is fixed). Our proof of decidability, and the complexity bounds we obtain, involves a combination of classical techniques of [24,28] with basic ideas from polynomial identity testing.

We present a case study in which costs are used to model the computation time required by an RSA encryption algorithm, and show that the vulnerability of the algorithm to timing attacks depends on the (in-)equivalence of probabilistic cost automata. In [14] two possible defenses against such timing leaks were suggested. We also analyse their effectiveness.

In Section 5 we consider pushdown automata. Probabilistic pushdown automata are a natural model of recursive probabilistic procedures, stochastic grammars and branching processes [10,16]. The equivalence problem of deterministic pushdown automata has been extensively studied [26,27]. We study the equivalence problem for *probabilistic visibly pushdown automata (VPA)* [3]. In a visibly pushdown automaton, whether the stack is popped or pushed is determined by the input symbol being read.

We show that the equivalence problem for probabilistic VPA is logspace equivalent to *Arithmetic Circuit Identity Testing (ACIT)*, which is the problem of determining equivalence of polynomials presented via arithmetic circuits [1]. Several polynomial-time randomized algorithms are known for **ACIT**, but it is a major open problem whether it can be solved in polynomial time by a deterministic algorithm. The inter-reducibility of probabilistic VPA equivalence and **ACIT** is reminiscent of the reduction of the positivity problem for arithmetic circuits to the reachability problem for recursive Markov chains [10]. However in this case the reduction is only in one direction—from circuits to recursive Markov chains.

In the technical development below it is convenient to consider \mathbb{Q} -weighted automata, which generalise probabilistic automata. All our results and examples are stated in terms of \mathbb{Q} -weighted automata. Missing proofs can be found in a technical report [12].

2 Preliminaries

2.1 Complexity Classes

Recall that **NC** is the subclass of **P** comprising those problems considered efficiently parallelisable. **NC** can be defined via *parallel random-access machines (PRAMs)*, which consist of a set of processors communicating through a shared memory. A problem is in **NC** if it can be solved in time $(\log n)^{O(1)}$ (polylogarithmic time) on a PRAM with $n^{O(1)}$ (polynomially many) processors. A more abstract definition of **NC** is as the class of languages which have **L**-uniform Boolean circuits of polylogarithmic depth and polynomial size. More specifically, denote by **NC^k** the class of languages which have circuits of depth $O(\log^k n)$. The complexity class **RNC** consists of those languages with randomized **NC** algorithms. We have the following inclusions none of which is known to be strict:

$$\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2 \subseteq \mathbf{NC} \subseteq \mathbf{RNC} \subseteq \mathbf{P}.$$

Problems in **NC** include directed reachability, computing the rank and determinant of an integer matrix, solving linear systems of equations and the tree-isomorphism problem. Problems that are **P**-hard under logspace reductions

include circuit value and max-flow. Such problems are not in **NC** unless **P** = **NC**. Problems in **RNC** include matching in graphs and max flow in 0/1-valued networks. In both cases these problems have resisted classification as either in **NC** or **P**-hard. See [11] for more details about **NC** and **RNC**.

2.2 Sequence Spaces

In this section we recall some results about spaces of sequences [23].

Given $s > 0$, define the following space of *formal power series*:

$$\ell_1(\mathbb{Z}^s) := \{f : \mathbb{Z}^s \rightarrow \mathbb{R} : \sum_{\mathbf{v} \in \mathbb{Z}^s} |f(\mathbf{v})| < \infty\} .$$

Then $\ell_1(\mathbb{Z}^s)$ is a complete vector space under the norm $\|f\| = \sum_{\mathbf{v} \in \mathbb{Z}^s} |f(\mathbf{v})|$. We can moreover endow $\ell_1(\mathbb{Z}^s)$ with a Banach algebra structure with multiplication

$$(f * g)(\mathbf{v}) := \sum_{\substack{\mathbf{u}, \mathbf{w} \in \mathbb{Z}^s \\ \mathbf{u} + \mathbf{w} = \mathbf{v}}} f(\mathbf{u})g(\mathbf{w}) .$$

Given $n > 0$ we also consider the space $\ell_1(\mathbb{Z}^s)^{n \times n}$ of $n \times n$ matrices with coefficients in $\ell_1(\mathbb{Z}^s)$. This is a complete normed linear space with respect to the infinity matrix norm

$$\|M\| := \max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} \|M_{i,j}\| .$$

If we define matrix multiplication in the standard way, using the algebra structure on $\ell_1(\mathbb{Z}^s)$, then $\|MN\| \leq \|M\|\|N\|$. In particular, if $\|M\| < 1$ then we can define a Kleene-star operation by $M^* := (I - M)^{-1} = \sum_{k=0}^{\infty} M^k$.

3 Weighted Automata

To permit effective representation of automata we assume that all transition probabilities are rational numbers. In our technical development it is convenient to work with *Q-weighted automata* [24], which are a generalisation of Rabin’s probabilistic automata.

A *Q-weighted automaton* $\mathcal{A} = (n, \Sigma, M, \alpha, \eta)$ consists of a positive integer $n \in \mathbb{N}$ representing the number of states, a finite alphabet Σ , a map $M : \Sigma \rightarrow \mathbb{Q}^{n \times n}$ assigning a transition matrix to each alphabet symbol, an initial (row) vector $\alpha \in \mathbb{Q}^n$, and a final (column) vector $\eta \in \mathbb{Q}^n$. We extend M to Σ^* as the matrix product $M(\sigma_1 \dots \sigma_k) := M(\sigma_1) \cdot \dots \cdot M(\sigma_k)$. The automaton \mathcal{A} assigns each word w a *weight* $\mathcal{A}(w) \in \mathbb{Q}$, where $\mathcal{A}(w) := \alpha M(w) \eta$. An automaton \mathcal{A} is said to be *zero* if $\mathcal{A}(w) = 0$ for all $w \in \Sigma^*$. Two automata \mathcal{B}, \mathcal{C} over the same alphabet Σ are said to be *equivalent* if $\mathcal{B}(w) = \mathcal{C}(w)$ for all $w \in \Sigma^*$. In the remainder of this section we present a randomised **NC**² algorithm for deciding equivalence of \mathbb{Q} -weighted automata and, in case of inequivalence, outputting a counterexample.

Given two automata \mathcal{B}, \mathcal{C} that are to be checked for equivalence, one can compute an automaton \mathcal{A} with $\mathcal{A}(w) = \mathcal{B}(w) - \mathcal{C}(w)$ for all $w \in \Sigma^*$. Then \mathcal{A} is zero if and only if \mathcal{B} and \mathcal{C} are equivalent. Given $\mathcal{B} = (n^{(\mathcal{B})}, \Sigma, M^{(\mathcal{B})}, \alpha^{(\mathcal{B})}, \eta^{(\mathcal{B})})$ and $\mathcal{C} = (n^{(\mathcal{C})}, \Sigma, M^{(\mathcal{C})}, \alpha^{(\mathcal{C})}, \eta^{(\mathcal{C})})$, set $\mathcal{A} = (n, \Sigma, M, \alpha, \eta)$ with $n := n^{(\mathcal{B})} + n^{(\mathcal{C})}$ and

$$M(\sigma) := \begin{pmatrix} M^{(\mathcal{B})}(\sigma) & 0 \\ 0 & M^{(\mathcal{C})}(\sigma) \end{pmatrix}, \quad \alpha := (\alpha^{(\mathcal{B})}, -\alpha^{(\mathcal{C})}), \quad \eta := \begin{pmatrix} \eta^{(\mathcal{B})} \\ \eta^{(\mathcal{C})} \end{pmatrix}.$$

This reduction allows us to focus on *zeroness*, i.e., the problem of determining whether a given \mathbb{Q} -weighted automaton $\mathcal{A} = (n, \Sigma, M, \alpha, \eta)$ is zero. (Since transition weights can be negative, zeroness is not the same as emptiness of the underlying unweighted automaton.) Note that a witness word $w \in \Sigma^*$ against zeroness of \mathcal{A} is also a witness against the equivalence of \mathcal{B} and \mathcal{C} . The following result from [28] is crucial.

Proposition 1. *If \mathcal{A} is not equal to the zero automaton then there exists a word $u \in \Sigma^*$ of length at most $n - 1$ such that $\mathcal{A}(u) \neq 0$.*

Our randomised \mathbf{NC}^2 procedure uses the Isolating Lemma of Mulmuley, Vazirani and Vazirani [18]. We use this lemma in a very similar way to [18], who are concerned with computing maximum matchings in graphs in \mathbf{RNC} .

Lemma 2. *Let \mathcal{F} be a family of subsets of a set $\{x_1, \dots, x_N\}$. Suppose that each element x_i is assigned a weight w_i chosen independently and uniformly at random from $\{1, \dots, 2N\}$. Define the weight of $S \in \mathcal{F}$ to be $\sum_{x_i \in S} w_i$. Then the probability that there is a unique minimum weight set in \mathcal{F} is at least $1/2$.*

We will apply the Isolating Lemma in conjunction with Proposition 1 to decide zeroness of a weighted automaton \mathcal{A} . Suppose \mathcal{A} has n states and alphabet Σ . Given $\sigma \in \Sigma$ and $1 \leq i \leq n$, choose a weight $w_{i,\sigma}$ independently and uniformly at random from the set $\{1, \dots, 2|\Sigma|n\}$. Define the weight of a word $u = \sigma_1 \dots \sigma_k$, $k \leq n$, to be $\text{wt}(u) := \sum_{i=1}^k w_{i,\sigma_i}$. (The reader should not confuse this with the weight $\mathcal{A}(u)$ assigned to u by the automaton \mathcal{A} .) Then we obtain a univariate polynomial P from automaton \mathcal{A} as follows:

$$P(x) = \sum_{k=0}^n \sum_{u \in \Sigma^k} \mathcal{A}(u) x^{\text{wt}(u)}.$$

If \mathcal{A} is equivalent to the zero automaton then clearly $P \equiv 0$. On the other hand, if \mathcal{A} is non-zero, then by Proposition 1 the set $\mathcal{F} = \{u \in \Sigma^{\leq n} : \mathcal{A}(u) \neq 0\}$ is non-empty. Thus there is a unique minimum-weight word $u \in \mathcal{F}$ with probability at least $1/2$ by the Isolating Lemma. In this case P contains the monomial $x^{\text{wt}(u)}$ with coefficient $\mathcal{A}(u)$ as its smallest-degree monomial. Thus $P \not\equiv 0$ with probability at least $1/2$.

It remains to observe that from the formula

$$P(x) = \alpha \left(\sum_{i=0}^n \prod_{j=1}^i \sum_{\sigma \in \Sigma} M(\sigma) x^{w_{j,\sigma}} \right) \eta$$

and the fact that iterated products of matrices of univariate polynomials can be computed in \mathbf{NC}^2 [7] we obtain an **RNC** algorithm for determining zeroness of weighted automata.

It is straightforward to extend the above algorithm to obtain an **RNC** procedure that not only decides zeroness of \mathcal{A} but also outputs a word u such that $\mathcal{A}(u) \neq 0$ in case \mathcal{A} is non-zero. Assume that \mathcal{A} is non-zero and that the random choice of weights has isolated a unique minimum-weight word $u = \sigma_1 \dots \sigma_k$ such that $\mathcal{A}(u) \neq 0$. To determine whether $\sigma \in \Sigma$ is the i -th letter of u we can increase the weight $w_{i,\sigma}$ by 1 while leaving all other weights unchanged and recompute the polynomial $P(x)$. Then σ is the i -th letter in u if and only if the minimum-degree monomial in P changes. All of these tests can be done independently, yielding an **RNC** procedure.

Theorem 3. *Given two weighted automata \mathcal{A} and \mathcal{B} , there is an **RNC** procedure that determines whether or not \mathcal{A} and \mathcal{B} are equivalent and that outputs a word w with $\mathcal{A}(w) \neq \mathcal{B}(w)$ in case \mathcal{A} and \mathcal{B} are inequivalent.*

4 Weighted Cost Automata

In this section we consider weighted automata with costs. Each transition has a cost, and the cumulative cost of a run is recorded in a tuple of counters. Transitions can also have negative costs, which can be considered as rewards. Note though that the counters do not affect the control flow of the automata. In Example 9 we use costs to record the passage of time in an encryption protocol. We explicitly include ε -transitions in our automata because they are convenient for applications (cf. Example 8) and we cannot rely on existing ε -elimination results in the presence of costs.

Let Σ be a finite alphabet not containing the symbol ε . A \mathbb{Q} -weighted cost automaton is a tuple $\mathcal{A} = (n, s, \Sigma, M, \alpha, \eta)$, where $n \in \mathbb{N}$ is the number of states; $s \in \mathbb{N}$ is the number of counters; $M : \Sigma \cup \{\varepsilon\} \rightarrow (C \rightarrow \mathbb{Q})^{n \times n}$ is the transition function, where $C = \{-1, 0, 1\}^s$ is the set of elementary cost vectors; $\alpha \in \mathbb{Q}^n$ is an initial (row) vector; $\eta \in \mathbb{Q}^n$ is a final (column) vector. In this definition, $M(\sigma)_{i,j}(\mathbf{v})$ represents the weight of a σ -transition from state i to j with cost vector $\mathbf{v} \in C$. For the semantics to be well-defined we assume that the total weight of all outgoing ε -labelled transitions from any given state is strictly less than 1.

In order to define the semantics of weighted cost automata it is convenient to use results on matrices of formal power series from Section 2. We can regard $M(\sigma)$ as an $n \times n$ matrix whose entries are elements of the space $\ell_1(\mathbb{Z}^s)$ of formal power series, where $M(\sigma)_{i,j}(\mathbf{v}) = 0$ for $\mathbf{v} \in \mathbb{Z}^s \setminus C$. Our convention on the total weight of ε -transitions is equivalent to the requirement that $\|M(\varepsilon)\| < 1$. We next extend M to a map $M : \Sigma^* \rightarrow (\ell_1(\mathbb{Z}^s))^{n \times n}$ such that, given a word $w \in \Sigma^*$ and states i, j , $M(w)_{i,j}(\mathbf{v})$ is the total weight of all w -labelled paths from state i to state j with accumulated cost $\mathbf{v} \in \mathbb{Z}^s$. Given a word $w = \sigma_1 \sigma_2 \dots \sigma_m \in \Sigma^*$, we define

$$M(w) := M(\varepsilon)^* M(\sigma_1) M(\varepsilon)^* \dots M(\sigma_m) M(\varepsilon)^* . \tag{1}$$

Finally, given $w \in \Sigma^*$ we define $\mathcal{A}(w) := \alpha M(w) \eta$. Then $\mathcal{A}(w)$ is an element of $\ell_1(\mathbb{Z}^s)$ such that $\mathcal{A}(w)(\mathbf{v})$ gives the total weight of all accepting runs with accumulated cost $\mathbf{v} \in \mathbb{Z}^s$.

Let $\mathbf{x} = (x_1, \dots, x_s)$ be a vector of variables, one for each counter. Our equivalence algorithm is based on a representation of $\mathcal{A}(w)$ as a rational function in \mathbf{x} , following classical ideas [20]. Given $\mathbf{v} \in \mathbb{Z}^s$ we denote by $\mathbf{x}^{\mathbf{v}}$ the monomial $x_1^{v_1} \cdots x_s^{v_s}$. (Note that we allow negative powers in monomials.) We say that $f \in \ell_1(\mathbb{Z}^s)$ has *finite support* if $f(\mathbf{v}) = 0$ for all but finitely many $\mathbf{v} \in \mathbb{Z}^s$. We identify such an f with the polynomial $\sum_{\mathbf{v} \in \mathbb{Z}^s} f(\mathbf{v}) \mathbf{x}^{\mathbf{v}}$. We furthermore say that $f \in \ell_1(\mathbb{Z}^s)$ is *rational* if there exist $g, h : \mathbb{Z}^s \rightarrow \mathbb{Q}$ with finite support such that $f * h = g$. We then identify f with the rational function

$$\sum_{\mathbf{v} \in \mathbb{Z}^s} g(\mathbf{v}) \mathbf{x}^{\mathbf{v}} / \sum_{\mathbf{v} \in \mathbb{Z}^s} h(\mathbf{v}) \mathbf{x}^{\mathbf{v}}.$$

Note that we can clear all negative exponents from the numerator and denominator of such an expression. Note also that sums and products of rational functions correspond to sums and products in $\ell_1(\mathbb{Z}^s)$ in the above representation.

Proposition 4. $M(w)$ can be represented as a matrix of rational functions in \mathbf{x} such that the numerator and denominator in each matrix entry have degrees at most $2n(s + 1) \cdot |w|$.

Proof. From equation (II) it suffices to show that $M(\varepsilon)^*$ can be represented as a matrix of rational functions with appropriate degree bounds. Recall that $M(\varepsilon)^* = (I - M(\varepsilon))^{-1}$, so it suffices to show that $I - M(\varepsilon)$ (considered as a matrix of polynomials) has an inverse that can be represented as a matrix of rational functions. But the determinant formula yields that $\det(I - M(\varepsilon))$ is a (non-zero) polynomial in \mathbf{x} , thus the cofactor formula for inverting matrices yields a representation of $(I - M(\varepsilon))^{-1}$ as a matrix of rational functions in \mathbf{x} of degree at most $2ns$. □

An automaton \mathcal{A} is said to be *zero* if $\mathcal{A}(w) \equiv 0$ for all $w \in \Sigma^*$. Two automata \mathcal{B}, \mathcal{C} over the same alphabet Σ with the same number of counters are said to be *equivalent* if $\mathcal{B}(w) \equiv \mathcal{C}(w)$ for all $w \in \Sigma^*$. As in Section 3, the equivalence problem can be reduced to the zeroness problem, so we focus on the latter.

The following proposition states that if there is a word witnessing that \mathcal{A} is non-zero, then there is a “short” such word.

Proposition 5. \mathcal{A} is zero if and only if $\mathcal{A}(w) \equiv 0$ for all $w \in \Sigma^*$ of length at most $n - 1$.

The proof, given in full in [12], is similar to the linear algebra arguments from [24,28], but involves an additional twist. The key idea is to substitute concrete values for the variables \mathbf{x} , thereby transforming from the setting of infinite-dimensional vector spaces of rational functions in \mathbf{x} to a finite dimensional setting where the arguments of [24,28] apply.

The decidability of zeroness (and hence equivalence) for weighted cost automata follows immediately from Proposition 5. However, using polynomial identity testing, we arrive at the following theorem.

Theorem 6. *The equivalence problem for weighted cost automata is decidable in randomised polynomial time.*

Proof. We have already observed that the equivalence problem can be reduced to the zeroness problem. We now reduce the zeroness problem to polynomial identity testing.

Given an automaton $\mathcal{A} = (n, s, \Sigma, M, \alpha, \eta)$, for each word $w \in \Sigma^*$ of length at most n we have a rational expression $\mathcal{A}(w)$ in variables $\mathbf{x} = (x_1, \dots, x_s)$ which has degree at most $d := 2n(s + 1) \cdot n$ by Proposition 4.

Now consider the set $R := \{1, 2, \dots, 2d\}$. Suppose that we pick $\mathbf{r} \in R^s$ uniformly at random. Denote by $\mathcal{A}(w)(\mathbf{r})$ the result of substituting \mathbf{r} for \mathbf{x} in the rational expression $\mathcal{A}(w)$. Clearly if \mathcal{A} is a zero automaton then $\mathcal{A}(w)(\mathbf{r}) = 0$ for all \mathbf{r} . On the other hand, if \mathcal{A} is non-zero then by Proposition 5 there exists a word $w \in \Sigma^*$ of length at most n such that $\mathcal{A}(w) \neq 0$. Since the degree of the rational expression $\mathcal{A}(w)$ is at most d it follows from the Schwartz-Zippel theorem [9,25,30] that the probability that $\mathcal{A}(w)(\mathbf{r}) = 0$ is at most $1/2$.

Thus our randomised procedure is to pick $\mathbf{r} \in R^s$ uniformly at random and to check whether $\mathcal{A}(w)(\mathbf{r}) = 0$ for some $w \in \Sigma^*$. It remains to show how we can do this check in polynomial time. To achieve this we show that there is a \mathbb{Q} -weighted automaton \mathcal{B} with no counters such that $\mathcal{A}(w)(\mathbf{r}) = \mathcal{B}(w)$ for all $w \in \Sigma^*$, since we can then check \mathcal{B} for zeroness using, e.g., Tzeng’s algorithm [28]. The automaton \mathcal{B} has the form $\mathcal{B} = (n^{(\mathcal{B})}, \Sigma, M^{(\mathcal{B})}, \alpha^{(\mathcal{B})}, \eta^{(\mathcal{B})})$, where $n^{(\mathcal{B})} = n$, $\alpha^{(\mathcal{B})} = \alpha$, $\eta^{(\mathcal{B})} = \eta$ and $M^{(\mathcal{B})}(\sigma) = \sum_{\mathbf{v} \in \mathbb{Z}^s} M(\sigma)(\mathbf{v})\mathbf{r}^{\mathbf{v}}$ for all $\sigma \in \Sigma$. \square

Corollary 7. *For each fixed number of counters the equivalence problem for weighted cost automata is decidable in deterministic polynomial time.*

See [12] for a proof.

Example 8. We consider probabilistic programs that randomly increase and decrease a single counter (initialised with 0) so that upon termination the counter has a random value $X \in \mathbb{Z}$. The programs should be such that X is a random variable with $X = Y - Z$ where Y and Z are independent random variables with a geometric distribution with parameters $p = 1/2$ and $p = 1/3$, respectively. (By that we mean that $\Pr(Y = k) = (1 - p)^k p$ for $k \in \{0, 1, \dots\}$, and similarly for Z .) Figure 1 shows code in the syntax of the APEX tool.

The program on the left consecutively runs two while loops: it first increments the counter according to a geometric distribution with parameter $1/2$ and then decrements the counter according to a geometric distribution with parameter $1/3$, so that the final counter value is distributed as desired. The program on the right is more efficient in that it runs only one of two while loops, depending on a single coin flip at the beginning. It may not be obvious though that the final counter value follows the same distribution as in the left program. We used


```

inc:com, dec:com |-                               inc:com, dec:com |-
  var%2 flip;                                     var%2 flip;
  flip := 0;                                       flip := coin[0:1/2,1:1/2];
  while (flip = 0) do {                             if (flip = 0) then {
    flip := coin[0:1/2,1:1/2];                       while (flip = 0) do {
    if (flip = 0) then {                               flip := coin[0:1/2,1:1/2];
      inc;                                             if (flip = 0) then {
    };                                               inc;
  };                                               };
  flip := 0;                                       }; else {
  while (flip = 0) do {                               flip := 0;
    flip := coin[0:2/3,1:1/3];                       while (flip = 0) do {
    if (flip = 0) then {                               dec;
      dec;                                             flip := coin[0:2/3,1:1/3];
    };                                               };
  }
:com                                               }
                                               :com

```

Fig. 1. Two APEX programs for producing a counter that is distributed as the difference between two geometrically distributed random variables

the APEX tool to translate the programs to the probabilistic cost automata \mathcal{B} and \mathcal{C} shown in Figure 2.

Since the input alphabets are empty, it suffices to consider the input word ε when comparing \mathcal{B} and \mathcal{C} for equivalence. If we construct the difference automaton $\mathcal{A} = (5, 1, \emptyset, M, \alpha, \eta)$ and invert the matrix of polynomials $I - M(\varepsilon)$, we obtain

$$\mathcal{A}(\varepsilon)(x) = \left(\frac{2}{x-2}, \frac{2}{(3x-2)(x-2)}, 1, \frac{-x}{2(x-2)}, \frac{3}{2(3x-2)} \right) \eta \equiv 0,$$

which proves equivalence of \mathcal{B} and \mathcal{C} . Notice that the actual algorithm would not compute $\mathcal{A}(\varepsilon)(x)$ as a polynomial, but it would compute $\mathcal{A}(\varepsilon)(r)$ only for a few concrete values $r \in \mathbb{Q}$. □

Example 9. RSA [22] is a widely-used cryptographic algorithm. Popular implementations of the RSA algorithm have been shown to be vulnerable to timing attacks that reveal private keys [14,5]. The preferred countermeasures are blinding techniques that randomise certain aspects of the computation, which are described in, e.g., [14]. We model the timing behaviour of the RSA algorithm using probabilistic cost automata, where costs encode time. These automata are produced by APEX, and are then used to check for timing leaks with and without blinding.

At the heart of RSA decryption is a modular exponentiation, which computes the value $m^d \bmod N$ where $m \in \{0, \dots, N-1\}$ is the encrypted message, $d \in \mathbb{N}$ is the private decryption exponent and $N \in \mathbb{N}$ is a modulus. An attacker wants to find out d . We model RSA decryption in APEX by implementing modular

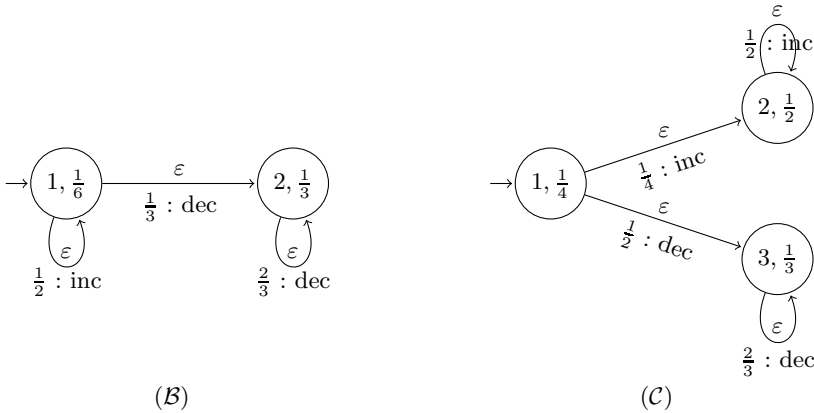


Fig. 2. Automata produced from the code in Figure 1. The states are labelled with their number and their “acceptance probability” (η -weight). In both automata, state 1 is the only initial state ($\alpha_1 = 1$ and $\alpha_i = 0$ for $i \neq 1$). The transitions are labelled with the input symbol ε , with a probability (weight) and a counter action (i.e. cost).

exponentiation by iterative squaring (see Figure 3). We consider the situation where the attacker is able to control the message m , and tries to derive d by observing the runtime distribution over different messages m . Following [14] we assume that the running time of multiplication depends on the operand values (because a source-level multiplication typically corresponds to a cascade of processor-level multiplications). By choosing the ‘right’ input message m , an attacker can observe which private keys are most likely.

We consider two blinding techniques mentioned in Kocher [14]. The first one is base blinding, i.e., the message is multiplied by r^d before exponentiation where d is a random number, which gives a result that can be fixed by dividing by r but makes it impossible for the attacker to control the basis of the exponentiation. The second one is exponent blinding, which adds a multiple of the group order $\varphi(N)$ of $\mathbb{Z}/N\mathbb{Z}$ to the exponent, which doesn’t change the result of the exponentiation¹ but changes the timing behaviour.

Figure 4 shows the automaton for $N = 10$, and private key $0, 1, 0, 1$ with message blinding enabled. The APEX program is given in Figure 3.

We investigate the effectiveness of blinding. Two private keys are indistinguishable if the resulting automata are equivalent. The more keys are indistinguishable the safer the algorithm. We analyse which private keys are identified by plain RSA, RSA with a blinded message and RSA with blinded exponent.

For example, in plain RSA, the following keys $0, 1, 0, 1$ and $1, 0, 0, 1$ are indistinguishable, keys $0, 1, 1, 0$ and $0, 0, 1, 1$ are indistinguishable with base blinding, lastly $1, 0, 0, 1$ and $1, 0, 1, 1$ are equivalent only with exponent blinding. Overall 9 different keys are distinguishable with plain RSA, 7 classes with base blinding and 4 classes with exponent blinding.

¹ Euler’s totient function φ satisfies $a^{\varphi(N)} \equiv 1 \pmod N$ for all $a \in \mathbb{Z}$.

```

const N := 10;    // modulus
const Bits := 4 ; // number of bits of the key

m :int%N, inc:com |-
var%2 exponent[Bits] = [0,1,0,1];
com power(x:int%N) {
  var%N s := 1;
  var%N R;
  for(var%(Bits + 1) k; k < Bits; ++k) do {
    R:=s;
    if(exponent[k]) then {
      R := R*x;
      if(5<=R) then { inc; inc } else { inc }
    }
    s := R*R;
  }
}
var%N message := m*rand[N]; // blinding
power(message) : com

```

Fig. 3. APEX code for RSA

5 Pushdown Automata and Arithmetic Circuits

In a visibly pushdown automaton [3] the stack operations are determined by the input word. Consequently VPA have a more tractable language theory than ordinary pushdown automata. The main result of this section shows that the equivalence problem for weighted VPA is logspace equivalent to the problem **ACIT** of determining whether a polynomial represented by an arithmetic circuit is identically zero.

A *visibly pushdown alphabet* $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$ consists of a finite set of *calls* Σ_c , a finite set of *returns* Σ_r , and a finite set of *internal actions* Σ_{int} . A visibly pushdown automaton over alphabet Σ is restricted so that it pushes onto the stack when it reads a call, pops the stack when it reads a return, and leaves the stack untouched when reading internal actions. Due to this restriction visibly pushdown automata only accept words in which calls and returns are appropriately matched. Define the set of *well-matched words* to be $\bigcup_{i \in \mathbb{N}} L_i$, where $L_0 = \Sigma_{int} + \{\varepsilon\}$ and $L_{i+1} = \Sigma_c L_i \Sigma_r + L_i L_i$.

A \mathbb{Q} -*weighted visibly pushdown automaton* on alphabet Σ is a tuple $\mathcal{A} = (n, \alpha, \eta, \Gamma, M)$, where n is the number of *states*, α is an n -dimensional *initial* (row) vector, η is an n -dimensional *final* (column) vector, Γ is a finite *stack alphabet*, and $M = (M_c, M_r, M_{int})$ is a tuple of *matrix-valued transition functions* with types $M_c : \Sigma_c \times \Gamma \rightarrow \mathbb{Q}^{n \times n}$, $M_r : \Sigma_r \times \Gamma \rightarrow \mathbb{Q}^{n \times n}$ and $M_{int} : \Sigma_{int} \rightarrow \mathbb{Q}^{n \times n}$. If $a \in \Sigma_c$ and $\gamma \in \Gamma$ then $M_c(a, \gamma)_{i,j}$ gives the weight of an a -labelled transition from state i to state j that pushes γ on the stack. If $a \in \Sigma_r$ and $\gamma \in \Gamma$ then $M_r(a, \gamma)_{i,j}$ gives the weight of an a -labelled transition from state i to j that pops γ from the stack.

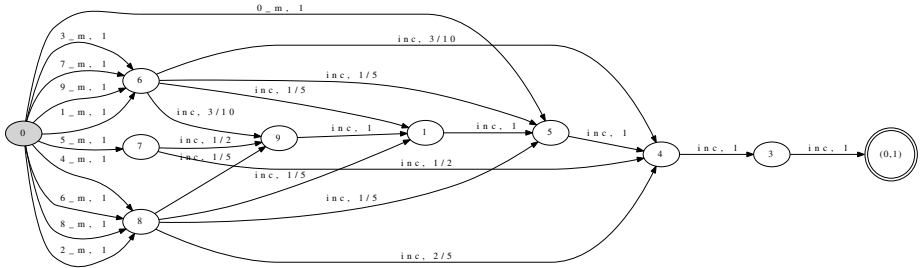


Fig. 4. Modeling RSA decryption with APEX

For each well-matched word $u \in \Sigma^*$ we define an $n \times n$ rational matrix $M^{(\mathcal{A})}(u)$ whose (i, j) -th entry denotes the total weight of all paths from state i to state j along input u . The definition of $M^{(\mathcal{A})}(u)$ follows the inductive definition of well-matched words. The base cases are $M^{(\mathcal{A})}(\varepsilon) = I$ and $M^{(\mathcal{A})}(a)_{i,j} = M_{int}(a)_{i,j}$. The inductive cases are

$$M^{(\mathcal{A})}(uv) = M^{(\mathcal{A})}(u) \cdot M^{(\mathcal{A})}(v)$$

$$M^{(\mathcal{A})}(aub) = \sum_{\gamma \in \Gamma} M_c(a, \gamma) \cdot M^{(\mathcal{A})}(u) \cdot M_r(b, \gamma),$$

for $a \in \Sigma_c, b \in \Sigma_r$.

The weight assigned by \mathcal{A} to a well-matched word w is defined to be $\mathcal{A}(w) := \alpha M^{(\mathcal{A})}(w) \eta$. We say that two weighted VPA \mathcal{A} and \mathcal{B} are *equivalent* if for each well-matched word w we have $\mathcal{A}(w) = \mathcal{B}(w)$.

An *arithmetic circuit* is a finite directed acyclic multigraph whose vertices, called *gates*, have indegree 0 or 2. Vertices of indegree 0 are called *input gates* and are labelled with a constant 0 or 1, or a variable from the set $\{x_i : i \in \mathbb{N}\}$. Vertices of indegree 2 are called *internal gates* and are labelled with one of the arithmetic operations $+, *$ or $-$. We assume that there is a unique gate with outdegree 0 called the *output*. Note that C is a multigraph, so there can be two edges between a pair of gates, i.e., both inputs to a given gate can lead from the same source. We call a circuit *variable-free* if all inputs gates are labelled 0 or 1.

The *Arithmetic Circuit Identity Testing (ACIT)* problem asks whether the output of a given circuit is equal to the zero polynomial. **ACIT** is known to be in **coRP** but it remains open whether there is a polynomial or even sub-exponential algorithm for this problem [1]. Utilising the fact that a variable-free arithmetic circuit of size $O(n)$ can compute 2^{2^n} , Allender *et al.* [1] give a logspace reduction of the general **ACIT** problem to the special case of variable-free circuits. Henceforth we assume without loss of generality that all circuits are variable-free. Furthermore we recall that **ACIT** can be reformulated as the problem of deciding whether two variable-free circuits using only the arithmetic operations $+$ and $*$ compute the same number [1].

The proof of the following proposition is given in [12].

Proposition 10. *ACIT is logspace reducible to the equivalence problem for weighted visibly pushdown automata.*

In the remainder of this section we give a converse reduction: from equivalence of weighted VPA to **ACIT**. The following result gives a decision procedure for the equivalence of two weighted VPA \mathcal{A} and \mathcal{B} .

Proposition 11. *\mathcal{A} is equivalent to \mathcal{B} if and only if $\mathcal{A}(w) = \mathcal{B}(w)$ for all words $w \in L_{n^2}$, where n is the sum of the number of states of \mathcal{A} and the number of states of \mathcal{B} .*

Proof. Recall that for each balanced word $u \in \Sigma^*$ we have rational matrices $M^{(\mathcal{A})}(u)$ and $M^{(\mathcal{B})}(u)$ giving the respective state-to-state transition weights of \mathcal{A} and \mathcal{B} on reading u . These two families of matrices can be combined into a single family

$$\mathcal{M} = \left\{ \begin{pmatrix} M^{(\mathcal{A})}(u) & \mathbf{0} \\ \mathbf{0} & M^{(\mathcal{B})}(u) \end{pmatrix} : u \text{ well-matched} \right\}$$

of $n \times n$ matrices. Let us also write \mathcal{M}_i for the subset of \mathcal{M} generated by those well-matched words $u \in L_i$.

Let $\alpha^{(\mathcal{A})}, \eta^{(\mathcal{A})}$ and $\alpha^{(\mathcal{B})}, \eta^{(\mathcal{B})}$ be the respective initial and final-state vectors of \mathcal{A} and \mathcal{B} . Then \mathcal{A} is equivalent to \mathcal{B} if and only if

$$(\alpha^{(\mathcal{A})} \ \alpha^{(\mathcal{B})})M \begin{pmatrix} \eta^{(\mathcal{A})} \\ -\eta^{(\mathcal{B})} \end{pmatrix} = 0 \tag{2}$$

for all $M \in \mathcal{M}$. It follows that \mathcal{A} is equivalent to \mathcal{B} if and only if (2) holds for all M in $\text{span}(\mathcal{M})$, where the span is taken in the rational vector space of $n \times n$ rational matrices. But $\text{span}(\mathcal{M}_i)$ is an ascending sequence of vector spaces:

$$\text{Span}(\mathcal{M}_0) \subseteq \text{Span}(\mathcal{M}_1) \subseteq \text{Span}(\mathcal{M}_2) \subseteq \dots$$

It follows from a dimension argument that this sequence stops in at most n^2 steps and we conclude that $\text{span}(\mathcal{M}) = \text{span}(\mathcal{M}_{n^2})$. □

Proposition 12. *Given a weighted visibly pushdown automaton \mathcal{A} and $n \in \mathbb{N}$ one can compute in logarithmic space a circuit that represents $\sum_{w \in L_{n^2}} \mathcal{A}(w)$.*

Proof. From the definition of the language L_i and the family of matrices $M^{(\mathcal{A})}$ we have:

$$\begin{aligned} \sum_{w \in L_{i+1}} M^{(\mathcal{A})}(w) &= \sum_{a \in \Sigma_c} \sum_{b \in \Sigma_r} \sum_{\gamma \in \Gamma} M^{(\mathcal{A})}(a, \gamma) \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right) M^{(\mathcal{A})}(b, \gamma) \\ &\quad + \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right) \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right). \end{aligned}$$

The above equation implies that we can compute in logarithmic space a circuit that represents $\sum_{w \in L_n} M^{(\mathcal{A})}(w)$. The result of the proposition immediately follows by premultiplying by the initial state vector and postmultiplying by the final state vector. □

A key property of weighted VPA is their closure under product.

Proposition 13. *Given weighted VPA \mathcal{A} and \mathcal{B} on the same alphabet Σ one can define a synchronous-product automaton, denoted $\mathcal{A} \times \mathcal{B}$, such that $(\mathcal{A} \times \mathcal{B})(w) = \mathcal{A}(w)\mathcal{B}(w)$ for all $w \in \Sigma^*$.*

The proof of Proposition 13, given in 12, exploits the fact that the stack height is determined by the input word, so the respective stacks of \mathcal{A} and \mathcal{B} operating in parallel can be simulated in a single stack.

Proposition 14. *The equivalence problem for weighted visibly pushdown automata is logspace reducible to ACIT.*

Proof. Let \mathcal{A} and \mathcal{B} be weighted visibly pushdown automata with a total of n states between them. Then

$$\begin{aligned} \sum_{w \in L_n} (\mathcal{A}(w) - \mathcal{B}(w))^2 &= \sum_{w \in L_n} \mathcal{A}(w)^2 + \mathcal{B}(w)^2 - 2\mathcal{A}(w)\mathcal{B}(w) \\ &= \sum_{w \in L_n} (\mathcal{A} \times \mathcal{A})(w) + (\mathcal{B} \times \mathcal{B})(w) - 2(\mathcal{A} \times \mathcal{B})(w) \end{aligned}$$

Thus \mathcal{A} is equivalent to \mathcal{B} iff $\sum_{w \in L_n} (\mathcal{A} \times \mathcal{A})(w) + (\mathcal{B} \times \mathcal{B})(w) = 2\sum_{w \in L_n} (\mathcal{A} \times \mathcal{B})(w)$. But Propositions 12 and 13 allow us to translate the above equation into an instance of ACIT. \square

The trick of considering sums-of-squares of acceptance weights in the above proof is inspired by 29, Lemma 1].

References

1. Allender, E.E., Bürgisser, P., Kjeldgaard-Pedersen, J., Bro Miltersen, P.: On the complexity of numerical analysis. *SIAM J. Comput.* 38(5), 1987–2006 (2009)
2. Almagor, S., Boker, U., Kupferman, O.: What's Decidable about Weighted Automata? In: Bultan, T., Hsiung, P.-A. (eds.) *ATVA 2011*. LNCS, vol. 6996, pp. 482–491. Springer, Heidelberg (2011)
3. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: *Proc. 36th Annual ACM Symposium on Theory of Computing, STOC*, pp. 202–211. ACM (2004)
4. Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. *Theoretical Computer Science* 36(3), 231–245 (2003)
5. Brumley, D., Boneh, D.: Remote timing attacks are practical. *Computer Networks* 48(5), 701–716 (2005)
6. Condon, A., Lipton, R.: On the complexity of space bounded interactive proofs (extended abstract). In: *Proceedings of FOCS*, pp. 462–467 (1989)
7. Cook, S.A.: A taxonomy of problems with fast parallel algorithms. *Information and Control* 64(1-3), 2–22 (1985)
8. Cortes, C., Mohri, M., Rastogi, A.: On the Computation of Some Standard Distances between Probabilistic Automata. In: Ibarra, O.H., Yen, H.-C. (eds.) *CIAA 2006*. LNCS, vol. 4094, pp. 137–149. Springer, Heidelberg (2006)

9. DeMillo, R., Lipton, R.: A probabilistic remark on algebraic program testing. *Inf. Process. Lett.* 7(4), 193–195 (1978)
10. Etesami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM* 56(1), 1:1–1:66 (2009)
11. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: *Limits to parallel computation: P-completeness theory*. Oxford University Press (1995)
12. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: On the complexity of the equivalence problem for probabilistic automata. Technical report, arxiv.org (2012), <http://arxiv.org/abs/1112.4644>
13. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: Language Equivalence for Probabilistic Automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 526–540. Springer, Heidelberg (2011)
14. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Kobitz, N. (ed.) *CRYPTO 1996*. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
15. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. Journal of Alg. and Comp.* 4(3), 232–249 (1994)
16. Kučera, A., Esparza, J., Mayr, R.: Model checking probabilistic pushdown automata. *Logical Methods in Computer Science* 2(1), 1–31 (2006)
17. Legay, A., Murawski, A.S., Ouaknine, J., Worrell, J.: On Automated Verification of Probabilistic Programs. In: Ramakrishnan, C.R., Rehof, J. (eds.) *TACAS 2008*. LNCS, vol. 4963, pp. 173–187. Springer, Heidelberg (2008)
18. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. In: *STOC*, pp. 345–354 (1987)
19. Murawski, A.S., Ouaknine, J.: On Probabilistic Program Equivalence and Refinement. In: Abadi, M., de Alfaro, L. (eds.) *CONCUR 2005*. LNCS, vol. 3653, pp. 156–170. Springer, Heidelberg (2005)
20. Niven, I.: Formal power series. *American Mathematical Monthly* 76(8), 871–889 (1969)
21. Rabin, M.O.: Probabilistic automata. *Inf. and Control* 6(3), 230–245 (1963)
22. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21, 120–126 (1978)
23. Rudin, W.: *Functional analysis*, 2nd edn. International Series in Pure and Applied Mathematics. McGraw-Hill Inc., New York (1991)
24. Schützenberger, M.-P.: On the definition of a family of automata. *Inf. and Control* 4, 245–270 (1961)
25. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* 27(4), 701–717 (1980)
26. Sénizergues, G.: The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) *ICALP 1997*. LNCS, vol. 1256, pp. 671–681. Springer, Heidelberg (1997)
27. Stirling, C.: Deciding DPDA Equivalence Is Primitive Recursive. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) *ICALP 2002*. LNCS, vol. 2380, pp. 821–832. Springer, Heidelberg (2002)
28. Tzeng, W.: A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing* 21(2), 216–227 (1992)
29. Tzeng, W.: On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.* 58(1), 43–46 (1996)
30. Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. In: Ng, K.W. (ed.) *EUROSAM 1979 and ISSAC 1979*. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979)

Author Index

- Adámek, Jiří 58, 89
Ahman, Danel 74
Atkey, Robert 42
- Bernardo, Marco 195
Biernacki, Dariusz 119
Bollig, Benedikt 391
Bonchi, Filippo 58
Bouyer, Patricia 301
Bove, Ana 104
Brenquier, Romain 301
Breuers, Stefan 150
Brihayé, Thomas 286
Bruyère, Véronique 286
- Caires, Luis 346
Calli, Andrea 1
Chapman, James 74
Chatterjee, Krishnendu 270
Chen, Di 437
Crafa, Silvia 225
Cyriac, Aiswarya 391
- D'Argenio, Pedro Rubén 452
De Nicola, Rocco 195
De Pril, Julie 286
Dybjær, Peter 104
- Gastin, Paul 391
Ghani, Neil 42
Gimbert, Hugo 286
Göller, Stefan 406
Goltz, Ursula 331
Gorin, Daniel 240
Gottlob, Georg 1
- Haase, Christoph 406
Hülsbusch, Mathias 58, 361
- Jacobs, Bart 42
Jagadeesan, Radha 180
Johann, Patricia 42
- Kartzow, Alexander 376
Kiefer, Stefan 467
- König, Barbara 58, 361
Kurz, Alexander 255
- Lee, Matias David 452
Lenglet, Sergueï 119
Löding, Christof 150
Loreti, Michele 195
- Markey, Nicolas 301
Mazza, Damiano 316
Milius, Stefan 58, 89
Moss, Lawrence S. 89
Murawski, Andrzej S. 467
- Nain, Sumit 421
Narayan Kumar, K. 391
Nestmann, Uwe 210
- Olschewski, Jörg 150
Orsi, Giorgio 1
Ouaknine, Joël 406, 467
- Peters, Kirstin 210
Petri, Gustavo 180
Pfenning, Frank 346
Pieris, Andreas 1
Preugschat, Sebastian 135
- Reuß, Andreas 165
Riely, James 180
Ross, Neil J. 316
- Schicke-Uffmann, Jens-Wolfhard 331
Schröder, Lutz 240
Seidl, Helmut 165
Sicard-Ramírez, Andrés 104
Silva, Alexandra 58
Sousa, Lurdes 89
Suzuki, Tomoyuki 255
- Toninho, Bernardo 346
Tuosto, Emilio 255
- Ummels, Michael 301
Uustalu, Tarmo 74
- van Breugel, Franck 437
van Glabbeek, Rob 331

Varacca, Daniele 225
Vardi, Moshe Y. 421

Wachter, Björn 467
Wilke, Thomas 135

Winkel, Glynn 26
Worrell, James 406, 437, 467

Yoshida, Nobuko 225