

Anca Muscholl (Ed.)

ARCOSS

LNCS 8412

Foundations of Software Science and Computation Structures

**17th International Conference, FOSSACS 2014
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2014
Grenoble, France, April 5–13, 2014, 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

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*

Deng Xiaotie, *City University of Hong Kong*

Jeannette M. Wing, *Microsoft Research, Redmond, WA, USA*

Anca Muscholl (Ed.)

Foundations of Software Science and Computation Structures

17th International Conference, FOSSACS 2014
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2014
Grenoble, France, April 5-13, 2014
Proceedings



Springer

Volume Editor

Anca Muscholl
LaBRI, University of Bordeaux, France
E-mail: anca@labri.fr

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-54829-1

e-ISBN 978-3-642-54830-7

DOI 10.1007/978-3-642-54830-7

Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014934148

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

© Springer-Verlag Berlin Heidelberg 2014

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, 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.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

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 2014 was the 17th instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference that was established in 1998, and this year consisted of six constituting conferences (CC, ESOP, FASE, FoSSaCS, TACAS, and POST) including eight invited speakers and two tutorial speakers. Before and after the main conference, numerous satellite workshops took place and attracted many researchers from all over the globe.

ETAPS is a confederation of several conferences, each with its own Program Committee (PC) and its own Steering Committee (if any). The conferences cover various aspects of software systems, ranging from theoretical foundations to programming language developments, compiler advancements, analysis tools, formal approaches to software engineering, and security. Organizing these conferences in a coherent, highly synchronized conference program, enables the participation in an exciting event, having the possibility to meet many researchers working in different directions in the field, and to easily attend the talks of different conferences.

The six main conferences together received 606 submissions this year, 155 of which were accepted (including 12 tool demonstration papers), yielding an overall acceptance rate of 25.6%. I thank all authors for their interest in ETAPS, all reviewers for the peer reviewing process, the PC members for their involvement, and in particular the PC co-chairs for running this entire intensive process. Last but not least, my congratulations to all authors of the accepted papers!

ETAPS 2014 was greatly enriched by the invited talks of Geoffrey Smith (Florida International University, USA) and John Launchbury (Galois, USA), both unifying speakers, and the conference-specific invited speakers (CC) Benoît Dupont de Dinechin (Kalray, France), (ESOP) Maurice Herlihy (Brown University, USA), (FASE) Christel Baier (Technical University of Dresden, Germany), (FoSSaCS) Petr Jančar (Technical University of Ostrava, Czech Republic), (POST) David Mazières (Stanford University, USA), and finally (TACAS) Orna Kupferman (Hebrew University Jerusalem, Israel). Invited tutorials were provided by Bernd Finkbeiner (Saarland University, Germany) and Andy Gordon (Microsoft Research, Cambridge, UK). My sincere thanks to all these speakers for their great contributions.

For the first time in its history, ETAPS returned to a city where it had been organized before: Grenoble, France. ETAPS 2014 was organized by the Université Joseph Fourier in cooperation with the following associations and societies: ETAPS e.V., EATCS (European Association for Theoretical Computer Science), EAPLS (European Association for Programming Languages and Systems), and EASST (European Association of Software Science and Technology). It had

support from the following sponsors: CNRS, Inria, Grenoble INP, PERSYVAL-Lab and Université Joseph Fourier, and Springer-Verlag.

The organization team comprised:

General Chair: Saddek Bensalem
 Conferences Chair: Alain Girault and Yassine Lakhnech
 Workshops Chair: Axel Legay
 Publicity Chair: Yliès Falcone
 Treasurer: Nicolas Halbwachs
 Webmaster: Marius Bozga

The overall planning for ETAPS is the responsibility of the Steering Committee (SC). The ETAPS SC consists of an executive board (EB) and representatives of the individual ETAPS conferences, as well as representatives of EATCS, EAPLS, and EASST. The Executive Board comprises Gilles Barthe (satellite events, Madrid), Holger Hermanns (Saarbrücken), Joost-Pieter Katoen (chair, Aachen and Twente), Gerald Lüttgen (treasurer, Bamberg), and Tarmo Uustalu (publicity, Tallinn). Other current SC members are: Martín Abadi (Santa Cruz and Mountain View), Erika Ábráham (Aachen), Roberto Amadio (Paris), Christel Baier (Dresden), Saddek Bensalem (Grenoble), Giuseppe Castagna (Paris), Albert Cohen (Paris), Alexander Egyed (Linz), Riccardo Focardi (Venice), Björn Franke (Edinburgh), Stefania Gnesi (Pisa), Klaus Havelund (Pasadena), Reiko Heckel (Leicester), Paul Klint (Amsterdam), Jens Knoop (Vienna), Steve Kremer (Nancy), Pasquale Malacaria (London), Tiziana Margaria (Potsdam), Fabio Martinelli (Pisa), Andrew Myers (Boston), Anca Muscholl (Bordeaux), Catuscia Palamidessi (Palaiseau), Andrew Pitts (Cambridge), Arend Rensink (Twente), Don Sanella (Edinburgh), Vladimiro Sassone (Southampton), Ina Schäfer (Braunschweig), Zhong Shao (New Haven), Gabriele Taentzer (Marburg), Cesare Tinelli (Iowa), Jan Vitek (West Lafayette), and Lenore Zuck (Chicago).

I sincerely thank all ETAPS SC members for all their hard work in making the 17th ETAPS a success. Moreover, thanks to all speakers, attendants, organizers of the satellite workshops, and Springer for their support. Finally, many thanks to Saddek Bensalem and his local organization team for all their efforts enabling ETAPS to return to the French Alps in Grenoble!

Preface

This volume contains the proceedings of the 17th International Conference on the Foundations of Software Science and Computation Structures, FOSSACS 2014, held in Grenoble, France, 5–13 April 2014. FOSSACS is an event of the Joint European Conferences on Theory and Practice of Software (ETAPS).

FOSSACS presents original papers on the foundations of software science. The conference invited submissions on theories and methods to support analysis, synthesis, transformation, and verification of programs and software systems. We received 128 abstracts and 106 full paper submissions; of these, 28 were selected for presentation at FOSSACS and inclusion in the proceedings. Also included is the abstract of a lecture by Petr Jančár, the FOSSACS 2014 invited speaker.

The PC members, and the external experts they consulted, wrote over 320 paper reviews, and the discussion phase of the meeting included a 3-day author rebuttal phase. The competition was very strong, and unfortunately many good papers could not be accepted.

I thank all the authors of papers submitted to FOSSACS 2014. I thank also all members of the PC for their excellent work, as well as the external reviewers for the expert help 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, I would like to thank the ETAPS 2014 Local Organizing Committee (chaired by Saddek Bensalem) and the ETAPS Steering Committee (chaired by Joost-Pieter Katoen) for their efficient coordination of all the activities leading up to FOSSACS 2014.

January 2014

Anca Muscholl

Organization

Program Committee

Luca Aceto	Reykjavik University, Iceland
Véronique Bruyère	University of Mons, Belgium
Véronique Cortier	LORIA, France
Kousha Etessami	University of Edinburgh, Scotland
Wan Fokkink	VU Amsterdam, The Netherlands
Holger Hermanns	Saarland University, Germany
Antonin Kucera	Masaryk University, Czech Republic
Slawomir Lasota	Warsaw University, Poland
Christof Löding	RWTH Aachen University, Germany
Anca Muscholl	University of Bordeaux, France
Andrew Pitts	University of Cambridge, UK
Alexander Rabinovich	Tel Aviv University, Israel
R. Ramanujam	The Institute of Mathematical Sciences, Chennai, India
Mark Reynolds	University of Western Australia, Australia
Simona Ronchi della Rocca	University of Turin, Italy
Grigore Rosu	University of Illinois at U.C., USA
Andrey Rybalchenko	TU Munich, Germany
Davide Sangiorgi	University of Bologna, Italy
Sven Schewe	University of Liverpool, UK
Thomas Schwentick	TU Dortmund, Germany
Luc Segoufin	LSV Cachan, France
Peter Selinger	Dalhousie University, Canada
Anil Seth	IIT Kanpur, India
James Worell	University of Oxford, UK
Wiesław Zielonka	University of Paris 7, France

Additional Reviewers

Atig, Mohamed Faouzi	Bollig, Benedikt
Bacci, Giorgio	Bonchi, Filippo
Banerjee, Mohua	Bouajjani, Ahmed
Basset, Nicolas	Brazdil, Tomas
Bataineh, Omar	Brotherston, James
Bernardo, Marco	Caltais, Georgiana
Bertrand, Nathalie	Cave, Andrew
Birget, Jean-Camille	Chadha, Rohit

Chen, Taolue
Cimini, Matteo
Ciobaca, Stefan
Clairambault, Pierre
Clemente, Lorenzo
Colcombet, Thomas
Corradini, Andrea
Cuijpers, Pieter
Czerwiński, Wojciech
D'Souza, Deepak
Dardha, Ornela
Davies, Rowan
Dawar, Anuj
De Paiva, Valeria
de Vink, Erik
Debois, Søren
Della Monica, Dario
Demangeon, Romain
Deng, Yuxin
Di Giamberardino, Paolo
Di Giusto, Cinzia
Diaconescu, Razvan
Drewes, Frank
Eisentraut, Christian
Espert, Javier
Fahrenberg, Uli
Fearnley, John
Feng, Yuan
Ferrer Fioriti, Luis Maria
Fiore, Marcelo
Forejt, Vojtech
Froeschle, Sibylle
Gabbay, Murdoch
Gaboardi, Marco
Gastin, Paul
Gay, Simon
Gebler, Daniel
Genaim, Samir
Genest, Blaise
Gimbert, Hugo
Given-Wilson, Thomas
Göller, Stefan
Goncharov, Sergey
Gore, Rajeev
Haar, Stefan
Haase, Christoph
Habermehl, Peter
Hague, Matthew
Hasegawa, Masahito
Hashemi, Vahid
Hatefi, Hassan
Hayman, Jonathan
Héam, Pierre-Cyrille
Heckel, Reiko
Heijltjes, Willem
Hildebrandt, Thomas
Hirschkoff, Daniel
Hofman, Piotr
Huang, Jeff
Hunter, Paul
Iosif, Radu
Jagadeesan, Radha
Jančar, Petr
Jennin, Jean-Baptiste
Kammar, Ohad
Kara, Ahmet
Karkare, Amey
Katsumata, Shin-Ya
Kissinger, Aleks
Klin, Bartek
Kobayashi, Naoki
Konur, Savas
Korenciak, Lubos
Kostylev, Egor V.
Koutny, Maciej
Krcal, Jan
Kretinsky, Jan
Kupke, Clemens
Lüttgen, Gerald
Laird, Jim
Lambers, Leen
Lanese, Ivan
LeFanu Lumsdaine, Peter
Lime, Didier
Lindley, Sam
Liu, Zhiqiang
Lluch Lafuente, Alberto
Lohrey, Markus
Lozes, Etienne
Manuel, Amaldev

Mardare, Radu
Marion, Jean-Yves
Martini, Simone
McCabe-Dansted, John
McKinley, Richard
Melliès, Paul-André
Michail, Othon
Miculan, Marino
Mikulski, Lukasz
Møgelberg, Rasmus Ejlers
Moore, Brandon
Mousavi, Mohammad Reza
Mukund, Madhavan
Murawski, Andrzej
Nandakumar, Satyadev
Nestmann, Uwe
Nies, Gilles
Novotný, Petr
Obdrzalek, Jan
Orchard, Dominic
Oualhadj, Youssouf
Pandya, Paritosh
Paolini, Luca
Park, Sungwoo
Parys, Pawel
Paul, Soumya
Perdrix, Simon
Perez, Jorge A.
Pérez, Guillermo
Petrisan, Daniela
Phillips, Iain
Piccolo, Mauro
Pimentel, Elaine
Popescu, Andrei
Pous, Damien
Puppis, Gabriele
Rabe, Markus N.
Ramsay, Steven
Reddy, Uday
Rehak, Vojtech
Rigo, Michel
Rot, Jurriaan
Sénizergues, Géraud
Sacerdoti Coen, Claudio
Sack, Joshua
Safránek, David
Salvati, Sylvain
Saurin, Alexis
Seely, Robert
Serbanuta, Traian Florin
Shoeppe, Ulrich
Silva, Alexandra
Simaitis, Aistis
Simon, Sunil
Sofronie-Stokkermans, Viorica
Song, Lei
Staton, Sam
Stefanescu, Andrei
Steinberg, Benjamin
Stewart, Alistair
Strejcek, Jan
Suresh, S.P.
Tan, Tony
Terui, Kazushige
Tiu, Alwen
Torunczyk, Szymon
Tribastone, Mirco
Trivedi, Ashutosh
Tschantz, Michael Carl
Turrini, Andrea
Tzevelekos, Nikos
Vafeiadis, Viktor
van Eeckelen, Marko
van Gool, Sam
Vaux, Lionel
Velner, Yaron
Villard, Jules
Wachter, Björn
Walukiewicz, Igor
Zavattaro, Gianluigi
Zeume, Thomas
Zhang, Lijun
Zimmermann, Martin

Table of Contents

Equivalences of Pushdown Systems Are Hard (Invited Contribution) . . .	1
<i>Petr Jančar</i>	

Probabilistic Systems

Active Diagnosis for Probabilistic Systems	29
<i>Nathalie Bertrand, Éric Fabre, Stefan Haar, Serge Haddad, and Loïc Hélouët</i>	
Analysis of Probabilistic Basic Parallel Processes	43
<i>Rémi Bonnet, Stefan Kiefer, and Anthony Widjaja Lin</i>	
Limit Synchronization in Markov Decision Processes	58
<i>Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi</i>	
Maximal Cost-Bounded Reachability Probability on Continuous-Time Markov Decision Processes	73
<i>Hongfei Fu</i>	

Semantics of Programming Languages

Type Reconstruction for the Linear π -Calculus with Composite and Equi-Recursive Types	88
<i>Luca Padovani</i>	
A Semantical and Operational Account of Call-by-Value Solvability	103
<i>Alberto Carraro and Giulio Guerrieri</i>	

Networks

Network-Formation Games with Regular Objectives	119
<i>Guy Avni, Orna Kupferman, and Tami Tamir</i>	
Playing with Probabilities in Reconfigurable Broadcast Networks	134
<i>Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier</i>	

Program Analysis

Unsafe Order-2 Tree Languages Are Context-Sensitive	149
<i>Naoki Kobayashi, Kazuhiro Inaba, and Takeshi Tsukada</i>	

Game Semantics for Nominal Exceptions	164
<i>Andrzej S. Murawski and Nikos Tzevelekos</i>	
Complexity of Model-Checking Call-by-Value Programs	180
<i>Takeshi Tsukada and Naoki Kobayashi</i>	

Games and Synthesis

Resource Reachability Games on Pushdown Graphs	195
<i>Martin Lang</i>	
Perfect-Information Stochastic Mean-Payoff Parity Games	210
<i>Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Youssouf Oualhadj</i>	
Latticed-LTL Synthesis in the Presence of Noisy Inputs	226
<i>Shaull Almagor and Orna Kupferman</i>	
The Complexity of Partial-Observation Stochastic Parity Games with Finite-Memory Strategies	242
<i>Krishnendu Chatterjee, Laurent Doyen, Sumit Nain, and Moshe Y. Vardi</i>	

Compositional Reasoning

On Negotiation as Concurrency Primitive II: Deterministic Cyclic Negotiations	258
<i>Javier Esparza and Jörg Desel</i>	
On Asymmetric Unification and the Combination Problem in Disjoint Theories	274
<i>Serdar Erbatur, Deepak Kapur, Andrew M. Marshall, Catherine Meadows, Paliath Narendran, and Christophe Ringeissen</i>	

Bisimulation

Axiomatizing Bisimulation Equivalences and Metrics from Probabilistic SOS Rules	289
<i>Pedro R. D’Argenio, Daniel Gebler, and Matias David Lee</i>	
Generalized Synchronization Trees	304
<i>James Ferlez, Rance Cleaveland, and Steve Marcus</i>	
Bisimulations for Communicating Transactions (Extended Abstract)	320
<i>Vasileios Koutavas, Carlo Spaccasassi, and Matthew Hennessy</i>	
Upper-Expectation Bisimilarity and Łukasiewicz μ -Calculus	335
<i>Matteo Mio</i>	

Categorical and Algebraic Models

Interacting Bialgebras Are Frobenius	351
<i>Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi</i>	
Generalized Eilenberg Theorem I: Local Varieties of Languages	366
<i>Jiří Adámek, Stefan Milius, Robert S.R. Myers, and Henning Urbat</i>	
Combining Bialgebraic Semantics and Equations	381
<i>Jurriaan Rot and Marcello Bonsangue</i>	
Models of a Non-associative Composition	396
<i>Guillaume Munch-Maccagnoni</i>	

Logics of Programming

Foundations for Decision Problems in Separation Logic with General Inductive Predicates	411
<i>Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max Kanovich, and Joël Ouaknine</i>	
A Coalgebraic Approach to Linear-Time Logics	426
<i>Corina Cirstea</i>	
A Relatively Complete Calculus for Structured Heterogeneous Specifications	441
<i>Till Mossakowski and Andrzej Tarlecki</i>	
Author Index	457

Equivalences of Pushdown Systems Are Hard

Petr Jančar*

Dept Comp. Sci., FEI, Techn. Univ. of Ostrava (VŠB-TUO),
17. Listopadu 15, 70833 Ostrava, Czech Rep.

petr.jancar@vsb.cz

<http://www.cs.vsb.cz/jancar>

Abstract. Language equivalence of deterministic pushdown automata (DPDA) was shown to be decidable by Sénizergues (1997, 2001); Stirling (2002) then showed that the problem is primitive recursive.

Sénizergues (1998, 2005) also generalized his proof to show decidability of bisimulation equivalence of (nondeterministic) PDA where ε -rules can be only deterministic and popping; this problem was shown to be nonelementary by Benedikt, Göller, Kiefer, and Murawski (2013), even for PDA with no ε -rules.

Here we refine Stirling’s analysis and show that DPDA equivalence is in TOWER, i.e., in the “least” nonelementary complexity class. The basic proof ideas remain the same but the presentation and the analysis are simplified, in particular by using a first-order term framework.

The framework of (nondeterministic) first-order grammars, with term root-rewriting rules, is equivalent to the model of PDA with restricted ε -rules to which Sénizergues’s decidability proof applies. We show that bisimulation equivalence is here Ackermann-hard, and thus not primitive recursive.

1 Introduction

Pushdown automata (PDA) are a standard and widely used model in computer science; we recall that a PDA is a (generally nondeterministic) finite automaton equipped with an (unbounded) stack, i.e., with a (LIFO) linear list accessible at only one end (at the “top”). PDA are naturally used to model (sequential) programs with recursive procedure calls, and they also form a basis for (automated) verification of some program properties. A traditional area that employs deterministic PDA (DPDA) is the syntactic analysis of programming languages.

Given such “devices” or “systems” (like PDA or DPDA), it is standard to ask if their (functional or behavioural) equivalence can be effectively, or even efficiently, checked. A classical equivalence is *language equivalence*, asking if two given systems accept the same sequences of (external) *input symbols*, or from another viewpoint, if they can perform the same sequences of *actions*.

While language equivalence of PDA was quickly recognized to be undecidable, the decidability question for DPDA had been a famous open problem since

* Supported by the Grant Agency of the Czech Rep., project GAČR:P202/11/0340.

1960s. The decidability was finally shown by Sénizergues [21], who got the Gödel Prize in 2002 for this achievement. Stirling [26] then showed that the problem is primitive recursive. We note that the known complexity lower bound for DPDA equivalence is just P-hardness (derived from P-hardness of the emptiness problem for context-free languages).

A fundamental behavioural equivalence is *bisimulation equivalence*, also called *bisimilarity*; roughly speaking, two states of a system (or of two systems) are bisimilar if performing an action a in one state can be matched by performing an action with the same name a in the other state so that the resulting states are also bisimilar. For deterministic systems this equivalence in principle coincides with language equivalence, but for nondeterministic ones it is finer.

Sénizergues [22] generalized his proof for DPDA to show decidability of bisimulation equivalence of (nondeterministic) PDA where ε -rules (internally changing the current state) can be only deterministic and stack-popping. (If we allow the option that a popping ε -rule can apply at the same time when an “external-action” rule can also apply, then bisimilarity becomes undecidable [14].) For this more general decidable problem no complexity upper bound has been shown. Regarding the lower bound, the previously known EXPTIME-hardness [18] has been recently shifted: Benedikt, Göller, Kiefer, and Murawski [2] showed that the problem is nonelementary, even for “real-time” PDA (RT-PDA), i.e. PDA with no ε -rules.

Contribution of this paper is summarized in the following points 1, 2.

1. Equivalence of DPDA is in TOWER. We refine Stirling’s analysis (from [26]) and show that DPDA equivalence is in TOWER, i.e., in the least (reasonably defined) nonelementary complexity class. We note that $\text{TOWER} = \mathbf{F}_3$ in the hierarchy of fast-growing complexity classes recently described by Schmitz [19]. The basic proof ideas remain the same (as in [26]) but the presentation and the analysis are simplified, in particular by using a first-order term framework, called *deterministic first-order grammars* (detFOG).

2. Bisimilarity of first-order grammars is Ackermann-hard. The general framework of (nondeterministic) first-order grammars (FOG), i.e. of finite sets of term *root-rewriting* rules, is equivalent to PDA with deterministic and popping ε -rules, i.e. to the model where decidability of bisimilarity was shown by Sénizergues [22]. We show that bisimulation equivalence is here even Ackermann-hard, and thus not primitive recursive. The proof is given by a reduction from the control-state reachability problem for reset (or lossy) counter machines for which Ackermann-hardness was shown by Schnoebelen (see [20] and references therein, and [28] for an independent proof related to relevance logic).

Further Comments. Some advantages of using first-order terms, i.e. a formalism with which (not only) every computer scientist is intimately familiar, were already demonstrated in [15]. Though the close relationship between (D)PDA and first-order schemes has been long known (see, e.g., [7]), the perspective viewing (D)PDA as (deterministic) finite sets of term *root-rewriting* rules seems not to have been fully exploited so far. We note that the decidability proof in [15] (for deterministic grammars) was constructed so that it constitutes a

good basis for extending the decidability proof to the nondeterministic case (to match [22]). Here we concentrate on bounding the length of shortest words witnessing nonequivalence, which cannot be easily generalized, as is now also indicated by the gap between TOWER and the Ackermann-hardness.

We summarize the relevant known results in the following table.

	DPDA = detFOG	RT-PDA	PDA~FOG	PDA
Lang-Eq	P-hard in TOWER	Undecidable	Undecidable	Undecidable
Bisi-Eq	P-hard in TOWER	TOWER-hard Decidable	ACK-hard Decidable	Undecidable

In the column PDA~FOG we refer to those PDA that are equivalent to first-order grammars, i.e., to PDA with only deterministic and popping ε -rules. PDA in the last column can have unrestricted ε -rules.

The results in the boldface are shown in this paper. The TOWER-membership is only one result, since language equivalence and bisimilarity in principle coincide for DPDA. As already mentioned, this result is derived by a finer look at Stirling's approach [26], where only a primitive recursive upper bound is claimed. The TOWER-hardness for RT-PDA is, in fact, a slight extrapolation of the result presented in [2]. The authors only claim nonelementary complexity but their proof can be adjusted to show TOWER-hardness in the sense of [19]. (This is the usual case with proofs showing nonelementary complexity lower bounds, as is also explained in [19]; in the particular case of RT-PDA this was also confirmed by a personal communication with S. Kiefer.)

We also note that the proof of Ackermann-hardness for first-order grammars presented here uses the feature (namely varying lengths of branches of syntactic trees of terms) corresponding to (restricted) ε -rules; hence TOWER-hardness remains the best known lower complexity bound for bisimilarity of RT-PDA.

The problem for reset counter machines, used in the hardness proof here, is in fact Ackermann-complete (or ACK-complete, or \mathbf{F}_ω -complete in the hierarchy of [19]); the upper bound was shown in [9]. The question of a similar upper bound for the bisimilarity problem is not discussed here.

Related Work and Some Open Questions. Here we only briefly mention some results and questions that are very close to the above discussed equivalence problems, with no attempt to give any sort of a full account. (For an updated survey of a specific area, namely bisimilarity checking of infinite-state systems, we can refer to [24].)

The main challenge is still to clarify the complexity status of DPDA equivalence, which is still far from being understood. The practical experiments by Henry and Sénizergues [12] have strengthened the feeling that the TOWER bound is indeed too large. More pleasant upper bounds were shown for subclasses of DPDA. A co-NP upper bound is known for finite-turn DPDA [23]. For simple grammars (real-time DPDA with a single control state), a polynomial algorithm deciding equivalence was shown in [13] (see [8] for a recent upper bound); it is worth to note that the *language inclusion* problem is *undecidable*

even in this simple case [10]. A recent result also shows NL-completeness of equivalence of deterministic one-counter automata [3] (answering the forty-year old polynomiality question posed by Valiant and Paterson).

A natural subclass of PDA are visibly pushdown automata, with EXPTIME-complete language equivalence problem (Alur and Madhusudan [1]); for bisimilarity the EXPTIME-completeness was shown by Srba [25], who also used Walukiewicz’s result on model checking pushdown systems [29]. For real-time one-counter automata bisimilarity is PSPACE-complete [4]. An interesting subclass is BPA (Basic Process Algebra), corresponding to real-time PDA with a single control state. In the so called normed case bisimilarity is polynomial (see the above mentioned [13], [8]), but in general bisimilarity is in 2-EXPTIME (claimed in [6] and explicitly proven in [16]) and EXPTIME-hard [17].

We can also mention the higher-order case. The decidability question for higher-order DPDA equivalence remains an open problem; some progress in this direction was made by Stirling in [27]. Bisimilarity of second-order RT-PDA is undecidable [5]. Another generalization of pushdown systems are ground term (or tree) rewrite systems (where rules replace subterms with subterms). Here the decidability of bisimilarity is open [11].

2 Preliminaries

In this section we define the basic notions; some standard definitions might be given in restricted forms when we do not need the full generality.

By \mathbb{N} we denote the set $\{0, 1, 2, \dots\}$ of nonnegative integers; we put $[i, j] = \{i, i+1, \dots, j\}$. For a set \mathcal{A} , by $\text{CARD}(\mathcal{A})$ we denote its cardinality (i.e. the number of elements when \mathcal{A} is finite). By \mathcal{A}^* we denote the set of finite sequences of elements of \mathcal{A} , which are also called *words* (over \mathcal{A}). By $|w|$ we denote the *length* of $w \in \mathcal{A}^*$. If $w = uv$ then u is a *prefix* of w , and v is a *suffix* of w . By ε we denote the *empty sequence* ($|\varepsilon| = 0$).

2.1 Bisimulation Equivalence in LTSs and in Deterministic LTSs

Labelled Transition Systems. A *labelled transition system* (an LTS) is a tuple $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$ where \mathcal{S} is a finite or countable set of *states*, Σ is a finite set of *actions* (or *letters*), and $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *a-transitions* (for each $a \in \Sigma$). We write $s \xrightarrow{a} s'$ instead of $(s, s') \in \xrightarrow{a}$. By $s \xrightarrow{w} s'$, where $w = a_1 a_2 \dots a_n \in \Sigma^*$, we denote that there is a *path* $s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n = s'$; if $s \xrightarrow{w} s'$, then s' is *reachable from* s . By writing $s \xrightarrow{w}$ we mean that s *enables* w , i.e., $s \xrightarrow{w} s'$ for some s' .

Deterministic LTSs. An LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$ is *deterministic*, a *det-LTS* for short, if for each pair $s \in \mathcal{S}$, $a \in \Sigma$ there is at most one s' such that $s \xrightarrow{a} s'$. Hence if w is enabled by s then there is precisely one s' such that $s \xrightarrow{w} s'$. Here we also use expressions like “the path $s \xrightarrow{w} s'$ ” or “the path $s \xrightarrow{w}$ ” since the respective path is unique.

Bisimilarity. We assume a (general) LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$. A set $\mathcal{B} \subseteq \mathcal{S} \times \mathcal{S}$ is a *bisimulation* if for any $(s, t) \in \mathcal{B}$ we have: for any $s \xrightarrow{a} s'$ there is $t \xrightarrow{a} t'$ such that $(s', t') \in \mathcal{B}$, and for any $t \xrightarrow{a} t'$ there is $s \xrightarrow{a} s'$ such that $(s', t') \in \mathcal{B}$. States $s, t \in \mathcal{S}$ are *bisimulation equivalent*, or *bisimilar*, written $s \sim t$, if there is a bisimulation \mathcal{B} containing (s, t) . In fact, $\sim \subseteq \mathcal{S} \times \mathcal{S}$ is the maximal bisimulation, the union of all bisimulations.

Trace Equivalence and Eq-levels in Det-LTSs. It is easy to check that in det-LTSs bisimulation equivalence coincides with so called *trace equivalence*, i.e., in any *deterministic* LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$ we have

$$s \sim t \text{ iff } \forall w \in \Sigma^* : s \xrightarrow{w} \Leftrightarrow t \xrightarrow{w}.$$

Hence s, t are equivalent iff they enable the same set of words, also called traces.

This suggests the following natural stratification of \sim ; it can be defined in the general (nondeterministic) case as well, but the (technically easier) deterministic case is sufficient for us. We thus assume a given det-LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$.

For any $k \in \mathbb{N}$ we put $\Sigma^{\leq k} = \{w; |w| \leq k\}$, and

$$s \sim_k t \text{ if } \forall w \in \Sigma^{\leq k} : s \xrightarrow{w} \Leftrightarrow t \xrightarrow{w}.$$

We note that $\sim_0 = \mathcal{S} \times \mathcal{S}$, and $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \dots$. Since our assumed \mathcal{L} is deterministic, it is also obvious that $\bigcap_{k \in \mathbb{N}} \sim_k = \sim$.

We use the notation $s \sim_k t$ also for $k = \omega$, identifying $s \sim_\omega t$ with $s \sim t$. For each pair (s, t) of states we define its *equivalence level*, its *eq-level* for short:

$$\text{EqLV}(s, t) = \max \{k \in \mathbb{N} \cup \{\omega\} \mid s \sim_k t\}.$$

We stipulate that $k < \omega$ and $\omega - k = \omega + k = \omega$ for any $k \in \mathbb{N}$. (Hence, e.g., $s \sim_{e-5} t$ means $s \sim t$ when $e = \omega$.)

Witnesses for Nonequivalent Pairs in Det-LTSs. Given a det-LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$, we note that $s \not\sim t$ implies that any *shortest* word wa ($a \in \Sigma$) witnessing their nonequivalence (i.e., enabled by precisely one of s, t) satisfies $|w| = \text{EqLV}(s, t)$. For technical convenience we introduce the following definition:

a word $w \in \Sigma^*$ is a *witness* for (s, t)

if it is a shortest word such that for some $a \in \Sigma$ we have that wa is enabled by precisely one of s, t . (A witness w for (s, t) is thus a shortest word satisfying $s \xrightarrow{w} s', t \xrightarrow{w} t'$ where $\text{EqLV}(s', t') = 0$, i.e. $s' \not\sim_1 t'$.) The *witness set* for (s, t) is the set of all witnesses for (s, t) (which is empty iff $s \sim t$).

We state some obvious facts in the next proposition. The crux is that by performing the same action $a \in \Sigma$ from s and t in a det-LTS the eq-level drops by at most one (if at all), and it does drop for some action when $\omega > \text{EqLV}(s, t) > 0$.

Proposition 1. *In any det-LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$ we have:*

1. *If $\text{EqLV}(s, s') > \text{EqLV}(s, t)$, then $\text{EqLV}(s, t) = \text{EqLV}(s', t)$ and the witness sets for (s, t) and (s', t) are the same.*

2. If u is a witness for (s, t) and $u = wu'$, then u' is a witness for (s', t') where $s \xrightarrow{w} s'$, $t \xrightarrow{w} t'$. Hence also $\text{EqLV}(s', t') = \text{EqLV}(s, t) - |w|$.
3. If $s \xrightarrow{v} s''$ and $t \xrightarrow{v} t''$, then $\text{EqLV}(s'', t'') \geq \text{EqLV}(s, t) - |v|$.

2.2 First-Order Terms, FO Grammars, and Det-FO Grammars

We aim to look at LTSs in which states are first-order terms. Transitions in such an LTS will be determined by a finite set of *root-rewriting* rules. We start with definitions of these ingredients.

First-Order (Regular) Terms. The terms are built from some specified function symbols, using *variables* from a fixed set $\text{VAR} = \{x_1, x_2, x_3, \dots\}$.

A *finite term* is either a variable x_i , or $A(G_1, \dots, G_m)$ where A is a function symbol with arity m and G_j are finite terms. Each finite term has its (rooted, finite, ordered) *syntactic tree*: for x_i it is just the root labelled with x_i ; for $A(G_1, \dots, G_m)$, the root is labelled with A , and the ordered root-successors are the trees corresponding to G_1, \dots, G_m , respectively. The height of a finite term E , denoted $\text{HEIGHT}(E)$, is the length of the longest branch of its syntactic tree.

Given a syntactic tree of a term, if we allow redirecting the arcs (i.e. changing their target nodes) arbitrarily, then we get a finite *graph presentation* (with a designated root but with possible cycles) of a *regular term*; its syntactic tree might be infinite, but the term has only finitely many *subterms*, where a subterm can have infinitely many *occurrences*, in arbitrarily large *depths*. By $\text{size}(E)$ we mean the size (the number of nodes, say) of the smallest graph presentation of a regular term E . (At the level of our later analysis, the details of such definitions are unimportant.)

In what follows, by a “term” we mean a “regular term” if we do not say explicitly that the term is finite. We reserve symbols E, F, G, H , and also T, U, V, W , for denoting (regular) terms.

Substitutions, and Their Associative Composition. By $\text{TERMS}_{\mathcal{N}}$ we denote the set of all (regular) terms over a (finite) set \mathcal{N} of function symbols (called “nonterminals” later). A *substitution* σ is a mapping $\sigma : \text{VAR} \rightarrow \text{TERMS}_{\mathcal{N}}$ whose *support* $\text{SUPP}(\sigma) = \{x_i \mid \sigma(x_i) \neq x_i\}$ is *finite*; we reserve the symbol σ for substitutions. By $\text{RANGE}(\sigma)$ we mean the set $\{\sigma(x_i) \mid x_i \in \text{SUPP}(\sigma)\}$. The finite-support restriction allows us to present any σ as a finite set of pairs. E.g., $\{(x_i, H)\}$ where $H \neq x_i$ is a substitution with the one-element support $\{x_i\}$.

By *applying a substitution* σ to a term E we get the term $E\sigma$ that arises from E by replacing each occurrence of x_i with $\sigma(x_i)$; given graph presentations, in the graph of E we just redirect each arc leading to x_i towards the root of $\sigma(x_i)$ (which includes the special “root-designating arc” when $E = x_i$). For $E = x_i$ we thus have $E\sigma = x_i\sigma = \sigma(x_i)$.

The natural *composition of substitutions* (where $\sigma = \sigma_1\sigma_2$ satisfies $\sigma(x_i) = (\sigma_1(x_i))\sigma_2$) is obviously associative. We thus write simply $E\sigma_1\sigma_2$ when meaning $(E\sigma_1)\sigma_2$ or $E(\sigma_1\sigma_2)$. For future use we might note that $\{(x_i, H)\}\sigma$ is the substitution arising from σ by replacing $\sigma(x_i)$ with $H\sigma$.

First-Order Grammars, and Det-FO Grammars. A *first-order grammar*, an *FO grammar* or just a *grammar* for short, is a tuple $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ where $\mathcal{N} = \{A_1, A_2, \dots\}$ is a finite set of ranked *nonterminals*, viewed as function symbols with arities, $\Sigma = \{a_1, a_2, \dots\}$ is a finite set of *actions* (or letters), and \mathcal{R} is a finite set of (root rewriting) *rules* of the form

$$A(x_1, x_2, \dots, x_m) \xrightarrow{a} E \quad (1)$$

where $A \in \mathcal{N}$, $\text{arity}(A) = m$, $a \in \Sigma$, and E is a *finite* term over \mathcal{N} in which each occurring variable is from the set $\{x_1, x_2, \dots, x_m\}$. (We exemplify the rules by $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$, $A(x_1, x_2, x_3) \xrightarrow{b} x_2$, $D(x_1, x_2) \xrightarrow{a} A(D(x_2, x_2), x_1, B)$; here the arities of A, B, C, D are 3, 0, 2, 2, respectively.)

A grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ is a *det-FO grammar* (deterministic first-order grammar) if for each pair $A \in \mathcal{N}$, $a \in \Sigma$ there is at most one rule of the type (1).

2.3 LTSs of Grammars, Equivalence Problem, Relation to PDA

LTSs Associated with Grammars and Det-FO Grammars. A grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ defines the LTS $\mathcal{L}_{\mathcal{G}} = (\text{TERMS}_{\mathcal{N}}, \Sigma, (\xrightarrow{a})_{a \in \Sigma})$ in which each rule

$$A(x_1, \dots, x_m) \xrightarrow{a} E \text{ induces transitions } (A(x_1, \dots, x_m))\sigma \xrightarrow{a} E\sigma$$

for all substitutions $\sigma : \text{VAR} \rightarrow \text{TERMS}_{\mathcal{N}}$.

(Examples of transitions induced by the previously given rules are $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$, $A(x_5, x_5, x_2) \xrightarrow{b} C(D(x_2, B), x_5)$, $A(U_1, U_2, U_3) \xrightarrow{b} C(D(U_3, B), U_2)$, $A(U_1, U_2, U_3) \xrightarrow{b} U_2$, etc.)

We complete the definition of $\mathcal{L}_{\mathcal{G}}$ by stipulating that

$$\text{EqLv}(x_i, H) = 0 \text{ if } H \neq x_i \text{ (in particular, } x_i \not\sim_1 x_j \text{ for } i \neq j).$$

To stay in the realm of pure LTSs, we could imagine that each used variable $x \in \text{VAR}$ is equipped with a fresh unique action a_x and with the rule $x \xrightarrow{a_x} x$. But we never consider such “transitions” in our reasoning, and we handle x_i as “dead terms” (not enabling any action). We thus (often tacitly) use the fact that $F \xrightarrow{w} G$ implies $F\sigma \xrightarrow{w} G\sigma$ (though not vice-versa in general).

Since the rhs (right-hand sides) in the rules (1) are finite terms, all *terms reachable from a finite term* are *finite*. (It turns out technically convenient to have the rhs finite while taking the set $\text{TERMS}_{\mathcal{N}}$ of all regular terms as the state set of $\mathcal{L}_{\mathcal{G}}$; the other options are in principle equivalent.)

We also observe that *the LTS $\mathcal{L}_{\mathcal{G}}$ is deterministic iff \mathcal{G} is a det-FO grammar*.

Equivalence Problems. By the *bisimilarity problem* (or the bisimulation equivalence problem) for *FO grammars* we mean the decision problem that asks, given a grammar \mathcal{G} and terms T_0, U_0 , whether $T_0 \sim U_0$ in $\mathcal{L}_{\mathcal{G}}$.

By the *equivalence problem for det-FO grammars* we mean the restriction of the bisimilarity problem to deterministic first-order grammars.

Relation of Grammars and Pushdown Automata. We have mentioned the relationship between (D)PDA and first-order schemes (see, e.g., [7]). A concrete transformation of a PDA (or of a DPDA) to an FO grammar (or to a det-FO grammar) can be found in [15]. We just sketch the idea, though it is not important here, and can be skipped; it suffices just to accept the main message mentioned afterwards.

A configuration $q_i Y_1 Y_2 \dots Y_k \perp$ of a PDA, where \perp is the bottom-of-the-stack symbol and the control states are q_1, q_2, \dots, q_m , can be viewed as the term $\mathcal{T}(q_i Y_1 Y_2 \dots Y_k \perp)$ defined inductively as follows: $\mathcal{T}(q_i \perp) = \perp$, $\mathcal{T}(q_i Y \alpha) = [q_i Y](\mathcal{T}(q_1 \alpha), \mathcal{T}(q_2 \alpha), \dots, \mathcal{T}(q_m \alpha))$. Hence we view each pair (q_i, Y) of a control state and a stack symbol as a nonterminal $[q_i Y]$ with arity m ; a special case is \perp with arity 0. A pushdown rule $q_i Y \xrightarrow{a} q_j \beta$ is rewritten to $q_i Y x \xrightarrow{a} q_j \beta x$ for a special formal symbol x , and the rule is transformed to $\mathcal{T}(q_i Y x) \xrightarrow{a} \mathcal{T}(q_j \beta x)$, where we define $\mathcal{T}(q_j x) = x_j$; hence $\mathcal{T}(q_i Y x) = [q_i Y](x_1, x_2, \dots, x_m)$. In fact, we still modify the operator \mathcal{T} when *deterministic popping ε -rules* $q_i Y \xrightarrow{\varepsilon} q_j$ are present. (In this case no other rule of the form $q_i Y \xrightarrow{\dots} \dots$ can be present.) For each such rule we do not create the grammar rule $\mathcal{T}(q_i Y x) \xrightarrow{\varepsilon} \mathcal{T}(q_j x)$ (recall that we have no ε -rules in our grammars) but we put $\mathcal{T}(q_i Y \alpha) = \mathcal{T}(q_j \alpha)$. (Hence the branches of the syntactic tree of $\mathcal{T}(q \alpha)$ can have varying lengths.)

The main message is that the classical language equivalence of DPDA is easily inter-reducible with the equivalence of det-FO grammars. (This also uses the fact that trace equivalence is a version of language equivalence to which the classical accepting-state equivalence can be easily reduced. Another standard fact is that in DPDA we can w.l.o.g. assume that all ε -rules are deterministic and popping.) A similar inter-reducibility holds for the bisimilarity problem for PDA and FO grammars, with the proviso that we restrict ourselves to (nondeterministic) PDA where all ε -rules (if any are present) are deterministic and popping.

2.4 Complexity Classes TOWER and ACK (Ackermann)

We now recall the notions needed for stating our complexity results. A hierarchy of “hard” complexity classes (where TOWER = \mathbf{F}_3 and ACK = \mathbf{F}_ω) as well as more details can be found in [19].

An *elementary function* $\mathbb{N}^k \rightarrow \mathbb{N}$ arises by a finite composition of constants, the elementary operations $+$, $-$, \cdot , *div* and the exponential operator \uparrow , where $m \uparrow n = m^n$. E.g., the triple-exponential function $f(n) = 2^{2^{2^n}}$ is elementary.

Tower-Bounded Functions, Class TOWER. Function $\text{Tower} : \mathbb{N} \rightarrow \mathbb{N}$ defined by $\text{Tower}(0) = 1$ and $\text{Tower}(n+1) = 2^{\text{Tower}(n)}$ is nonelementary. We say that a *function* $f : \mathbb{N} \rightarrow \mathbb{N}$ is *Tower-bounded* if there is an elementary function g such that $f(n) \leq \text{Tower}(g(n))$. By TOWER we denote the class of decision problems solvable by Turing machines with Tower-bounded time (or space).

Class ACK, and Ackermann-Hardness. Let the family f_0, f_1, f_2, \dots of functions be defined by putting $f_0(n) = n+1$ and $f_{k+1}(n) = f_k(f_k(\dots f_k(n) \dots))$ where f_k is applied $n+1$ times. By (our version of) the *Ackermann function* we

mean the function f_A defined by $f_A(n) = f_n(n)$; it is a “simplest” non-primitive recursive function. A decision problem belongs to the class ACK if it is solvable in time (or space) $f_A(g(n))$ where g is a primitive recursive function. It is the ACK-hardness, called Ackermann-hardness, what is important here. We refer to [19] for a full discussion, but for our use the following restricted version suffices.

We define HP-ACK as the problem that asks, given a Turing machine M , an input w , and some $n \in \mathbb{N}$, whether M halts on w within $f_A(n)$ steps. We say that a *problem* \mathcal{P} is *Ackermann-hard* if HP-ACK is reducible to \mathcal{P} , or to the complementary problem $\text{co-}\mathcal{P}$, by a standard polynomial many-one reduction.

3 Equivalence of Det-FO Grammars Is in TOWER

In this section we show that the lengths of witnesses for pairs of nonequivalent terms in LTSs associated with det-FO grammars are Tower-bounded. It is then obvious that the equivalence problem for det-FO grammars is in TOWER.

To make this precise, we define the function $\text{MAXFEL} : \mathbb{N} \rightarrow \mathbb{N}$ (“Maximal Finite Eq-Level”) as given below. We stipulate $\text{max}\emptyset = 0$, and by $\text{size}(\mathcal{G}, T, U)$ we mean the size of a standard presentation of grammar \mathcal{G} and terms T, U . For any $j \in \mathbb{N}$, we put

$$\text{MAXFEL}(j) = \max \{ e \mid \text{there are a det-FO grammar } \mathcal{G} \text{ and terms } T, U \text{ such that } T \not\sim U \text{ in } \mathcal{L}_{\mathcal{G}}, \text{size}(\mathcal{G}, T, U) \leq j, \text{ and } \text{EqLv}(T, U) = e \}.$$

Theorem 2. *Function MAXFEL (for det-FO grammars) is Tower-bounded.*

Corollary 3. *The equivalence problem for det-FO grammars, as well as for DPDA, is in TOWER.*

In Section 3.2 we describe a proof of Theorem 2 at a partly informal level. Some more formalized proof parts are then given in Section 3.3. Before starting with the proof, we first observe some important facts in Section 3.1. These facts are more or less straightforward, and an “impatient” reader might thus have only a quick look at Section 3.1 and read Section 3.2 immediately, returning to Section 3.1 if/when needed.

3.1 Compositionality of Terms, and Safe Changes of Substitutions

Given a det-FO grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$, the LTS $\mathcal{L}_{\mathcal{G}}$ is *deterministic*, and it thus has the properties captured by Prop. 1. We now note some other properties, using the *structure of states* of $\mathcal{L}_{\mathcal{G}}$, i.e. the structure of terms. (Recall that by a “term” we mean a “regular term”, unless we explicitly say a “finite term”.)

It is useful to extend the relations \sim_k and \sim to substitutions. For $\sigma, \sigma' : \text{VAR} \rightarrow \text{TERMS}_{\mathcal{N}}$ and $k \in \mathbb{N} \cup \{\omega\}$ we put

$$\sigma \sim_k \sigma' \text{ if } \sigma(x_i) \sim_k \sigma'(x_i) \text{ for all } x_i \in \text{VAR}.$$

We also put $\text{EqLv}(\sigma, \sigma') = \max \{ k \mid \sigma \sim_k \sigma' \}$.

Congruence Properties, and the Limit of a Repeated Substitution

The items 1, 2 in the next proposition (Prop. 4) state the simple fact that \sim_k are congruences (for all $k \in \mathbb{N} \cup \{\omega\}$) in our term-setting. The item 3 states a basic fact underpinning our use of *regular* terms.

The point where and why we come to regular terms even when starting with finite terms can be *roughly* described as follows: If we can replace subterms $\sigma(x_i)$ in a pair $(E\sigma, F\sigma)$ with $H\sigma$, while keeping the same eq-level, i.e. having $\text{EqLV}(E\sigma, F\sigma) = \text{EqLV}(E\{(x_i, H)\}\sigma, F\{(x_i, H)\}\sigma)$, then we can repeat such replacing of σ with $\{(x_i, H)\}\sigma$ forever, keeping the same eq-level all the time. The limit of this process is the pair $(E'\sigma, F'\sigma)$ where $E' = E\{(x_i, H)\}\{(x_i, H)\}\cdots$ and $F' = F\{(x_i, H)\}\{(x_i, H)\}\cdots$. In $(E'\sigma, F'\sigma)$ the value $\sigma(x_i)$ is irrelevant, and we can thus “remove” x_i from the support of σ (as described below).

We note that the limit $H' = H\{(x_i, H)\}\{(x_i, H)\}\{(x_i, H)\}\cdots$ is a well-defined regular term, when H is a (regular) term. If $H = x_i$, then $H' = x_i$, and otherwise H' is the unique term satisfying $H' = H\{(x_i, H')\}$; we note that $H' = H$ if x_i does not occur in H , which includes the case $H = x_j$ for $j \neq i$. (A graph presentation of H' arises from a graph presentation of H by redirecting any arc leading to x_i towards the root.)

We note that $H \neq x_i$ implies that x_i does not occur in H' . We also observe that if x_i does not occur in a term G then $\sigma(x_i)$ plays no role in the composition $\{(x_i, G)\}\sigma$. In this case $\{(x_i, G)\}\sigma = \{(x_i, G)\}\sigma_{[-x_i]}$ where $\sigma_{[-x_i]}$ arises from σ by removing x_i from the support (if it is there), i.e.,

$$\sigma_{[-x_i]}(x_i) = x_i \text{ and } \sigma_{[-x_i]}(x_j) = \sigma(x_j) \text{ for } j \neq i.$$

Proposition 4

1. If $E \sim_k F$, then $E\sigma \sim_k F\sigma$. Hence $\text{EqLV}(E, F) \leq \text{EqLV}(E\sigma, F\sigma)$.
2. If $\sigma \sim_k \sigma'$ then $E\sigma \sim_k E\sigma'$. Hence $\text{EqLV}(\sigma, \sigma') \leq \text{EqLV}(E\sigma, E\sigma')$.
Moreover, if $\sigma \not\sim \sigma'$ and $E \notin \text{VAR}$ then $\text{EqLV}(\sigma, \sigma') < \text{EqLV}(E\sigma, E\sigma')$.
3. If $\sigma(x_i) \sim_k H\sigma$ and $H \neq x_i$, then $\sigma \sim_k \{(x_i, H')\}\sigma_{[-x_i]}$ where $H' = H\{(x_i, H)\}\{(x_i, H)\}\cdots$.

Proof. The points 1 and 2 can be easily shown by induction on k ; for $k = \omega$ we use the fact that $\sim = \bigcap_{j \in \mathbb{N}} \sim_j$. It is also obvious that $\text{EqLV}(E\sigma, E\sigma') \geq |w| + \text{EqLV}(\sigma, \sigma')$ if w is a shortest word such that $E \xrightarrow{w} x_j$ for some $x_j \in \text{VAR}$; if there is no such w , then $E\sigma \sim E\sigma'$.

3. Suppose $\sigma(x_i) \sim_k H\sigma$ and $H \neq x_i$, and put $\sigma' = \{(x_i, H')\}\sigma_{[-x_i]}$; hence $\sigma' = \{(x_i, H')\}\sigma$ since x_i does not occur in H' . We need to show $\sigma \sim_k \sigma'$.

Since $\sigma'(x_j) = \sigma(x_j)$ for $j \neq i$, we have $\text{EqLV}(\sigma, \sigma') = \text{EqLV}(\sigma(x_i), \sigma'(x_i)) = \text{EqLV}(\sigma(x_i), H'\sigma)$. Hence it suffices to show $\text{EqLV}(\sigma(x_i), H'\sigma) \geq k$; we assume $\sigma(x_i) \not\sim H'\sigma$, since otherwise we are done.

Since $H' = H\{(x_i, H')\}$, we deduce that $H'\sigma = H\sigma'$. Using 2 (and the fact $\sigma'(x_j) = \sigma(x_j)$ when $H = x_j$), we deduce that $\text{EqLV}(H\sigma, H\sigma') > \text{EqLV}(\sigma, \sigma') = \text{EqLV}(\sigma(x_i), H'\sigma)$.

Hence $\text{EqLV}(\sigma(x_i), H'\sigma) = \text{EqLV}(\sigma(x_i), H\sigma) \geq k$ (by using Prop. 1(1)). \square

Getting an “Equation” $\sigma(x_i) \sim_k H\sigma$

The (technical) item 1 in the next proposition (Prop. 5) is trivial. The item 2 “completes” the item 1 in Prop. 4: Roughly speaking, if we can “increase” $\text{EqLv}(E, F)$ by applying some σ to both E and F , then the reason is that any witness w for (E, F) reaches some x_i on one side and $H \neq x_i$ on the other side. We have $\text{EqLv}(x_i, H) = 0$ but $\text{EqLv}(\sigma(x_i), H\sigma)$ might be larger.

Proposition 5

1. If $E \xrightarrow{w} x_i, F \xrightarrow{w} H$ or $E \xrightarrow{w} H, F \xrightarrow{w} x_i$ where $H \neq x_i$, then for any σ we have $\text{EqLv}(\sigma(x_i), H\sigma) \geq \text{EqLv}(E\sigma, F\sigma) - |w|$.
2. If $\text{EqLv}(E, F) < \text{EqLv}(E\sigma, F\sigma)$ for some σ , then for any witness w for (E, F) there are some $x_i \in \text{SUPP}(\sigma)$ and $H \neq x_i$ such that $E \xrightarrow{w} x_i, F \xrightarrow{w} H$ or $E \xrightarrow{w} H, F \xrightarrow{w} x_i$.

Proof. 1. Since $E \xrightarrow{w} x_i$ implies $E\sigma \xrightarrow{w} \sigma(x_i)$ and $F \xrightarrow{w} H$ implies $F\sigma \xrightarrow{w} H\sigma$, the claim follows from Prop. 1(3).

2. By induction on $e = \text{EqLv}(E, F)$. We cannot have $(E, F) = (x_i, x_i)$ since $E \not\sim F$; if $\{E, F\} = \{x_i, H\}$ for $H \neq x_i$, then we are done: $e = 0, w = \varepsilon$, and if $x_i \notin \text{SUPP}(\sigma)$ then $\{E\sigma, F\sigma\} = \{x_i, H\sigma\}$, in which case $\text{EqLv}(x_i, H\sigma) > 0$ implies that $H = x_j$ (for $j \neq i$) and $\sigma(x_j) = x_i$, whence $x_j \in \text{SUPP}(\sigma)$.

We thus assume that both $\text{ROOT}(E)$ and $\text{ROOT}(F)$ are nonterminals. If $e = 0$ (i.e., the roots enable different sets of actions), then $\text{EqLv}(E\sigma, F\sigma) = 0$ – a contradiction; hence $e > 0$. Then for each $a \in \Sigma$ where $E \xrightarrow{a} E', F \xrightarrow{a} F'$ and $\text{EqLv}(E', F') = e - 1$ (hence for each a “starting” a witness for (E, F)) we have $\text{EqLv}(E'\sigma, F'\sigma) \geq \text{EqLv}(E\sigma, F\sigma) - 1 > e - 1$. By the induction hypothesis any witness w' for (E', F') satisfies the claim, and thus any witness aw' for (E, F) satisfies the claim as well. \square

Safe Changes of Subterms in a Pair of Terms (Keeping the Eq-Level)

The form of the next proposition is tailored to match our later use. The crux of the item 1 (of Prop. 6) trivially follows from the already established facts: we can “safely” replace a subterm V in one term of a pair (T, U) with another subterm V' if $\text{EqLv}(V, V') > \text{EqLv}(T, U)$. By “safely” we mean that the eq-level does not change: if the arising pair is (T', U) , say, then $\text{EqLv}(T', U) = \text{EqLv}(T, U)$; moreover, even the witness sets for (T, U) and (T', U) are the same.

The item 2 is slightly subtler: if we want to safely replace $(E\sigma, F\sigma)$ with $(E\sigma', F\sigma')$ (thus replacing the relevant subterms on both sides simultaneously), then a (substantially) weaker condition for $\text{EqLv}(\sigma, \sigma')$ suffices. Roughly speaking, from $(E\sigma', F\sigma')$ we should first perform a word at least as long as a witness for (E, F) before the change of substitutions might matter (regarding the eq-level). For a safe replacing, i.e. for guaranteeing $\text{EqLv}(E\sigma', F\sigma') = \text{EqLv}(E\sigma, F\sigma)$, it thus suffices that $\text{EqLv}(\sigma, \sigma') > \text{EqLv}(E\sigma, F\sigma) - \text{EqLv}(E, F)$.

Proposition 6

1. If $\text{EqLv}(\sigma, \sigma') > \text{EqLv}(G\sigma, U)$, then the witness sets for $(G\sigma, U)$ and $(G\sigma', U)$ are the same (and $\text{EqLv}(G\sigma, U) = \text{EqLv}(G\sigma', U)$).

2. Assume $\text{EqLV}(E, F) = k < \omega$ and $\text{EqLV}(E\sigma, F\sigma) = e$ (hence $k \leq e$). If $\sigma \sim_{e-k+1} \sigma'$ then $\text{EqLV}(E\sigma', F\sigma') = e$.

Proof

1. Since $\text{EqLV}(G\sigma, G\sigma') \geq \text{EqLV}(\sigma, \sigma')$, the claim follows from Prop. 1(1).
 2. If $e = \omega$, then we have $\sigma \sim \sigma'$, and thus $E\sigma' \sim E\sigma \sim F\sigma \sim F\sigma'$. Suppose now a counterexample with the minimal e ; hence $\text{EqLV}(E\sigma', F\sigma') = e' > e$. (If $e' < e$, then swapping σ, σ' contradicts our minimality assumption.)

If $\{E, F\} = \{x_i, H\}$, then $H \neq x_i$, $k = 0$, and $\text{EqLV}(\sigma(x_i), H\sigma) = e$. Since $\sigma \sim_{e+1} \sigma'$, we have $\text{EqLV}(\sigma'(x_i), H\sigma') = e$, a contradiction. Otherwise (when $\{E, F\} \neq \{x_i, H\}$, and thus $\text{ROOT}(E)$ and $\text{ROOT}(F)$ are nonterminals) we must have $1 \leq k \leq e < e'$, and there is $a \in \Sigma$ such that $E \xrightarrow{a} E', F \xrightarrow{a} F'$ and $\text{EqLV}(E'\sigma, F'\sigma) = e-1$. We thus have

$$k-1 \leq k' = \text{EqLV}(E', F') \leq \text{EqLV}(E'\sigma, F'\sigma) = e-1 < e' - 1 \leq \text{EqLV}(E'\sigma', F'\sigma').$$

Since $\sigma \sim_{e-1-k'+1} \sigma'$, we contradict our minimality assumption. \square

We now state a simple corollary of the previous facts; its form will be particularly useful in an inductive argument based on decreasing the support of certain substitutions.

Corollary 7. *Suppose $\text{EqLV}(E\sigma, F\sigma) > \text{EqLV}(E\sigma'\sigma, F\sigma'\sigma)$. Then $E \not\sim F$ and for any witness w for (E, F) there are $x_i, H \neq x_i$ such that $E \xrightarrow{w} x_i, F \xrightarrow{w} H$, or vice versa. For any such x_i, H we have*

$$\text{EqLV}(E\sigma'\sigma, F\sigma'\sigma) = \text{EqLV}(E\sigma'\{(x_i, H')\}\sigma_{[-x_i]}, F\sigma'\{(x_i, H')\}\sigma_{[-x_i]}),$$

where $H' = H\{(x_i, H)\}\{(x_i, H)\} \cdots$.

Proof. Let the assumption hold. We can thus write

$$k = \text{EqLV}(E, F) \leq \text{EqLV}(E\sigma'\sigma, F\sigma'\sigma) = e' < e = \text{EqLV}(E\sigma, F\sigma).$$

Hence $E \not\sim F$. Let us fix a witness w for (E, F) ; it has the associated x_i, H by Prop 5(2), and $|w| = k$. We deduce $\sigma(x_i) \sim_{e-k} H\sigma$ (by Prop 5(1)), and $\sigma \sim_{e-k} \{(x_i, H')\}\sigma_{[-x_i]}$ (by Prop. 4(3)).

Since $k \leq k' = \text{EqLV}(E\sigma', F\sigma') \leq e' < e$, we have $\sigma \sim_{e'-k'+1} \{(x_i, H')\}\sigma_{[-x_i]}$. The claim thus follows from Prop. 6(2). \square

3.2 Proof of Theorem 2 (Partly Informal)

Convention (on Small Numbers, and on (A, i) -Sink Words)

In the following reasoning we assume a fixed det-FO grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$. To ease the presentation, we also use informally-sounding words like “small” or “short”. Nevertheless, we will *not further formalize* such usages, since the respective expressions have rigorous meanings: we view a *number* as *small* if it is bounded by an elementary function of $\text{size}(\mathcal{G})$ (independently of any initial terms

T_0, U_0). I.e., we (implicitly) claim in each such case that there is an elementary function $f : \mathbb{N} \rightarrow \mathbb{N}$, independent of our concrete grammar \mathcal{G} , such that the value $f(\text{size}(\mathcal{G}))$ is an upper bound for our “small number”. When we say that a *word*, or a *sequence* in general, is *short*, then we mean that its length is small. A *set* is *small* if its cardinality is small. A *term* is *small* when its (presentation) size is small.

E.g., we can easily check that if U is shortly reachable from W , i.e., if there is a short w such that $W \xrightarrow{w} U$, then there are small *finite* terms H, F such that $W = H\sigma$, $U = F\sigma$ (F being shortly reachable from H) where $\text{RANGE}(\sigma)$ contains only subterms of W occurring in small depths in W . This is based on the fact that performing one transition from a term W results in $E\sigma$ where E is the rhs of a rule in \mathcal{R} and $\text{RANGE}(\sigma)$ contains only depth-1 subterms of W ; we note that $\text{size}(E)$ is bounded by the small number TRANSINC (“Increase by a Transition”) that is equal to the maximal size of the right-hand sides of the rules in \mathcal{R} .

We can explicitly note that $W \xrightarrow{w} U$ implies $\text{size}(U) \leq \text{size}(W) + |w| \cdot \text{TRANSINC}$; if W is finite then also $\text{HEIGHT}(U) \leq \text{HEIGHT}(W) + |w| \cdot \text{TRANSINC}$.

A useful exercise is to note that if there is a word $w \in \Sigma^*$ such that $A(x_1, \dots, x_m) \xrightarrow{w} x_i$, called an (A, i) -*sink word*, then a shortest such word is short. (More details for this claim are in Section 3.3.) We say that any (A, i) -sink word *exposes* the i th *root-successor* in any term $A(V_1, \dots, V_m)$. If there is no (A, i) -sink word, then the i th root-successor V_i in $A(V_1, \dots, V_m)$ is *non-exposable*, and plays no role (i.e., by any change of V_i another equivalent term arises). In fact, we will assume that *all root-successors are exposable* (since the grammar \mathcal{G} can be harmlessly adjusted to satisfy this), and

we fix a shortest (A, i) -sink word $w_{[A, i]}$

for all $A \in \mathcal{N}$, $i \in [1, \text{arity}(A)]$. We also define the following small number:

$$M_0 = 1 + \max \{ |w_{[A, i]}|; A \in \mathcal{N}, i \in [1, \text{arity}(A)] \}. \quad (2)$$

Start of the Proof of Theorem 2

We have fixed a det-FO grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$, and we now assume a witness u_0 for (T_0, U_0) where T_0, U_0 are *finite* terms; hence $T_0 \not\sim U_0$ in $\mathcal{L}_{\mathcal{G}}$ and $|u_0| = \text{EQLV}(T_0, U_0)$. (The restriction to *finite* T_0, U_0 is here technically convenient, not really crucial.) If $u_0 = a_1 a_2 \dots a_k$, then $(T_0, U_0), u_0$ *generate the sequence*

$$(T_0, U_0) \xrightarrow{a_1} (T_1, U_1) \xrightarrow{a_2} \dots \xrightarrow{a_k} (T_k, U_k) \quad (3)$$

where we write $(T, U) \xrightarrow{a} (T', U')$ instead of $T \xrightarrow{a} T', U \xrightarrow{a} U'$. We note that $\text{EQLV}(T_i, U_i) = k - i$; the *sequence* $(T_0, U_0), (T_1, U_1), \dots, (T_k, U_k)$ is thus *eqlevel-decreasing*, i.e., it satisfies $\text{EQLV}(T_0, U_0) > \text{EQLV}(T_1, U_1) > \dots > \text{EQLV}(T_k, U_k)$. Moreover, $a_{i+1} a_{i+2} \dots a_k$ is a witness for (T_i, U_i) .

We prove Theorem 2 by deriving a Tower-bounded function (of $\text{size}(\mathcal{G}, T_0, U_0)$) that bounds the length k of (3); we use two macro-steps described below.

First Macro-step: Controlled Balancing. Both paths $T_0 \xrightarrow{u_0} T_k, U_0 \xrightarrow{u_0} U_k$ in (3) can have “sinking” and “non-sinking” segments distributed quite differently. By sinking we mean “exposing subterms” (corresponding to popping, i.e. stack-height decreasing, in DPDA), by non-sinking we mean the opposite (the stack-height can only increase). We will now show particular “balancing steps” that allow us to stepwise modify the pairs (T_j, U_j) in (3) so that the component terms in the final modified $(\overline{T}_j, \overline{U}_j)$ are more “balanced” (i.e., more “close to each other”) while the eq-levels are kept unchanged (i.e., $\text{EqLv}(\overline{T}_j, \overline{U}_j) = \text{EqLv}(T_j, U_j)$). We will have $(\overline{T}_0, \overline{U}_0) = (T_0, U_0)$, and all $\overline{T}_j, \overline{U}_j$ will be finite terms.

A slight control of balancing steps will give rise to a certain subsequence $(\overline{T}_{i_0}, \overline{U}_{i_0}), (\overline{T}_{i_1}, \overline{U}_{i_1}), \dots, (\overline{T}_{i_\ell}, \overline{U}_{i_\ell})$ of the sequence of pairs in the “balanced version” of (3), where $i_0 = 0$. The subsequence satisfies that each pair $(\overline{T}_{i_j}, \overline{U}_{i_j})$ is “close” to a certain finite term W_j , called a “(balancing) pivot”; this also entails that the heights of \overline{T}_{i_j} and \overline{U}_{i_j} are bounded by $\text{HEIGHT}(W_j) + x$ where x is a small number.

Moreover, the sequence W_0, W_1, \dots, W_ℓ of pivots will be “sufficiently representative” in the sense that for any $r \in [0, k]$ we can bound $\text{HEIGHT}(\overline{T}_r)$ and $\text{HEIGHT}(\overline{U}_r)$ by $\text{HEIGHT}(W_j) + y$ where y is a small number and j is the largest such that $i_j \leq r$. In other words, the heights of terms in the segment $(\overline{T}_{i_j}, \overline{U}_{i_j}), (\overline{T}_{i_{j+1}}, \overline{U}_{i_{j+1}}), (\overline{T}_{i_{j+2}}, \overline{U}_{i_{j+2}}), \dots, (\overline{T}_r, \overline{U}_r)$, where $r = i_{j+1} - 1$ if $j < \ell$ and $r = k$ if $j = \ell$, are bounded by $\text{HEIGHT}(W_j) + z$ for a small number z .

Hence it suffices to bound ℓ (the length of the subsequence) by a Tower-bounded function (of $\text{size}(\mathcal{G}, T_0, U_0)$); this will be achieved by the second macro-step, by using the “pivot-path” $W_0 \xrightarrow{w_1} W_1 \xrightarrow{w_2} \dots \xrightarrow{w_\ell} W_\ell$ that will be precisely defined at the end of this first macro-step.

Balancing Steps

We say that a path $V \xrightarrow{u}$ is *root-performable* if $A(x_1, \dots, x_m) \xrightarrow{u}$ where $A = \text{ROOT}(V)$. For technical convenience we define a *non-sink segment* as a root-performable path $V \xrightarrow{w} V'$ where $|w| = M_0$. (Thus $V \xrightarrow{w} V'$ might expose a root-successor in V , but only by the last step if at all; so $V \xrightarrow{w} V'$ surely “misses” the possibility to expose some root-successor in V as quickly as possible.)

Let $T \xrightarrow{w} T'$ be a non-sink segment for a part

$$(T, U) \xrightarrow{w} (T', U')$$

of (3); hence $(T, U) = (T_{r-M_0}, U_{r-M_0})$, $(T', U') = (T_r, U_r)$, and $w = a_{r-M_0+1} a_{r-M_0+2} \dots a_r$, for some $r \in [M_0, k]$. Then $T = A(V_1, \dots, V_m)$, $A(x_1, \dots, x_m) \xrightarrow{w} G$, and $T \xrightarrow{w[A, j]} V_j$ for each $j \in [1, m]$. We recall that $|w_{[A, j]}| < |w| = M_0$.

Since $T \sim_{M_0} U$, we must have $U \xrightarrow{w[A, j]} V'_j$ for some V'_j , and $\text{EqLv}(V_j, V'_j) > \text{EqLv}(T', U')$ (by Prop. 1(2,3)).

We thus have $T' = G\sigma$ where $\sigma(x_j) = V_j$ for $j \in [1, m]$, and we can

$$\text{replace the pair } (T', U') = (G\sigma, U') \text{ with } (T'', U') = (G\sigma', U')$$

where $\sigma'(x_j) = V'_j$. After replacing (T', U') with (T'', U') , the whole respective suffix of (3) changes: we generate it by (T'', U') , u' where $u' = a_{r+1} a_{r+2} \cdots a_k$. This is safe in the sense that the witness sets (and the eq-levels) for (T'', U') and (T', U') are the same (by Prop. 6(1)).

In our concrete notation, we have transformed (3) to

$$(T_0, U_0) \xrightarrow{a_1} \cdots (T_{r-1}, U_{r-1}) \xrightarrow{a_r} (\widehat{T}_r, U_r) \xrightarrow{a_{r+1}} (\widehat{T}_{r+1}, U_{r+1}) \cdots \xrightarrow{a_k} (\widehat{T}_k, U_k)$$

where $T_{r-1} \xrightarrow{a_r} \widehat{T}_r$ might be not a valid transition but the eq-levels have not changed, i.e., $\text{EqLV}(\widehat{T}_{r+i}, U_{r+i}) = \text{EqLV}(T_{r+i}, U_{r+i})$ for $i \in [0, k-r]$. (The notation \widehat{T}_j should not be mixed with \overline{T}_j discussed previously; any T_j can undergo several changes, though at most one “balancing”, before becoming the “final” \overline{T}_j .)

Since w is short ($|w| = M_0$), the pair (T'', U') (i.e. (\widehat{T}_r, U_r)) is “balanced” in the sense that both terms are “close” to one term, namely to U : we have $(T'', U') = (G\sigma', U')$ where G is a small finite term (since shortly reachable from $A(x_1, \dots, x_m)$), and $\text{RANGE}(\sigma')$ contains only terms that are shortly reachable from U , and U' is itself shortly reachable from U . We capture the discussed closeness by the notation

$$U \models (T'', U'),$$

and we also say that U is the *pivot* of our *balancing step*, while (T'', U') is called the *balancing result*.

Analogously to the described *left-balancing step*, where we replace $(G\sigma, U')$ with $(G\sigma', U')$, we define a *right-balancing step*, where $(U \xrightarrow{w} U')$ is a non-sink segment and) we replace $(T', G\sigma)$ with $(T', G\sigma')$; here T is the pivot and we have $T \models (T', G\sigma')$.

An important feature of our (ternary) predicate \models is that

$$W \models (T, U) \text{ implies that } W = H\sigma, T = E\sigma, U = F\sigma$$

for some small finite terms H, E, F , where $\text{RANGE}(\sigma)$ contains only subterms of W occurring in small depths in W ; moreover, at least one of E, F is shortly reachable from H (and thus at least one of T, U is shortly reachable from W). (A precise definition of $W \models (T, U)$ is given in Section 3.3.)

Quadruple-Sequences, and Pivot Paths

We use the described balancing steps as follows. We first artificially create a “pivot” $W_0 = B(T_0, U_0)$ for a (possibly added) nonterminal B so that we have $W_0 \models (T_0, U_0)$. Our aim is to transform (3), by stepwise balancing, into a certain sequence of a different kind, namely to

$$((W_0, T_0, U_0), u_0) \rightsquigarrow ((W_1, T'_1, U'_1), u_1) \rightsquigarrow \cdots \rightsquigarrow ((W_\ell, T'_\ell, U'_\ell), u_\ell) \quad (4)$$

where also $W_i \models (T'_i, U'_i)$ for $i \in [1, \ell]$ and u_i is a proper suffix of u_{i-1} and a witness for (T'_i, U'_i) . (The sequence $(T_0, U_0), (T'_1, U'_1), \dots, (T'_\ell, U'_\ell)$ will be the subsequence $(\overline{T}_{i_0}, \overline{U}_{i_0}), (\overline{T}_{i_1}, \overline{U}_{i_1}), \dots, (\overline{T}_{i_\ell}, \overline{U}_{i_\ell})$ discussed earlier.)

We start with the one-element sequence $((W_0, T_0, U_0), u_0)$. Then we traverse (3) from left to right, and we perform a first possible balancing step (if there is any non-sink segment in $T_0 \xrightarrow{u_0}$ or $U_0 \xrightarrow{u_0}$). We prolong our sequence to $((W_0, T_0, U_0), u_0) \rightsquigarrow ((W_1, T'_1, U'_1), u_1)$ where W_1 is the pivot and (T'_1, U'_1) the balancing result of this step; u_1 is the suffix of u_0 that is a witness for (T'_1, U'_1) .

Now we continue, by traversing the sequence generated by $(T'_1, U'_1), u_1$, or by $(T'_j, U'_j), u_j$ in general; this generated sequence corresponds to the current version of the respective suffix of the stepwise modified (3). We look for the first possible balancing step in this sequence, with *one proviso*: if we have $W_j \models (G\sigma', U'_j)$ after a left-balancing step, then the next balancing step can be a right-balancing (thus changing the pivot-side) only if the “rest-head” G has been in the meantime erased, i.e., some $\sigma'(x_i)$ (that is shortly reachable from W_j) has been exposed. Hence shortly after a left-balancing step corresponding to $W_j \models (G\sigma', U'_j)$ either another left-balancing step is performed or G is erased and balancing at both sides is again allowed.

After any right-balancing step, an analogous proviso applies. This guarantees that W_{j+1} is reachable from W_j , by a certain path $W_j \xrightarrow{w_{j+1}} W_{j+1}$ in which at most *a small number of non-sink segments occurs*: either the path $W_j \xrightarrow{w_{j+1}} W_{j+1}$ is short, or it has a short prefix after which no non-sink segment occurs. (More details are in Section 3.3.)

We finish creating the sequence (4) when the paths $T'_\ell \xrightarrow{u_\ell}$, $U'_\ell \xrightarrow{u_\ell}$ do not allow further balancing; it is clear that these paths then must be either short or sinking all the time (in which case the heights of terms in the paths $T'_\ell \xrightarrow{u_\ell}$, $U'_\ell \xrightarrow{u_\ell}$ are successively decreasing).

To summarize, from the sequence (3), whose length k we want to bound, we have come to the sequence (4) that also has the associated *pivot-path*

$$W_0 \xrightarrow{w_1} W_1 \xrightarrow{w_2} \dots \xrightarrow{w_\ell} W_\ell, \quad (5)$$

where the number of non-sink segments in each subpath $W_{j-1} \xrightarrow{w_j} W_j$ is small.

Second Step: Deriving a Tower-bound from a Pivot Path. As we made clear, for each term W there is only a small number of pairs (T, U) such that $W \models (T, U)$. Since $(T_0, U_0), (T'_1, U'_1), (T'_2, U'_2), \dots, (T'_\ell, U'_\ell)$ in (4) is an eqlevel-decreasing sequence, we have no repeat of a pair here, and thus the number of occurrences of each concrete W_j in the sequence W_0, W_1, \dots, W_ℓ is small.

We can intuitively note that if the sequence (3) is “very long” (w.r.t. $\text{size}(\mathcal{G}, T_0, U_0)$) then the maximal $\text{HEIGHT}(W_j)$ is “much larger” than $\text{HEIGHT}(W_0)$. In this case the pivot path (5) must have “long increasing segments”. In a long increasing segment the pivots W_j are “frequent” since each (sub)path $W_{j-1} \xrightarrow{w_j} W_j$ has at most a small number of non-sink segments.

We formalize this intuition by help of “stair sequences” that correspond to the standard “stair-growing” stack-contents in a path from one PDA-configuration to a larger configuration.

Stair Sequences. Let us present (5) as

$$W_0 \xrightarrow{w'_1} V_1 \xrightarrow{w''_1} W_1 \xrightarrow{w'_2} V_2 \xrightarrow{w''_2} W_2 \cdots \xrightarrow{w'_\ell} V_\ell \xrightarrow{w''_\ell} W_\ell \quad (6)$$

where V_j is the term in the path $W_{j-1} \xrightarrow{w_j} W_j$ for which the respective w'_j is the shortest possible such that the path $V_j \xrightarrow{w''_j} W_j$ does not expose any root-successor in V_j (and is thus also root-performable). There is always such V_j ; in some cases we might have $V_j = W_{j-1}$ ($w'_j = \varepsilon$) or $V_j = W_j$ ($w''_j = \varepsilon$).

Hence $V_j = A(x_1, \dots, x_m)\sigma$ and $W_j = G\sigma$ where $A(x_1, \dots, x_m) \xrightarrow{w''_j} G$ and $\text{RANGE}(\sigma)$ contains root-successors in V_j ; since the number of non-sink segments in $W_{j-1} \xrightarrow{w_j} W_j$ is small, G is a small finite term. We say that a subsequence

$$i_1 < i_2 < \cdots < i_r \quad (7)$$

of the sequence $1, 2, \dots, \ell$ is a *stair sequence* if for each $j \in [1, r-1]$ the subpath

$$V_{i_j} \xrightarrow{w''_{i_j}} V_{i_{j+1}} \xrightarrow{w'_{i_{j+1}}} \cdots \xrightarrow{w''_{i_r-1}} V_{i_r}$$

of (6) does not expose any root-successor in V_{i_j} (and is thus root-performable). Moreover, for convenience we require that the sequence is maximal in the sense that we would violate the above condition by inserting any i where $i_j < i < i_{j+1}$ for some $j \in [1, r-1]$. This requirement guarantees the *small-stair property*: for any $j \in [1, r-1]$ we have $V_{i_j} = A(x_1, \dots, x_m)\sigma$ and $V_{i_{j+1}} = F\sigma$ where $A(x_1, \dots, x_m) \xrightarrow{w''_{i_j}} V_{i_{j+1}} \xrightarrow{w'_{i_{j+1}}} \cdots \xrightarrow{w''_{i_r-1}} F$ and F is a small finite term (though *not all* terms in the path $A(x_1, \dots, x_m) \rightarrow \cdots \rightarrow F$ are claimed to be small).

It Suffices to Get a Tower-bound on the Lengths of Stair Sequences

Later we show that the lengths of stair sequences are bounded by a Tower-bounded function f of $\text{size}(\mathcal{G})$, independently of T_0, U_0 ; now we assume such f . For any pivot W_j (in the sequence (5)) we then obviously have

$$\text{HEIGHT}(W_j) \leq \text{HEIGHT}(W_0) + \text{stair} \cdot (1 + f(\text{size}(\mathcal{G})))$$

where *stair* is an appropriate small number.

Since the number of (finite) terms with the height at most $H \in \mathbb{N}$ is bounded by $y \uparrow (m \uparrow H)$ for some small numbers m, y (recall that \uparrow is the exponential operator), we easily deduce that the number of elements of the sequence (4) is bounded by a Tower-bounded function (of $\text{size}(\mathcal{G}, T_0, U_0)$).

For $j \in [0, \ell]$ we consider $((W_j, T'_j, U'_j), u_j)$ in (4), where we put $(T'_0, U'_0) = (T_0, U_0)$. Let $u_j = wu_{j+1}$; we put $w = u_\ell$ when $j = \ell$. It is easy to check that the heights of terms on the paths $T'_j \xrightarrow{w} \cdots$ and $U'_j \xrightarrow{w} \cdots$ are bounded by $\text{HEIGHT}(W_j) + x$ where x is a small number. Since the sequence generated by $(T'_j, U'_j), w$ is eqlevel-decreasing, and thus has no repeat, we get that $|w|$ is bounded by the value $(y \uparrow (m \uparrow (\text{HEIGHT}(W_j) + x)))^2$ for some small m, x, y .

We thus routinely derive a Tower-bounded function (of $\text{size}(\mathcal{G}, T_0, U_0)$) that bounds the length of u_0 and thus $\text{EQLV}(T_0, U_0)$. Hence after we show the

promised Tower-bounded function f (of $\text{size}(\mathcal{G})$) that bounds the lengths of stair sequences, the proof of Theorem 2 will be finished.

A Tower-bound on the Lengths of Stair Sequences

Let us fix a stair sequence $i_1 < i_2 < \dots < i_r$, referring to the notation around (7). We now show that $r \leq f(\text{size}(\mathcal{G}))$ for a Tower-bounded function f (independent of \mathcal{G}).

By our definition, which also implies the small-stair property, we can easily observe that the sequence $V_{i_1}, V_{i_2}, V_{i_3}, \dots, V_{i_r}$ can be presented as

$$A_1\sigma', A_2\sigma'_1\sigma', A_3\sigma'_2\sigma'_1\sigma', \dots, A_r\sigma'_{r-1}\sigma'_{r-2}\dots\sigma'_1\sigma'$$

where we write shortly A_j instead of $A_j(x_1, \dots, x_{m_j})$ and where

$A_j(x_1, \dots, x_{m_j}) \xrightarrow{w} A_{j+1}(x_1, \dots, x_{m_{j+1}})\sigma'_j$ for the respective w from (6), i.e. $w = w'_{i_j} \dots w'_{i_{j+1}}$. Hence the supports of σ' and σ'_j are subsets of $\{x_1, x_2, \dots, x_m\}$ where m is the maximal arity of nonterminals, and $\text{RANGE}(\sigma'_j)$ contains the root-successors in the small finite term F such that $A_j(x_1, \dots, x_{m_j}) \xrightarrow{w} F$. Hence σ'_j are small, i.e., they are small sets of small pairs of the form (x_i, E) .

By the definition of (6), we can present $W_{i_1}, W_{i_2}, W_{i_3}, \dots, W_{i_r}$ as

$$G_1\sigma', G_2\sigma'_1\sigma', G_3\sigma'_2\sigma'_1\sigma', \dots, G_r\sigma'_{r-1}\sigma'_{r-2}\dots\sigma'_1\sigma' \quad (8)$$

where G_j are small finite terms. Recalling the discussion around the introduction of $W \models (T, U)$ (namely the form $W = H\sigma, T = E\sigma, U = F\sigma$), it is useful to rewrite (8) as

$$H_1\sigma, H_2\rho_1\sigma, H_3\rho_2\rho_1\sigma, \dots, H_r\rho_{r-1}\rho_{r-2}\dots\rho_1\sigma$$

where H_j are also small finite terms, but maybe with larger heights than G_j , guaranteeing that the sequence $(T'_{i_1}, U'_{i_1}), (T'_{i_2}, U'_{i_2}), \dots, (T'_{i_r}, U'_{i_r})$ (extracted from (4)) can be presented as

$$(E_1\sigma, F_1\sigma), (E_2\rho_1\sigma, F_2\rho_1\sigma), \dots, (E_r\rho_{r-1}\rho_{r-2}\dots\rho_1\sigma, F_r\rho_{r-1}\rho_{r-2}\dots\rho_1\sigma) \quad (9)$$

where all E_j, F_j are small. This forces us to increase the supports of σ and ρ_j comparing to σ' and σ'_j (since $\text{RANGE}(\sigma)$ contains deeper subterms of V_{i_1}) but it is obvious that we can take the supports of σ and of all ρ_j as subsets of

$$\text{SUP}_0 = \{x_1, x_2, \dots, x_{n_0}\} \quad (10)$$

where n_0 is small. Moreover, all terms in $\text{RANGE}(\rho_j)$ are small (but this is not claimed for σ). (We use the symbol “ ρ ” instead of a variant of “ σ ” to stress the small size of all ρ_j .) Given σ , (9) is fully determined by the sequence of triples

$$(E_1, F_1, \rho_0) (E_2, F_2, \rho_1) \dots (E_r, F_r, \rho_{r-1}) \quad (11)$$

which can be viewed as a word in a small alphabet AL (consisting of the respective triples); we use ρ_0 for uniformity, defining it as the empty-support substitution.

We now note a useful combinatorial fact. Let us define a function $h : \mathbb{N} \rightarrow \mathbb{N}$ inductively as follows:

$h(0) = 1$, and $h(j+1) = h(j) \cdot (1 + q^{h(j)})$ where $q = \text{CARD}(\text{AL})$.

Viewing q as a constant, h is obviously a Tower-bounded function. The pigeon-hole principle implies that in any $h(1)$ -long segment of (11), i.e. in any segment with length $h(1) = 1 + \text{CARD}(\text{AL})$, there are two different occurrences of one element of AL . Moreover, in any $h(j+1)$ -long segment there are two different, non-overlapping, occurrences of one $h(j)$ -long segment.

We aim to show that $r \leq h(n_0+1)$, where r is the length of our stair sequence, as well as of the sequences (9) and (11), and $n_0 = \text{CARD}(\text{SUP}_0)$ is introduced in (10). Since n_0 and $q = \text{CARD}(\text{AL})$ are small numbers (bounded by elementary functions of $\text{size}(\mathcal{G})$), the value $h(n_0+1)$ is obviously bounded by $g(\text{size}(\mathcal{G}))$ for a Tower-bounded function g . Hence after we show $r \leq h(n_0+1)$ (in the following last part of this section), the proof of Theorem 2 will be finished.

Length r of Any Stair Sequence Is Less Than $h(n_0+1)$

We first introduce certain “recurrent-pattern” sequences, now using *general regular* terms and substitutions as ingredients, with *no size-restrictions*.

We define (n, ℓ) -presentations where $n, \ell \in \mathbb{N}$ (not to be mixed with ℓ in (5)). Each (n, ℓ) -presentation presents a sequence with 2^ℓ elements where an element is a pair of (regular) terms. A sequence that can be presented by an (n, ℓ) -presentation is called an (n, ℓ) -sequence. An (n, ℓ) -presentation consists of

- a set $\text{SUP} \subseteq \text{VAR}$ where $\text{CARD}(\text{SUP}) \leq n$,
- a pair (E, F) of (regular) terms, and
- substitutions $\sigma_1, \sigma_2, \dots, \sigma_\ell$ and σ whose supports are subsets of SUP .

If $\ell = 0$, then the presented $(n, 0)$ -sequence is the one-element sequence $(E\sigma, F\sigma)$. If $\ell > 0$, then the presented sequence (with 2^ℓ elements) arises by concatenating two $(n, \ell-1)$ -sequences: the first half (with $2^{\ell-1}$ elements) is presented by SUP , (E, F) , $\sigma_1, \sigma_2, \dots, \sigma_{\ell-1}$, σ , and the second half by SUP , (E, F) , $\sigma_1, \sigma_2, \dots, \sigma_{\ell-1}$, and $\sigma' = \sigma_\ell \sigma$.

An example of an $(n, 2)$ -sequence is

$$(E\sigma, F\sigma), (E\sigma_1\sigma, F\sigma_1\sigma), (E\sigma_2\sigma, F\sigma_2\sigma), (E\sigma_1\sigma_2\sigma, F\sigma_1\sigma_2\sigma) \quad (12)$$

if the supports of $\sigma, \sigma_1, \sigma_2$ are subsets of SUP with $\text{CARD}(\text{SUP}) \leq n$. If also $\text{SUPP}(\sigma_3) \subseteq \text{SUP}$, and we replace σ in (12) with $\sigma_3\sigma$, we get

$$(E\sigma_3\sigma, F\sigma_3\sigma), (E\sigma_1\sigma_3\sigma, F\sigma_1\sigma_3\sigma), (E\sigma_2\sigma_3\sigma, F\sigma_2\sigma_3\sigma), (E\sigma_1\sigma_2\sigma_3\sigma, F\sigma_1\sigma_2\sigma_3\sigma).$$

Put together, the above 8 pairs constitute an $(n, 3)$ -sequence, presented by SUP , (E, F) , $\sigma_1, \sigma_2, \sigma_3$, and σ .

We now prove the next claim, which also implies that any $h(n_0+1)$ -long segment of the (eqlevel-decreasing) sequence (9) contains an (n_0, n_0+1) -subsequence (i.e. a subsequence that is an (n_0, n_0+1) -sequence).

Claim. *Any $h(\ell)$ -long segment of (9) contains an (n_0, ℓ) -subsequence.*

Proof. We give an inductive definition (based on ℓ) of so called *good* (n_0, ℓ) -presentations. It will be guaranteed that each $h(\ell)$ -long segment of (9) has an (n_0, ℓ) -subsequence with a *good* (n_0, ℓ) -presentation.

Any $h(0)$ -long segment is an element $(E_i \rho_{i-1} \rho_{i-2} \dots \rho_1 \sigma, F_i \rho_{i-1} \rho_{i-2} \dots \rho_1 \sigma)$; it trivially constitutes an $(n_0, 0)$ -sequence, and we define its good $(n_0, 0)$ -presentation as $\text{SUP}, (E, F), \bar{\sigma}$ where $(E, F) = (E_i, F_i)$ and $\bar{\sigma} = \rho_{i-1} \rho_{i-2} \dots \rho_1 \sigma$.

For a $h(\ell+1)$ -long segment S of (9) we define a respective good $(n_0, \ell+1)$ -presentation as follows. We take two $h(\ell)$ -long non-overlapping subsegments S_1, S_2 of S such that the respective “images” of S_1 and S_2 in (11) are the same. (This is possible by the definition of h .)

Let $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_\ell, \bar{\sigma}$ be a good (n_0, ℓ) -presentation of a subsequence of S_1 , starting at (the relative) position p_1^{rel} inside S_1 and at (the absolute) position p_1^{abs} in (9); let p_2^{abs} denote the position in (9) that corresponds to the relative position p_1^{rel} in S_2 . Our (inductive) construction of good presentations guarantees that $(E, F) = (E_{p_1^{abs}}, F_{p_1^{abs}})$ and $\bar{\sigma} = \rho_{p_1^{abs}-1} \rho_{p_1^{abs}-2} \dots \rho_1 \sigma$. Another (inductive) property of good presentations is that $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_\ell, \sigma_{\ell+1} \bar{\sigma}$, where where we put $\sigma_{\ell+1} = \rho_{p_2^{abs}-1} \rho_{p_2^{abs}-2} \dots \rho_{p_1^{abs}}$, is a good (n_0, ℓ) -presentation of a subsequence of S_2 (since the images of S_1 and S_2 in (11) are the same). (Our definition in the case $\ell = 0$ indeed guarantees these two properties.)

Then $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_{\ell+1}, \bar{\sigma}$ is defined to be a good $(n_0, \ell+1)$ -presentation, presenting a subsequence of the $h(\ell+1)$ -long segment S . The above used properties of good (n_0, ℓ) -presentations are obviously guaranteed for the defined good $(n_0, \ell+1)$ -presentation as well. \square

We observe that all elements of a $(0, \ell)$ -sequence are the same. There is thus no eqlevel-decreasing $(0, \ell)$ -sequence for $\ell > 0$. The next claim shows that an eqlevel-decreasing (n, ℓ) -sequence with $\ell > 0$ gives rise to an eqlevel-decreasing $(n-1, \ell-1)$ -sequence (arising from the original even-index elements by subterm replacing that does not change the respective eq-levels). This implies that

there is no eqlevel-decreasing (n, ℓ) -sequence where $n < \ell$;

in particular, there is no eqlevel-decreasing (n_0, n_0+1) -sequence. Using the previous claim, we deduce that there is no $h(n_0+1)$ -long segment in (9); this implies $r < h(n_0+1)$, and the proof of Theorem 2 is finished.

Claim. *Let e_j denote the eq-level of the j th element of an (n, ℓ) -sequence, and assume $\ell > 0$ and $e_1 > e_2 > \dots > e_{2\ell}$. Then $n > 0$ and there is an $(n-1, \ell-1)$ -sequence in which the eq-level of its j th element is e_{2j} .*

Proof. Let Seq be an (n, ℓ) -sequence, presented by $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_\ell$, and σ , where $\ell > 0$ and $e_1 > e_2 > \dots > e_{2\ell}$ are the eq-levels of the elements of Seq in the respective order. We must obviously have $n > 0$.

Since $\text{EqLv}(E\sigma, F\sigma) = e_1 > e_2 = \text{EqLv}(E\sigma_1\sigma, F\sigma_1\sigma)$, we can fix some $x_i \in \text{SUP}$ and $H \neq x_i$, where $E \xrightarrow{w} x_i$ and $F \xrightarrow{w} H$, or vice versa, for a witness w for (E, F) . (This follows from Prop. 5(2).) We put

$$H' = H\{(x_i, H)\}\{(x_i, H)\} \dots, \text{ and } \sigma'_j = \sigma_j \{(x_i, H')\} \text{ (for all } j \in [1, \ell]).$$

We note that $\sigma'_j \bar{\sigma} = \sigma'_j \bar{\sigma}_{[-x_i]}$ for any $\bar{\sigma}$ (where $\bar{\sigma}_{[-x_i]}$ arises from $\bar{\sigma}$ by putting $\bar{\sigma}_{[-x_i]}(x_i) = x_i$). We say that the above (n, ℓ) -presentation of Seq (accompanied) with x_i, H gives rise to the $(n-1, \ell-1)$ -sequence Seq' presented by

$$\text{SUP} \setminus \{x_i\}, (E\sigma'_1, F\sigma'_1), (\sigma'_2)_{[-x_i]}, (\sigma'_3)_{[-x_i]}, \dots, (\sigma'_\ell)_{[-x_i]}, \sigma_{[-x_i]}. \quad (13)$$

We now show that Seq' satisfies the desired condition, i.e., the eq-levels of its elements are $e_2 > e_4 > e_6 > \dots > e_{2\ell}$. We prove this by induction on ℓ .

By Cor. 7 we deduce that $\text{EqLv}(E\sigma'_1\sigma_{[-x_i]}, F\sigma'_1\sigma_{[-x_i]}) = e_2$, since

$$\text{EqLv}(E\sigma_1\sigma, F\sigma_1\sigma) = \text{EqLv}(E\sigma_1\{(x_i, H')\}\sigma_{[-x_i]}, F\sigma_1\{(x_i, H')\}\sigma_{[-x_i]}).$$

If $\ell = 1$, then we are done.

If $\ell > 1$, then the $(n, \ell-1)$ -presentation $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_{\ell-1}, \sigma$ (presenting the first half of Seq) with x_i, H gives rise to the $(n-1, \ell-2)$ -sequence Seq'_1 that is the first half of Seq' ; hence Seq'_1 is presented by $\text{SUP} \setminus \{x_i\}, (E\sigma'_1, F\sigma'_1), (\sigma'_2)_{[-x_i]}, (\sigma'_3)_{[-x_i]}, \dots, (\sigma'_{\ell-1})_{[-x_i]}, \sigma_{[-x_i]}$. By the induction hypothesis, the eq-levels in Seq'_1 are $e_2, e_4, \dots, e_{2\ell-1}$.

The $(n, \ell-1)$ -presentation $\text{SUP}, (E, F), \sigma_1, \sigma_2, \dots, \sigma_{\ell-1}, \sigma_\ell\sigma$ (presenting the second half of Seq) with x_i, H gives rise to the $(n-1, \ell-2)$ -sequence Seq''_2 presented by $\text{SUP} \setminus \{x_i\}, (E\sigma'_1, F\sigma'_1), (\sigma'_2)_{[-x_i]}, (\sigma'_3)_{[-x_i]}, \dots, (\sigma'_{\ell-1})_{[-x_i]}, (\sigma_\ell\sigma)_{[-x_i]}$. By the induction hypothesis, the eq-levels in Seq''_2 are $e_{2\ell-1+2}, e_{2\ell-1+4}, \dots, e_{2\ell}$.

By another (repeated) use of Cor. 7, the eq-levels in Seq''_2 do not change when we replace $(\sigma_\ell\sigma)_{[-x_i]}$ with $(\sigma_\ell\{(x_i, H')\}\sigma_{[-x_i]})_{[-x_i]} = (\sigma'_\ell)_{[-x_i]}\sigma_{[-x_i]}$ in the presentation. By this replacing we get the sequence Seq'_2 that is the second half of Seq' , in fact. The proof is thus finished. \square

3.3 Formalizing Informal Parts of the Proof of Theorem 2

In this section we just add more formal details to some parts of the proof in Section 3.2 where the reasoning might look too informal. We assume a given det-FO grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$.

Sink Words, Normal-Form Grammars, Small Number M_0

For every $A \in \mathcal{N}$ and $i \in [1, m]$ where $m = \text{arity}(A)$ we say that $w \in \Sigma^*$ is an (A, i) -sink word if $A(x_1, \dots, x_m) \xrightarrow{w} x_i$. We can compute a shortest (A, i) -sink word $w_{[A, i]}$ for each (A, i) for which such a word exists. The lengths of some $w_{[A, i]}$ can be exponential in $\text{size}(\mathcal{G})$, but we can compute these lengths (and concise presentations of $w_{[A, i]}$) by a polynomial algorithm based on dynamic programming. The essential fact is that the length of a shortest (A, i) -sink word is $1 + |v|$ where v is a shortest word such that $E \xrightarrow{v} x_i$ for the right-hand side E of a rule $A(x_1, \dots, x_m) \xrightarrow{a} E$ in \mathcal{R} . If $E = x_i$ then $v = \varepsilon$; otherwise v can be composed from the (A, i) -sink words corresponding to a respective branch in the syntactic tree of E .

We assume that the grammar \mathcal{G} is in the normal form, i.e., there is $w_{[A, i]}$ for each (A, i) . This is harmless: if there is no such word for a concrete pair A, i , then we can decrease the arity of A and modify the rules in \mathcal{R} accordingly, while the LTS $\mathcal{L}_{\mathcal{G}}$ remains unchanged, in fact.

It is now also quite clear that M_0 defined by (2) is indeed a small number.

Closeness Predicate \models **Defined via** $\models_B, \models_R, \models_L$

We recall the small number TRANSINC (the maximal size of the rhs of a rule in \mathcal{R}) that also bounds the term-height increase caused by performing one transition, and we define another small number:

$$M_1 = \text{TRANSINC} \cdot (M_0)^2 + 2 \cdot M_0.$$

The points 1-4 below define (a technically convenient form of) the predicate $W \models (T, U)$, divided into three (not necessarily disjoint) cases \models_B (“both sides reachable”), \models_R (“right-hand side reachable”), and \models_L (“left-hand side reachable”). We say that a term T is k -reachable from W if $W \xrightarrow{|w|} T$ where $|w| \leq k$.

- 1) $W \models_B (T, U)$ if each of T, U is M_1 -reachable from W .
- 2) $W \models_R (T, U)$ if U is M_0 -reachable from W and $T = G\sigma$ where $\text{HEIGHT}(G) \leq M_0 \cdot \text{TRANSINC}$ and $\sigma(x_i)$ is M_0 -reachable from W for each x_i in G .
- 3) $W \models_L (T, U)$ if T is M_0 -reachable from W and $U = G\sigma$ where $\text{HEIGHT}(G) \leq M_0 \cdot \text{TRANSINC}$ and $\sigma(x_i)$ is M_0 -reachable from W for each x_i in G .
- 4) $W \models (T, U)$ if $W \models_B (T, U)$ or $W \models_L (T, U)$ or $W \models_R (T, U)$.

We can easily verify a claim from Section 3.2: if $W \models (T, U)$, then $W = H\sigma$, $T = E\sigma$, $U = F\sigma$ for some small finite terms H, E, F , where $\text{RANGE}(\sigma)$ contains only subterms of W occurring in small depths in W .

Relation \rightsquigarrow on the Set of Quadruples $((W, T, U), u)$

Now we define the relation \rightsquigarrow (used in (4)), by the following (deduction) rules divided into the cases a), b), c). In fact, formally we introduce the relations $\rightsquigarrow^{\text{BAL}}$ (“balance”) and $\rightsquigarrow^{\text{POST}}$ (“postpone”) where $\rightsquigarrow = \rightsquigarrow^{\text{BAL}} \cup \rightsquigarrow^{\text{POST}} \cdot \rightsquigarrow^{\text{BAL}}$. We assume that

$$W \models (T, U) \text{ and } u \text{ is a witness for } (T, U),$$

and we describe $((W', T', U'), u')$ such that $((W, T, U), u) \rightsquigarrow^{\text{BAL}} ((W', T', U'), u')$ or $((W, T, U), u) \rightsquigarrow^{\text{POST}} ((W', T', U'), u')$; at the same time we verify that $W' \models (T', U')$ and that u' is a proper suffix of u and a witness for (T', U') .

a) $W \models_B (T, U)$.

Suppose that u has the *shortest* prefix w such that one of the paths $T \xrightarrow{w}$, $U \xrightarrow{w}$ finishes by a non-sink segment.

- i) Suppose $w = u_1 u_2$ where $T \xrightarrow{u_1} T_1 \xrightarrow{u_2} T_2$ and $T_1 \xrightarrow{u_2} T_2$ is a non-sink segment; we thus have $|u_2| = M_0$, $T_1 = A(V_1, \dots, V_m)$, and $T_2 = G\sigma$ where $A(x_1, \dots, x_m) \xrightarrow{u_2} G$ and $\sigma(x_i) = V_i$ (for $i \in [1, m]$).

Let $U \xrightarrow{u_1} U_1 \xrightarrow{u_2} U_2$ and $u = u_1 u_2 u'$. Then we deduce

$$((W, T, U), u) \rightsquigarrow^{\text{BAL}} ((U_1, G\sigma', U_2), u')$$

putting $\sigma'(x_i) = V_i'$ where $U_1 \xrightarrow{w(A, i)} V_i'$.

We can verify that $U_1 \models_R (G\sigma', U_2)$. Moreover, u' is a witness for $(G\sigma', U_2)$ (by using Prop. 6(1)).

- ii) If $w = u_1 u_2$ where $U \xrightarrow{u_1} U_1 \xrightarrow{u_2} U_2$ and $U_1 \xrightarrow{u_2} U_2$ is a non-sink segment, then we proceed symmetrically and deduce

$$((W, T, U), u) \overset{\text{BAL}}{\rightsquigarrow} ((T_1, T_2, G\sigma'), u').$$

Here $T_1 \models_L (T_2, G\sigma')$, and u' is a witness for $(T_2, G\sigma')$.

b) $W \not\models_B (T, U)$ and $W \models_R (T, U)$.

Hence $W \models_R (G\sigma, U)$ where $G\sigma$ is a presentation of T in the form of the definition of \models_R (in the point 2); thus $\text{HEIGHT}(G) \leq M_0 \cdot \text{TRANSINC}$, and G is not a variable since $W \not\models_B (T, U)$. Suppose now that u has the shortest prefix w such that one of the following two conditions holds:

i) $G \xrightarrow{w} x_i$ (hence $G\sigma \xrightarrow{w} \sigma(x_i)$ where $\sigma(x_i)$ is M_0 -reachable from W), or
 ii) $w = u_1 u_2$ where $G \xrightarrow{u_1} G_1 \xrightarrow{u_2} G_2$ and $G_1 \xrightarrow{u_2} G_2$ is a non-sink segment. We note that if there is no such w , then u is short. Here we assume $u = wu'$. In the case i), $G \xrightarrow{w} x_i$, we take U' such that $U \xrightarrow{w} U'$ and we deduce

$$((W, G\sigma, U), u) \overset{\text{POST}}{\rightsquigarrow} ((W, \sigma(x_i), U'), u').$$

We note that $W \models_B (\sigma(x_i), U')$: since $|w| \leq (1 + M_0 \cdot \text{TRANSINC}) \cdot M_0$ and U is M_0 -reachable from W , we have that U' is M_1 -reachable from W ; and $\sigma(x_i)$ is even M_0 -reachable from W .

In the case ii) we proceed as in a), i.e., for $U \xrightarrow{u_1} U_1 \xrightarrow{u_2} U_2$ we deduce

$$((W, G\sigma, U), u) \overset{\text{BAL}}{\rightsquigarrow} ((U_1, G'\sigma', U_2), u')$$

accordingly (where $G_1\sigma = A'(x_1, \dots, x_{m'})\sigma'' \xrightarrow{u_2} G'\sigma'' = G_2\sigma$ is the non-sink segment). Here $U_1 \models_R (G'\sigma', U_2)$ and u' is a witness for $(G'\sigma', U_2)$.

c) $W \not\models_B (T, U)$ and $W \models_L (T, G\sigma)$.

This case is analogous to b). We can here deduce

$$((W, T, G\sigma), u) \overset{\text{POST}}{\rightsquigarrow} ((W, T', \sigma(x_i)), u')$$

where $W \models_B (T', \sigma(x_i))$ and u' is a witness for $(T', \sigma(x_i))$, or

$$((W, T, G\sigma), u) \overset{\text{BAL}}{\rightsquigarrow} ((T_1, T_2, G'\sigma'), u')$$

where $T_1 \models_L (T_2, G'\sigma')$ and u' is a witness for $(T_2, G'\sigma')$.

We recall that $\rightsquigarrow = \overset{\text{BAL}}{\rightsquigarrow} \cup \overset{\text{POST}}{\rightsquigarrow} \cdot \overset{\text{BAL}}{\rightsquigarrow}$, and we note that $\overset{\text{POST}}{\rightsquigarrow} \cdot \overset{\text{POST}}{\rightsquigarrow}$ is empty.

We can easily verify the next claims:

1. If $((W, T, U), u) \overset{\text{BAL}}{\rightsquigarrow} ((W', T', U'), u')$ by a), then W' is M_0 -reachable from a subterm of a term (T or U) that is M_1 -reachable from W .
2. If $((W, T, U), u) \overset{\text{POST}}{\rightsquigarrow} ((W', T', U'), u')$ then $W' = W$.
3. If $((W, T, U), u) \overset{\text{BAL}}{\rightsquigarrow} ((W', T', U'), u')$ by b) or c), then W' is M_1 -reachable from W .

Hence $((W, T, U), u) \rightsquigarrow ((W', T', U'), u')$ implies that $W \xrightarrow{v_1} W'_1 \xrightarrow{v_2} W'_2 \xrightarrow{v_3} W'$ for some v_1, v_2, v_3 and W'_1, W'_2 where $|v_1| \leq M_1$, $|v_3| \leq M_0$, and $W'_1 \xrightarrow{v_2} W'_2$ sinks from W'_1 to its subterm W'_2 : we take v_2 as the sequence of the appropriate (A, i) -sink words $w_{[A, i]}$ (along the respective branch in the syntactic tree of W'_1).

We fix such a path $W \xrightarrow{w} W'$, where $w = v_1 v_2 v_3$, for each $((W, T, U), u) \rightsquigarrow ((W', T', U'), u')$. It is obvious that $W \xrightarrow{w} W'$ contains at most a small number of non-sink segments.

We have thus formalized deriving a pivot-path (5) from the sequence (4).

4 Bisimilarity of FO Grammars Is Ackermann-Hard

Here we prove the next theorem, referring to the notions discussed in Section 2.4.

Theorem 8. *Bisimilarity of first-order grammars is Ackermann-hard.*

We do not give a direct reduction from HP-ACK, since using Lemma 9 below is much more convenient. We define the necessary notions first.

Reset Counter Machines (RCMs). An RCM is a tuple $\mathcal{M} = (d, Q, \delta)$ where d is the *dimension*, yielding d nonnegative *counters* c_1, c_2, \dots, c_d , Q is a finite set of (*control*) *states*, and $\delta \subseteq Q \times \text{OP} \times Q$ is a finite set of *instructions*, where the set OP of *operations* contains $\text{INC}(c_i)$ (increment c_i), $\text{DEC}(c_i)$ (decrement c_i), and $\text{RESET}(c_i)$ (set c_i to 0), for $i = 1, 2, \dots, d$. We view $Q \times \mathbb{N}^d$ as the set CONF of *configurations* of \mathcal{M} . The *transition relation* $\longrightarrow \subseteq \text{CONF} \times \text{CONF}$ is induced by δ in the obvious way: If $(p, op, q) \in \delta$ then we have $(p, (n_1, \dots, n_d)) \longrightarrow (q, (n'_1, \dots, n'_d))$ in the following cases:

- $op = \text{INC}(c_i)$, $n'_i = n_i + 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \text{DEC}(c_i)$, $n_i > 0$, $n'_i = n_i - 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \text{RESET}(c_i)$, $n'_i = 0$, and $n'_j = n_j$ for all $j \neq i$.

By \longrightarrow^* we denote the reflexive and transitive closure of \longrightarrow .

Control-State Reachability Problem for RCMs

We define the *RCM control-state reachability problem* in the following form: given an RCM $\mathcal{M} = (d, Q, \delta)$ and (control) states $p_{\text{IN}}, p_{\text{F}}$, we ask if p_{F} is reachable from $(p_{\text{IN}}, (0, 0, \dots, 0))$, i.e., if there are $m_1, m_2, \dots, m_d \in \mathbb{N}$ such that $(p_{\text{IN}}, (0, 0, \dots, 0)) \longrightarrow^* (p_{\text{F}}, (m_1, m_2, \dots, m_d))$.

Lemma 9. [20] *RCM control-state reachability problem is Ackermann-hard.*

We mentioned this problem, and its ACK-completeness, in Section 1. (The crux of the hardness proof is an efficient construction that, given $n \in \mathbb{N}$, provides RCMs $\mathcal{M}_1, \mathcal{M}_2$ that “weakly compute” the function f_n and its inverse, for f_n used in the definition of the Ackermann function. By “weakly” we mean that some computations can also return smaller values than expected.)

RCM Control-State Reachability Reduces to First-Order Bisimilarity

We finish by proving the next lemma; this establishes Theorem 8, by using Lemma 9. The given reduction is obviously polynomial. (In fact, it can be checked to be a logspace reduction, but this is a minor point in the view of the fact that even a primitive-recursive reduction would suffice here.)

Lemma 10. *The RCM control-state reachability problem is polynomially reducible to the complement of the bisimilarity problem for first-order grammars.*

Proof. Let us consider an instance $\mathcal{M} = (d, Q, \delta)$, $p_{\text{IN}}, p_{\text{F}}$ of the RCM control-state reachability problem, and imagine the following game between Attacker (he) and Defender (she). This is the first version of a game that will be afterwards

implemented as a standard bisimulation game. Attacker aims to show that p_F is reachable from $(p_{IN}, (0, 0, \dots, 0))$, while Defender opposes this.

The game uses $2d$ *game-counters*, which are never decremented; each counter c_i of \mathcal{M} yields two game-counters, namely c_i^I and c_i^D , for counting the numbers of Increments and Decrements of c_i , respectively, since the last reset or since the beginning if there has been no reset of c_i so far. The *initial position* is $(p_{IN}, ((0, 0), \dots, (0, 0)))$, with all $2d$ game-counters (organized in pairs) having the value 0.

A game *round* from position $(p, ((n_1, n'_1), \dots, (n_d, n'_d)))$ proceeds as described below. It will become clear that it suffices to consider only the cases $n_i \geq n'_i$; the position then corresponds to the \mathcal{M} 's configuration $(p, (n_1 - n'_1, \dots, n_d - n'_d))$.

If $p = p_F$, then Attacker wins; if $p \neq p_F$ and there is no instruction $(p, op, q) \in \delta$, then Defender wins. Otherwise Attacker chooses $(p, op, q) \in \delta$, and the continuation depends on op as follows:

1. If $op = \text{INC}(c_i)$, then the next-round position arises (from the previous one) by replacing p with q and by performing $c_i^I := c_i^I + 1$ (the counter of increments of c_i is incremented, i.e., n_i is replaced with $n_i + 1$).
2. If $op = \text{RESET}(c_i)$, then the next-round position arises by replacing p with q and by performing $c_i^I := 0$ and $c_i^D := 0$ (hence both n_i and n'_i are replaced with 0).
3. If $op = \text{DEC}(c_i)$, then *Defender chooses* one of the following options:
 - (a) the next-round position arises by replacing p with q and by performing $c_i^D := c_i^D + 1$ (the counter of decrements of c_i is incremented, i.e., n'_i is replaced with $n'_i + 1$), or
 - (b) (Defender claims that this *decrement* is *illegal* since $n_i = n'_i$ and) the next position becomes just (n_i, n'_i) . In this case a (deterministic) check if $n_i = n'_i$ is performed, by successive synchronized decrements at both sides. If indeed $n_i = n'_i$ (the counter-bottoms are reached at the same moment), then Defender wins; otherwise (when $n_i \neq n'_i$) Attacker wins.

If $(p_{IN}, (0, 0, \dots, 0)) \xrightarrow{*} (p_F, (m_1, m_2, \dots, m_d))$ for some m_1, m_2, \dots, m_d , i.e., if the answer to RCM control-state reachability is YES, then Attacker has a winning strategy: he just follows the corresponding sequence of instructions. He thus also always chooses $\text{DEC}(c_i)$ *legally*, i.e. only in the cases where $n_i > n'_i$, and Defender loses if she ever chooses 3(b). If the answer is NO (p_F is not reachable), and Attacker follows a legal sequence of instructions, then he either loses in a “dead” state or the play is infinite; if Attacker chooses an illegal decrement, then in the first such situation we obviously have $n_i = n'_i$ for the respective counter c_i , and Defender can force her win via 3(b).

Since the game-counters can be only incremented or reset, it is a routine to implement the above game as a bisimulation game in the grammar-framework (using a standard method of “Defender’s forcing” for implementing the choice in 3). We now describe the corresponding grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$.

The set \mathcal{N} of nonterminals will include a unary nonterminal I , a nullary nonterminal \perp , and the nonterminals with arity $2d$ that are induced by control

states of \mathcal{M} as follows: each $p \in Q$ induces $A_p, A_{(p,i)}, B_p, B_{(p,i,1)}, B_{(p,i,2)}$, where $i = 1, 2, \dots, d$.

We intend that a game-position $(p, ((n_1, n'_1), \dots, (n_d, n'_d)))$ corresponds to the pair of terms

$$\left(A_p(I^{n_1} \perp, I^{n'_1} \perp, \dots, I^{n_d} \perp, I^{n'_d} \perp), B_p(I^{n_1} \perp, I^{n'_1} \perp, \dots, I^{n_d} \perp, I^{n'_d} \perp) \right) \quad (14)$$

where $I^k \perp$ is a shorthand for $I(I(\dots I(\perp) \dots))$ with I occurring k times; we put $I^0 \perp = \perp$. The RCM control-state reachability instance $\mathcal{M}, p_{\text{IN}}, p_{\text{F}}$ will be reduced to the (non)bisimilarity-problem instance $\mathcal{G}, A_{p_{\text{IN}}}(\perp, \dots, \perp), B_{p_{\text{IN}}}(\perp, \dots, \perp)$.

We put $\Sigma = \delta \uplus \{a, b\}$, i.e., the actions of \mathcal{G} correspond to the instructions (or instruction names) of \mathcal{M} , and we also use auxiliary (fresh) actions a, b .

The set of rules \mathcal{R} contains a sole rule for I , namely $I(x_1) \xrightarrow{a} x_1$, and no rule for \perp ; hence $I^n \perp \sim I^{n'} \perp$ iff $n = n'$. Each instruction $\text{INS} = (p, op, q) \in \delta$ induces the rules in \mathcal{R} as follows:

1. If $op = \text{INC}(c_i)$, then the induced rules are

$$A_p(x_1, \dots, x_{2d}) \xrightarrow{\text{INS}} A_q(x_1, \dots, x_{2(i-1)}, I(x_{2i-1}), x_{2i}, \dots, x_{2d}), \text{ and}$$

$$B_p(x_1, \dots, x_{2d}) \xrightarrow{\text{INS}} B_q(x_1, \dots, x_{2(i-1)}, I(x_{2i-1}), x_{2i}, \dots, x_{2d}).$$

2. If $op = \text{RESET}(c_i)$, then the induced rules are

$$A_p(x_1, \dots, x_{2d}) \xrightarrow{\text{INS}} A_q(x_1, \dots, x_{2(i-1)}, \perp, \perp, x_{2i+1}, \dots, x_{2d}),$$

$$B_p(x_1, \dots, x_{2d}) \xrightarrow{\text{INS}} B_q(x_1, \dots, x_{2(i-1)}, \perp, \perp, x_{2i+1}, \dots, x_{2d}).$$

3. If $op = \text{DEC}(c_i)$, then the induced rules are below; here we use the shorthand

$A \xrightarrow{a} B$ when meaning $A(x_1, \dots, x_{2d}) \xrightarrow{a} B(x_1, \dots, x_{2d})$:

$$A_p \xrightarrow{\text{INS}} A_{(q,i)}, A_p \xrightarrow{\text{INS}} B_{(q,i,1)}, A_p \xrightarrow{\text{INS}} B_{(q,i,2)}, B_p \xrightarrow{\text{INS}} B_{(q,i,1)},$$

$$B_p \xrightarrow{\text{INS}} B_{(q,i,2)},$$

$$A_{(q,i)}(x_1, \dots, x_{2d}) \xrightarrow{a} A_q(x_1, \dots, x_{2i-1}, I(x_{2i}), x_{2i+1}, \dots, x_{2d}),$$

$$B_{(q,i,1)}(x_1, \dots, x_{2d}) \xrightarrow{a} B_q(x_1, \dots, x_{2i-1}, I(x_{2i}), x_{2i+1}, \dots, x_{2d}),$$

$$B_{(q,i,2)}(x_1, \dots, x_{2d}) \xrightarrow{a} A_q(x_1, \dots, x_{2i-1}, I(x_{2i}), x_{2i+1}, \dots, x_{2d}),$$

$$A_{(q,i)}(x_1, \dots, x_{2d}) \xrightarrow{b} x_{2i-1}, B_{(q,i,1)}(x_1, \dots, x_{2d}) \xrightarrow{b} x_{2i-1},$$

$$B_{(q,i,2)}(x_1, \dots, x_{2d}) \xrightarrow{b} x_{2i}.$$

Moreover, \mathcal{R} contains $A_{p_{\text{F}}}(x_1, \dots, x_{2d}) \xrightarrow{a} \perp$ (but not $B_{p_{\text{F}}}(x_1, \dots, x_{2d}) \xrightarrow{a} \perp$).

Now we recall the standard (turn-based) *bisimulation game*, starting with the pair $(A_{p_{\text{IN}}}(\perp, \dots, \perp), B_{p_{\text{IN}}}(\perp, \dots, \perp))$. In the round starting with (T_1, T_2) , Attacker chooses a transition $T_j \xrightarrow{a} T'_j$ and then Defender chooses $T_{3-j} \xrightarrow{a} T'_{3-j}$ (for the same $a \in \Sigma$); the next round starts with the pair (T'_1, T'_2) . If a player gets stuck, then (s)he loses; an infinite play is a win of Defender. It is obvious that Defender has a winning strategy in this game iff $A_{p_{\text{IN}}}(\perp, \dots, \perp) \sim B_{p_{\text{IN}}}(\perp, \dots, \perp)$.

We now easily check that this bisimulation game indeed implements the above described game; a game-position $(p, ((n_1, n'_1), \dots, (n_d, n'_d)))$ is implemented as the pair (14). The points 1 and 2 directly correspond to the previous points 1 and 2. If Attacker chooses an instruction $\text{INS} = (p, \text{DEC}(c_i), q)$, then he must

use the respective rule $A_p \xrightarrow{\text{INS}} A_{(q,i)}$ in 3, since otherwise Defender installs syntactic equality, i.e. a pair (T, T) . It is now Defender who chooses $B_p \xrightarrow{\text{INS}} B_{(q,i,1)}$ (corresponding to the previous 3(a)) or $B_p \xrightarrow{\text{INS}} B_{(q,i,2)}$ (corresponding to 3(b)). Attacker then must choose action a in the first case, and action b in the second case; otherwise we get syntactic equality. The first case thus results in the pair $(A_q(\dots), B_q(\dots))$ corresponding to the next game-position (where c_i^D has been incremented), and the second case results in the pair $(I^{n_i} \perp, I^{n'_i} \perp)$; we have already observed that $I^{n_i} \perp \sim I^{n'_i} \perp$ iff $n_i = n'_i$.

Finally we observe that in any pair $(A_{p_F}(\dots), B_{p_F}(\dots))$ Attacker wins immediately (since the transition $A_{p_F}(\dots) \xrightarrow{a} \perp$ can not be matched).

We have thus established that p_F is reachable from $(p_{\text{IN}}, (0, \dots, 0))$ if, and only if, $A_{p_{\text{IN}}}(\perp, \dots, \perp) \not\sim B_{p_{\text{IN}}}(\perp, \dots, \perp)$. \square

Acknowledgement. The Ackermann-hardness result was achieved during my visit at LSV ENS Cachan, and I am grateful to Sylvain Schmitz and Philippe Schnoebelen for fruitful discussions.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. STOC 2004, pp. 202–211. ACM (2004)
2. Benedikt, M., Göller, S., Kiefer, S., Murawski, A.S.: Bisimilarity of pushdown automata is nonelementary. In: Proc. LICS 2013, pp. 488–498. IEEE Computer Society (2013)
3. Böhm, S., Göller, S., Jančar, P.: Equivalence of deterministic one-counter automata is NL-complete. In: Proc. STOC 2013, pp. 131–140. ACM (2013)
4. Böhm, S., Göller, S., Jančar, P.: Bisimulation equivalence and regularity for real-time one-counter automata. *J. Comput. Syst. Sci.* 80, 720–743 (2014), <http://dx.doi.org/10.1016/j.jcss.2013.11.003>
5. Broadbent, C.H., Göller, S.: On bisimilarity of higher-order pushdown automata: Undecidability at order two. In: FSTTCS 2012. LIPIcs, vol. 18, pp. 160–172. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012)
6. Burkart, O., Caucal, D., Steffen, B.: An elementary bisimulation decision procedure for arbitrary context-free processes. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 423–433. Springer, Heidelberg (1995)
7. Courcelle, B.: Recursive applicative program schemes. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, vol. B, pp. 459–492. Elsevier, MIT Press (1990)
8. Czerwinski, W., Lasota, S.: Fast equivalence-checking for normed context-free processes. In: Proc. of FSTTCS 2010. LIPIcs, vol. 8, pp. 260–271. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010)
9. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson’s lemma. In: Proc. LICS 2011, pp. 269–278. IEEE Computer Society (2011)
10. Friedman, E.P.: The inclusion problem for simple languages. *Theor. Comput. Sci.* 1(4), 297–316 (1976)

11. Göller, S., Lohrey, M.: The First-Order Theory of Ground Tree Rewrite Graphs. *Logical Methods in Computer Science* 10(1) (2014), [http://dx.doi.org/10.2168/LMCS-10\(1:7\)2014](http://dx.doi.org/10.2168/LMCS-10(1:7)2014)
12. Henry, P., Sénizergues, G.: LALBLC A program testing the equivalence of dpda's. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 169–180. Springer, Heidelberg (2013)
13. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theor. Comput. Sci.* 158(1&2), 143–159 (1996)
14. Jančar, P., Srba, J.: Undecidability of bisimilarity by Defender's forcing. *J. ACM* 55(1) (2008)
15. Jančar, P.: Decidability of DPDA language equivalence via first-order grammars. In: Proc. LICS 2012, pp. 415–424. IEEE Computer Society (2012)
16. Jančar, P.: Bisimilarity on basic process algebra is in 2-ExpTime (an explicit proof). *Logical Methods in Computer Science* 9(1) (2013)
17. Kiefer, S.: BPA bisimilarity is EXPTIME-hard. *Inf. Process. Lett.* 113(4), 101–106 (2013)
18. Kučera, A., Mayr, R.: On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Inf. Comput.* 208(7), 772–796 (2010)
19. Schmitz, S.: Complexity hierarchies beyond elementary. CoRR abs/1312.5686 (2013)
20. Schnoebelen, P.: Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 616–628. Springer, Heidelberg (2010)
21. Sénizergues, G.: $L(A)=L(B)$? Decidability results from complete formal systems. *Theoretical Computer Science* 251(1-2), 1–166 (2001); a preliminary version appeared at Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.): ICALP 1997. LNCS, vol. 1256. Springer, Heidelberg (1997)
22. Sénizergues, G.: The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.* 34(5), 1025–1106 (2005) (a preliminary version appeared at FOCS 1998)
23. Sénizergues, G.: The equivalence problem for t-turn DPDA is co-NP. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 478–489. Springer, Heidelberg (2003)
24. Srba, J.: Roadmap of infinite results. in: *Current Trends In Theoretical Computer Science, The Challenge of the New Century*, vol. 2, pp. 337–350. World Scientific Publishing Co. (2004), updated version at <http://users-cs.au.dk/srba/roadmap/>
25. Srba, J.: Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science* 5(1) (2009)
26. 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)
27. Stirling, C.: Second-order simple grammars. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 509–523. Springer, Heidelberg (2006)
28. Urquhart, A.: The complexity of decision procedures in relevance logic II. *J. Symb. Log.* 64(4), 1774–1802 (1999)
29. Walukiewicz, I.: Pushdown processes: Games and model-checking. *Inf. Comput.* 164(2), 234–263 (2001)

Active Diagnosis for Probabilistic Systems^{*}

Nathalie Bertrand¹, Éric Fabre¹,
Stefan Haar^{1,2}, Serge Haddad², and Loïc Hélouët¹

¹ Inria, France

² LSV, ENS Cachan & CNRS & Inria, France

Abstract. The diagnosis problem amounts to deciding whether some specific “fault” event occurred or not in a system, given the observations collected on a run of this system. This system is then diagnosable if the fault can always be detected, and the active diagnosis problem consists in controlling the system in order to ensure its diagnosability. We consider here a stochastic framework for this problem: once a control is selected, the system becomes a stochastic process. In this setting, the active diagnosis problem consists in deciding whether there exists some observation-based strategy that makes the system diagnosable with probability one. We prove that this problem is EXPTIME-complete, and that the active diagnosis strategies are belief-based. The *safe* active diagnosis problem is similar, but aims at enforcing diagnosability while preserving a positive probability to non faulty runs, i.e. without enforcing the occurrence of a fault. We prove that this problem requires non belief-based strategies, and that it is undecidable. However, it belongs to NEXPTIME when restricted to belief-based strategies. Our work also refines the decidability/undecidability frontier for verification problems on partially observed Markov decision processes.

1 Introduction

Diagnosis for discrete event systems was introduced in [11], and can be described as follows: a labeled transition system performs a run, which may contain some specific events called *faults*. Some of the transition labels are observable, so one gets information about the performed run through its trace, i.e. its sequence of observed labels. The diagnosis problem then amounts to determining whether a fault event occurred or not given the observed trace. The trace is called faulty (resp. correct) if all runs that can have produced it contain (resp. do not contain) a fault. In the remaining cases the trace is called ambiguous. Along with the diagnosis problem comes the diagnosability question: does there exist an infinite ambiguous trace (thus forbidding diagnosis)? For finite transition systems, checking diagnosability was proved to have a polynomial complexity [15].

^{*} This work was supported by project ImpRo ANR-2010-BLAN-0317 and the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement 257462 HYCON2 NOE.

Diagnosis and diagnosability checking have been extended to numerous models (Petri nets [3], pushdown systems [9], etc.) and settings (centralized, decentralized, distributed), and have had an impact on important application areas, e.g. for telecommunication network failure diagnosis. Several contributions have considered enforcing the diagnosability of a system. Under the generic name of active diagnosis, the problems take quite different shapes. They range from the selection of minimal sets of observable labels that make the system diagnosable [4], to the design of controllers that select a diagnosable sublanguage of a system [10], and to online aspects that either turn on and off sensors [4, 13] or modify an action plan [5] in order to reduce the amount of ambiguity. Probabilistic systems have also received some attention [7, 12], with two essential motivations: determining the likelihood of a fault given an observed trace and defining diagnosability for probabilistic systems. Two definitions have been proposed: The A-diagnosability, which requires that the ambiguous traces have a null probability, and the weaker AA-diagnosability, which requires that fault likelihood will converge to one with probability one. Interestingly, the A-diagnosability does not depend on the specific values of transition probabilities, but only on their support: it is thus a structural property of a system, which can be checked in polynomial time on finite state systems.

Here we address the question of active diagnosis for stochastic systems. We elaborate on two recent contributions. The first one [8] improves the work in [10] and designs an observation-based controller that enables a subset of actions in the system in order to make it diagnosable while preserving its liveness. Optimal constructions are then proposed the most relevant for our work being the characterization of unambiguous traces by a deterministic Büchi automaton with minimal size. The second one [1] considers probabilistic Büchi automata, a subclass of partially observed Markov decision processes (POMDP), and proves that checking the existence of strategies that almost surely achieve a Büchi condition on POMDP is EXPTIME-complete. The result was later extended in [2]. This motivates the use of POMDP as semantics for the models we consider.

The first contribution of this paper is a framework for the active diagnosis problem of probabilistic systems. The models we consider are weighted and labeled transition systems, where some transitions represent a fault. Some of the transition labels are observable, and similarly some are controllable. From a given state of the system, and given a set of enabled labels, one derives a transition probability by normalization of transition weights. The active diagnosis problem amounts to designing a label activation strategy that enforces the stochastic diagnosability of the system while preserving its liveness. As a second contribution, this problem is proved to be decidable, and EXPTIME complete. The resulting strategies are belief-based, i.e. they only depend on the set of possible states of the system given past observations, regardless of the exact values of transition weights. As a third contribution, we introduce and analyze the *safe* active diagnosis problem. It extends the active diagnosis by enforcing a positive probability of correct runs. In other words, this rules out strategies that would reach diagnosability only by enforcing the occurrence of a fault. We prove

that safe active diagnosis may require non belief-based strategies, and that the existence of such strategies is an undecidable problem. This result refines the decidability/undecidability frontier for POMDP: the existence of a strategy simultaneously ensuring a Büchi condition almost-surely and a safety condition with positive probability is undecidable. This may seem surprising since the existence of strategies for each objective taken separately is decidable. As a last contribution, we prove that, restricted to belief-based strategies, the safe active diagnosis problem becomes decidable and belongs to NEXPTIME.

The paper is organized as follows: section 2 introduces the active diagnosis problem for probabilistic systems, and compares it with the state of the art. Section 3 proposes resolution techniques for active diagnosis. Section 4 analyzes the safe active diagnosis problem. Section 5 concludes this work. A long version of this paper including proofs is available at <http://hal.inria.fr/hal-00930919>

2 The Active Diagnosis Problem

This section recalls diagnosis problems from the literature, and formalizes the new problems we are interested in.

2.1 Passive (Probabilistic) Diagnosis

When dealing with stochastic discrete event systems diagnosis, systems are often modeled using labeled transition systems.

Definition 1. A probabilistic labeled transition system (*pLTS*) is a tuple $\mathcal{A} = \langle Q, q_0, \Sigma, T, \mathbf{P} \rangle$ where:

- Q is a set of states with $q_0 \in Q$ the initial state;
- Σ is a finite set of events;
- $T \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- \mathbf{P} is the transition matrix from T to $\mathbb{Q}_{\geq 0}$ fulfilling for all $q \in Q$:

$$\sum_{(q,a,q') \in T} \mathbf{P}[q, a, q'] = 1.$$

Observe that a pLTS is a labeled transition system (LTS) equipped with transition probabilities. The transition relation of the underlying LTS is defined by: $q \xrightarrow{a} q'$ for $(q, a, q') \in T$; this transition is then said to be *enabled* in q . A *run* over the word $\sigma = a_1 a_2 \dots \in \Sigma^\omega$ is a sequence of states $(q_i)_{i \geq 0}$ such that $q_i \xrightarrow{a_{i+1}} q_{i+1}$ for all $i \geq 0$, and we write $q_0 \xrightarrow{\sigma}$ if such a run exists. A finite run over $w \in \Sigma^*$ is defined analogously, and we write $q \xrightarrow{w} q'$ if such a run ends at state q' . A state q is *reachable* if there exists a run $q_0 \xrightarrow{w} q$ for some $w \in \Sigma^*$. On the other hand, forgetting the labels and merging the transitions with same source and target, one obtains a discrete time Markov chain (DTMC).

Definition 2 (Languages of a pLTS). Let $\mathcal{A} = \langle Q, q_0, \Sigma, T, \mathbf{P} \rangle$ be a pLTS. The finite language $\mathcal{L}^*(\mathcal{A}) \subseteq \Sigma^*$ of \mathcal{A} and the infinite language $\mathcal{L}^\omega(\mathcal{A}) \subseteq \Sigma^\omega$ of \mathcal{A} are defined by:

$$\mathcal{L}^*(\mathcal{A}) = \{ w \in \Sigma^* \mid \exists q : q_0 \xrightarrow{w} q \} \quad \mathcal{L}^\omega(\mathcal{A}) = \{ \sigma \in \Sigma^\omega \mid q_0 \xrightarrow{\sigma} \}$$

Observations. In order to formalize problems related to diagnosis, we partition Σ into two disjoint sets Σ_o and Σ_u , the sets of *observable* and of *unobservable events*, respectively. Moreover, we distinguish a special *fault* event $f \in \Sigma_u$. Let σ be a finite word; its length is denoted $|\sigma|$. For $\Sigma' \subseteq \Sigma$, define $\mathcal{P}_{\Sigma'}(\sigma)$, the projection of σ on Σ' , inductively by: $\mathcal{P}_{\Sigma'}(\varepsilon) = \varepsilon$; for $a \in \Sigma'$, $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)a$; and $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)$ for $a \notin \Sigma'$. Write $|\sigma|_{\Sigma'}$ for $|\mathcal{P}_{\Sigma'}(\sigma)|$, and for $a \in \Sigma$, write $|\sigma|_a$ for $|\sigma|_{\{a\}}$. When σ is an infinite word, its projection is the limit of the projections of its finite prefixes. This projection can be either finite or infinite. As usual the projection is extended to languages. In the rest of the paper, we will only use \mathcal{P}_{Σ_o} , the projection onto observable events, and hence we will drop the subscript and simply write \mathcal{P} instead of \mathcal{P}_{Σ_o} .

With respect to the partition of $\Sigma = \Sigma_o \uplus \Sigma_u$, a pLTS \mathcal{A} is *convergent* if $\mathcal{L}^\omega(\mathcal{A}) \cap \Sigma^* \Sigma_u^\omega = \emptyset$ (i.e. there is no infinite sequence of unobservable events from any reachable state). When \mathcal{A} is convergent, then for all $\sigma \in \mathcal{L}^\omega(\mathcal{A})$, one has $\mathcal{P}(\sigma) \in \Sigma_o^\omega$. In the rest of the paper we assume that pLTS are convergent and we will call a *sequence* a finite or infinite word over Σ , and an *observed sequence* a finite or infinite sequence over Σ_o . Clearly, the projection of a sequence on Σ_o yields an observed sequence. Intuitively, a sequence describes the behavior of a system during an execution, and an observed sequence represents how such a run is perceived. Now, the role of diagnosis is to decide, for any observed sequence, whether a fault has occurred or not.

Ambiguity. A finite (resp. infinite) sequence σ is *correct* if it belongs to $(\Sigma \setminus \{f\})^*$ (resp. $(\Sigma \setminus \{f\})^\omega$). Otherwise σ is called *faulty*. A correct sequence and a faulty sequence may have the same observed projection, yielding ambiguity.

Definition 3 (Classification of observed sequences). *Let \mathcal{A} be a pLTS. An observed sequence $\sigma \in \Sigma_o^\omega$ is called ambiguous if there exist two sequences $\sigma_1, \sigma_2 \in \mathcal{L}^\omega(\mathcal{A})$ such that $\mathcal{P}(\sigma_1) = \mathcal{P}(\sigma_2) = \sigma$, σ_1 is correct and σ_2 is faulty. An observed sequence $\sigma' \in \mathcal{P}(\mathcal{L}^\omega(\mathcal{A}))$ is surely faulty if $\mathcal{P}^{-1}(\sigma') \cap \mathcal{L}^\omega(\mathcal{A}) \subseteq \Sigma^* f \Sigma^\omega$. An observed sequence $\sigma' \in \mathcal{P}(\mathcal{L}^\omega(\mathcal{A}))$ is surely correct if $\mathcal{P}^{-1}(\sigma') \cap \mathcal{L}^\omega(\mathcal{A}) \subseteq (\Sigma \setminus \{f\})^\omega$. These notions are defined analogously for finite observed sequences.*

Example. Consider the (convergent) pLTS to the left in Fig. 1, where $\Sigma_u = \{f, u\}$. We assume uniform distributions so we do not represent the probability matrix \mathbf{P} . This pLTS contains infinite ambiguous sequences: immediately after a is observed, an ambiguity appears, and this ambiguity remains in all infinite observed sequences without occurrence of d and finishing with ab^ω . Removing the loop at q_2 and/or q_4 makes all infinite ambiguous sequences disappear.

In the sequel, we will use the characterization of unambiguous sequences using deterministic Büchi automata [8].

Definition 4 (Büchi automaton). *A Büchi automaton over Σ is a tuple $\mathcal{B} = \langle Q, q_0, \Sigma, T, F \rangle$ with $\langle Q, q_0, \Sigma, T \rangle$ its underlying LTS and $F \subseteq Q$ an acceptance condition. A run $(q_i)_{i \geq 0}$ is accepting if $q_i \in F$ for infinitely many values of i . The language $\mathcal{L}(\mathcal{B})$ consists of all words in Σ^ω for which there exists an accepting*

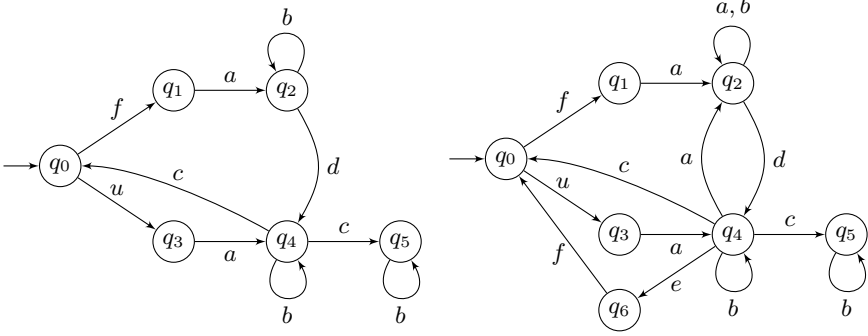


Fig. 1. Two examples of pLTS (cLTS), with $\Sigma_u = \{f, u\}$ and $\Sigma_o = \{a, b, c, d, e\}$

run. A Büchi automaton is deterministic if for all q, a , $\{q' \mid q \xrightarrow{a} q'\}$ is either a singleton or the empty set.

Theorem 1 ([8]). Given a pLTS \mathcal{A} with n states, one can build in exponential time a deterministic Büchi automaton \mathcal{B} with $2^{O(n)}$ states whose language is the set of unambiguous sequences of \mathcal{A} .

We briefly sketch the structure of \mathcal{B} . Its states are triples $\langle U, V, W \rangle$, where $U, V, W \subseteq Q$, $U \cup V \cup W \neq \emptyset$ and $V \cap W = \emptyset$, and its transitions are labeled by events from Σ_o , that is \mathcal{B} recognizes observed sequences. The initial state of \mathcal{B} is $\langle \{q_0\}, \emptyset, \emptyset \rangle$. Given an observed sequence σ reaching state $\langle U, V, W \rangle$, U is the set of states of \mathcal{A} reached by a correct sequence with projection σ , and $V \cup W$ is the set of states of \mathcal{A} reached by a faulty sequence with projection σ . When $U = \emptyset$, σ is the projection of faulty sequences of \mathcal{A} . The decomposition between V and W reflects the fact that \mathcal{B} tries to “solve the ambiguity” between U and W (when both are non empty), while V corresponds to a waiting room of states reached by faulty sequences that will be examined when the current ambiguity is resolved. Given some new observation a , a transition from $\langle U, V, W \rangle$ to the new state $\langle U', V', W' \rangle$ is defined as follows. U' is the set of states reached from U by a correct sequence with projection a . Let Y be the set of states reached from U by a faulty sequence with projection a , or reached from V by a sequence with projection a . When W is non empty then W' is the set of states reached from W by a sequence with projection a and $V' = Y$. Otherwise, the faulty sequences ending in states memorized by W cannot be extended by a sequences with projection a , and we set $V' = \emptyset$ and $W' = Y$. The ambiguity between U and W has been resolved, but new ambiguity may arise between U' and W' . Accepting states in F are triples $\langle U, V, W \rangle$ with $U = \emptyset$ or $W = \emptyset$. Hence, all infinite observed sequence of \mathcal{A} passing infinitely often through F are not ambiguous (they resolve ambiguities one after another) and are accepted by \mathcal{B} .

We are now in position to define diagnosability. It is well-known that given a pLTS \mathcal{A} and a Büchi automaton \mathcal{B} , the set of sequences of \mathcal{A} accepted by \mathcal{B} is measurable [14]. So the following definition is sound.

Definition 5 (Diagnosability). *A pLTS \mathcal{A} is diagnosable if the set of sequences yielding ambiguous observed sequences has null measure. It is safely diagnosable if it is diagnosable and the set of correct sequences has positive measure.*

The notion of a safely diagnosable pLTS is introduced to ensure that fault occurrence is not almost sure. This property is important: a diagnosable system which is not safely diagnosable contains only faulty infinite runs. In the rest of the paper, we will consider active diagnosis, that is, ways to force a system to become diagnosable using a controller. If a controlled system is not safely diagnosable, then the diagnosis solution enforced by the controller is not acceptable.

Example. Consider again the pLTS to the left in Fig. 1. The only ambiguous observed (infinite) sequences necessarily terminate with ab^ω . But the probability to produce such a sequence is null, as the system will reach q_5 with probability one. In other words, ambiguity vanishes at the first occurrence of d or cb . Since cb occurs with probability one, this pLTS is diagnosable. This pLTS is also safely diagnosable, as it can produce correct sequences with a positive probability: there is a positive probability to reach q_5 by sequence uac . If one removes state q_5 and its connected transitions, the system remains diagnosable, but is not safely diagnosable anymore: as the graph of the pLTS is strongly connected, every transition will be visited (infinitely often) with probability 1 implying that f occurs.

2.2 Active Probabilistic Diagnosis

In order to allow control over the actions of a system while preserving the possibility of a probabilistic semantic, we introduce controllable weighted labelled transition system where probabilities are replaced by weights.

Definition 6. *A controllable weighted labelled transition system (cLTS) is a tuple $\mathcal{C} = \langle Q, q_0, \Sigma, T \rangle$ where:*

- Q is a finite set of states with $q_0 \in Q$ the initial state;
- the event alphabet Σ is partitionned into observable Σ_o and unobservable Σ_u events, and also partitionned into controllable Σ_c and uncontrollable Σ_e (e for environment) events;
- $\Sigma_u = \{f, u\}$ contains a faulty event, and a non-faulty one;
- $T : S \times \Sigma \times S \rightarrow \mathbb{N}$ is the transition function, labelling transitions with integer weights.

A cLTS has an underlying LTS where the transition relation is defined by $q \xrightarrow{a} q'$ if $T(q, a, q') > 0$. All previous definitions that do not depend on probabilities equally apply to cLTS. We denote by $\text{Ena}(q)$ the set of events that are enabled in q : $\text{Ena}(q) = \{a \in \Sigma \mid \exists q', T(q, a, q') > 0\}$. We assume that the cLTS is convergent and *live*: for all q , $\text{Ena}(q) \neq \emptyset$.

Let $\mathcal{C} = \langle Q, q_0, \Sigma, T \rangle$ be a cLTS. For $q \in Q$ and $\Sigma^\bullet \subseteq \Sigma$, we define

$$G^{\Sigma^\bullet}(q) = \sum_{a \in \Sigma^\bullet, q' \in Q} T(q, a, q')$$

as the (possibly null) global outgoing weight from q restricted to Σ^\bullet -events. Similarly, we define a normalization of the transition relation restricted to Σ^\bullet by

$$T^{\Sigma^\bullet}(q, a, q') = \begin{cases} \frac{T(q, a, q')}{G^{\Sigma^\bullet}(q)} & \text{if } a \in \Sigma^\bullet \text{ and } T(q, a, q') > 0 \\ 0 & \text{otherwise} \end{cases}$$

For a given finite set X , we define by $\text{Dist}(X)$ the set of probabilistic distributions over X . Let $x \in X$, we denote by $\mathbf{1}_x$ the Dirac distribution on x . For a distribution $\delta \in \text{Dist}(X)$, the support of δ is the set $\text{Supp}(\delta) = \{x \in X \mid \delta(x) > 0\}$.

A *strategy* for a cLTS \mathcal{C} is a mapping $\pi : \Sigma_o^* \rightarrow \text{Dist}(2^\Sigma)$ such that for every $\sigma \in \Sigma_o^*$, for every $\Sigma' \in \text{Supp}(\pi(\sigma))$, $\Sigma' \supseteq \Sigma_e$. A strategy consists in, given some observation, randomly choosing a subset of allowed events that includes the uncontrollable events. Given a cLTS \mathcal{C} and a strategy π , we consider configurations of the form $(\sigma, q, \Sigma^\bullet) \in \Sigma_o^* \times Q \times 2^\Sigma$ where σ is the observed sequence, q is the current state and Σ^\bullet is a set of events allowed by π after observing σ . We define inductively the set $\text{Reach}_\pi(\mathcal{C})$ of reachable configurations under π :

- for all $\Sigma^\bullet \in \text{Supp}(\pi(\varepsilon))$, $(\varepsilon, q_0, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$;
- for all $(\sigma, q, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, for all $a \in \Sigma_u \cap \Sigma^\bullet$, such that $q \xrightarrow{a} q'$ $(\sigma, q', \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, denoted $(\sigma, q, \Sigma^\bullet) \xrightarrow{a}_\pi (\sigma, q', \Sigma^\bullet)$;
- for all $(\sigma, q, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, for all $a \in \Sigma_o \cap \Sigma^\bullet$ such that $q \xrightarrow{a} q'$ and $\Sigma^{\bullet'} \in \text{Supp}(\pi(\sigma a))$, $(\sigma a, q', \Sigma^{\bullet'}) \in \text{Reach}_\pi(\mathcal{C})$, denoted $(\sigma, q, \Sigma^\bullet) \xrightarrow{a}_\pi (\sigma a, q', \Sigma^{\bullet'})$.

A strategy π is said to be *live* if for every configuration $(\sigma, q, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, $G^{\Sigma^\bullet}(q) \neq 0$. Live strategies are the only relevant strategies as the other strategies introduce deadlocks. We are now in position to introduce the semantics of a cLTS. It is defined w.r.t. to some live strategy π as a pLTS. Its set of states is $\text{Reach}_\pi(\mathcal{C})$ with an initial state whose goal is to randomly select w.r.t. π the initial control. The transition probabilities are defined by T^{Σ^\bullet} accordingly to the current control Σ^\bullet except that when an observable action occurs it must be combined with the random choice (w.r.t. π) of the next control.

Definition 7. Let \mathcal{C} be a CLTS and π be a live strategy, the pLTS \mathcal{C}_π induced by strategy π on \mathcal{C} is defined as $\mathcal{C}_\pi = \langle Q_\pi, \Sigma, q_{0\pi}, T_\pi, \mathbf{P}_\pi \rangle$ where:

- $Q_\pi = \{q_{0\pi}\} \cup \text{Reach}_\pi(\mathcal{C})$;
- for all $(\varepsilon, q_0, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, $(q_{0\pi}, u, (\varepsilon, q_0, \Sigma^\bullet)) \in T_\pi$;
- for all $(\sigma, q, \Sigma^\bullet), (\sigma', q', \Sigma^{\bullet'}) \in \text{Reach}_\pi(\mathcal{C})$, $((\sigma, q, \Sigma^\bullet), a, (\sigma', q', \Sigma^{\bullet'})) \in T_\pi$ iff $(\sigma, q, \Sigma^\bullet) \xrightarrow{a}_\pi (\sigma', q', \Sigma^{\bullet'})$;
- for all $(\varepsilon, q_0, \Sigma^\bullet) \in \text{Reach}_\pi(\mathcal{C})$, $\mathbf{P}_\pi(q_{0\pi}, u, (\varepsilon, q_0, \Sigma^\bullet)) = \pi(\varepsilon)(\Sigma^\bullet)$;
- for all $((\sigma, q, \Sigma^\bullet), a, (\sigma, q', \Sigma^{\bullet'})) \in T_\pi$, for all $a \in \Sigma_u \cap \Sigma^\bullet$, $\mathbf{P}_\pi((\sigma, q, \Sigma^\bullet), a, (\sigma, q', \Sigma^{\bullet'})) = T^{\Sigma^\bullet}(q, a, q')$;

- for all $\left((\sigma, q, \Sigma^\bullet), a, (\sigma a, q', \Sigma^{\bullet'}) \right) \in T_\pi$, for all $a \in \Sigma_o \cap \Sigma^\bullet$,
- $\mathbf{P}_\pi \left((\sigma, q, \Sigma^\bullet), a, (\sigma a, q', \Sigma^{\bullet'}) \right) = T^{\Sigma^\bullet}(q, a, q') \cdot \pi(\sigma.a)(\Sigma^{\bullet'})$.

We can now formalize the decision problems we are interested in.

Definition 8 ((Safe) Active probabilistic diagnosis). *Given a cLTS $\mathcal{C} = \langle Q, q_0, \Sigma, T \rangle$, the active probabilistic diagnosis problem asks, whether there exists a live strategy π in \mathcal{C} such that the pLTS \mathcal{C}_π is diagnosable. The safe active probabilistic diagnosis problem asks whether there exists a live strategy π in \mathcal{C} such that the pLTS \mathcal{C}_π is safely diagnosable. The synthesis problems consists in building a live strategy π in \mathcal{C} such that the pLTS \mathcal{C}_π is (safely) diagnosable.*

Example. Consider the cLTS to the right in Fig. 1 with all weights equal to 1 and $\Sigma_o = \Sigma_c$. Without control, the system is not diagnosable as the observed sequence $aadc b^\omega$ is ambiguous, and it has a positive probability. So the strategy should disable action a for each correct observed sequence ending by ab^* . In addition, if this strategy always forbids c , the system becomes diagnosable, but the occurrence of a fault is enforced: so it is not safely diagnosable. Alternatively, if the strategy always forbids e , the system becomes safely diagnosable, as we obtain a pLTS “weakly probabilistically bisimilar” to the one on the left in Fig. 1.

3 Analysis of the Active Probabilistic Diagnosis Problem

To solve the active probabilistic diagnosis problem, we reduce it to a decidable problem on POMDP: namely, the existence of a strategy ensuring a Büchi objective with probability one [1, 2].

Definition 9 (POMDP). *A partially observable Markov decision process (POMDP) is a tuple $\mathbf{M} = \langle Q, q_0, \text{Obs}, \text{Act}, T \rangle$ where*

- Q is a finite set of states with q_0 the initial state;
- $\text{Obs} : Q \rightarrow \mathcal{O}$ assigns an observation $O \in \mathcal{O}$ to each state.
- Act is a finite set of actions;
- $T : Q \times \text{Act} \rightarrow \text{Dist}(Q)$ is a partial transition function. Letting $\text{Ena}(q) = \{a \in \text{Act} \mid T(q, a) \text{ is defined}\}$, we assume that:
 - for all $q \in Q$, $\text{Ena}(q) \neq \emptyset$, and
 - whenever $\text{Obs}(q) = \text{Obs}(q') = O$, then $\text{Ena}(q) = \text{Ena}(q')$ and slightly abusing our notation, we will denote by $\text{Ena}(O)$ the set of events enabled in every state with observation O .

A *decision rule* is an item of $\text{Dist}(\text{Act})$ that resolves non-determinism by randomization. A *strategy* maps histories of observations to decision rules. Formally, a strategy is a function $\pi : \mathcal{O}^+ \rightarrow \text{Dist}(\text{Act})$ such that for all $O_1 \cdots O_i$, $\text{Supp}(\pi(O_1 \cdots O_i)) \subseteq \text{Ena}(O_i)$. Given a strategy π and an initial distribution δ over states, a POMDP \mathbf{M} becomes a stochastic process that can be represented

by a possibly infinite pLTS denoted $M(\pi)$. One denotes $\mathbb{P}_\pi^\delta(\text{Ev})$ the probability that event Ev is realized in this process.

A *belief* is a subset of $\text{Obs}^{-1}(\text{O})$ for some observation O that corresponds to the possible reachable states w.r.t. some sequence of observations. The initial belief is $\{q_0\}$ and given a current belief B , a decision rule δ and an observation O , the belief $\Delta(B, (\delta, \text{O}))$ obtained after δ has been applied and O has been observed is defined by: $\bigcup_{q \in B, a \in \text{Supp}(\delta)} \text{Supp}(T(q, a)) \cap \text{Obs}^{-1}(\text{O})$. A strategy which only depends on the current belief is called a *belief-based strategy*.

In order to provide a POMDP M_C for the diagnosis problems of a cLTS \mathcal{C} , we face several difficulties. First, in a cLTS the observations are related to actions while in a POMDP they are related to states. Fortunately all the information related to ambiguity is included in the deterministic Büchi automaton described in section 2. Thus (with one exception) the states are pairs of a state of the Büchi automaton and a state of the cLTS. In \mathcal{C} , the control is performed by allowing a subset of events. Thus actions of M_C are subset of events that includes the uncontrollable events. Given some control Σ' , for defining the transition probability of M_C from (l, q) to (l', q') , one must consider all paths in \mathcal{C} labelled by events of Σ' from q to q' such that the last event (say b) is the single observable one. The probability of any such path is obtained by the product of the individual step probabilities. The latter are then defined by the normalization of weights w.r.t. Σ' . They cannot be infinite paths of unobservable events due to the convergence of \mathcal{C} . However some path can reach a state where no event of Σ' is possible. In other words, the control Σ' applied in (l, q) has a non null probability to reach a deadlock (i.e. the chosen decision rule leads to a non live strategy for the cLTS). In order to capture this behaviour and to obtain a non defective probability distribution, we add an additional state **lost**, that corresponds to such deadlocks. The next definition formalizes our approach.

Definition 10. *The POMDP $M_C = \langle Q^M, q_0^M, \text{Obs}, \text{Act}, T^M \rangle$ derived from a cLTS $\mathcal{C} = \langle Q, q_0, \Sigma, T \rangle$ and its associated deterministic Büchi automaton $\mathcal{B} = \langle L, l_0, \Sigma_o, T^B, F \rangle$ is defined by:*

- $Q^M = L \times Q \uplus \{\mathbf{lost}\}$ with $q_0^M = ((l_0, q_0))$;
- the set of observations is $\mathcal{O} = L \cup \{\mathbf{lost}\}$, with $\text{Obs}((l, q)) = l$ and $\text{Obs}(\mathbf{lost}) = \mathbf{lost}$;
- $\text{Act} = \{\Sigma' \mid \Sigma' \supseteq \Sigma_e\}$;
- for all $(l, q) \in Q^M$ and $\Sigma' \in \text{Act}$, $T^M((l, q), \Sigma') = \mu$ where:

- $\mu((l', q'))$ is defined by (with $q_n = q$ when $n = 0$):

$$\sum_{\substack{b \xrightarrow{\Sigma'} l' \\ b \in \Sigma' \cap \Sigma_o}} \sum_{\substack{q \xrightarrow{a_1} q_1 \cdots \xrightarrow{a_n} q_n \xrightarrow{b} q' \\ a_1 \cdots a_n \in \Sigma' \cap \Sigma_u}} T^{\Sigma'}(q, a_1, q_1) \cdot \left(\prod_{i=1}^{n-1} T^{\Sigma'}(q_i, a_{i+1}, q_{i+1}) \right) \cdot T^{\Sigma'}(q_n, b, q')$$

- $\mu(\text{lost})$ is defined by:

$$\sum_{\substack{q \xrightarrow{a_1} q_1 \dots \xrightarrow{a_n} q_n \\ a_1 \dots a_n \in \Sigma' \cap \Sigma_u \\ G^{\Sigma'}(q_n) = 0}} T^{\Sigma'}(q, a_1, q_1) \cdot \prod_{i=1}^{n-1} T^{\Sigma'}(q_i, a_{i+1}, q_{i+1})$$

- $T^M(\text{lost}, \Sigma') = \mathbf{1}_{\text{lost}}$ for all $\Sigma' \in \text{Act}$.

Given \mathcal{C} , the construction of the Büchi automaton \mathcal{B} is performed in exponential time. The construction of $M_{\mathcal{C}}$ is also done in exponential time. Indeed, there is an exponential blowup for Act but again w.r.t. \mathcal{C} . Finally, while the distributions μ of action effects are presented in the definition as sums over paths of \mathcal{C} , each one can be computed by a matrix inversion in polynomial time (as done in discrete time Markov chains).

The next lemma is a straightforward consequence of the properties of \mathcal{B} and the above definition of $M_{\mathcal{C}}$. Here we use LTL notations to denote sets of paths in a POMDP, such as \diamond , \square and $\square\diamond$ for eventually, always and infinitely often respectively.

Lemma 1. *\mathcal{C} is actively diagnosable if and only if there exists a strategy π in $M_{\mathcal{C}}$ such that $\mathbb{P}_{\pi}^{q_0}(M_{\mathcal{C}} \models \square\diamond(W = \emptyset \vee U = \emptyset)) = 1$.*

Moreover, \mathcal{C} is safely actively diagnosable if and only if there exists a strategy π in $M_{\mathcal{C}}$ such that $\mathbb{P}_{\pi}^{q_0}(M_{\mathcal{C}} \models \square\diamond(W = \emptyset \vee U = \emptyset)) = 1$ and $\mathbb{P}_{\pi}^{q_0}(M_{\mathcal{C}} \models \square(U \neq \emptyset)) > 0$.

In the statement of Lemma 1, $W = \emptyset \vee U = \emptyset$ is a shorthand to denote the set of states $(\langle U, V, W \rangle, q)$ in $M_{\mathcal{C}}$ such that either $W = \emptyset$ or $U = \emptyset$; similarly, $U \neq \emptyset$ represents the set of states $(\langle U, V, W \rangle, q)$ such that $U \neq \emptyset$. As a consequence of Lemma 1, the active diagnosis problem for controllable LTS reduces to the existence of an almost-sure winning strategy for a Büchi objective on some exponential size POMDP.

Theorem 2. *The active probabilistic diagnosis decision and synthesis problems are EXPTIME-complete. There exists a family $(C_n)_{n \in \mathbb{N}}$ of actively diagnosable cLTS with the size of C_n in $O(n)$, and such that any winning strategy for M_{C_n} diagnosable requires at least $2^{\Omega(n)}$ memory-states.*

The EXPTIME upper bound may seem surprising, since $M_{\mathcal{C}}$ is exponential in the size of \mathcal{C} , and the procedure to decide whether there exists a strategy in a POMDP to ensure a Büchi objective with probability 1 is in EXPTIME, due to the use of beliefs. However, in the POMDP $M_{\mathcal{C}}$ we consider, the information on the belief is already contained in the state $(\langle U, V, W \rangle, q)$, as $U \cup V \cup W$. Therefore, a second exponential blowup, due to the beliefs, is avoided and the active probabilistic diagnosis problem remains in EXPTIME.

4 Analysis of the Safe Active Probabilistic Diagnosis Problem

As will be shown below, the status of the active diagnosis problem changes when the safety requirement is added. The next proposition highlights this difference and it is the basis for the undecidability result of Theorem 3.

Proposition 1. *There exists a cLTS which is safely actively diagnosable and such that all belief-based strategies are losing.*

Proof. Let us consider the cLTS of Figure 2 with $\Sigma_u = \{u, f\}$ and $\Sigma_e = \{u, f, c\}$, and where all weights are equal to 1.

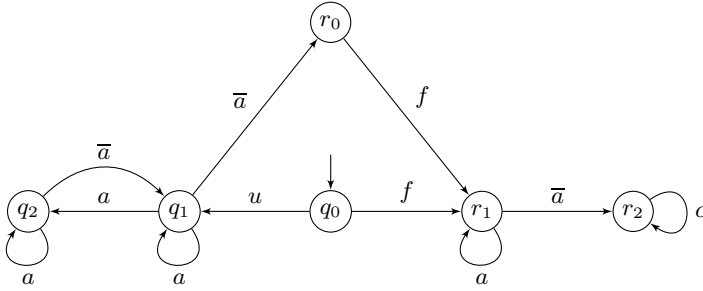


Fig. 2. A cLTS with only non belief-based strategies for safe diagnosis

Pick any sequence of positive integers $\{\alpha_i\}_{i \geq 1}$ such that $\prod_{i \geq 1} 1 - 2^{-\alpha_i} > 0$. Define $A = \{a\} \cup \Sigma_e$ and $\bar{A} = \{\bar{a}\} \cup \Sigma_e$. We claim that the strategy π that consists in selecting, after n observations, the n^{th} subset in the following sequence $A^{\alpha_1} \bar{A} A^{\alpha_2} \bar{A} \dots$, is winning. Observe that after an observable sequence of length $i \leq \alpha_1$, the system is either after a faulty sequence in r_1 with probability $\frac{1}{2}$, or after a correct sequence in q_1 with probability 2^{-i-1} , or after a correct sequence in q_2 with probability $\frac{1}{2}(1 - 2^{-i})$. So, after an observable sequence of length $\alpha_1 + 1$, the system is either after a faulty sequence in r_2 with probability $\frac{1}{2}$, or after a faulty sequence in r_1 (via r_0) with probability $2^{-\alpha_1-1}$, or after a correct sequence in q_1 with probability $\frac{1}{2}(1 - 2^{-\alpha_1})$. At the next step, the faulty sequence in r_2 is then detected by the occurrence of c .

Iterating this process we conclude that:

- any fault that may occur after π is applied up to $A^{\alpha_1} \bar{A} A^{\alpha_2} \bar{A} \dots A^{\alpha_i} \bar{A}$, is detected after π is applied up to $A^{\alpha_1} \bar{A} A^{\alpha_2} \bar{A} \dots A^{\alpha_{i+1}} \bar{A} A$. So the (full) strategy $\pi = A^{\alpha_1} \bar{A} A^{\alpha_2} \bar{A} \dots$ surely detects faults.
- the probability that there is an infinite correct sequence is equal to $\frac{1}{2} \prod_{i \geq 1} 1 - 2^{-\alpha_i} > 0$, due to our choice of the α_i 's. Therefore, correct sequences have positive probability under π .

Consider a belief-based strategy π . There are three possible subsets of allowed events: A , \bar{A} and Σ . The decision rule associated with belief $\{q_0\}$ must allow a in order to get the possibility of a correct sequence which, in case a occurs, leads to belief $\{q_1, q_2, r_1\}$. We should clarify here that beliefs do not correspond to the possible current states. They represent the possible states after the last observed event. For instance, when the belief is $\{q_0\}$, the current state may either be q_0 , or q_1 after action u , or r_1 after fault f . Consider the (randomized) decision rule of π associated with belief $\{q_1, q_2, r_1\}$: $p_A \cdot A + p_{\bar{A}} \cdot \bar{A} + p_{\Sigma} \cdot \Sigma$ (denoted \mathbf{p}). If $p_A = 1$, then the possible first fault remains undetected, and π is losing. So \bar{a} may occur leading to belief $\{q_1, r_0, r_2\}$.

Consider the decision rule of π associated with belief $\{q_1, r_0, r_2\}$: $p'_A \cdot A + p'_{\bar{A}} \cdot \bar{A} + p'_{\Sigma} \cdot \Sigma$ (denoted \mathbf{p}'). If $p'_A = 1$, then at the next instant, there is no possible correct sequence, and π is losing.

So $p'_A < 1$ and $p_A < 1$. Assume now that the current distribution of states is $\alpha q_1 + \beta r_0 + (1 - \alpha - \beta)r_2$ (with belief $\{q_1, r_0, r_2\}$). The distribution after the next occurrence of \bar{a} is defined by $\alpha_{\mathbf{p}, \mathbf{p}'} \alpha q_1 + (1 - \alpha_{\mathbf{p}, \mathbf{p}'}) \alpha r_0 + (1 - \alpha)r_2$, where $\alpha_{\mathbf{p}, \mathbf{p}'}$ only depends on \mathbf{p} and \mathbf{p}' . A correct sequence implies an infinite number of \bar{a} ; after n occurrences of \bar{a} the probability of a correct sequence is bounded by $\alpha_{\mathbf{p}, \mathbf{p}'}^n$. So the probability of an infinite correct sequence is null, and π is losing. \square

Theorem 3. *The safe active diagnosis problem for cLTS is undecidable.*

Proof (sketch). We perform a reduction from the following undecidable problem: given a blind POMDP and a set F of states, does there exist a strategy that ensures the Büchi objective $\square \diamond F$ with positive probability. The structure of the cLTS we construct is similar to the one of the example from Fig. 2, except that the states q_1 and q_2 are replaced with two copies of the POMDP. Consistently a and \bar{a} are replaced by two copies of the alphabet of the POMDP with one of them bared. From F states in the first copy, with a non bared action one moves to the second one, and from any state, with bared actions, one moves back from the second copy to the first one, or moves from the first copy to r_0 .

The following immediate corollary is interesting since both the existence of a strategy achieving a Büchi objective almost surely, and the existence of strategy achieving a safety objective with positive probability are decidable for POMDP [2, 6].

Corollary 1. *The problem whether, given a POMDP M with subsets of states F and I , there exists a strategy π with $\mathbb{P}_{\pi}(M \models \square \diamond F) = 1$ and $\mathbb{P}_{\pi}(M \models \square I) > 0$, is undecidable.*

Given that the general safe active diagnosis problem is undecidable, and that belief-based strategies are not sufficient to achieve safe diagnosability, we consider now the restriction of the safe active diagnosis problem to belief-based strategies. Similarly to the case of active diagnosis, we reduce the safe active probabilistic diagnosis for belief-based-strategies to some verification question on POMDP.

Theorem 4. *The safe active probabilistic diagnosis problem restricted to belief-based strategies is in NEXPTIME and EXPTIME-hard.*

5 Conclusion

We studied the active diagnosis and safe active diagnosis problems for probabilistic discrete event systems, within a unifying POMDP framework. While the active diagnosis problem is EXPTIME-complete, the safe active diagnosis problem is undecidable in general, and belongs to NEXPTIME when restricted to belief-based strategies. Since the lower and upper bounds do not coincide for the latter problem, we strive to close the gap between these bounds in future work. More generally, we will investigate the safe active diagnosis problem restricted to finite-memory strategies. Another problem, closely related to diagnosability, is the predictability problem: given any observation, can we detect that the occurrence of a fault *before* it happens? Last, given the tight relation probabilistic diagnosis has with verification problems for POMDP, we plan to investigate further POMDP problems with multiple objectives.

References

1. Baier, C., Bertrand, N., Grösser, M.: Probabilistic ω -automata. *Journal of the ACM* 59(1), 1–52 (2012)
2. Bertrand, N., Genest, B., Gimbert, H.: Qualitative determinacy and decidability of stochastic games with signals. In: *Proceedings of LICS 2009*, pp. 319–328. IEEE Computer Society (2009)
3. Cabasino, M., Giua, A., Lafortune, S., Seatzu, C.: Diagnosability analysis of unbounded Petri nets. In: *Proceedings of CDC 2009*, pp. 1267–1272. IEEE (2009)
4. Cassez, F., Tripakis, S.: Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae* 88, 497–540 (2008)
5. Chanthery, E., Pencolé, Y.: Monitoring and active diagnosis for discrete-event systems. In: *Proceedings of SP 2009*, pp. 1545–1550. Elsevier (2009)
6. Chatterjee, K., Doyen, L., Gimbert, H., Henzinger, T.A.: Randomness for free. In: Hliněný, P., Kučera, A. (eds.) *MFCS 2010*. LNCS, vol. 6281, pp. 246–257. Springer, Heidelberg (2010)
7. Fabre, E., Jezequel, L.: On the construction of probabilistic diagnosers. In: *Proceeding of WODES 2010*, pp. 229–234. Elsevier (2010)
8. Haar, S., Haddad, S., Melliti, T., Schwoon, S.: Optimal constructions for active diagnosis. In: *Proceedings of FSTTCS 2013*. LIPIcs, vol. 24, pp. 527–539. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
9. Morvan, C., Pinchinat, S.: Diagnosability of pushdown systems. In: Namjoshi, K., Zeller, A., Ziv, A. (eds.) *HVC 2009*. LNCS, vol. 6405, pp. 21–33. Springer, Heidelberg (2011)
10. Sampath, M., Lafortune, S., Teneketzis, D.: Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control* 43(7), 908–929 (1998)
11. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete-event systems. *IEEE Trans. Aut. Cont.* 40(9), 1555–1575 (1995)

12. Thorsley, D., Teneketzis, D.: Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control* 50(4), 476–492 (2005)
13. Thorsley, D., Teneketzis, D.: Active acquisition of information for diagnosis and supervisory control of discrete-event systems. *Journal of Discrete Event Dynamic Systems* 17, 531–583 (2007)
14. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: *Proceedings of FOCS 1985*, pp. 327–338. IEEE Computer Society Press (1985)
15. Yoo, T.-S., Lafortune, S.: Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Trans. Automat. Contr.* 47(9), 1491–1495 (2002)

Analysis of Probabilistic Basic Parallel Processes

Rémi Bonnet¹, Stefan Kiefer^{1,*}, and Anthony Widjaja Lin^{1,2,**}

¹ University of Oxford, UK

² Academia Sinica, Taiwan

Abstract. Basic Parallel Processes (BPPs) are a well-known subclass of Petri Nets. They are the simplest common model of concurrent programs that allows unbounded spawning of processes. In the probabilistic version of BPPs, every process generates other processes according to a probability distribution. We study the decidability and complexity of fundamental qualitative problems over probabilistic BPPs — in particular reachability with probability 1 of different classes of target sets (e.g. upward-closed sets). Our results concern both the Markov-chain model, where processes are scheduled randomly, and the MDP model, where processes are picked by a scheduler.

1 Introduction

We study probabilistic basic parallel processes (pBPP), which is a stochastic model for concurrent systems with unbounded process spawning. Processes can be of different types, and each type has a fixed probability distribution for generating new sub-processes. A pBPP can be described using a notation similar to that of stochastic context-free grammars. For instance,

$$X \xrightarrow{0.2} XX \quad X \xrightarrow{0.3} XY \quad X \xrightarrow{0.5} \varepsilon \quad Y \xrightarrow{0.7} X \quad Y \xrightarrow{0.3} Y$$

describes a system with two types of processes. Processes of type X can generate two processes of type X , one process of each type, or zero processes with probabilities 0.2, 0.3, and 0.5, respectively. Processes of type Y can generate one process, of type X or Y , with probability 0.7 and 0.3. The order of processes on the right-hand side of each rule is not important. Readers familiar with process algebra will identify this notation as a probabilistic version of Basic Parallel Processes (BPPs), which is widely studied in automated verification, see e.g. [7,11,6,13,12,9],

A *configuration* of a pBPP indicates, for each type X , how many processes of type X are present. Writing Γ for the finite set of types, a configuration is thus an element of \mathbb{N}^Γ . In a configuration $\alpha \in \mathbb{N}^\Gamma$ with $\alpha(X) \geq 1$ an X -process may be scheduled. Whenever a process of type X is scheduled, a rule with X on the left-hand side is picked randomly according to the probabilities of the rules, and then an X -process is replaced by processes as on the right-hand side.

* Stefan Kiefer is supported by a Royal Society University Research Fellowship.

** Anthony W. Lin was supported by EPSRC Research Fellowship (EP/H026878/1).

In the example above, if an X -process is scheduled, then with probability 0.3 it is replaced by a new X -process and by a new Y -process. This leads to a new configuration, α' , with $\alpha'(X) = \alpha(X)$ and $\alpha'(Y) = \alpha(Y) + 1$.

Which type is scheduled in a configuration $\alpha \in \mathbb{N}^T$ depends on the model under consideration. One possibility is that the type to be scheduled is selected randomly among those types X with $\alpha(X) \geq 1$. In this way, a pBPP induces an (infinite-state) Markov chain. We consider two versions of this Markov chain: in one version the type to be scheduled is picked using a uniform distribution on those types with at least one waiting process; in the other version the type is picked using a uniform distribution on the waiting processes. For instance, in configuration α with $\alpha(X) = 1$ and $\alpha(Y) = 2$, according to the “type” version, the probability of scheduling X is $1/2$, whereas in the “process” version, the probability is $1/3$. Both models seem to make equal sense, so we consider them both in this paper. As it turns out their difference is unimportant for our results.

In many contexts (e.g. probabilistic distributed protocols — see [15,14]), it is more natural that this scheduling decision is not taken randomly, but by a *scheduler*. Then the pBPP induces a Markov decision process (MDP), where a scheduler picks a type X to be scheduled, but the rule with X on the left-hand side is selected probabilistically according to the probabilities on the rules.

In this paper we provide decidability results concerning *coverability* with probability 1, or “almost-sure” coverability, which is a fundamental qualitative property of pBPPs. We say a configuration $\beta \in \mathbb{N}^T$ *covers* a configuration $\phi \in \mathbb{N}^T$ if $\beta \geq \phi$ holds, where \geq is meant componentwise. For instance, ϕ may model a configuration with one producer and one consumer; then $\beta \geq \phi$ means that a transaction between a producer and a consumer can take place. Another example is a critical section that can be entered only when a lock is obtained. Given a pBPP, an initial configuration α , and target configurations ϕ_1, \dots, ϕ_k , the *coverability problem* asks whether with probability 1 it is the case that starting from α a configuration β is reached that covers some ϕ_i . One can equivalently view the problem as almost-sure reachability of an upward-closed set.

In Section 3 we show using a Karp-Miller-style construction that the coverability problem for pBPP Markov chains is decidable. We provide a nonelementary lower complexity bound. In Section 4 we consider the coverability problem for MDPs. There the problem appears in two flavours, depending on whether the scheduler is “angelic” or “demonic”. In the angelic case we ask whether there *exists* a scheduler so that a target is almost-surely covered. We show that this problem is decidable, and if such a scheduler does exist one can synthesize one. In the demonic case we ask whether a target is almost-surely covered, no matter what the scheduler (an operating system, for instance) does. For the question to make sense we need to exclude unfair schedulers, i.e., those that never schedule a waiting process. Using a robust fairness notion (k -fairness), which does not depend on the exact probabilities in the rules, we show that the demonic problem is also decidable. In Section 5 we show for the Markov chain and for both versions of the MDP problem that the coverability problem becomes P-time solvable, if the target configurations ϕ_i consist of only one process each (i.e., are unit vectors). Such target

configurations naturally arise in concurrent systems (e.g. freedom from deadlock: whether *at least one* process eventually goes into a critical section). Finally, in Section 6 we show that the almost-sure reachability problem for semilinear sets, which generalizes the coverability problem, is undecidable for pBPP Markov chains and MDPs. Some missing proofs can be found in [2].

Related Work. (Probabilistic) BPPs can be viewed as (stochastic) Petri nets where each transition has exactly one input place. Stochastic Petri nets, in turn, are equivalent to probabilistic vector addition systems with states (pVASSs), whose reachability and coverability problems were studied in [1]. This work is close to ours; in fact, we build on fundamental results of [1]. Whereas we show that coverability for the Markov chain induced by a pBPP is decidable, it is shown in [1] that the problem is undecidable for general pVASSs. In [1] it is further shown for general pVASSs that coverability becomes decidable if the target sets are “ Q -states”. If we apply the same restriction on the target sets, coverability becomes polynomial-time decidable for pBPPs, see Section 5. MDP problems are not discussed in [1].

The MDP version of pBPPs was studied before under the name *task systems* [3]. There, the scheduler aims at a “space-efficient” scheduling, which is one where the maximal number of processes is minimised. Goals and techniques of this paper are very different from ours.

Certain classes of non-probabilistic 2-player games on Petri nets were studied in [16]. Our MDP problems can be viewed as games between two players, Scheduler and Probability. One of our proofs (the proof of Theorem 11) is inspired by proofs in [16].

The notion of k -fairness that we consider in this paper is not new. Similar notions have appeared in the literature of concurrent systems under the name of “bounded fairness” (e.g. see [5] and its citations).

2 Preliminaries

We write $\mathbb{N} = \{0, 1, 2, \dots\}$. For a countable set X we write $\text{dist}(X)$ for the set of *probability distributions* over X ; i.e., $\text{dist}(X)$ consists of those functions $f : X \rightarrow [0, 1]$ such that $\sum_{x \in X} f(x) = 1$.

Markov Chains. A *Markov chain* is a pair $\mathcal{M} = (Q, \delta)$, where Q is a countable (finite or infinite) set of states, and $\delta : Q \rightarrow \text{dist}(Q)$ is a probabilistic transition function that maps a state to a probability distribution over the successor states. Given a Markov chain we also write $s \xrightarrow{p} t$ or $s \rightarrow t$ to indicate that $p = \delta(s)(t) > 0$. A *run* is an infinite sequence $s_0 s_1 \dots \in Q^\omega$ with $s_i \rightarrow s_{i+1}$ for $i \in \mathbb{N}$. We write $\text{Run}(s_0 \dots s_k)$ for the set of runs that start with $s_0 \dots s_k$. To every initial state $s_0 \in S$ we associate the probability space $(\text{Run}(s_0), \mathcal{F}, \mathcal{P})$ where \mathcal{F} is the σ -field generated by all basic cylinders $\text{Run}(s_0 \dots s_k)$ with $s_0 \dots s_k \in Q^*$, and $\mathcal{P} : \mathcal{F} \rightarrow [0, 1]$ is the unique probability measure such that $\mathcal{P}(\text{Run}(s_0 \dots s_k)) = \prod_{i=1}^k \delta(s_{i-1})(s_i)$. For a state $s_0 \in Q$ and a set $F \subseteq Q$, we write $s_0 \models \diamond F$

for the event that a run started in s_0 hits F . Formally, $s_0 \models \diamond F$ can be seen as the set of runs $s_0 s_1 \cdots$ such that there is $i \geq 0$ with $s_i \in F$. Clearly we have $\mathcal{P}(s_0 \models \diamond F) > 0$ if and only if in \mathcal{M} there is a path from s_0 to a state in F . Similarly, for $Q_1, Q_2 \subseteq Q$ we write $s_0 \models Q_1 \cup Q_2$ to denote the set of runs $s_0 s_1 \cdots$ such that there is $j \geq 0$ with $s_j \in Q_2$ and $s_i \in Q_1$ for all $i < j$. We have $\mathcal{P}(s_0 \models Q_1 \cup Q_2) > 0$ if and only if in \mathcal{M} there is a path from s_0 to a state in Q_2 using only states in Q_1 . A Markov chain is *globally coarse* with respect to a set $F \subseteq Q$ of configurations, if there exists $c > 0$ such that for all $s_0 \in Q$ we have that $\mathcal{P}(s_0 \models \diamond F) > 0$ implies $\mathcal{P}(s_0 \models \diamond F) \geq c$.

Markov Decision Processes. A Markov decision process (MDP) is a tuple $\mathcal{D} = (Q, A, En, \delta)$, where Q is a countable set of states, A is a finite set of actions, $En : Q \rightarrow 2^A \setminus \emptyset$ is an action enabledness function that assigns to each state s the set $En(s)$ of actions enabled in s , and $\delta : S \times A \rightarrow \text{dist}(S)$ is a probabilistic transition function that maps a state s and an action $a \in En(s)$ enabled in s to a probability distribution over the successor states. A *run* is an infinite alternating sequence of states and actions $s_0 a_1 s_1 a_2 \cdots$ such that for all $i \geq 1$ we have $a_i \in En(s_{i-1})$ and $\delta(s_{i-1}, a_i)(s_i) > 0$. For a finite word $w = s_0 a_1 \cdots s_{k-1} a_k s_k \in Q(AQ)^*$ we write $\text{last}(w) = s_k$. A *scheduler* for \mathcal{D} is a function $\sigma : Q(AQ)^* \rightarrow \text{dist}(A)$ that maps a run prefix $w \in Q(AQ)^*$, representing the history of a play, to a probability distribution over the actions enabled in $\text{last}(w)$. We write $\text{Run}(w)$ for the set of runs that start with $w \in Q(AQ)^*$. To an initial state $s_0 \in S$ and a scheduler σ we associate the probability space $(\text{Run}(s_0), \mathcal{F}, \mathcal{P}_\sigma)$, where \mathcal{F} is the σ -field generated by all basic cylinders $\text{Run}(w)$ with $w \in \{s_0\}(AQ)^*$, and $\mathcal{P}_\sigma : \mathcal{F} \rightarrow [0, 1]$ is the unique probability measure such that $\mathcal{P}(\text{Run}(s_0)) = 1$, and $\mathcal{P}(\text{Run}(was)) = \mathcal{P}(\text{Run}(w)) \cdot \sigma(w)(a) \cdot \delta(\text{last}(w), a)(s)$ for all $w \in \{s_0\}(AQ)^*$ and all $a \in A$ and all $s \in Q$. A scheduler σ is called *deterministic* if for all $w \in Q(AQ)^*$ there is $a \in A$ with $\sigma(w)(a) = 1$. A scheduler σ is called *memoryless* if for all $w, w' \in Q(AQ)^*$ with $\text{last}(w) = \text{last}(w')$ we have $\sigma(w) = \sigma(w')$. When specifying events, i.e., measurable subsets of $\text{Run}(s_0)$, the actions are often irrelevant. Therefore, when we speak of runs $s_0 s_1 \cdots$ we mean the runs $s_0 a_1 s_1 a_2 \cdots$ for arbitrary $a_1, a_2, \dots \in A$. E.g., in this understanding we view $s_0 \models \diamond F$ with $s_0 \in Q$ and $F \subseteq Q$ as an event.

Probabilistic BPPs and Their Configurations. A probabilistic BPP (pBPP) is a tuple $\mathcal{S} = (\Gamma, \hookrightarrow, \text{Prob})$, where Γ is a finite set of *types*, $\hookrightarrow \subseteq \Gamma \times \mathbb{N}^\Gamma$ is a finite set of *rules* such that for every $X \in \Gamma$ there is at least one rule of the form $X \hookrightarrow \alpha$, and Prob is a function that to every rule $X \hookrightarrow \alpha$ assigns its probability $\text{Prob}(X \hookrightarrow \alpha) \in (0, 1] \cap \mathbb{Q}$ so that for all $X \in \Gamma$ we have $\sum_{X \hookrightarrow \alpha} \text{Prob}(X \hookrightarrow \alpha) = 1$. We write $X \xrightarrow{p} \alpha$ to denote that $\text{Prob}(X \hookrightarrow \alpha) = p$. A *configuration* of \mathcal{S} is an element of \mathbb{N}^Γ . We write $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ for componentwise addition and subtraction of two configurations α_1, α_2 . When there is no confusion, we may identify words $u \in \Gamma^*$ with the configuration $\alpha \in \mathbb{N}^\Gamma$ such that for all $X \in \Gamma$ we have that $\alpha(X) \in \mathbb{N}$ is the number of occurrences of X in u . For instance, we write XXY or XYX for the configuration α with $\alpha(X) = 2$ and $\alpha(Y) = 1$ and $\alpha(Z) = 0$ for $Z \in \Gamma \setminus \{X, Y\}$. In particular, we may write ε for α with $\alpha(X) = 0$

for all $X \in \Gamma$. For configurations α, β we write $\alpha \leq \beta$ if $\alpha(X) \leq \beta(X)$ holds for all $X \in \Gamma$; we write $\alpha < \beta$ if $\alpha \leq \beta$ but $\alpha \neq \beta$. For a configuration α we define the *number of types* $|\alpha|_{type} = |\{X \in \Gamma \mid \alpha(X) \geq 1\}|$ and the *number of processes* $|\alpha|_{proc} = \sum_{X \in \Gamma} \alpha(X)$. Observe that we have $|\alpha|_{type} \leq |\alpha|_{proc}$. A set $F \subseteq \mathbb{N}^\Gamma$ of configurations is called *upward-closed* (*downward-closed*, respectively) if for all $\alpha \in F$ we have that $\alpha \leq \beta$ implies $\beta \in F$ ($\alpha \geq \beta$ implies $\beta \in F$, respectively). For $\alpha \in \mathbb{N}^\Gamma$ we define $\alpha \uparrow := \{\beta \in \mathbb{N}^\Gamma \mid \beta \geq \alpha\}$. For $F \subseteq \mathbb{N}^\Gamma$ and $\alpha \in F$ we say that α is a *minimal element* of F , if there is no $\beta \in F$ with $\beta < \alpha$. It follows from Dickson's lemma that every upward-closed set has finitely many minimal elements; i.e., F is upward-closed if and only if $F = \phi_1 \uparrow \cup \dots \cup \phi_n \uparrow$ holds for some $n \in \mathbb{N}$ and $\phi_1, \dots, \phi_n \in \mathbb{N}^\Gamma$.

Markov Chains Induced by a pBPP. To a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ we associate the Markov chains $\mathcal{M}_{type}(\mathcal{S}) = (\mathbb{N}^\Gamma, \delta_{type})$ and $\mathcal{M}_{proc}(\mathcal{S}) = (\mathbb{N}^\Gamma, \delta_{proc})$ with $\delta_{type}(\varepsilon, \varepsilon) = \delta_{proc}(\varepsilon, \varepsilon) = 1$ and for $\alpha \neq \varepsilon$

$$\delta_{type}(\alpha, \gamma) = \sum_{\substack{X \xrightarrow{p} \beta \text{ s.t. } \alpha(X) \geq 1 \\ \text{and } \gamma = \alpha - X + \beta}} \frac{p}{|\alpha|_{type}} \quad \text{and} \quad \delta_{proc}(\alpha, \gamma) = \sum_{\substack{X \xrightarrow{p} \beta \text{ s.t.} \\ \gamma = \alpha - X + \beta}} \frac{\alpha(X) \cdot p}{|\alpha|_{proc}}.$$

In words, the new configuration γ is obtained from α by replacing an X -process with a configuration randomly sampled according to the rules $X \xrightarrow{p} \beta$. In $\mathcal{M}_{type}(\mathcal{S})$ the selection of X is based on the number of types in α , whereas in $\mathcal{M}_{proc}(\mathcal{S})$ it is based on the number of processes in α . We have $\delta_{type}(\alpha, \gamma) = 0$ iff $\delta_{proc}(\alpha, \gamma) = 0$. We write \mathcal{P}_{type} and \mathcal{P}_{proc} for the probability measures in $\mathcal{M}_{type}(\mathcal{S})$ and $\mathcal{M}_{proc}(\mathcal{S})$, respectively.

The MDP Induced by a pBPP. To a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ we associate the MDP $\mathcal{D}(\mathcal{S}) = (\mathbb{N}^\Gamma, \Gamma \cup \{\perp\}, En, \delta)$ with a fresh action $\perp \notin \Gamma$, and $En(\alpha) = \{X \in \Gamma \mid \alpha(X) \geq 1\}$ for $\varepsilon \neq \alpha \in \mathbb{N}^\Gamma$ and $En(\varepsilon) = \{\perp\}$, and $\delta(\alpha, X)(\alpha - X + \beta) = p$ whenever $\alpha(X) \geq 1$ and $X \xrightarrow{p} \beta$, and $\delta(\varepsilon, \perp)(\varepsilon) = 1$. As in the Markov chain, the new configuration γ is obtained from α by replacing an X -process with a configuration randomly sampled according to the rules $X \xrightarrow{p} \beta$. But in contrast to the Markov chain the selection of X is up to a scheduler.

3 The Coverability Problem for the Markov Chain

In this section we study the *coverability problem* for the Markov chains induced by a pBPP. We say a run $\alpha_0 \alpha_1 \dots$ of a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ *covers* a configuration $\phi \in \mathbb{N}^\Gamma$, if $\alpha_i \geq \phi$ holds for some $i \in \mathbb{N}$. The coverability problem asks whether it is almost surely the case that some configuration from a finite set $\{\phi_1, \dots, \phi_n\}$ will be covered. More formally, the coverability problem is the following. Given a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$, an initial configuration $\alpha_0 \in \mathbb{N}^\Gamma$, and finitely many configurations ϕ_1, \dots, ϕ_n , does $\mathcal{P}_{type}(\alpha_0 \models \diamond F) = 1$ hold,

where $F = \phi_1 \uparrow \cup \dots \cup \phi_n \uparrow$? Similarly, does $\mathcal{P}_{proc}(\alpha_0 \models \diamond F) = 1$ hold? We will show that those two questions always have the same answer.

In Section 3.1 we show that the coverability problem is decidable. In Section 3.2 we show that the complexity of the coverability problem is nonelementary.

3.1 Decidability

For deciding the coverability problem we use the approach of [1]. The following proposition is crucial for us:

Proposition 1. *Let $\mathcal{M} = (Q, \delta)$ be a Markov chain and $F \subseteq Q$ such that \mathcal{M} is globally coarse with respect to F . Let $\bar{F} = Q \setminus F$ be the complement of F and let $\tilde{F} := \{s \in Q \mid \mathcal{P}(s \models \diamond F) = 0\} \subseteq \bar{F}$ denote the set of states from which F is not reachable in \mathcal{M} . Let $s_0 \in Q$. Then we have $\mathcal{P}(s_0 \models \diamond F) = 1$ if and only if $\mathcal{P}(s_0 \models \bar{F} \cup \tilde{F}) = 0$.*

Proof. Immediate from [1, Lemmas 3.7, 5.1 and 5.2]. □

In other words, Proposition 1 states that F is almost surely reached if and only if there is no path to \tilde{F} that avoids F . Proposition 1 will allow us to decide the coverability problem by computing only reachability relations in \mathcal{M} , ignoring the probabilities.

Recall that for a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$, the Markov chains $\mathcal{M}_{type}(\mathcal{S})$ and $\mathcal{M}_{proc}(\mathcal{S})$ have the same structure; only the transition probabilities differ. In particular, if $F \subseteq \mathbb{N}^\Gamma$ is upward-closed, the set \tilde{F} , as defined in Proposition 1, is the same for $\mathcal{M}_{type}(\mathcal{S})$ and $\mathcal{M}_{proc}(\mathcal{S})$. Moreover, we have the following proposition (full proof in [2]).

Proposition 2. *Let $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ be a pBPP. Let $F \subseteq \mathbb{N}^\Gamma$ be upward-closed. Then the Markov chains $\mathcal{M}_{type}(\mathcal{S})$ and $\mathcal{M}_{proc}(\mathcal{S})$ are globally coarse with respect to F .*

Proof (sketch). The statement about $\mathcal{M}_{type}(\mathcal{S})$ follows from [1, Theorem 4.3]. For the statement about $\mathcal{M}_{proc}(\mathcal{S})$ it is crucial to argue that starting with any configuration $\alpha \in \mathbb{N}^\Gamma$ it is the case with probability 1 that every type X with $\alpha(X) \geq 1$ is eventually scheduled. Since F is upward-closed it follows that for all $\alpha, \beta \in \mathbb{N}^\Gamma$ with $\alpha \leq \beta$ we have $\mathcal{P}_{proc}(\alpha \models \diamond F) \leq \mathcal{P}_{proc}(\beta \models \diamond F)$. Then the statement follows from Dickson's lemma.

For an illustration of the challenge, consider the pBPP with $X \xrightarrow{1} XX$ and $Y \xrightarrow{1} YY$, and let $F = XX \uparrow$. Clearly we have $\mathcal{P}_{proc}(X \models \diamond F) = 1$, as the X -process is scheduled immediately. Now let $\alpha_0 = XY$. Since $\alpha_0 \geq X$, the inequality claimed above implies $\mathcal{P}_{proc}(\alpha_0 \models F) = 1$. Indeed, the probability that the X -process in α_0 is *never* scheduled is at most $\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdot \dots$, which is 0. Hence $\mathcal{P}_{proc}(\alpha_0 \models \diamond F) = 1$. □

The following proposition follows by combining Propositions 1 and 2.

Proposition 3. *Let $\mathcal{S} = (\Gamma, \hookrightarrow, \text{Prob})$ be a pBPP. Let $F \subseteq \mathbb{N}^\Gamma$ be upward-closed. Let $\alpha_0 \in \mathbb{N}^\Gamma$. We have:*

$$\begin{aligned} \mathcal{P}_{type}(\alpha_0 \models \diamond F) = 1 &\iff \mathcal{P}_{type}(\alpha_0 \models \bar{F}U\tilde{F}) = 0 \\ &\iff \mathcal{P}_{proc}(\alpha_0 \models \bar{F}U\tilde{F}) = 0 \iff \mathcal{P}_{proc}(\alpha_0 \models \diamond F) = 1 \end{aligned}$$

By Proposition 3 we may in the following omit the subscript from $\mathcal{P}_{type}, \mathcal{P}_{proc}, \mathcal{M}_{type}, \mathcal{M}_{proc}$ if it does not matter. We have the following theorem.

Theorem 4. *The coverability problem is decidable: given a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, \text{Prob})$, an upward-closed set $F \subseteq \mathbb{N}^\Gamma$, and a configuration $\alpha_0 \in \mathbb{N}^\Gamma$, it is decidable whether $\mathcal{P}(\alpha_0 \models \diamond F) = 1$ holds.*

Proof. The complement of \tilde{F} (i.e., the set of configurations from which F is reachable) is upward-closed, and its minimal elements can be computed by a straightforward fixed-point computation (this is even true for the more general model of pVASS, e.g., see [1, Remark 4.2]). By Proposition 3 it suffices to decide whether $\mathcal{P}(\alpha_0 \models \bar{F}U\tilde{F}) > 0$ holds. Define $R := \{\alpha \in \tilde{F} \mid \alpha \text{ is reachable from } \alpha_0 \text{ via } \bar{F}\text{-configurations}\}$. Observe that $\mathcal{P}(\alpha_0 \models \bar{F}U\tilde{F}) > 0$ if and only if $R \cap \tilde{F} \neq \emptyset$. We can now give a Karp-Miller-style algorithm for checking that $R \cap \tilde{F} \neq \emptyset$: (i) Starting from α_0 , build a tree of configurations reachable from α_0 via \bar{F} -configurations (i.e., at no stage F -configurations are added to this tree) — for example, in a breadth-first search manner — but stop expanding a leaf node α_k as soon as we discover that the branch $\alpha_0 \rightarrow \dots \rightarrow \alpha_k$ satisfies the following: $\alpha_j \leq \alpha_k$ for some $j < k$. (ii) As soon as a node $\alpha \in \tilde{F}$ is generated, terminate and output “yes”. (iii) When the tree construction is completed without finding nodes in \tilde{F} , terminate and output “no”.

To prove correctness of the above algorithm, we first prove termination. To this end, it suffices to show that the constructed tree is finite. To see this, observe first that every branch in the constructed tree is of finite length. This is an immediate consequence of Dickon’s lemma and our policy of terminating a leaf node α that satisfies $\alpha' \leq \alpha$, for some ancestor α' of α in this tree. Now since all branches of the tree are finite, König’s lemma shows that the tree itself must be finite (since each node has finite degree).

To prove partial correctness, it suffices to show that the policy of terminating a leaf node α that satisfies $\alpha' \leq \alpha$, for some ancestor α' of α in this tree, is valid. That is, we want to show that if $R \cap \tilde{F} \neq \emptyset$ then a witnessing vector $\gamma \in R \cap \tilde{F}$ will be found by the algorithm. We have the following lemma whose proof is in [2].

Lemma 5. *Let $\alpha_0 \in \tilde{F}$ and let $\gamma \in \mathbb{N}^\Gamma$. Let $\alpha_0 \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_k$ be a shortest path in $\mathcal{M}(\mathcal{S})$ such that $\alpha_0, \dots, \alpha_k \in \tilde{F}$ and $\alpha_k \leq \gamma$. Then for all i, j with $0 \leq i < j \leq k$ we have $\alpha_i \not\leq \alpha_j$.*

Let $R \cap \tilde{F} \neq \emptyset$ and let $\gamma \in \mathbb{N}^\Gamma$ be a minimal element of $R \cap \tilde{F}$. By Lemma 5 our algorithm does not prune any shortest path from α_0 to γ . Hence it outputs “yes”. \square

3.2 Nonelementary Lower Bound

We have the following lower-bound result:

Theorem 6. *The complexity of the coverability problem is nonelementary.*

The proof is technically involved.

Proof (sketch). We claim that there exists a nonelementary function f such that given a 2-counter machine M running in space $f(k)$, we can compute a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ of size $\leq k$, an upward-closed set $F \subseteq \mathbb{N}^\Gamma$ (with at most k minimal elements, described by numbers at most k), and a type $X_0 \in \Gamma$, such that $\mathcal{P}(X_0 \models \diamond F) = 1$ holds if and only if M does not terminate. Recall that by Proposition 3 we have that $\mathcal{P}(X_0 \models \diamond F) = 1$ is equivalent to $\mathcal{P}(X_0 \models \bar{F} \cup \tilde{F}) = 0$.

Since the exact values of the probabilities do not matter, it suffices to construct a (nonprobabilistic) BPP \mathcal{S} . Further, by adding processes that can spawn everything (and hence cannot take part in \tilde{F} -configurations) one can change the condition of reaching \tilde{F} to reaching a downward closed set $G \subseteq \bar{F}$. So the problem we are reducing to is: does there exist a path in \mathcal{S} that is contained in \bar{F} and goes from X_0 to a downward closed set G .

By defining F suitably we can add various restrictions on the behaviour of our BPP. For example, the following example allows X to be turned into Y if and only if there is no Z present:

$$X \hookrightarrow YW \quad W \hookrightarrow \varepsilon \quad F = WZ\uparrow$$

Doubling the number of a given process is straightforward, and it is also possible to divide the number of a given process by two. Looking only at runs inside \bar{F} , the following BPP can turn all its X -processes into half as many X' -processes. (Note that more X' -processes could be spawned, but because of the monotonicity of the system, the “best” runs are those that spawn a minimal number of processes.)

$$\begin{aligned} X &\hookrightarrow TP & T &\hookrightarrow \bar{P} & P &\hookrightarrow \varepsilon & \bar{P} &\hookrightarrow \varepsilon \\ P_1 &\hookrightarrow \bar{P}_2 & \bar{P}_2 &\hookrightarrow P_2 & P_2 &\hookrightarrow \bar{P}_1 & \bar{P}_1 &\hookrightarrow P_1 X' \\ F &= P\bar{P}_1\uparrow \cup P\bar{P}_2\uparrow \cup \bar{P}P_1\uparrow \cup \bar{P}P_2\uparrow \cup T^2\uparrow \\ \alpha_{init} &= X^n \bar{P}_1 \end{aligned}$$

Let us explain this construction. In order to make an X -process disappear, we need to create temporary processes P and \bar{P} . However, these processes are incompatible, respectively, with \bar{P}_i and P_i . Thus, destroying an X -process requires the process P_1 to move into \bar{P}_1 and then into P_2 . By repeatedly destroying X -processes, this forces the creation of half as many X' -processes.

It is essential for our construction to have a loop-gadget that performs a cycle of processes $A \hookrightarrow B \hookrightarrow C \hookrightarrow A$ exactly k times (“ k -loop”). By activating/disabling transitions based on the absence/presence of an A -, B - or

C -process, we can force an operation to be performed k times. For example, assuming the construction of a k -loop gadget, the following BPP doubles the number of X -processes k times:

$$\begin{aligned}
 X &\leftrightarrow Y & Y &\leftrightarrow ZZ & Z &\leftrightarrow X \\
 & & & & & \text{(rules for } k\text{-loop on } A/B/C\text{)} \\
 F &= XB\uparrow \cup YC\uparrow \cup ZA\uparrow
 \end{aligned}$$

For the loop to perform $A \leftrightarrow B$, all X -processes have to be turned into Y . Similarly, performing $B \leftrightarrow C \leftrightarrow A$ requires the Y -processes to be turned into Z , then into X . Thus, in order to perform one iteration of the loop, one needs to double the number of X -processes.

To implement such a loop we need two more gadgets: one for creating k processes, and one for consuming k processes. By turning a created process into a consumed process one at a time, we obtain the required cycle. Here is an example:

$$\begin{aligned}
 I &\leftrightarrow A & A &\leftrightarrow B & B &\leftrightarrow C & C &\leftrightarrow \varepsilon \\
 \bar{A} &\leftrightarrow \bar{B} & \bar{B} &\leftrightarrow \bar{C}F & \bar{C} &\leftrightarrow A \\
 & & & & & \text{(rules for a gadget to consume } k \text{ processes } F\text{)} \\
 & & & & & \text{(rules for a gadget to spawn } k \text{ processes } I\text{)} \\
 F &= A\bar{A}\uparrow \cup B\bar{B}\uparrow \cup C\bar{C}\uparrow \cup AA\uparrow \cup BB\uparrow \cup CC\uparrow \\
 \alpha_{init} &= \bar{A}
 \end{aligned}$$

By combining a k -loop with a multiplier or a divider, we can spawn or consume 2^k processes. This allows us to create a 2^k -loop. By iterating this construction, we get a BPP of exponential size (each loop requires two lower-level loops) that is able to spawn or consume $2^{2^{\dots^k}}$ processes.

It remains to simulate our 2-counter machine M . The main idea is to spawn an initial budget b of processes, and to make sure that this number stays the same along the run. Zero-tests are easy to implement; the difficulty lies in the increments and decrements. The solution is to maintain, for each simulated counter, two pools of processes X and \bar{X} , such that if the counter is supposed to have value k , then we have processes $X^k \bar{X}^{b-k}$. Now, incrementing consists in turning all these processes into backup processes, except one \bar{X} -process. Then, we turn this process into an X -process, and return all backup processes to their initial type.

In [2] we provide complete details of the proofs, including graphical representations of the processes involved. \square

4 The Coverability Problem for the MDP

In the following we investigate the controlled version of the pBPP model. Recall from Section 2 that a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ induces an MDP $\mathcal{D}(\mathcal{S})$ where in a configuration $\varepsilon \neq \alpha \in \mathbb{N}^\Gamma$ a scheduler σ picks a type X with $\alpha(X) \geq 1$. The successor configuration is then obtained randomly from α according to the rules in \mathcal{S} with X on the left-hand side.

We investigate (variants of) the decision problem that asks, given $\alpha_0 \in \mathbb{N}^\Gamma$ and an upward-closed set $F \subseteq \mathbb{N}^\Gamma$, whether $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$ holds for some scheduler (or for all schedulers, respectively).

4.1 The Existential Problem

In this section we consider the scenario where we ask for a scheduler that makes the system reach an upward-closed set with probability 1. We prove the following theorem:

Theorem 7. *Given a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ and a configuration $\alpha_0 \in \mathbb{N}^\Gamma$ and an upward-closed set $F \subseteq \mathbb{N}^\Gamma$, it is decidable whether there exists a scheduler σ with $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$. If such a scheduler exists, one can compute a deterministic and memoryless scheduler σ with $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$.*

Proof (sketch). The proof (in [2]) is relatively long. The idea is to abstract the MDP $\mathcal{D}(\mathcal{S})$ (with \mathbb{N}^Γ as state space) to an “equivalent” finite-state MDP. The state space of the finite-state MDP is $Q := \{0, 1, \dots, K\}^\Gamma \subseteq \mathbb{N}^\Gamma$, where K is the largest number that appears in the minimal elements of F . For finite-state MDPs, reachability with probability 1 can be decided in polynomial time, and an optimal deterministic and memoryless scheduler can be synthesized.

When setting up the finite-state MDP, special care needs to be taken of transitions that would lead from Q to a configuration α outside of Q , i.e., $\alpha \in \mathbb{N}^\Gamma \setminus Q$. Those transitions are redirected to a probability distribution on T_α with $T_\alpha \subseteq Q$, so that each configuration in T_α is “equivalent” to some configuration $\beta \in \mathbb{N}^\Gamma$ that could be reached from α in the infinite-state MDP $\mathcal{D}(\mathcal{S})$, if the scheduler follows a particular optimal strategy in $\mathcal{D}(\mathcal{S})$. (One needs to show that indeed with probability 1 such a β is reached in the infinite-state MDP, if the scheduler acts according to this strategy.) This optimal strategy is based on the observation that whenever in configuration $\beta \in \mathbb{N}^\Gamma$ with $\beta(X) > K$ for some X , then type X can be scheduled. This is without risk, because after scheduling X , at least K processes of type X remain, which is enough by the definition of K . The benefit of scheduling such X is that processes appearing on the right-hand side of X -rules may be generated, possibly helping to reach F . For computing T_α , we rely on decision procedures for the reachability problem in Petri nets, which prohibits us from giving an upper complexity bound. \square

4.2 The Universal Problem

In this section we consider the scheduler as adversarial in the sense that it tries to avoid the upward-closed set F . We say “the scheduler wins” if it avoids F forever.

We ask if the scheduler can win with positive probability: given α_0 and F , do we have $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$ for all schedulers σ ? For the question to make sense, we need to rephrase it, as we show now. Consider the pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ with $\Gamma = \{X, Y\}$ and the rules $X \xrightarrow{1} XX$ and $Y \xrightarrow{1} YY$. Let $F = XX\uparrow$. If $\alpha_0 = X$, then, clearly, we have $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$ for all schedulers σ . However, if $\alpha_0 = XY$, then there is a scheduler σ with $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 0$: take the scheduler σ that always schedules Y and never X . Such a scheduler is intuitively *unfair*. If an operating system acts as a scheduler, a minimum requirement would be that waiting processes are scheduled eventually.

We call a run $\alpha_0 X_1 \alpha_1 X_2 \dots$ in the MDP $\mathcal{D}(\mathcal{S})$ *fair* if for all $i \geq 0$ and all $X \in \Gamma$ with $\alpha_i(X) \geq 1$ we have $X = X_j$ for some $j > i$. We call a scheduler σ *classically fair* if it produces only fair runs.

Example 8. Consider the pBPP with $X \xrightarrow{1} Y$ and $Y \xrightarrow{0.5} Y$ and $Y \xrightarrow{0.5} X$. Let $F = YY\uparrow$. Let $\alpha_0 = XX$. In configuration $\alpha = XY$ the scheduler has to choose between two options: It can pick X , resulting in the successor configuration $YY \in F$, which is a “loss” for the scheduler. Alternatively, it picks Y , which results in α or α_0 , each with probability 0.5. If it results in α , nothing has changed; if it results in α_0 , we say a “a round is completed”. Consider the scheduler σ that acts as follows. When in configuration $\alpha = XY$ and in the i th round, it picks Y until either the next round (the $(i + 1)$ st round) is completed or Y has been picked i times in this round. In the latter case it picks X and thus loses. Clearly, σ is classically fair (provided that it behaves in a classically fair way after it loses, for instances using round-robin). The probability of losing in the i th round is 2^{-i} . Hence the probability of losing is $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1 - \prod_{i=1}^{\infty} (1 - 2^{-i}) < 1$. (For this inequality, recall that for a sequence $(a_i)_{i \in \mathbb{N}}$ with $a_i \in (0, 1)$ we have $\prod_{i \in \mathbb{N}} (1 - a_i) = 0$ if and only if the series $\sum_{i \in \mathbb{N}} a_i$ diverges.) One can argue along these lines that any classically fair scheduler needs to play longer and longer rounds in order to win with positive probability. In particular, such schedulers need infinite memory.

It is hardly conceivable that an operating system would “consider” such schedulers. Note that the pBPP from the previous example has a finite state space.

In the probabilistic context, a commonly used alternative notion is *probabilistic fairness*, see e.g. [10,17] or [4] for an overview (the term *probabilistic fairness* is used differently in [4]). We call a scheduler σ *probabilistically fair* if with probability 1 it produces a fair run.

Example 9. For the pBPP from the previous example, consider the scheduler σ that picks Y until the round is completed. Then $\mathcal{P}_\sigma(\alpha \models \diamond F) = 0$ and σ is probabilistically fair.

The following example shows that probabilistic fairness for pBPPs can be unstable with respect to perturbations in the probabilities.

Example 10. Consider a pBPP with

$$X \xrightarrow{1} Y \quad Y \xrightarrow{1} XZ \quad Z \xrightarrow{p} ZZ \quad Z \xrightarrow{1-p} \varepsilon \quad \text{for some } p \in (0, 1)$$

and $F = YZ\uparrow$ and $\alpha_0 = XZ$.

Let $p \leq 0.5$. Then, by an argument on the ‘‘gambler’s ruin problem’’ (see e.g. [8, Chapter XIV]), with probability 1 each Z -process produces only finitely many other Z -processes in its ‘‘subderivation tree’’. Consider the scheduler σ that picks Z as long as there is a Z -process. With probability 1 it creates a run of the following form:

$$(XZ) \cdots (X)(Y)(XZ) \cdots (X)(Y)(XZ) \cdots (X)(Y)(XZ) \cdots$$

Such runs are fair, so σ is probabilistically fair and wins with probability 1.

Let $p > 0.5$. Then, by the same random-walk argument, with probability 1 some Z -process (i.e., at least one of the Z -processes created by Y) produces infinitely many other Z -processes. So any probabilistically fair scheduler σ produces, with probability 1, a Y -process before all Z -processes are gone, and thus loses.

We conclude that a probabilistically fair scheduler σ with $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) < 1$ exists if and only if $p \leq 0.5$.

The example suggests that deciding whether there exists a probabilistically fair scheduler σ with $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) < 1$ requires arguments on (in general) multidimensional random walks. In addition, the example shows that probabilistic fairness is not a robust notion when the exact probabilities are not known.

We aim at solving those problems by considering a stronger notion of fair runs: Let $k \in \mathbb{N}$. We call a run $\alpha_0 X_1 \alpha_1 X_2 \dots$ *k-fair* if for all $i \geq 0$ and all $X \in \Gamma$ with $\alpha_i(X) \geq 1$ we have that $X \in \{X_{i+1}, X_{i+2}, \dots, X_k\}$. In words, if $\alpha_i(X) \geq 1$, the type X has to be scheduled within time k . We call a scheduler *k-fair* if it produces only *k-fair* runs.

Theorem 11. *Given a pBPP $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$, an upward-closed set F , a number $k \in \mathbb{N}$, and a configuration $\alpha_0 \in \mathbb{N}^\Gamma$, it is decidable whether for all *k-fair* schedulers σ we have $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$.*

The proof is inspired by proofs in [16], and combines new insights with the technique of Theorem 4, see [2]. We remark that the proof shows that the exact values of the positive probabilities do not matter.

5 Q-States Target Sets

In this section, we provide a sensible restriction of input target sets which yields polynomial-time solvability of our problems. Let $Q = \{X_1, \dots, X_n\} \subseteq \Gamma$. The *Q-states set* is the upward-closed set $F = X_1\uparrow \cup \dots \cup X_n\uparrow$. There are two reasons to consider *Q-states target sets*. Firstly, *Q-states target sets* are sufficiently

expressive to capture common examples in the literature of distributed protocols, e.g., freedom from deadlock and resource starvation (standard examples include the dining philosopher problem in which case *at least one* philosopher must eat). Secondly, Q -states target sets have been considered in the literature of Petri nets: e.g., in [1]¹ the authors showed that qualitative reachability for probabilistic Vector Addition Systems with States with Q -states target sets becomes decidable whereas the same problem is undecidable with upward-closed target sets.

Theorem 12. *Let $\mathcal{S} = (\Gamma, \hookrightarrow, Prob)$ be a pBPP. Let $Q \subseteq \Gamma$ represent an upward-closed set $F \subseteq \mathbb{N}^\Gamma$. Let $\alpha_0 \in \mathbb{N}^\Gamma$ and $k \geq |\Gamma|$.*

- (a) *The coverability problem with Q -states target sets is solvable in polynomial time; i.e., we can decide in polynomial time whether $\mathcal{P}(\alpha_0 \models \diamond F) = 1$ holds.*
 (b) *We have:*

$$\begin{aligned} & \mathcal{P}(\alpha_0 \models \diamond F) = 1 \\ \iff & \mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1 \text{ holds for some scheduler } \sigma \\ \iff & \mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1 \text{ holds for all } k\text{-fair schedulers } \sigma. \end{aligned}$$

As a consequence of part (a), the existential and the k -fair universal problem are decidable in polynomial time.

Proof. Denote by $Q' \subseteq \Gamma$ the set of types $X \in \Gamma$ such that there are $\ell \in \mathbb{N}$, a path $\alpha_0, \dots, \alpha_\ell$ in the Markov chain $\mathcal{M}(\mathcal{S})$, and a type $Y \in Q$ such that $\alpha_0 = X$ and $\beta = \alpha_\ell \geq Y$. Clearly we have $Q \subseteq Q' \subseteq \Gamma$, and Q' can be computed in polynomial time.

In the following, view \mathcal{S} as a context-free grammar with empty terminal set (ignore the probabilities, and put the symbols on the right-hand sides in an arbitrary order). Remove from \mathcal{S} all rules of the form: (i) $X \hookrightarrow \alpha$ where $X \in Q$ or $\alpha(Y) \geq 1$ for some $Y \in Q$, and (ii) $X \hookrightarrow \alpha$ where $X \in \Gamma \setminus Q'$. Furthermore, add rules $X \hookrightarrow \varepsilon$ where $X \in \Gamma \setminus Q'$. Check (in polynomial time) whether in the grammar the empty word ε is produced by α_0 .

We have that ε is produced by α_0 if and only if $\mathcal{P}(\alpha_0 \models \diamond F) < 1$. This follows from Proposition 3, as the complement of \tilde{F} is the Q' -states set. Hence part (a) of the theorem follows.

For part (b), let $\mathcal{P}(\alpha_0 \models \diamond F) < 1$. By part (a) we have that ε is produced by α_0 . Then for all schedulers σ we have $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) < 1$. Trivially, as a special case, this holds for some k -fair scheduler. (Note that k -fair schedulers exist, as $k \geq |\Gamma|$.)

Conversely, let $\mathcal{P}(\alpha_0 \models \diamond F) = 1$. By part (a) we have that ε is not produced by α_0 . Then, no matter what the scheduler does, the set F remains reachable. So all k -fair schedulers will, with probability 1, hit F eventually. \square

¹ Our definition seems different from [1], but equivalent from standard embedding of Vector Addition Systems with States to Petri Nets.

6 Semilinear Target Sets

In this section, we prove that the qualitative reachability problems that we considered in the previous sections become undecidable when we extend upward-closed to semilinear target sets.

Theorem 13. *Let $\mathcal{S} = (\Gamma, \hookrightarrow, \text{Prob})$ be a pBPP. Let $F \subseteq \mathbb{N}^F$ be a semilinear set. Let $\alpha_0 \in \mathbb{N}^F$. The following problems are undecidable:*

- (a) *Does $\mathcal{P}(\alpha_0 \models \diamond F) = 1$ hold?*
- (b) *Does $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$ hold for all 7-fair schedulers σ ?*
- (c) *Does $\mathcal{P}_\sigma(\alpha_0 \models \diamond F) = 1$ hold for some scheduler σ ?*

The proofs are reductions from the control-state-reachability problem for 2-counter machines, see [2].

7 Conclusions and Future Work

In this paper we have studied fundamental qualitative coverability and other reachability properties for pBPPs. For the Markov-chain model, the coverability problem for pBPPs is decidable, which is in contrast to general pVASSs. We have also shown a nonelementary lower complexity bound. For the MDP model, we have proved decidability of the existential and the k -fair version of the universal coverability problem. The decision algorithms for the MDP model are not (known to be) elementary, as they rely on Petri-net reachability and a Karp-Miller-style construction, respectively. It is an open question whether there exist elementary algorithms. Another open question is whether the universal MDP problem without any fairness constraints is decidable.

We have given examples of problems where the answer depends on the exact probabilities in the pBPP. This is also true for the reachability problem for finite sets: Given a pBPP and $\alpha_0 \in \mathbb{N}^F$ and a *finite* set $F \subseteq \mathbb{N}^F$, the reachability problem for finite sets asks whether we have $\mathcal{P}(\alpha_0 \models \diamond F) = 1$ in the Markov chain $\mathcal{M}(\mathcal{S})$. Similarly as in Example 10 the answer may depend on the exact probabilities: consider the pBPP with $X \xrightarrow{p} XX$ and $X \xrightarrow{1-p} \varepsilon$, and let $\alpha_0 = XX$ and $F = \{X\}$. Then we have $\mathcal{P}(\alpha_0 \models \diamond F) = 1$ if and only if $p \leq 1/2$. The same is true in both the existential and the universal MDP version of this problem. Decidability of all these problems is open, but clearly decision algorithms would have to use techniques that are very different from ours, such as analyses of multidimensional random walks.

On a more conceptual level we remark that the problems studied in this paper are qualitative in two senses: (a) we ask whether certain events happen with probability 1 (rather than > 0.5 etc.); and (b) the exact probabilities in the rules of the given pBPP do not matter. Even if the system is nondeterministic and not probabilistic, properties (a) and (b) allow for an interpretation of our results in terms of nondeterministic BPPs, where the nondeterminism is constrained by the laws of probability, thus imposing a special but natural kind of fairness. It would be interesting to explore this kind of “weak” notion of probability for other (infinite-state) systems.

References

1. Abdulla, P.A., Ben Henda, N., Mayr, R.: Decisive Markov chains. *Logical Methods in Computer Science* 3(4:7) (2007)
2. Bonnet, R., Kiefer, S., Lin, A.W.: Analysis of probabilistic basic parallel processes. Technical report, arxiv.org (2014), <http://arxiv.org/abs/1401.4130>
3. Brázdil, T., Esparza, J., Kiefer, S., Luttenberger, M.: Space-efficient scheduling of stochastically generated tasks. *Information and Computation* 210, 87–110 (2012)
4. de Alfaro, L.: From fairness to chance. *Electronic Notes in Theoretical Computer Science* 22, 55–87 (1999)
5. Dershowitz, N., Jayasimha, D.N., Park, S.: Bounded fairness. In: Dershowitz, N. (ed.) *Verification: Theory and Practice*. LNCS, vol. 2772, pp. 304–317. Springer, Heidelberg (2004)
6. Esparza, J.: Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.* 31(1), 13–25 (1997)
7. Esparza, J., Kiehn, A.: On the model checking problem for branching time logics and basic parallel processes. In: Wolper, P. (ed.) *CAV 1995*. LNCS, vol. 939, pp. 353–366. Springer, Heidelberg (1995)
8. Feller, W.: *An introduction to probability theory and its applications*, vol. I. John Wiley & Sons (1968)
9. Fröschle, S.B., Jančar, P., Lasota, S., Sawa, Z.: Non-interleaving bisimulation equivalences on basic parallel processes. *Inf. Comput.* 208(1), 42–62 (2010)
10. Hart, S., Sharir, M., Pnueli, A.: Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems* 5(3), 356–380 (1983)
11. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theor. Comput. Sci.* 158(1&2), 143–159 (1996)
12. Hüttel, H., Kobayashi, N., Suto, T.: Undecidable equivalences for basic parallel processes. *Inf. Comput.* 207(7), 812–829 (2009)
13. Jančar, P.: Strong bisimilarity on basic parallel processes is PSPACE-complete. In: *Proceedings of LICS*, pp. 218–227 (2003)
14. Lynch, N.A., Saias, I., Segala, R.: Proving time bounds for randomized distributed algorithms. In: *PODC*, pp. 314–323 (1994)
15. Norman, G.: Analysing randomized distributed algorithms. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) *Validation of Stochastic Systems*. LNCS, vol. 2925, pp. 384–418. Springer, Heidelberg (2004)
16. Raskin, J.-F., Samuelides, M., Van Begin, L.: Games for counting abstractions. *Electronic Notes in Theoretical Computer Science* 128(6), 69–85 (2005)
17. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite state programs. In: *Proceedings of FOCS*, pp. 327–338 (1985)

Limit Synchronization in Markov Decision Processes^{*}

Laurent Doyen¹, Thierry Massart², and Mahsa Shirmohammadi^{1,2}

¹ LSV, ENS Cachan & CNRS, France

² Université Libre de Bruxelles, Belgium

Abstract. Markov decision processes (MDP) are finite-state systems with both strategic and probabilistic choices. After fixing a strategy, an MDP produces a sequence of probability distributions over states. The sequence is eventually synchronizing if the probability mass accumulates in a single state, possibly in the limit. Precisely, for $0 \leq p \leq 1$ the sequence is p -synchronizing if a probability distribution in the sequence assigns probability at least p to some state, and we distinguish three synchronization modes: (i) *sure* winning if there exists a strategy that produces a 1-synchronizing sequence; (ii) *almost-sure* winning if there exists a strategy that produces a sequence that is, for all $\epsilon > 0$, a $(1-\epsilon)$ -synchronizing sequence; (iii) *limit-sure* winning if for all $\epsilon > 0$, there exists a strategy that produces a $(1-\epsilon)$ -synchronizing sequence. We consider the problem of deciding whether an MDP is sure, almost-sure, or limit-sure winning, and we establish the decidability and optimal complexity for all modes, as well as the memory requirements for winning strategies. Our main contributions are as follows: (a) for each winning modes we present characterizations that give a PSPACE complexity for the decision problems, and we establish matching PSPACE lower bounds; (b) we show that for sure winning strategies, exponential memory is sufficient and may be necessary, and that in general infinite memory is necessary for almost-sure winning, and unbounded memory is necessary for limit-sure winning; (c) along with our results, we establish new complexity results for alternating finite automata over a one-letter alphabet.

1 Introduction

Markov decision processes (MDP) are finite-state stochastic models used in the design of systems that exhibit both controllable and stochastic behavior, such as in planning, randomized algorithms, and communication protocols [2,13,4]. The controllable choices along the execution are fixed by a strategy, and the stochastic choices describe the system response. When a strategy is fixed in an MDP, the *symbolic* outcome is a sequence of probability distributions over states of the MDP, which differs from the traditional semantics where a probability measure

^{*} This work was partially supported by the Belgian Fonds National de la Recherche Scientifique (FNRS), and by the PICS project *Quaverif* funded by the French Centre National de la Recherche Scientifique (CNRS).

is considered over sets of sequences of states. This view is adequate in many applications, such as systems biology, sensor networks, robot planning, etc. [15,5], where the system consists of several copies of the same process (molecules, sensors, robots, etc.), and the relevant information along the execution of the system is the number of processes in each state, or the relative frequency (i.e., the probability) of each state. In recent works, the verification of quantitative properties of the symbolic outcome was shown undecidable [18]. Decidability is obtained for special subclasses [6], or through approximations [1].

In this paper, we consider a general class of strategies that select actions depending on the full history of the system execution. In the context of several identical processes, the same strategy is used in every process, but the internal state of each process need not be the same along the execution, since probabilistic transitions may have different outcome in each process. Therefore, the execution of the system is best described by the sequence of probability distributions over states along the execution. Previously, the special case of word-strategies have been considered, that at each step select the same control action in all states, and thus only depend on the number of execution steps of the system. Several problems for MDPs with word-strategies (also known as probabilistic automata) are undecidable [3,14,18,11]. In particular the limit-sure reachability problem, which is to decide whether a given state can be reached with probability arbitrarily close to one, is undecidable for probabilistic automata [14].

We establish the decidability and optimal complexity of deciding synchronizing properties for the symbolic outcome of MDPs under general strategies. Synchronizing properties require that the probability distributions tend to accumulate all the probability mass in a single state, or in a set of states. They generalize synchronizing properties of finite automata [20,10]. Formally for $0 \leq p \leq 1$, a sequence $\bar{X} = X_0 X_1 \dots$ of probability distributions $X_i : Q \rightarrow [0, 1]$ over state space Q of an MDP is *eventually p -synchronizing* if for some $i \geq 0$, the distribution X_i assigns probability at least p to some state. Analogously, it is *always p -synchronizing* if in all distributions X_i , there is a state with probability at least p . For $p = 1$, these definitions are the qualitative analogous for sequences of distributions of the traditional reachability and safety conditions [9]. In particular, an eventually 1-synchronizing sequence witnesses that there is a length ℓ such that all paths of length ℓ in the MDP reach a single state, which is thus reached synchronously no matter the probabilistic choices.

Viewing MDPs as one-player stochastic games, we consider the following traditional winning modes (see also Table 1): (i) *sure* winning, if there is a strategy that generates an {eventually, always} 1-synchronizing sequence; (ii) *almost-sure* winning, if there exists a strategy that generates a sequence that is, for all $\epsilon > 0$, {eventually, always} $(1-\epsilon)$ -synchronizing; (iii) *limit-sure* winning, if for all $\epsilon > 0$, there is a strategy that generates an {eventually, always} $(1-\epsilon)$ -synchronizing sequence.

We show that the three winning modes form a strict hierarchy for eventually synchronizing: there are limit-sure winning MDPs that are not almost-sure

Table 1. Winning modes and synchronizing objectives (where $\mathcal{M}_n^\alpha(T)$ denotes the probability that under strategy α , after n steps the MDP \mathcal{M} is in a state of T)

	Always	Eventually
Sure	$\exists\alpha \ \forall n \ \mathcal{M}_n^\alpha(T) = 1$	$\exists\alpha \ \exists n \ \mathcal{M}_n^\alpha(T) = 1$
Almost-sure	$\exists\alpha \ \inf_n \mathcal{M}_n^\alpha(T) = 1$	$\exists\alpha \ \sup_n \mathcal{M}_n^\alpha(T) = 1$
Limit-sure	$\sup_\alpha \inf_n \mathcal{M}_n^\alpha(T) = 1$	$\sup_\alpha \sup_n \mathcal{M}_n^\alpha(T) = 1$

winning, and there are almost-sure winning MDPs that are not sure winning. For always synchronizing, the three modes coincide.

For each winning mode, we consider the problem of deciding if a given initial distribution is winning. We establish the decidability and optimal complexity bounds for all winning modes. Under general strategies, the decision problems have much lower complexity than with word-strategies. We show that all decision problems are decidable, in polynomial time for always synchronizing, and PSPACE-complete for eventually synchronizing. This is also in contrast with almost-sure winning in the traditional semantics of MDPs, which is solvable in polynomial time for both safety and reachability objectives [8]. Our complexity results are shown in Table 2.

We complete the picture by providing optimal memory bounds for winning strategies. We show that for sure winning strategies, exponential memory is sufficient and may be necessary, and that in general infinite memory is necessary for almost-sure winning, and unbounded memory is necessary for limit-sure winning.

Some results in this paper rely on insights related to games and alternating automata that are of independent interest. First, the sure-winning problem for eventually synchronizing is equivalent to a two-player game with a synchronized reachability objective, where the goal for the first player is to ensure that a target state is reached after a number of steps that is independent of the strategy of the opponent (and thus this number can be fixed in advance by the first player). This condition is stronger than plain reachability, and while the winner in two-player reachability games can be decided in polynomial time, deciding the winner for synchronized reachability is PSPACE-complete. This result is obtained by turning the synchronized reachability game into a one-letter alternating automaton for which the emptiness problem (i.e., deciding if there exists a word accepted by the automaton) is PSPACE-complete [16,17]. Second, our PSPACE lower bound for the limit-sure winning problem in eventually synchronizing uses a PSPACE-completeness result that we establish for the *universal finiteness problem*, which is to decide, given a one-letter alternating automata, whether from every state the accepted language is finite.

A full version of this paper with all proofs is available [12].

2 Markov Decision Processes and Synchronization

A *probability distribution* over a finite set S is a function $d : S \rightarrow [0, 1]$ such that $\sum_{s \in S} d(s) = 1$. The *support* of d is the set $\text{Supp}(d) = \{s \in S \mid d(s) > 0\}$.

We denote by $\mathcal{D}(S)$ the set of all probability distributions over S . For $T \neq \emptyset$, the *uniform distribution* on T assigns probability $\frac{1}{|T|}$ to every state in T . Given $s \in S$, the *Dirac distribution* on s assigns probability 1 to s , and by a slight abuse of notation, we usually denote it simply by s .

2.1 Markov Decision Processes

A *Markov decision process* (MDP) $\mathcal{M} = \langle Q, \mathbf{A}, \delta \rangle$ consists of a finite set Q of states, a finite set \mathbf{A} of actions, and a probabilistic transition function $\delta : Q \times \mathbf{A} \rightarrow \mathcal{D}(Q)$. A state q is *absorbing* if $\delta(q, a)$ is the Dirac distribution on q for all actions $a \in \mathbf{A}$.

We describe the behavior of an MDP as a one-player stochastic game played for infinitely many rounds. Given an initial distribution $\mu_0 \in \mathcal{D}(Q)$, the game starts in the first round in state q with probability $\mu_0(q)$. In each round, the player chooses an action $a \in \mathbf{A}$, and if the game is in state q , the next round starts in the successor state q' with probability $\delta(q, a)(q')$.

Given $q \in Q$ and $a \in \mathbf{A}$, denote by $\text{post}(q, a)$ the set $\text{Supp}(\delta(q, a))$, and given $T \subseteq Q$ let $\text{Pre}(T) = \{q \in Q \mid \exists a \in \mathbf{A} : \text{post}(q, a) \subseteq T\}$ be the set of states from which the player has an action to ensure that the successor state is in T . For $k > 0$, let $\text{Pre}^k(T) = \text{Pre}(\text{Pre}^{k-1}(T))$ with $\text{Pre}^0(T) = T$.

A *path* in \mathcal{M} is an infinite sequence $\pi = q_0 a_0 q_1 a_1 \dots$ such that $q_{i+1} \in \text{post}(q_i, a_i)$ for all $i \geq 0$. A finite prefix $\rho = q_0 a_0 q_1 a_1 \dots q_n$ of a path (or simply a finite path) has length $|\rho| = n$ and last state $\text{Last}(\rho) = q_n$. We denote by $\text{Play}(\mathcal{M})$ and $\text{Pref}(\mathcal{M})$ the set of all paths and finite paths in \mathcal{M} respectively.

For the decision problems considered in this paper, only the support of the probability distributions in the transition function is relevant (i.e., the exact value of the positive probabilities does not matter); therefore, we can encode an MDP as an \mathbf{A} -labelled transition system (Q, R) with $R \subseteq Q \times \mathbf{A} \times Q$ such that $(q, a, q') \in R$ is a transition if $q' \in \text{post}(q, a)$.

Strategies. A *randomized strategy* for \mathcal{M} (or simply a strategy) is a function $\alpha : \text{Pref}(\mathcal{M}) \rightarrow \mathcal{D}(\mathbf{A})$ that, given a finite path ρ , returns a probability distribution $\alpha(\rho)$ over the action set, used to select a successor state q' of ρ with probability $\sum_{a \in \mathbf{A}} \alpha(\rho)(a) \cdot \delta(q, a)(q')$ where $q = \text{Last}(\rho)$.

A strategy α is *pure* if for all $\rho \in \text{Pref}(\mathcal{M})$, there exists an action $a \in \mathbf{A}$ such that $\alpha(\rho)(a) = 1$; and *memoryless* if $\alpha(\rho) = \alpha(\rho')$ for all ρ, ρ' such that $\text{Last}(\rho) = \text{Last}(\rho')$. We view pure strategies as functions $\alpha : \text{Pref}(\mathcal{M}) \rightarrow \mathbf{A}$, and memoryless strategies as functions $\alpha : Q \rightarrow \mathcal{D}(\mathbf{A})$. Finally, a strategy α uses *finite-memory* if it can be represented by a finite-state transducer $T = \langle \text{Mem}, m_0, \alpha_u, \alpha_n \rangle$ where Mem is a finite set of modes (the memory of the strategy), $m_0 \in \text{Mem}$ is the initial mode, $\alpha_u : \text{Mem} \times \mathbf{A} \times Q \rightarrow \text{Mem}$ is an update function, that given the current memory, last action and state updates the memory, and $\alpha_n : \text{Mem} \times Q \rightarrow \mathcal{D}(\mathbf{A})$ is a next-move function that selects the probability distribution $\alpha_n(m, q)$ over actions when the current mode is m and the current state of \mathcal{M} is q . For pure strategies, we assume that $\alpha_n : \text{Mem} \times Q \rightarrow \mathbf{A}$. The *memory size* of the strategy

is the number $|\text{Mem}|$ of modes. For a finite-memory strategy α , let $\mathcal{M}(\alpha)$ be the Markov chain obtained as the product of \mathcal{M} with the transducer defining α .

2.2 State Semantics

In the traditional state semantics, given an initial distribution $\mu_0 \in \mathcal{D}(Q)$ and a strategy α in an MDP \mathcal{M} , a *path-outcome* is a path $\pi = q_0 a_0 q_1 a_1 \dots$ in \mathcal{M} such that $q_0 \in \text{Supp}(\mu_0)$ and $a_i \in \text{Supp}(\alpha(q_0 a_0 \dots a_{i-1} q_i))$ for all $i \geq 0$. The probability of a finite prefix $\rho = q_0 a_0 q_1 a_1 \dots q_n$ of π is $\mu_0(q_0) \cdot \prod_{j=0}^{n-1} \alpha(q_0 a_0 \dots q_j)(a_j) \cdot \delta(q_j, a_j)(q_{j+1})$. We denote by $\text{Outcomes}(\mu_0, \alpha)$ the set of all path-outcomes from μ_0 under strategy α . An *event* $\Omega \subseteq \text{Play}(\mathcal{M})$ is a measurable set of paths, and given an initial distribution μ_0 and a strategy α , the probabilities $\text{Pr}^\alpha(\Omega)$ of events Ω are uniquely defined [19]. In particular, given a set $T \subseteq Q$ of target states, and $k \in \mathbb{N}$, we denote by $\square T = \{q_0 a_0 q_1 \dots \in \text{Play}(\mathcal{M}) \mid \forall i : q_i \in T\}$ the safety event of always staying in T , by $\diamond T = \{q_0 a_0 q_1 \dots \in \text{Play}(\mathcal{M}) \mid \exists i : q_i \in T\}$ the event of reaching T , and by $\diamond^k T = \{q_0 a_0 q_1 \dots \in \text{Play}(\mathcal{M}) \mid q_k \in T\}$ the event of reaching T after exactly k steps. Hence, $\text{Pr}^\alpha(\diamond T)$ is the probability to reach T under strategy α .

We consider the following classical winning modes. Given an initial distribution μ_0 and an event Ω , we say that \mathcal{M} is:

- *sure winning* if there exists a strategy α such that $\text{Outcomes}(\mu_0, \alpha) \subseteq \Omega$;
- *almost-sure winning* if there exists a strategy α such that $\text{Pr}^\alpha(\Omega) = 1$;
- *limit-sure winning* if $\sup_\alpha \text{Pr}^\alpha(\Omega) = 1$.

It is known for safety objectives $\square T$ in MDPs that the three winning modes coincide, and for reachability objectives $\diamond T$ that an MDP is almost-sure winning if and only if it is limit-sure winning. For both objectives, the set of initial distributions for which an MDP is sure (resp., almost-sure or limit-sure) winning can be computed in polynomial time [8].

2.3 Distribution Semantics

In contrast to the state semantics, we consider the outcome of an MDP \mathcal{M} under a fixed strategy as a sequence of probability distributions over states defined as follows [18]. Given an initial distribution $\mu_0 \in \mathcal{D}(Q)$ and a strategy α in \mathcal{M} , the *symbolic outcome* of \mathcal{M} from μ_0 is the sequence $(\mathcal{M}_n^\alpha)_{n \in \mathbb{N}}$ of probability distributions defined by $\mathcal{M}_k^\alpha(q) = \text{Pr}^\alpha(\diamond^k \{q\})$ for all $k \geq 0$ and $q \in Q$. Hence, \mathcal{M}_k^α is the probability distribution over states after k steps under strategy α . Note that $\mathcal{M}_0^\alpha = \mu_0$.

Informally, synchronizing objectives require that the probability of some state (or some group of states) tends to 1 in the sequence $(\mathcal{M}_n^\alpha)_{n \in \mathbb{N}}$. Given a set $T \subseteq Q$, consider the functions $\text{sum}_T : \mathcal{D}(Q) \rightarrow [0, 1]$ and $\text{max}_T : \mathcal{D}(Q) \rightarrow [0, 1]$ that compute $\text{sum}_T(X) = \sum_{q \in T} X(q)$ and $\text{max}_T(X) = \max_{q \in T} X(q)$. For $f \in \{\text{sum}_T, \text{max}_T\}$ and $p \in [0, 1]$, we say that a probability distribution X

Table 2. Computational complexity of the membership problem, and memory requirement for the strategies (for always synchronizing, the three modes coincide)

	Always		Eventually	
	Complexity	Memory requirement	Complexity	Memory requirement
Sure	PTIME-C	memoryless	PSPACE-C	exponential
Almost-sure			PSPACE-C	infinite
Limit-sure			PSPACE-C	unbounded

is p -synchronized according to f if $f(X) \geq p$, and that a sequence $\bar{X} = X_0 X_1 \dots$ of probability distributions is:

- (a) *always p -synchronizing* if X_i is p -synchronized for all $i \geq 0$;
- (b) *event (or eventually) p -synchronizing* if X_i is p -synchronized for some $i \geq 0$.

For $p = 1$, we view these definitions as the qualitative analogous for sequences of distributions of the traditional safety and reachability conditions for sequences of states [9]. Now, we define the following winning modes. Given an initial distribution μ_0 and a function $f \in \{sum_T, max_T\}$, we say that for the objective of $\{\text{always, eventually}\}$ synchronizing from μ_0 , \mathcal{M} is:

- *sure winning* if there exists a strategy α such that the symbolic outcome of α from μ_0 is $\{\text{always, eventually}\}$ 1-synchronizing according to f ;
- *almost-sure winning* if there exists a strategy α such that for all $\epsilon > 0$ the symbolic outcome of α from μ_0 is $\{\text{always, eventually}\}$ $(1 - \epsilon)$ -synchronizing according to f ;
- *limit-sure winning* if for all $\epsilon > 0$, there exists a strategy α such that the symbolic outcome of α from μ_0 is $\{\text{always, eventually}\}$ $(1 - \epsilon)$ -synchronizing according to f ;

We often use $X(T)$ instead of $sum_T(X)$, as in Table 1 where the definitions of the various winning modes and synchronizing objectives for $f = sum_T$ are summarized. In Section 2.4, we present an example to illustrate the definitions.

2.4 Decision Problems

For $f \in \{sum_T, max_T\}$ and $\lambda \in \{\text{always, event}\}$, the *winning region* $\langle\langle 1 \rangle\rangle_{sure}^\lambda(f)$ is the set of initial distributions such that \mathcal{M} is sure winning for λ -synchronizing (we assume that \mathcal{M} is clear from the context). We define analogously the winning regions $\langle\langle 1 \rangle\rangle_{almost}^\lambda(f)$ and $\langle\langle 1 \rangle\rangle_{limit}^\lambda(f)$. For a singleton $T = \{q\}$ we have $sum_T = max_T$, and we simply write $\langle\langle 1 \rangle\rangle_\mu^\lambda(q)$ (where $\mu \in \{\text{sure, almost, limit}\}$). We are interested in the algorithmic complexity of the *membership problem*, which is to decide, given a probability distribution μ_0 , whether $\mu_0 \in \langle\langle 1 \rangle\rangle_\mu^\lambda(f)$. As we show below, it is easy to establish the complexity of the membership problems for always synchronizing, while it is more tricky for eventually synchronizing. The complexity results are summarized in Table 2.

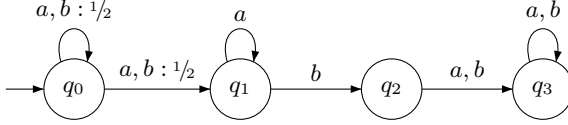


Fig. 1. An MDP \mathcal{M} such that $\langle\langle 1 \rangle\rangle_{\text{sure}}^{\text{event}}(q_1) \neq \langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{event}}(q_1)$ and $\langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{event}}(q_2) \neq \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(q_2)$

Always Synchronizing. We first remark that for always synchronizing, the three winning modes coincide.

Lemma 1. *Let T be a set of states. For all functions $f \in \{\max_T, \text{sum}_T\}$, we have $\langle\langle 1 \rangle\rangle_{\text{sure}}^{\text{always}}(f) = \langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{always}}(f) = \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{always}}(f)$.*

It follows from the proof of Lemma 1 that the winning region for always synchronizing according to sum_T coincides with the set of winning initial distributions for the safety objective $\square T$ in the traditional state semantics, which can be computed in polynomial time [7]. Moreover, always synchronizing according to \max_T is equivalent to the existence of an infinite path staying in T in the transition system $\langle Q, R \rangle$ of the MDP restricted to transitions $(q, a, q') \in R$ such that $\delta(q, a)(q') = 1$, which can also be decided in polynomial time. In both cases, pure memoryless strategies are sufficient.

Theorem 1. *The membership problem for always synchronizing can be solved in polynomial time, and pure memoryless strategies are sufficient.*

Eventually Synchronizing. For all functions $f \in \{\max_T, \text{sum}_T\}$, the following inclusions hold: $\langle\langle 1 \rangle\rangle_{\text{sure}}^{\text{event}}(f) \subseteq \langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{event}}(f) \subseteq \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(f)$ and we show that the inclusions are strict in general. Consider the MDP in Fig. 1 with initial state q_0 and target $T = \{q_1\}$. For all strategies, the probability in q_0 is always positive, implying that the MDP is not sure-winning in $\{q_1\}$. However, the MDP is almost-sure winning in $\{q_1\}$ using a strategy that always plays a . Now, consider target $T = \{q_2\}$. For all $\epsilon > 0$, we can have probability at least $1 - \epsilon$ in q_2 by playing a long enough, and then b . For a fixed strategy, this probability never tends to 1 since if the probability $p > 0$ in q_2 is positive at a certain step, then it remains bounded by $1 - p < 1$ for all next steps. Therefore, the MDP is not almost-sure winning in $\{q_2\}$, but it is limit-sure winning.

Lemma 2. *There exists an MDP \mathcal{M} and states q_1, q_2 such that:*

- (i) $\langle\langle 1 \rangle\rangle_{\text{sure}}^{\text{event}}(q_1) \subsetneq \langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{event}}(q_1)$, and
- (ii) $\langle\langle 1 \rangle\rangle_{\text{almost}}^{\text{event}}(q_2) \subsetneq \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(q_2)$.

The rest of this paper is devoted to the solution of the membership problem for eventually synchronizing. We make some preliminary remarks to show that it is sufficient to solve the membership problem according to $f = \text{sum}_T$ and for MDPs with a single initial state. Our results will also show that pure strategies are sufficient in all modes.

Remark. For eventually synchronizing and each winning mode, we show that the membership problem with function max_T is polynomial-time equivalent to the membership problem with function $sum_{T'}$ with a singleton T' . First, for $\mu \in \{\text{sure, almost, limit}\}$, we have $\langle\langle 1 \rangle\rangle_\mu^{event}(max_T) = \bigcup_{q \in T} \langle\langle 1 \rangle\rangle_\mu^{event}(q)$, showing that the membership problems for max are polynomial-time reducible to the corresponding membership problem for $sum_{T'}$ with singleton T' . The reverse reduction is as follows. Given an MDP \mathcal{M}' , a singleton $T' = \{q\}$ and an initial distribution μ'_0 , we can construct an MDP \mathcal{M} and initial distribution μ_0 such that $\mu'_0 \in \langle\langle 1 \rangle\rangle_\mu^{event}(q)$ iff $\mu_0 \in \langle\langle 1 \rangle\rangle_\mu^{event}(max_T)$ where $T = Q$ is the state space of \mathcal{M} . The idea is to construct \mathcal{M} and μ_0 as a copy of \mathcal{M}' and μ'_0 where all states except q are duplicated, and the initial and transition probabilities are evenly distributed between the copies. Therefore if the probability tends to 1 in some state, it has to be in q .

Remark. To solve the membership problems for eventually synchronizing with function sum_T , it is sufficient to provide an algorithm that decides membership of Dirac distributions (i.e., assuming MDPs have a single initial state), since to solve the problem for an MDP \mathcal{M} with initial distribution μ_0 , we can equivalently solve it for a copy of \mathcal{M} with a new initial state q_0 from which the successor distribution on all actions is μ_0 . Therefore, it is sufficient to consider initial Dirac distributions μ_0 .

3 One-Letter Alternating Automata

In this section, we consider *one-letter alternating automata* (1L-AFA) as they have a structure of alternating graph analogous to MDP (i.e., when ignoring the probabilities). We review classical decision problems for 1L-AFA, and establish the complexity of a new problem, the *universal finiteness problem* which is to decide if from every initial state the language of a given 1L-AFA is finite. These results of independent interest are useful to establish the PSPACE lower bounds for eventually synchronizing in MDPs.

One-Letter Alternating Automata. Let $B^+(Q)$ be the set of positive Boolean formulas over Q , i.e. Boolean formulas built from elements in Q using \wedge and \vee . A set $S \subseteq Q$ satisfies a formula $\varphi \in B^+(Q)$ (denoted $S \models \varphi$) if φ is satisfied when replacing in φ the elements in S by **true**, and the elements in $Q \setminus S$ by **false**.

A *one-letter alternating finite automaton* is a tuple $\mathcal{A} = \langle Q, \delta_{\mathcal{A}}, \mathcal{F} \rangle$ where Q is a finite set of states, $\delta_{\mathcal{A}} : Q \rightarrow B^+(Q)$ is the transition function, and $\mathcal{F} \subseteq Q$ is the set of accepting states. We assume that the formulas in transition function are in disjunctive normal form. Note that the alphabet of the automaton is omitted, as it has a single letter. In the language of a 1L-AFA, only the length of words is relevant. For all $n \geq 0$, define the set $Acc_{\mathcal{A}}(n, \mathcal{F}) \subseteq Q$ of states from which the word of length n is accepted by \mathcal{A} as follows:

- $\text{Acc}_{\mathcal{A}}(0, \mathcal{F}) = \mathcal{F}$;
- $\text{Acc}_{\mathcal{A}}(n, \mathcal{F}) = \{q \in Q \mid \text{Acc}_{\mathcal{A}}(n-1, \mathcal{F}) \models \delta(q)\}$ for all $n > 0$.

The set $\mathcal{L}(\mathcal{A}_q) = \{n \in \mathbb{N} \mid q \in \text{Acc}_{\mathcal{A}}(n, \mathcal{F})\}$ is the *language* accepted by \mathcal{A} from initial state q .

For fixed n , we view $\text{Acc}_{\mathcal{A}}(n, \cdot)$ as an operator on 2^Q that, given a set $\mathcal{F} \subseteq Q$ computes the set $\text{Acc}_{\mathcal{A}}(n, \mathcal{F})$. Note that $\text{Acc}_{\mathcal{A}}(n, \mathcal{F}) = \text{Acc}_{\mathcal{A}}(1, \text{Acc}_{\mathcal{A}}(n-1, \mathcal{F}))$ for all $n \geq 1$. Denote by $\text{Pre}_{\mathcal{A}}(\cdot)$ the operator $\text{Acc}_{\mathcal{A}}(1, \cdot)$. Then for all $n \geq 0$ the operator $\text{Acc}_{\mathcal{A}}(n, \cdot)$ coincides with $\text{Pre}_{\mathcal{A}}^n(\cdot)$, the n -th iterate of $\text{Pre}_{\mathcal{A}}(\cdot)$.

Decision Problems. We present the classical emptiness and finiteness problems for alternating automata, and we introduce a variant of the finiteness problem that will be useful for solving synchronizing problems for MDPs.

- The *emptiness problem* for 1L-AFA is to decide, given a 1L-AFA \mathcal{A} and an initial state q , whether $\mathcal{L}(\mathcal{A}_q) = \emptyset$. The emptiness problem can be solved by checking whether $q \in \text{Pre}_{\mathcal{A}}^n(\mathcal{F})$ for some $n \geq 0$. It is known that the emptiness problem is PSPACE-complete, even for transition functions in disjunctive normal form [16,17].
- The *finiteness problem* is to decide, given a 1L-AFA \mathcal{A} and an initial state q , whether $\mathcal{L}(\mathcal{A}_q)$ is finite. The sequence $\text{Pre}_{\mathcal{A}}^n(\mathcal{F})$ is ultimately periodic, and for all $n \geq 0$, there exists $n_0 \leq 2^{|Q|}$ such that $\text{Pre}_{\mathcal{A}}^{n_0}(\mathcal{F}) = \text{Pre}_{\mathcal{A}}^n(\mathcal{F})$. Therefore, the finiteness problem can be solved in (N)PSPACE by guessing $n, k \leq 2^{|Q|}$ such that $\text{Pre}_{\mathcal{A}}^{n+k}(\mathcal{F}) = \text{Pre}_{\mathcal{A}}^n(\mathcal{F})$ and $q \in \text{Pre}_{\mathcal{A}}^n(\mathcal{F})$. The finiteness problem is PSPACE-complete by a simple reduction from the emptiness problem: from an instance (\mathcal{A}, q) of the emptiness problem, construct (\mathcal{A}', q') where $q' = q$ and $\mathcal{A}' = \langle Q, \delta', \mathcal{F} \rangle$ is a copy of $\mathcal{A} = \langle Q, \delta, \mathcal{F} \rangle$ with a self-loop on q (formally, $\delta'(q) = q \vee \delta(q)$ and $\delta'(r) = \delta(r)$ for all $r \in Q \setminus \{q\}$). It is easy to see that $\mathcal{L}(\mathcal{A}_q) = \emptyset$ iff $\mathcal{L}(\mathcal{A}'_{q'})$ is finite.
- The *universal finiteness problem* is to decide, given a 1L-AFA \mathcal{A} , whether $\mathcal{L}(\mathcal{A}_q)$ is finite for all states q . This problem can be solved by checking whether $\text{Pre}_{\mathcal{A}}^n(\mathcal{F}) = \emptyset$ for some $n \leq 2^{|Q|}$, and thus it is in PSPACE. Note that if $\text{Pre}_{\mathcal{A}}^n(\mathcal{F}) = \emptyset$, then $\text{Pre}_{\mathcal{A}}^m(\mathcal{F}) = \emptyset$ for all $m \geq n$.

Given the PSPACE-hardness proofs of the emptiness and finiteness problems, it is not easy to see that the universal finiteness problem is PSPACE-hard.

Lemma 3. *The universal finiteness problem for 1L-AFA is PSPACE-hard.*

Relation with MDPs. The underlying structure of a Markov decision process $\mathcal{M} = \langle Q, \mathcal{A}, \delta \rangle$ is an alternating graph, where the successor q' of a state q is obtained by an existential choice of an action a and a universal choice of a state $q' \in \text{Supp}(\delta(q, a))$. Therefore, it is natural that some questions related to MDPs have a corresponding formulation in terms of alternating automata. We show that such connections exist between synchronizing problems for MDPs and language-theoretic questions for alternating automata, such as emptiness and universal finiteness. Given a 1L-AFA $\mathcal{A} = \langle Q, \delta_{\mathcal{A}}, \mathcal{F} \rangle$, assume without loss

of generality that the transition function $\delta_{\mathcal{A}}$ is such that $\delta_{\mathcal{A}}(q) = c_1 \vee \dots \vee c_m$ has the same number m of conjunctive clauses for all $q \in Q$. From \mathcal{A} , construct the MDP $\mathcal{M}_{\mathcal{A}} = \langle Q, \mathbf{A}, \delta_{\mathcal{M}} \rangle$ where $\mathbf{A} = \{a_1, \dots, a_m\}$ and $\delta_{\mathcal{M}}(q, a_k)$ is the uniform distribution over the states occurring in the k -th clause c_k in $\delta_{\mathcal{A}}(q)$, for all $q \in Q$ and $a_k \in \mathbf{A}$. Then, we have $\text{Acc}_{\mathcal{A}}(n, \mathcal{F}) = \text{Pre}_{\mathcal{M}}^n(\mathcal{F})$ for all $n \geq 0$. Similarly, from an MDP \mathcal{M} and a set T of states, we can construct a 1L-AFA $\mathcal{A} = \langle Q, \delta_{\mathcal{A}}, \mathcal{F} \rangle$ with $\mathcal{F} = T$ such that $\text{Acc}_{\mathcal{A}}(n, \mathcal{F}) = \text{Pre}_{\mathcal{M}}^n(T)$ for all $n \geq 0$ (let $\delta_{\mathcal{A}}(q) = \bigvee_{a \in \mathbf{A}} \bigwedge_{q' \in \text{post}(q, a)} q'$ for all $q \in Q$).

Several decision problems for 1L-AFA can be solved by computing the sequence $\text{Acc}_{\mathcal{A}}(n, \mathcal{F})$, and we show that some synchronizing problems for MDPs require the computation of the sequence $\text{Pre}_{\mathcal{M}}^n(\mathcal{F})$. Therefore, the above relation between 1L-AFA and MDPs establishes bridges that we use in Section 4 to transfer complexity results from 1L-AFA to MDPs.

4 Eventually Synchronization

In this section, we show the PSPACE-completeness of the membership problem for eventually synchronizing objectives and the three winning modes. By the remarks at the end of Section 2, we consider the membership problem with function sum and Dirac initial distributions (i.e., single initial state).

4.1 Sure Eventually Synchronization

Given a target set T , the membership problem for sure-winning eventually synchronizing objective in T can be solved by computing the sequence $\text{Pre}^n(T)$ of iterated predecessor. A state q_0 is sure-winning for eventually synchronizing in T if $q_0 \in \text{Pre}^n(T)$ for some $n \geq 0$.

Lemma 4. *Let \mathcal{M} be an MDP and T be a target set. For all states q_0 , we have $q_0 \in \langle \{1\} \rangle_{\text{sure}}^{\text{event}}(\text{sum}_T)$ if and only if there exists $n \geq 0$ such that $q_0 \in \text{Pre}_{\mathcal{M}}^n(T)$.*

By Lemma 4, the membership problem for sure eventually synchronizing is equivalent to the emptiness problem of 1L-AFA, and thus PSPACE-complete (even when T is a singleton). Moreover if $q_0 \in \text{Pre}_{\mathcal{M}}^n(T)$, a finite-memory strategy with n modes that at mode i in a state q plays an action a such that $\text{post}(q, a) \subseteq \text{Pre}^{i-1}(T)$ is sure winning for eventually synchronizing. There exists a family of MDPs \mathcal{M}_n ($n \in \mathbb{N}$) that are sure winning for eventually synchronization, and where the sure winning strategies require exponential memory [12]. Essentially, the structure of \mathcal{M}_n is an initial uniform probabilistic transition to n components H_1, \dots, H_n where H_i is a cycle of length p_i the i -th prime number, and sure eventually synchronization requires memory size $p_n^\# = \prod_{i=1}^n p_i$. The following theorem summarizes the results for sure eventually synchronizing.

Theorem 2. *For sure eventually synchronizing in MDPs:*

1. (Complexity). *The membership problem is PSPACE-complete.*
2. (Memory). *Exponential memory is necessary and sufficient for both pure and randomized strategies, and pure strategies are sufficient.*

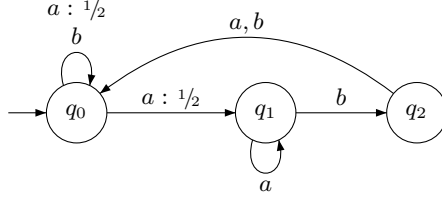


Fig. 2. An MDP where infinite memory is necessary for almost-sure eventually synchronizing strategies

4.2 Almost-Sure Eventually Synchronization

We show an example where infinite memory is necessary to win for almost-sure eventually synchronizing. Consider the MDP in Fig. 2 with initial state q_0 . We construct a strategy that is almost-sure eventually synchronizing in $\{q_2\}$, showing that $q_0 \in \langle\langle 1 \rangle\rangle_{almost}^{event}(q_2)$. First, observe that for all $\epsilon > 0$ we can have probability at least $1 - \epsilon$ in q_2 after finitely many steps: playing n times a and then b leads to probability $1 - \frac{1}{2^n}$ in q_2 . Thus the MDP is limit-sure eventually synchronizing in q_2 . Moreover the remaining probability mass is in q_0 . It turns out that from any (initial) distribution with support $\{q_0, q_2\}$, the MDP is again limit-sure eventually synchronizing in q_2 (and with support in $\{q_0, q_2\}$). Therefore we can take a smaller value of ϵ and play a strategy to have probability at least $1 - \epsilon$ in q_2 , and repeat this for $\epsilon \rightarrow 0$. This strategy ensures almost-sure eventually synchronizing in q_2 . The next result shows that infinite memory is necessary for almost-sure winning in this example.

Lemma 5. *There exists an almost-sure eventually synchronizing MDP for which all almost-sure eventually synchronizing strategies require infinite memory.*

It turns out that in general, almost-sure eventually synchronizing strategies can be constructed from a family of limit-sure eventually synchronizing strategies if we can also ensure that the probability mass remains in the winning region (as in the MDP in Fig. 2). We present a characterization of the winning region for almost-sure winning based on an extension of the limit-sure eventually synchronizing objective *with exact support*. This objective requires to ensure probability arbitrarily close to 1 in the target set T , and moreover that after the same number of steps the support of the probability distribution is contained in a given set U . Formally, given an MDP \mathcal{M} , let $\langle\langle 1 \rangle\rangle_{limit}^{event}(sum_T, U)$ for $T \subseteq U$ be the set of all initial distributions such that for all $\epsilon > 0$ there exists a strategy α and $n \in \mathbb{N}$ such that $\mathcal{M}_n^\alpha(T) \geq 1 - \epsilon$ and $\mathcal{M}_n^\alpha(U) = 1$. We say that α is limit-sure eventually synchronizing in T with support in U .

We will present an algorithmic solution to limit-sure eventually synchronizing objectives with exact support in Section 4.3. Our characterization of the winning region for almost-sure winning is as follows. There must exist a support U such that (i) the MDP is sure winning for eventually synchronizing in target U ,

and (ii) from distributions with support in U , it is possible to get probability arbitrarily close to 1 in T , and support back in U . In the example of Fig. 2 with $T = \{q_2\}$, we can take $U = \{q_0, q_2\}$.

Lemma 6. *Let \mathcal{M} be an MDP and T be a target set. For all states q_0 , we have $q_0 \in \langle\langle 1 \rangle\rangle_{almost}^{event}(sum_T)$ if and only if there exists a set U such that:*

- $q_0 \in \langle\langle 1 \rangle\rangle_{sure}^{event}(sum_U)$, and
- $d_U \in \langle\langle 1 \rangle\rangle_{limit}^{event}(sum_T, U)$ where d_U is the uniform distribution over U .

As we show in Section 4.3 that the membership problem for limit-sure eventually synchronizing with exact support can be solved in PSPACE, it follows from the characterization in Lemma 6 that the membership problem for almost-sure eventually synchronizing is in PSPACE, using the following (N)PSPACE algorithm: guess the set U , and check that $q_0 \in \langle\langle 1 \rangle\rangle_{sure}^{event}(sum_U)$, and that $d_U \in \langle\langle 1 \rangle\rangle_{limit}^{event}(sum_T, U)$ where d_U is the uniform distribution over U (both can be done in PSPACE by Theorem 2 and Theorem 4). We present a matching lower bound using a reduction from the membership problem for sure eventually synchronization [12], which is PSPACE-complete by Theorem 2.

Lemma 7. *The membership problem for $\langle\langle 1 \rangle\rangle_{almost}^{event}(sum_T)$ is PSPACE-hard even if T is a singleton.*

The results of this section are summarized as follows.

Theorem 3. *For almost-sure eventually synchronizing in MDPs:*

1. (Complexity). *The membership problem is PSPACE-complete.*
2. (Memory). *Infinite memory is necessary in general for both pure and randomized strategies, and pure strategies are sufficient.*

4.3 Limit-Sure Eventually Synchronization

In this section, we present the algorithmic solution for limit-sure eventually synchronizing with exact support. Note that the limit-sure eventually synchronizing objective is a special case where the support is the state space of the MDP. Consider the MDP in Fig. 1 which is limit-sure eventually synchronizing in $\{q_2\}$, as shown in Lemma 2. For $i = 0, 1, \dots$, the sequence $\text{Pre}^i(T)$ of predecessors of $T = \{q_2\}$ is ultimately periodic: $\text{Pre}^0(T) = \{q_2\}$, and $\text{Pre}^i(T) = \{q_1\}$ for all $i \geq 1$. Given $\epsilon > 0$, a strategy to get probability $1 - \epsilon$ in q_2 first accumulates probability mass in the *periodic* subsequence of predecessors (here $\{q_1\}$), and when the probability mass is greater than $1 - \epsilon$ in q_1 , the strategy injects the probability mass in q_2 (through the aperiodic prefix of the sequence of predecessors). This is the typical shape of a limit-sure eventually synchronizing strategy. Note that in this scenario, the MDP is also limit-sure eventually synchronizing in every set $\text{Pre}^i(T)$ of the sequence of predecessors. A special case is when it is possible to get probability 1 in the sequence of predecessors after finitely

many steps. In this case, the probability mass injected in T is 1 and the MDP is even sure-winning. The algorithm for deciding limit-sure eventually synchronization relies on the above characterization, generalized in Lemma 8 to limit-sure eventually synchronizing with exact support, saying that limit-sure eventually synchronizing in T with support in U is equivalent to either limit-sure eventually synchronizing in $\text{Pre}^k(T)$ with support in $\text{Pre}^k(U)$ (for arbitrary k), or sure eventually synchronizing in T (and therefore also in U).

Lemma 8. *For all $T \subseteq U$ and all $k \geq 0$, we have $\langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(\text{sum}_T, U) = \langle\langle 1 \rangle\rangle_{\text{sure}}^{\text{event}}(\text{sum}_T) \cup \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(\text{sum}_R, Z)$ where $R = \text{Pre}^k(T)$ and $Z = \text{Pre}^k(U)$.*

Thanks to Lemma 8, since sure-winning is already solved in Section 4.1, it suffices to solve the limit-sure eventually synchronizing problem for target $R = \text{Pre}^k(T)$ and support $Z = \text{Pre}^k(U)$ with arbitrary k , instead of T and U . We can choose k such that both $\text{Pre}^k(T)$ and $\text{Pre}^k(U)$ lie in the periodic part of the sequence of pairs of predecessors $(\text{Pre}^i(T), \text{Pre}^i(U))$. We can assume that $k \leq 3^{|\mathcal{Q}|}$ since $\text{Pre}^i(T) \subseteq \text{Pre}^i(U) \subseteq \mathcal{Q}$ for all $i \geq 0$. For such value of k the limit-sure problem is conceptually simpler: once some probability is injected in $R = \text{Pre}^k(T)$, it can loop through the sequence of predecessors and visit R infinitely often (every r steps, where $r \leq 3^{|\mathcal{Q}|}$ is the period of the sequence of pairs of predecessors). It follows that if a strategy ensures with probability 1 that the set R can be reached by finite paths whose lengths are congruent modulo r , then the whole probability mass can indeed synchronously accumulate in R in the limit. Therefore, limit-sure eventually synchronizing in R reduces to standard limit-sure reachability with target set R and the additional requirement that the numbers of steps at which the target set is reached be congruent modulo r . In the case of limit-sure eventually synchronizing with support in Z , we also need to ensure that no mass of probability leaves the sequence $\text{Pre}^i(Z)$. In a state $q \in \text{Pre}^i(Z)$, we say that an action $a \in \mathbf{A}$ is Z -safe at position i if¹ $\text{post}(q, a) \subseteq \text{Pre}^{i-1}(Z)$. In states $q \notin \text{Pre}^i(Z)$ there is no Z -safe action at position i .

To encode the above requirements, we construct an MDP $\mathcal{M}_Z \times [r]$ that allows only Z -safe actions to be played (and then mimics the original MDP), and tracks the position (modulo r) in the sequence of predecessors, thus simply decrementing the position on each transition since all successors of a state $q \in \text{Pre}^i(Z)$ on a safe action are in $\text{Pre}^{i-1}(Z)$. Formally, if $\mathcal{M} = \langle Q, \mathbf{A}, \delta \rangle$ then $\mathcal{M}_Z \times [r] = \langle Q', \mathbf{A}, \delta' \rangle$ where:

- $Q' = Q \times \{r-1, \dots, 1, 0\} \cup \{\text{sink}\}$; intuitively, we expect that $q \in \text{Pre}^i(Z)$ in the reachable states $\langle q, i \rangle$ consisting of a state q of \mathcal{M} and a *position* i in the predecessor sequence;
- δ' is defined as follows (assuming an arithmetic modulo r on positions) for all $\langle q, i \rangle \in Q'$ and $a \in \mathbf{A}$: if a is a Z -safe action in q at position i , then $\delta'(\langle q, i \rangle, a)(\langle q', i-1 \rangle) = \delta(q, a)(q')$, otherwise $\delta'(\langle q, i \rangle, a)(\text{sink}) = 1$ (and sink is absorbing).

¹ Since $\text{Pre}^r(Z) = Z$ and $\text{Pre}^r(R) = R$, we assume a modular arithmetic for exponents of Pre, that is $\text{Pre}^x(\cdot)$ is defined as $\text{Pre}^{x \bmod r}(\cdot)$. For example $\text{Pre}^{-1}(Z)$ is $\text{Pre}^{r-1}(Z)$.

Note that the size of the MDP $\mathcal{M}_Z \times [r]$ is exponential in the size of \mathcal{M} (since r is at most $3^{|\mathcal{Q}|}$).

Lemma 9. *Let \mathcal{M} be an MDP and $R \subseteq Z$ be two sets of states such that $\text{Pre}^r(R) = R$ and $\text{Pre}^r(Z) = Z$ where $r > 0$. Then a state q_0 is limit-sure eventually synchronizing in R with support in Z ($q_0 \in \langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(\text{sum}_R, Z)$) if and only if there exists $0 \leq t < r$ such that $\langle q_0, t \rangle$ is limit-sure winning for the reachability objective $\diamond(R \times \{0\})$ in the MDP $\mathcal{M}_Z \times [r]$.*

Since deciding limit-sure reachability is PTIME-complete, it follows from Lemma 9 that limit-sure synchronization (with exact support) can be decided in EXPTIME. We can show that the problem can be solved in PSPACE by exploiting the special structure of the exponential MDP in Lemma 9.

Lemma 10. *The membership problem for limit-sure eventually synchronization with exact support is in PSPACE.*

To establish the PSPACE-hardness for limit-sure eventually synchronizing in MDPs, we use a reduction from the universal finiteness problem for 1L-AFAs.

Lemma 11. *The membership problem for $\langle\langle 1 \rangle\rangle_{\text{limit}}^{\text{event}}(\text{sum}_T)$ is PSPACE-hard even if T is a singleton.*

The example in Fig. 2 can be used to show that the memory needed by a family of strategies to win limit-sure eventually synchronizing objective (in target $T = \{q_2\}$) is unbounded.

Theorem 4. *For limit-sure eventually synchronizing (with or without exact support) in MDPs:*

1. (Complexity). *The membership problem is PSPACE-complete.*
2. (Memory). *Unbounded memory is required for both pure and randomized strategies, and pure strategies are sufficient.*

Acknowledgment. We are grateful to Winfried Just and German A. Enciso for helpful discussions on Boolean networks and for the gadget in the proof of Lemma 3.

References

1. Agrawal, M., Akshay, S., Genest, B., Thiagarajan, P.S.: Approximate verification of the symbolic dynamics of Markov chains. In: LICS, pp. 55–64. IEEE (2012)
2. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. J. Algorithm 11(3), 441–461 (1990)
3. 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)

4. Baier, C., Bertrand, N., Schnoebelen, P.: On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 347–361. Springer, Heidelberg (2006)
5. Baldoni, R., Bonnet, F., Milani, A., Raynal, M.: On the solvability of anonymous partial grids exploration by mobile robots. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 428–445. Springer, Heidelberg (2008)
6. Chadha, R., Korthikanti, V.A., Viswanathan, M., Agha, G., Kwon, Y.: Model checking MDPs with a unique compact invariant set of distributions. In: Proc. of QEST, pp. 121–130. IEEE Computer Society (2011)
7. Chatterjee, K., Henzinger, T.A.: A survey of stochastic ω -regular games. *J. Comput. Syst. Sci.* 78(2), 394–413 (2012)
8. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
9. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. *Theor. Comput. Sci.* 386(3), 188–217 (2007)
10. Doyen, L., Massart, T., Shirmohammadi, M.: Infinite synchronizing words for probabilistic automata. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 278–289. Springer, Heidelberg (2011)
11. Doyen, L., Massart, T., Shirmohammadi, M.: Infinite synchronizing words for probabilistic automata (Erratum). CoRR, abs/1206.0995 (2012)
12. Doyen, L., Massart, T., Shirmohammadi, M.: Limit synchronization in Markov decision processes. CoRR, abs/1310.2935 (2013)
13. Fokkink, W., Pang, J.: Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM. *Journal of Universal Computer Science* 12(8), 981–1006 (2006)
14. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: Decidable and undecidable problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, part II. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010)
15. Henzinger, T.A., Mateescu, M., Wolf, V.: Sliding window abstraction for infinite Markov chains. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 337–352. Springer, Heidelberg (2009)
16. Holzer, M.: On emptiness and counting for alternating finite automata. In: Developments in Language Theory, pp. 88–97 (1995)
17. Jancar, P., Sawa, Z.: A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.* 104(5), 164–167 (2007)
18. Korthikanti, V.A., Viswanathan, M., Agha, G., Kwon, Y.: Reasoning about MDPs as transformers of probability distributions. In: Proc. of QEST, pp. 199–208. IEEE Computer Society (2010)
19. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. of FOCS, pp. 327–338. IEEE Computer Society (1985)
20. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Maximal Cost-Bounded Reachability Probability on Continuous-Time Markov Decision Processes*

Hongfei Fu**

Lehrstuhl für Informatik II, RWTH Aachen University, Germany

Abstract. In this paper, we consider multi-dimensional maximal cost-bounded reachability probability over continuous-time Markov decision processes (CTMDPs). Our major contributions are as follows. Firstly, we derive an integral characterization which states that the maximal cost-bounded reachability probability function is the least fixed-point of a system of integral equations. Secondly, we prove that the maximal cost-bounded reachability probability can be attained by a measurable deterministic cost-positional scheduler. Thirdly, we provide a numerical approximation algorithm for maximal cost-bounded reachability probability. We present these results under the setting of both early and late schedulers. Besides, we correct a fundamental proof error in the PhD Thesis by Martin Neuhäüßer on maximal time-bounded reachability probability by completely new proofs for the more general case of multi-dimensional maximal cost-bounded reachability probability.

1 Introduction

The class of continuous-time Markov decision processes (CTMDPs) (or controlled Markov chains) [13,12] is a stochastic model that incorporates both features from continuous-time Markov chains (CTMCs) [6] and discrete-time Markov decision processes (MDPs) [13]. A CTMDP extends a CTMC in the sense that it allows non-deterministic choices, and it extends an MDP in the sense that it incorporates negative exponential time-delays. Due to its modelling capability of real-time probabilistic behaviour and non-determinism, CTMDPs are widely used in dependability analysis and performance evaluation [2].

In a CTMDP, non-determinism is resolved by *schedulers* [16]. Informally, a scheduler determines the non-deterministic choices depending on the finite trajectory of the CTMDP so far and possibly the sojourn time of the current state. A scheduler is assumed to be *measurable* so that it induces a well-defined probability space over the infinite trajectories of the underlying CTMDP. Measurable schedulers are further divided into categories of *early schedulers* and *late schedulers* [10,16]. A scheduler that makes the choice solely by the trajectory so far is called an early scheduler, while a scheduler that utilizes both the trajectory and

* Partially funded by the EU FP7 projects CARP and SENSATION. Full version available at [7].

** Supported by a CSC scholarship.

the sojourn time (at the current state) is called a late scheduler. With schedulers, one can reason about quantitative information such as the maximal/minimal probability/expectation of certain property.

In this paper, we focus on the problem to compute *max/min resource-bounded reachability probability* on a CTMDP. Typical resource types considered here are time and cost, where a time bound can be deemed as a special cost bound with unit-cost 1. In general, the task is to compute or approximate the optimal (max/min) reachability probability to certain target states within a given resource bound (e.g., a time bound).

Optimal time-bounded reachability probability over CTMDPs has been widely studied in recent years. Neuhäuser *et al.* [11] proved that the maximal time-bounded reachability probability function is the least fixed point of a system of integral equations. Rabe and Schewe [14] showed that the max/min time-bounded reachability probability can be attained by a deterministic piecewise-constant time-positional scheduler. Efficient approximation algorithms are also developed by, e.g., Neuhäuser *et al.* [11], Brázdil *et al.* [3], Hatefi *et al.* [8] and Rabe *et al.* [5].

As to optimal cost-bounded reachability probability, much less is known. To the best of the author's knowledge, the only prominent result is by Baier *et al.* [1], which establishes a certain duality property between time and cost bound. Their result is restrictive in the sense that (i) it assumes that the CTMDP have everywhere positive unit-cost values, (ii) it only takes into account one-dimensional cost-bound aside the time-bound, and (iii) it does not really provide an approximation algorithm when both time- and cost-bounds are present.

Besides resource-bounded reachability probability, we would like to mention another research field on CTMDPs with costs (or dually, rewards), which is (discounted) accumulated reward over finite/infinite horizon (cf. [4,12], just to mention a little).

Our Contribution. We consider multi-dimensional maximal cost-bounded reachability probability (*abbr.* MMCRP) over CTMDPs, under the setting of both early and late schedulers, for which the unit-cost is constant. We first prove that the MMCRP function is the least fixed-point of a system of integral equations. Then we prove that deterministic cost-positional measurable schedulers suffice to achieve the MMCRP value. Finally, we describe a numerical algorithm which approximates the MMCRP value with an error bound. The approximation algorithm relies on a differential characterization which in turn is derived from the least fixed-point characterization. The complexity of the approximation algorithm is polynomial in the size of the CTMDP and the reciprocal of the error bound, and exponential in the dimension of cost vectors.

Besides, we point out a fundamental proof error in the treatment of maximal time-bounded reachability probability on continuous-time Markov decision processes [9,11]. We fix this error in the more general setting of maximal cost-bounded reachability probability by completely new proofs.

Structure of the Paper. Section 2 introduces some preliminaries of CTMDPs. Section 3 illustrates the definition of schedulers and the probability spaces they induce. In Section 4, we define the notion of maximal cost-bounded reachability probability and derive the least-fixed-point characterization, while we also point out the proof error in [9,11]. In Section 5, we prove that the maximal cost-bounded reachability probability can be reached by a measurable deterministic cost-positional scheduler. In Section 6, we derive a differential characterization which is crucial to our approximation algorithm. In Section 7, we present our approximation algorithm. Finally, Section 8 concludes the paper.

Due to page limit, we omit all the proofs and only present the results for late schedulers. The details can be found at [7].

2 Continuous-Time Markov Decision Processes

In the whole paper, we will use the following convention for notations. We will denote by $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers and by \mathbb{N}_0 the set of non-negative integers. We use x, d, t, τ to range over real numbers, m, n, i, j to range over \mathbb{N}_0 , and bold-face letters $\mathbf{x}, \mathbf{c}, \mathbf{d}$ to range over (column) real vectors. Given $\mathbf{c} \in \mathbb{R}^k$ ($k \in \mathbb{N}$), we denote by \mathbf{c}_i ($1 \leq i \leq k$) the i -th coordinate of \mathbf{c} . We denote by $\mathbf{0}$ the real vector whose coordinates are all equal to 0 (with the implicitly known dimension). We extend $\{\leq, <, \geq, >\}$ to real vectors and functions in a pointwise fashion: for two real vectors \mathbf{c}, \mathbf{d} , $\mathbf{c} \leq \mathbf{d}$ iff $\mathbf{c}_i \leq \mathbf{d}_i$ for all i ; for two real-valued functions g, h , $g \leq h$ iff $g(y) \leq h(y)$ for all y . Given a set Y , we let $\mathbf{1}_Y$ be the indicator function of Y , i.e., $\mathbf{1}_Y(y) = 1$ if $y \in Y$ and $\mathbf{1}_Y(y) = 0$ for $y \in X - Y$, where $X \supseteq Y$ is an implicitly known set. Given a positive real number $\lambda > 0$, let $f_\lambda(t) := \lambda \cdot e^{-\lambda \cdot t}$ ($t \geq 0$) be the probability density function of the negative exponential distribution with rate λ . Besides, we will use g, h to range over general functions.

2.1 The Model

Definition 1. A Continuous-Time Markov Decision Process (CTMDP) is a tuple $(L, \text{Act}, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ where

- L is a finite set of states (or locations);
- Act is a finite set of actions;
- $\mathbf{R} : L \times \text{Act} \times L \rightarrow \mathbb{R}_{\geq 0}$ is the rate matrix;
- $\{\mathbf{w}_i : L \times \text{Act} \rightarrow \mathbb{R}_{\geq 0}\}_{1 \leq i \leq k}$ is the family of k unit-cost functions ($k \in \mathbb{N}$);

An action $a \in \text{Act}$ is enabled at state $s \in L$ if $\mathbf{E}(s, a) := \sum_{u \in L} \mathbf{R}(s, a, u)$ is non-zero. The set of enabled actions at $s \in L$ is denoted by $\text{En}(s)$. We assume that for each state $s \in L$, $\text{En}(s) \neq \emptyset$.

Let $(L, \text{Act}, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ be a CTMDP. For each $s, s' \in L$ and $a \in \text{En}(s)$, we define $\mathbf{P}(s, a, s') := \frac{\mathbf{R}(s, a, s')}{\mathbf{E}(s, a)}$ to be the discrete transition probability from s to s' via a . We denote by $\mathbf{w}(s, a)$ the real vector $\{\mathbf{w}_i(s, a)\}_{1 \leq i \leq k}$ for each

$(s, a) \in L \times Act$. Given $s \in L$ and $a \in Act$, we denote by $\mathcal{D}[s]$ the Dirac distribution (over L) at s (i.e., $\mathcal{D}s = 1$ and $\mathcal{D}[s](s') = 0$ for $s' \in L - \{s\}$) and by $\mathcal{D}[a]$ the Dirac distribution (over Act) at a . Moreover, we define (with $\min \emptyset := 1$):

- $\mathbf{w}_{\min} := \min\{\mathbf{w}_i(s, a) \mid 1 \leq i \leq k, s \in L, a \in \text{En}(s), \mathbf{w}_i(s, a) > 0\}$;
- $\mathbf{w}_{\max} := \max\{\mathbf{w}_i(s, a) \mid 1 \leq i \leq k, s \in L, a \in \text{En}(s)\}$;
- $\mathbf{E}_{\max} := \max\{\mathbf{E}(s, a) \mid s \in L, a \in \text{En}(s)\}$;

We will use s, s' (resp. a, b) to range over states (resp. actions) of a CTMDP.

Often, a CTMDP is accompanied with an initial distribution which specifies the initial stochastic environment (for the CTMDP).

Definition 2. Let $\mathcal{M} = (L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ be a CTMDP. An initial distribution (for \mathcal{M}) is a function $\alpha : L \rightarrow [0, 1]$ such that $\sum_{s \in L} \alpha(s) = 1$.

Intuitively, the execution of a CTMDP $(L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ with a scheduler is as follows. At the beginning, an initial state s is chosen (as the current state) w.r.t the initial distribution α . Then the scheduler chooses an action a enabled at s either before or after a time-delay occurs at the state s . After the time-delay, the current state is switched to an arbitrary state $s' \in L$ with probability $\mathbf{P}(s, a, s')$, and so forth. Besides, each cost function \mathbf{w}_i assigns to each state-action pair (s, a) the i -th constant unit-cost $\mathbf{w}_i(s, a)$ (per time unit) when the CTMDP dwells at state s . Basically, the scheduler makes the decision of the action to be chosen when entering a new state, and has two distinct objectives: either to maximize a certain property or (in contrast) to minimize a certain property. In this paper, we will focus on the objective to maximize a cost-bounded reachability probability for a certain target set of states.

In this paper, we focus on an important subclass of CTMDPs, called *locally-uniform* CTMDPs (cf. [10]).

Definition 3. A CTMDP $(L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ is locally-uniform if $\mathbf{E}(s, a) = \mathbf{E}(s, b)$ and $\mathbf{w}_i(s, a) = \mathbf{w}_i(s, b)$ for all $1 \leq i \leq k, s \in L$ and $a, b \in \text{En}(s)$.

Intuitively, a locally uniform CTMDP has the property that the time-delay and the cost is independent of the action chosen at each state. For locally-uniform CTMDPs, we simply use $\mathbf{E}(s)$ to denote $\mathbf{E}(s, a)$ ($a \in \text{En}(s)$ is arbitrary), and $\mathbf{w}(s), \mathbf{w}_i(s)$ for $\mathbf{w}(s, a), \mathbf{w}_i(s, a)$ likewise.

2.2 Paths and Histories

In this part, we introduce the notion of paths and histories. Intuitively, paths reflect infinite executions of a CTMDP, whereas histories reflect finite executions of a CTMDP. Below we fix a CTMDP $\mathcal{M} = (L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$.

Definition 4. A (*n infinite*) path π is an infinite sequence

$$\pi = \left\langle s_0 \xrightarrow{a_0, t_0} s_1 \xrightarrow{a_1, t_1} s_2 \dots \right\rangle$$

such that $s_i \in L$, $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \text{Act}$ for all $i \geq 0$; We denote s_i, t_i and a_i by $\pi[i], \pi\langle i \rangle$ and $\pi(i)$, respectively. A (finite) history ξ is a finite sequence

$$\xi = \left\langle s_0 \xrightarrow{a_0, t_0} s_1 \xrightarrow{a_1, t_1} s_2 \dots s_m \right\rangle \quad (m \geq 0)$$

such that $s_i \in L$, $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \text{Act}$ for all $0 \leq i \leq m-1$, and $s_m \in L$; We denote s_i, t_i, a_i and m by $\xi[i], \xi\langle i \rangle, \xi(i)$ and $|\xi|$, respectively. Moreover, we define $\xi \downarrow := \xi[|\xi|]$ to be the last state of the history ξ .

Below we introduce more notations on paths and histories. We denote the set of paths and histories (of \mathcal{M}) by $\text{Paths}(\mathcal{M})$ and $\text{Hists}(\mathcal{M})$, respectively. We define $\text{Hists}^n(\mathcal{M}) := \{\xi \in \text{Hists}(\mathcal{M}) \mid |\xi| = n\}$ to be the set of all histories with length n ($n \geq 0$). For each $n \in \mathbb{N}_0$ and $\pi \in \text{Paths}(\mathcal{M})$, we define the history $\pi[0..n]$ to be the finite prefix of π up to n ; Formally,

$$\pi[0..n] := \left\langle \pi[0] \xrightarrow{\pi(0), \pi\langle 0 \rangle} \dots \pi[n] \right\rangle .$$

Given $\pi \in \text{Paths}(\mathcal{M})$ and $(s, a, t) \in L \times \text{Act} \times \mathbb{R}_{\geq 0}$, we denote by $s \xrightarrow{a, t} \pi$ the path obtained by “putting” the prefix “ $s \xrightarrow{a, t}$ ” before π ; Formally,

$$s \xrightarrow{a, t} \pi := \left\langle s \xrightarrow{a, t} \pi[0] \xrightarrow{\pi(0), \pi\langle 0 \rangle} \pi[1] \xrightarrow{\pi(1), \pi\langle 1 \rangle} \dots \right\rangle .$$

Analogously, we define $s \xrightarrow{a, t} \xi$ (for $\xi \in \text{Hists}(\mathcal{M})$) to be the history obtained by “putting” “ $s \xrightarrow{a, t}$ ” before the history ξ .

Intuitively, a path π reflects a whole execution (trajectory) of the CTMDP where $\pi[i]$ is the current state at the i -th stage, $\pi(i)$ is the action chosen at $\pi[i]$ and $\pi\langle i \rangle$ is the dwell-time (time-delay) on $\pi[i]$. On the other hand, a history ξ is a finite prefix of a path which reflects the execution up to $|\xi|$ stages.

Below we extend sets of histories to sets of paths in a cylindrical fashion.

Definition 5. Suppose $n \in \mathbb{N}_0$ and $\Xi \subseteq \text{Hists}^n(\mathcal{M})$. The cylinder extension of Ξ , denoted $\text{Cyl}(\Xi)$, is defined as follows:

$$\text{Cyl}(\Xi) := \{\pi \in \text{Paths}(\mathcal{M}) \mid \pi[0..n] \in \Xi\} .$$

In this paper, we concern costs on paths and histories. The cost is assigned linearly w.r.t the unit-cost and the time spent in a state. The following definition presents the details.

Definition 6. Given a path $\pi \in \text{Paths}(\mathcal{M})$ and a set $G \subseteq L$ of states, we denote by $\mathbf{C}_j(\pi, G)$ ($1 \leq j \leq k$) the j -th accumulated cost along π until G is reached; Formally, if $\pi[m] \in G$ for some $m \geq 0$ then

$$\mathbf{C}_j(\pi, G) := \sum_{i=0}^m \mathbf{w}_j(\pi[i], \pi(i)) \cdot \pi(i)$$

where $n \in \mathbb{N}_0 \cup \{-1\}$ is the smallest integer such that $\pi[n+1] \in G$; otherwise $\mathbf{C}_j(\pi, G) := +\infty$. Given a history $\xi \in \text{Hists}(\mathcal{M})$, we denote by $\mathbf{C}_j(\xi)$ ($1 \leq j \leq k$) the accumulated cost of ξ w.r.t the j -th unit-cost function; Formally,

$$\mathbf{C}_j(\xi) := \sum_{i=0}^{|\xi|-1} \mathbf{w}_j(\xi[i], \xi(i)) \cdot \xi(i).$$

We denote by $\mathbf{C}(\pi, G)$ and $\mathbf{C}(\xi)$ the vectors $\{\mathbf{C}_j(\pi, G)\}_{1 \leq j \leq k}$ and $\{\mathbf{C}_j(\xi)\}_{1 \leq j \leq k}$.

2.3 Measurable Spaces on Paths and Histories

In the following, we define the measurable spaces for paths and histories, following the definitions of [16,10]. Below we fix a CTMDP $\mathcal{M} = (L, \text{Act}, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$. Firstly, we introduce the notion of combined actions and its measurable space.

Definition 7. A combined action is a tuple (a, t, s) where $a \in \text{Act}$, $t \in \mathbb{R}_{\geq 0}$ and $s \in L$. The measurable space $(\Gamma_{\mathcal{M}}, \mathcal{U}_{\mathcal{M}})$ over combined actions is defined as follows:

- $\Gamma_{\mathcal{M}} := \text{Act} \times \mathbb{R}_{\geq 0} \times L$ is the set of combined actions;
- $\mathcal{U}_{\mathcal{M}} := 2^{\text{Act}} \otimes \mathcal{B}(\mathbb{R}_{\geq 0}) \otimes 2^L$ is the product σ -algebra for which $\mathcal{B}(\mathbb{R}_{\geq 0})$ is the Borel σ -field on $\mathbb{R}_{\geq 0}$.

The following definition introduces the notion of templates which will be used to define the measurable spaces.

Definition 8. A template θ is a finite sequence $\theta = \langle s, U_1, \dots, U_m \rangle$ ($m \geq 0$) such that $s \in L$ and $U_i \in \mathcal{U}_{\mathcal{M}}$ for $1 \leq i \leq m$; The length of θ , denoted by $|\theta|$, is defined to be m . The set of histories $\text{Hists}(\theta)$ spanned by a template θ is defined by:

$$\text{Hists}(\langle s, U_1, \dots, U_m \rangle) := \left\{ \xi \in \text{Hists}^m(\mathcal{M}) \mid \xi[0] = s \text{ and } (\xi(i), \xi(i), \xi[i+1]) \in U_{i+1} \text{ for all } 0 \leq i < m \right\}.$$

Now we introduce the measurable spaces on paths and histories, as in the following definition.

Definition 9. The measurable space $(\Omega_{\mathcal{M}}^n, \mathcal{S}_{\mathcal{M}}^n)$ over $\text{Hists}^n(\mathcal{M})$ ($n \in \mathbb{N}_0$) is defined as follows: $\Omega_{\mathcal{M}}^n = \text{Hists}^n(\mathcal{M})$ and $\mathcal{S}_{\mathcal{M}}^n$ is generated by the family

$$\{\text{Hists}(\theta) \mid \theta \text{ is a template and } |\theta| = n\}$$

of subsets of $\text{Hists}^n(\mathcal{M})$.

The measurable space $(\Omega_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}})$ over $\text{Paths}(\mathcal{M})$ is defined as follows: $\Omega_{\mathcal{M}} = \text{Paths}(\mathcal{M})$ and $\mathcal{S}_{\mathcal{M}}$ is the smallest σ -algebra generated by the family

$$\{\text{Cyl}(\Xi) \mid \Xi \in \mathcal{S}_{\mathcal{M}}^n \text{ for some } n \geq 0\}$$

of subsets of $\text{Paths}(\mathcal{M})$.

3 Schedulers and Their Probability Spaces

The stochastic feature of a CTMDP is endowed by a (measurable) scheduler which resolves the action when a state is entered. In the following, we briefly introduce late schedulers for CTMDPs. Most notions in this part stem from [16,10]. Below we fix a locally-uniform CTMDP $\mathcal{M} = (L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$.

Definition 10. A late scheduler D is a function

$$D : Hists(\mathcal{M}) \times \mathbb{R}_{\geq 0} \times Act \rightarrow [0, 1]$$

such that for each $\xi \in Hists(\mathcal{M})$ and $t \in \mathbb{R}_{\geq 0}$, the following conditions hold:

- $\sum_{a \in Act} D(\xi, t, a) = 1$;
- for all $a \in Act$, $D(\xi, t, a) > 0$ implies $a \in \text{En}(\xi \downarrow)$.

D is measurable iff for all $n \geq 0$ and $a \in Act$, the function $D(\cdot, \cdot, a)$ is measurable w.r.t $(\Omega_{\mathcal{M}}^n \times \mathbb{R}_{\geq 0}, \mathcal{S}_{\mathcal{M}}^n \otimes \mathcal{B}(\mathbb{R}_{\geq 0}))$, provided that the domain of $D(\cdot, \cdot, a)$ is restricted to $Hists^n(\mathcal{M}) \times \mathbb{R}_{\geq 0}$.

Intuitively, a late scheduler D chooses a distribution over actions immediately after the time-delay at the current state s (i.e., the last state of a history) is ended; the time-delay observes the negative exponential distribution with rate $\mathbf{E}(s)$. The decision $D(\xi, t, \cdot)$ is based on the history ξ and the dwell time t at $\xi \downarrow$; the next state is determined stochastically w.r.t $\mathbf{P}(\xi \downarrow, a, \cdot)$, where a is in turn determined w.r.t $D(\xi, t, \cdot)$. The local-uniformity allows a late scheduler to make such decision, without mathematical ambiguity on the accumulated cost and the probability density function for the time-delay. Moreover, the measurability condition will be needed to define a probability measure for the measurable space $(\Omega_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}})$.

Each measurable late scheduler will induce a probability measure on combined actions, when applied to a specific history. Below we introduce the probability measure induced by measurable late schedulers.

Definition 11. Let $\xi \in Hists(\mathcal{M})$ be a history and D a measurable late scheduler. The probability measure $\mu_{\mathcal{M}}^D(\xi, \cdot)$ for the measurable space $(\Gamma_{\mathcal{M}}, \mathcal{U}_{\mathcal{M}})$ is defined as follows:

$$\mu_{\mathcal{M}}^D(\xi, U) := \int_{\mathbb{R}_{\geq 0}} f_{\mathbf{E}(\xi \downarrow)}(t) \cdot \left\{ \sum_{a \in \text{En}(\xi \downarrow)} D(\xi, t, a) \cdot \left[\sum_{s \in L} \mathbf{1}_U(a, t, s) \cdot \mathbf{P}(\xi \downarrow, a, s) \right] \right\} dt$$

for each $U \in \mathcal{U}_{\mathcal{M}}$.

Now we define the probability spaces on histories and paths. Firstly, we define the probability space on histories. To this end, we introduce the notion of concatenation as follows.

Definition 12. Let $\xi \in Hists(\mathcal{M})$ be a history and $(a, t, s) \in \Gamma_{\mathcal{M}}$ be a combined action. We define $\xi \circ (a, t, s) \in Hists(\mathcal{M})$ to be the history obtained by concatenating (a, t, s) to $\xi \downarrow$ (i.e. $\xi \circ (a, t, s) = \xi[0] \dots \xi \downarrow \xrightarrow{a, t} s$) .

Then the probability space on histories of fixed length is given as follows.

Definition 13. *Suppose D is a measurable late scheduler and α is an initial distribution. The sequence $\{\Pr_{\mathcal{M},D,\alpha}^n : \mathcal{S}_{\mathcal{M}}^n \rightarrow [0, 1]\}_{n \geq 0}$ of probability measures is inductively as follows:*

$$\Pr_{\mathcal{M},D,\alpha}^0(\Xi) := \sum_{s \in \Xi} \alpha(s) ;$$

$$\Pr_{\mathcal{M},D,\alpha}^{n+1}(\Xi) := \int_{\Omega_{\mathcal{M}}^n} \left[\int_{\Gamma_{\mathcal{M}}} \mathbf{1}_{\Xi}(\xi \circ \gamma) \mu_{\mathcal{M}}^D(\xi, d\gamma) \right] \Pr_{\mathcal{M},D,\alpha}^n(d\xi) ;$$

for each $\Xi \in \mathcal{S}_{\mathcal{M}}^n$.

Finally, the probability space on paths is given as follows.

Definition 14. *Let D be a measurable late scheduler and α be an initial distribution. The probability space $(\Omega_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}}, \Pr_{\mathcal{M},D,\alpha})$ is defined as follows:*

- $\Omega_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{M}}$ is defined as in Definition 9;
- $\Pr_{\mathcal{M},D,\alpha}$ is the unique probability measure such that

$$\Pr_{\mathcal{M},D,\alpha}(\text{Cyl}(\Xi)) = \Pr_{\mathcal{M},D,\alpha}^n(\Xi)$$

for all $n \geq 0$ and $\Xi \in \mathcal{S}_{\mathcal{M}}^n$.

We end this section with a fundamental property asserting that the role of initial distribution α can be decomposed into Dirac distributions on individual states.

Proposition 1. *For each measurable late scheduler D and each initial distribution α , $\Pr_{\mathcal{M},D,\alpha}(\Pi) = \sum_{s \in L} \alpha(s) \cdot \Pr_{\mathcal{M},D,\mathcal{D}[s]}(\Pi)$ for all $\Pi \in \mathcal{S}_{\mathcal{M}}$.*

4 Maximal Cost-Bounded Reachability Probability

In this section, we consider maximal cost-bounded reachability probabilities. Below we fix a locally-uniform CTMDP $\mathcal{M} = (L, \text{Act}, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ and a set $G \subseteq L$. For the sake of simplicity, we omit the ‘ \mathcal{M} ’ which appear in the subscript of ‘Pr’.

Definition 15. *Let D be a measurable late scheduler. Define the function $\text{prob}_G^D : L \times \mathbb{R}^k \rightarrow [0, 1]$ by: $\text{prob}_G^D(s, \mathbf{c}) := \Pr_{D,\mathcal{D}[s]}(\Pi_G^{\mathbf{c}})$ where*

$$\Pi_G^{\mathbf{c}} := \{\pi \in \text{Paths}(\mathcal{M}) \mid \mathbf{C}(\pi, G) \leq \mathbf{c}\} .$$

Define $\text{prob}_G^{\max} : L \times \mathbb{R}^k \rightarrow [0, 1]$ by: $\text{prob}_G^{\max}(s, \mathbf{c}) := \sup_{D \in \mathcal{L}_{\mathcal{M}}} \text{prob}_G^D(s, \mathbf{c})$ for $s \in L$ and $\mathbf{c} \in \mathbb{R}^k$, where $\mathcal{L}_{\mathcal{M}}$ is the set of all measurable late schedulers.

From the definition, we can see that Π_G^c is the set of paths which can reach G within cost \mathbf{c} , $\text{prob}_G^{\max}(s, \mathbf{c})$ is the maximal probability of Π_G^c with initial distribution $\mathcal{D}(s)$ (i.e., fixed initial state s) ranging over all late schedulers. It is not hard to verify that Π_G^c is measurable, thus all functions in Definition 15 are well-defined. It is worth noting that if $\mathbf{c} \not\geq \mathbf{0}$, then both $\text{prob}_G^D(s, \mathbf{c})$ and $\text{prob}_G^{\max}(s, \mathbf{c})$ is zero.

The following theorem mainly presents the fixed-point characterization for prob_G^{\max} , while it also states that prob_G^{\max} is Lipschitz continuous.

Theorem 1. *The function prob_G^{\max} is the least fixed-point (w.r.t \leq) of the high-order operator $\mathcal{T}_G : [L \times \mathbb{R}^k \rightarrow [0, 1]] \rightarrow [L \times \mathbb{R}^k \rightarrow [0, 1]]$ defined as follows:*

- $\mathcal{T}_G(h)(s, \mathbf{c}) := \mathbf{1}_{\mathbb{R}_{\geq 0}^k}(\mathbf{c})$ if $s \in G$;
- If $s \notin G$ then

$$\mathcal{T}_G(h)(s, \mathbf{c}) := \int_0^\infty f_{\mathbf{E}(s)}(t) \cdot \max_{a \in \text{En}(s)} \left[\sum_{s' \in L} \mathbf{P}(s, a, s') \cdot h(s', \mathbf{c} - t \cdot \mathbf{w}(s)) \right] dt ;$$

for each $h : L \times \mathbb{R}^k \rightarrow [0, 1]$. Moreover,

$$|\text{prob}_G^{\max}(s, \mathbf{c}) - \text{prob}_G^{\max}(s, \mathbf{c}')| \leq \frac{\mathbf{E}_{\max}}{\mathbf{w}_{\min}} \cdot \|\mathbf{c} - \mathbf{c}'\|_\infty$$

for all $\mathbf{c}, \mathbf{c}' \geq \mathbf{0}$ and $s \in L$.

The Lipschitz constant $\frac{\mathbf{E}_{\max}}{\mathbf{w}_{\min}}$ will be crucial to the error bound of our approximation algorithm.

Now we describe the proof error in [9,11]. The error lies in the proof of [9, Lemma 5.1 on Pages 119] which tries to prove that the time-bounded reachability probability functions are continuous. In detail, the error is at the proof for right-continuity of the functions. Let us take the sentence ‘‘This implies ... for some $\xi \leq \frac{\epsilon}{2}$.’’ from line -3 to line -2 on page 119 as (*). (*) is wrong in general, as one can treat D 's as natural numbers, and define

$$\text{Pr}_n(\text{‘‘reach } G \text{ within } z\text{’’}) := \begin{cases} n \cdot z & \text{if } z \in [0, \frac{1}{n}] \\ 1 & \text{if } z \in (\frac{1}{n}, \infty) \end{cases} .$$

Then $\sup_n \text{Pr}_n(\text{‘‘reach } G \text{ within } z\text{’’})$ equals 1 for $z > 0$ and 0 for $z = 0$. Thus $\sup_D \text{Pr}_D(\text{‘‘reach } G \text{ within } z\text{’’})$ on $z \geq 0$ is right-discontinuous at $z = 0$, which does not satisfy (*) (treat D as a natural number). Note that a concrete counterexample does not exist as [9, Lemma 5.1] is correct due to this paper; it is the proof that is flawed. Also note that Lemma 5.1 is important as the least fixed-point characterization [9, Theorem 5.1 on Page 120] and the optimal scheduler [9, Theorem 5.2 on page 124] directly rely on it. We fix the error in the more general setting of cost-bounded reachability probability by providing new proofs.

5 Optimal Schedulers

In this section, we establish optimal late schedulers for maximal cost-bounded reachability probability. We show that there exists a deterministic cost-positional late scheduler that achieves the maximal cost-bounded reachability probability. Below we fix a locally-uniform CTMDP $\mathcal{M} = (L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$. We first introduce the notion of deterministic cost-positional schedulers.

Definition 16. *A measurable late scheduler D is deterministic cost-positional iff (i) $D(\xi, t, \bullet) = D(\xi', t', \bullet)$ whenever $\xi \downarrow = \xi' \downarrow$ and $\mathbf{C}(\xi) + t \cdot \mathbf{w}(\xi \downarrow) = \mathbf{C}(\xi') + t' \cdot \mathbf{w}(\xi' \downarrow)$, and (ii) $D(\xi, t, \bullet)$ is Dirac for all histories ξ and $t \geq 0$.*

Intuitively, a deterministic cost-positional scheduler makes its decision solely on the current state and the cost accumulated so far, and its decision is always Dirac. The following theorem shows that such a scheduler suffices to achieve maximal cost-bounded reachability probability.

Theorem 2. *For all $\mathbf{c} \in \mathbb{R}^k$ and $G \subseteq L$, there exists a deterministic cost-positional measurable late scheduler D such that $\text{prob}_G^{\max}(s, \mathbf{c}) = \text{Pr}_{\mathsf{D}, \mathcal{D}[s]}(\Pi_G^{\mathbf{c}})$ for all $s \in L$.*

6 Differential Characterization for Maximal Reachability Probabilities

In this section, we derive differential characterization for the function prob_G^{\max} . The differential characterization will be fundamental to our approximation algorithm. Below we fix a locally-uniform CTMDP $(L, Act, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$ and a set $G \subseteq L$.

The differential characterization relies on a notion of directional derivative as follows.

Definition 17. *Let $s \in L - G$ and $\mathbf{c} \geq \mathbf{0}$. Define*

$$\nabla^+ \text{prob}_G^{\max}(s, \mathbf{c}) := \lim_{t \rightarrow 0^+} \frac{\text{prob}_G^{\max}(s, \mathbf{c} + t \cdot \mathbf{w}(s)) - \text{prob}_G^{\max}(s, \mathbf{c})}{t} .$$

If $\mathbf{c}_i > 0$ whenever $\mathbf{w}_i(s) > 0$ ($1 \leq i \leq k$), define

$$\nabla^- \text{prob}_G^{\max}(s, \mathbf{c}) := \lim_{t \rightarrow 0^-} \frac{\text{prob}_G^{\max}(s, \mathbf{c} + t \cdot \mathbf{w}(s)) - \text{prob}_G^{\max}(s, \mathbf{c})}{t} ;$$

Otherwise, let $\nabla^- \text{prob}_G^{\max}(s, \mathbf{c}) = \nabla^+ \text{prob}_G^{\max}(s, \mathbf{c})$.

The following theorem gives a characterization for the directional derivative.

Theorem 3. *For all $s \in L - G$ and $\mathbf{c} \geq \mathbf{0}$, $\nabla^+ \text{prob}_G^{\max}(s, \mathbf{c}) = \nabla^- \text{prob}_G^{\max}(s, \mathbf{c})$. Moreover,*

$$\nabla^+ \text{prob}_G^{\max}(s, \mathbf{c}) = \max_{a \in \text{En}(s)} \sum_{s' \in L} \mathbf{R}(s, a, s') \cdot (\text{prob}_G^{\max}(s', \mathbf{c}) - \text{prob}_G^{\max}(s, \mathbf{c})) .$$

As $\nabla^+ \text{prob}_G^{\max}(s, \mathbf{c}) = \nabla^- \text{prob}_G^{\max}(s, \mathbf{c})$, we will solely use $\nabla \text{prob}_G^{\max}(s, \mathbf{c})$ to denote both of them.

Theorem 3 allows one to approximate $\text{prob}_G^{\max}(s, \mathbf{c} + t \cdot \mathbf{w}(s))$ by $\text{prob}_G^{\max}(s, \mathbf{c})$ and $\nabla^+ \text{prob}_G^{\max}(s, \mathbf{c})$. An exception is the case $\mathbf{w}(s) = \mathbf{0}$. Below we tackle this situation.

Proposition 2. *Let $Y_G := \{s \in L \mid \mathbf{w}(s) = \mathbf{0}\}$. For each $\mathbf{c} \geq \mathbf{0}$, the function $s \mapsto \text{prob}_G^{\max}(s, \mathbf{c})$ is the least fixed-point (w.r.t \leq) of the high-order operator $\mathcal{Y}_{\mathbf{c}, G} : [Y_G \rightarrow [0, 1]] \rightarrow [Y_G \rightarrow [0, 1]]$ defined as follows:*

$$\mathcal{Y}_{\mathbf{c}, G}(h)(s) := \max_{a \in \text{En}(s)} \left[\sum_{s' \in Y_G} \mathbf{P}(s, a, s') \cdot h(s') + \sum_{s' \in L - Y_G} \mathbf{P}(s, a, s') \cdot \text{prob}_G^{\max}(s', \mathbf{c}) \right].$$

7 Numerical Approximation Algorithms

In this section, we develop an approximation algorithm to compute the maximal cost-bounded reachability probability under late schedulers. In the following we fix a locally-uniform CTMDP $\mathcal{M} = (L, \text{Act}, \mathbf{R}, \{\mathbf{w}_i\}_{1 \leq i \leq k})$. Our numerical algorithm will achieve the following tasks:

Input: a set $G \subseteq L$, a state $s \in L$, a vector $\mathbf{c} \in \mathbb{N}_0^k$ and an error bound $\epsilon > 0$;

Output: a value $x \in [0, 1]$ such that $|\text{prob}_G^{\max}(s, \mathbf{c}) - x| \leq \epsilon$.

For computational purposes, we assume that each $\mathbf{w}_i(s)$ is an integer; rational numbers (and simultaneously the input cost bound vector) can adjusted to integers by multiplying a common multiplier of the denominators, without changing the maximal probability value to be approximated.

We base our approximation scheme on Theorem 3 and Proposition 2. In the following we fix a set $G \subseteq L$.

Below we illustrate the discretization and the approximation scheme for late schedulers. Note that $\text{prob}_G^{\max}(s, \mathbf{c}) = 1$ whenever $s \in G$ and $\mathbf{c} \geq \mathbf{0}$. Thus we do not need to incorporate those points into the discretization.

Definition 18. *Let $\mathbf{c} \in \mathbb{N}_0^k$ and $N \in \mathbb{N}$. Define*

$$\text{Disc}(\mathbf{c}, N) := \{\mathbf{d} \in \mathbb{R}^k \mid \mathbf{0} \leq \mathbf{d} \leq \mathbf{c} \text{ and } N \cdot \mathbf{d}_i \in \mathbb{N}_0 \text{ for all } 1 \leq i \leq k\}.$$

The set $\text{D}_N^{\mathbf{c}}$ of discretized grid points is defined as follows:

$$\text{D}_N^{\mathbf{c}} := (L - G) \times \text{Disc}(\mathbf{c}, N).$$

The following definition presents the approximation scheme on $\text{D}_N^{\mathbf{c}}$.

Definition 19. *Define $X_G := (L - G) - Y_G$. The approximation scheme $\Upsilon_{\mathbf{c}, N}^G$ on $\text{D}_N^{\mathbf{c}}$ consists of the following items:*

- exactly one rounding argument for each element of D_N^c ;
- a system of equations for elements in $X_G \times \text{Disc}(\mathbf{c}, N)$;
- a linear program on Y_G for each $\mathbf{d} \in \text{Disc}(\mathbf{c}, N)$.

Rounding Arguments: For each element $y \in D_N^c$, the rounding argument for y is as follows:

$$\overline{\text{prob}}_G(y) := \frac{K}{N^2} \text{ if } \text{prob}_G(y) \in \left[\frac{K}{N^2}, \frac{K+1}{N^2} \right) \text{ for some integer } 0 \leq K \leq N^2 .$$

Equations: The system of equations is described as follows. For all $(s, \mathbf{d}) \in D_N^c$ with $\mathbf{w}(s) \neq \mathbf{0}$ and $\mathbf{d} - \frac{1}{N} \cdot \mathbf{w}(s) \geq \mathbf{0}$, there is a linear equation

$$\frac{\text{prob}_G(s, \mathbf{d}) - \overline{\text{prob}}_G(s, \text{pre}(\mathbf{d}, s))}{\frac{1}{N}} = \max_{a \in \text{En}(s)} \sum_{s' \in L} \mathbf{R}(s, a, s') \cdot (\overline{\text{prob}}_G(s', \text{pre}(\mathbf{d}, s)) - \overline{\text{prob}}_G(s, \text{pre}(\mathbf{d}, s))) \quad (\text{E1})$$

where $\text{pre}(\mathbf{d}, s) := \mathbf{d} - \frac{1}{N} \cdot \mathbf{w}(s)$. For all $(s, \mathbf{d}) \in D_N^c$ with $\mathbf{w}(s) \neq \mathbf{0}$ and $\mathbf{d} - \frac{1}{N} \cdot \mathbf{w}(s) \not\geq \mathbf{0}$, there is a linear equation

$$\text{prob}_G(s, \mathbf{d}) = 0 . \quad (\text{E2})$$

Linear Programs: For each $\mathbf{d} \in \text{Disc}(\mathbf{c}, N)$, the collection $\{\text{prob}_G(s, \mathbf{d})\}_{s \in Y_G}$ of values is the unique optimum solution of the linear program as follows:

- $\min \sum_{s \in Y_G} \text{prob}_G(s, \mathbf{d})$, subject to:
- $\text{prob}_G(s, \mathbf{d}) \geq \sum_{s' \in L} \mathbf{P}(s, a, s') \cdot \text{prob}_G(s', \mathbf{d})$ for all $s \in Y_G$ and $a \in \text{En}(s)$;
 - $\text{prob}_G(s, \mathbf{d}) \in [0, 1]$ for all $s \in Y_G$;

where the values $\{\text{prob}_G(s, \mathbf{d})\}_{s \in X_G}$ are assumed to be known. All $\text{prob}_G(s, \mathbf{d})$'s and $\overline{\text{prob}}_G(s, \mathbf{d})$ above with $s \in G$ are predefined to be 1.

Generally, $\text{prob}_G(s, \mathbf{d})$ approximates $\text{prob}_G^{\max}(s, \mathbf{d})$ and $\overline{\text{prob}}_G(s, \mathbf{d})$ approximates the same value with a rounding operation. A detailed computational sequence of the approximation scheme is described in Algorithm 1.

In principle, we compute the ‘‘higher’’ grid point $\text{prob}_G(s, \mathbf{d} + \frac{1}{N} \cdot \mathbf{w}(s))$ by $\text{prob}_G(s, \mathbf{d})$ and (E1), and then update other ‘‘higher’’ points through the linear program. The rounding argument is incorporated to avoid precision explosion caused by linear programming. The following proposition shows that Algorithm 1 indeed terminates after a finite number of steps.

Proposition 3. *Algorithm 1 terminates after a finite number of steps for all $\mathbf{c} \in \mathbb{N}_0^k$ and $N \in \mathbb{N}$.*

The following theorem states that the approximation scheme really approximates prob_G^{\max} . To ease the notation, we shall use $\text{prob}_G(s, \mathbf{d})$ or $\overline{\text{prob}}_G(s, \mathbf{d})$ to denote both the variable at the grid point and the value it holds under the approximation scheme.

Algorithm 1. The Computation of $\mathcal{Y}_{\mathbf{c},N}^G$ (for late schedulers)

-
- 1: Set all grid points in $\{(s, \mathbf{d}) \in \mathbb{D}_N^{\mathbf{c}} \mid \mathbf{d} - \frac{1}{N} \cdot \mathbf{w}(s) \not\geq \mathbf{0}\}$ to zero by (E2);
 - 2: Compute all $\text{prob}_G(s, \mathbf{d})$ that can be directly obtained through the linear program;
 - 3: Compute all $\overline{\text{prob}}_G(s, \mathbf{d})$ that can be directly obtained by the rounding argument;
 - 4: Compute all $\text{prob}_G(s, \mathbf{d})$ that can be directly obtained through (E1), and back to Step 2. until all grid points in $\mathbb{D}_N^{\mathbf{c}}$ are computed;
-

Theorem 4. Let $\mathbf{c} \in \mathbb{N}_0^k$ and $N \in \mathbb{N}$ with $N \geq \mathbf{E}_{\max}$. For each $(s, \mathbf{d}) \in \mathbb{D}_N^{\mathbf{c}}$,

$$|\text{prob}_G(s, \mathbf{d}) - \text{prob}_G^{\max}(s, \mathbf{d})| \leq \left(\frac{2 \cdot \mathbf{E}_{\max}^2 \cdot \mathbf{w}_{\max}}{N \cdot \mathbf{w}_{\min}} + \frac{1}{N} \right) \cdot \left[\sum_{i=1}^k \mathbf{d}_i \right] + \frac{\mathbf{E}_{\max}}{N}$$

and

$$\begin{aligned} |\overline{\text{prob}}_G(s, \mathbf{d}) - \text{prob}_G^{\max}(s, \mathbf{d})| &\leq \\ &\left(\frac{2 \cdot \mathbf{E}_{\max}^2 \cdot \mathbf{w}_{\max}}{N \cdot \mathbf{w}_{\min}} + \frac{1}{N} \right) \cdot \left[\sum_{i=1}^k \mathbf{d}_i \right] + \frac{\mathbf{E}_{\max}}{N} + \frac{1}{N^2} \end{aligned}$$

From Theorem 4, we derive our approximation algorithm as follows.

Corollary 1. There exists an algorithm such that given any $\epsilon > 0$, $s \in L$, $G \subseteq L$ and $\mathbf{c} \in \mathbb{N}_0^k$, the algorithm outputs a $d \in [0, 1]$ which satisfies that $|d - \text{prob}_G^{\max}(s, \mathbf{c})| \leq \epsilon$. Moreover, the algorithm runs in

$$\mathcal{O}\left(\left(\max\left\{\mathbf{E}_{\max}, \frac{M}{\epsilon}\right\}\right)^k \cdot \left(\prod_{i=1}^k \mathbf{c}_i\right) \cdot \left(|\mathcal{M}| + \log \frac{M}{\epsilon}\right)^8\right)$$

time, where $M := \left(2 \cdot \mathbf{E}_{\max}^2 \cdot \frac{\mathbf{w}_{\max}}{\mathbf{w}_{\min}} + 1\right) \cdot \left[\sum_{i=1}^k \mathbf{c}_i\right] + \mathbf{E}_{\max} + 1$ and $|\mathcal{M}|$ is the size of \mathcal{M} .

Proof. The algorithm is an simple application of Theorem 4. If $s \in G$, the algorithm just returns 1. Otherwise, the algorithm just calls Algorithm 1 with $N := \lfloor \max\{\mathbf{E}_{\max}, \frac{M}{\epsilon}\} \rfloor + 1$ and set $d = \text{prob}_G(s, \mathbf{c})$; By Theorem 4, we directly obtain that $|d - \text{prob}_G^{\max}(s, \mathbf{c})| \leq M \cdot \frac{1}{N}$. For each $\mathbf{d} \in \text{Disc}(\mathbf{c}, N)$, the total computation of $\{\text{prob}_G(s, \mathbf{d})\}_{s \in X_G \cup Y_G}$ and $\{\overline{\text{prob}}_G(s, \mathbf{d})\}_{s \in X_G \cup Y_G}$ takes $\mathcal{O}\left(|\mathcal{M}| + \log \frac{M}{\epsilon}\right)^8$ time since the most time consuming part is the linear program which takes $\mathcal{O}\left(|\mathcal{M}| + \log N\right)^8$ time (cf. [15]). Thus the total running time of the algorithm is $\mathcal{O}\left(\left(\max\left\{\mathbf{E}_{\max}, \frac{M}{\epsilon}\right\}\right)^k \cdot \left(\prod_{i=1}^k \mathbf{c}_i\right) \cdot \left(|\mathcal{M}| + \log \frac{M}{\epsilon}\right)^8\right)$ since the size of $\text{Disc}(\mathbf{c}, N)$ is $\mathcal{O}\left(N^k \cdot \left(\prod_{i=1}^k \mathbf{c}_i\right)\right)$.

8 Conclusion

In this paper, we established an integral characterization for multi-dimensional maximal cost-bounded reachability probabilities in continuous-time Markov decision processes, the existence of deterministic cost-positional optimal scheduler

and an algorithm to approximate the cost-bounded reachability probability with an error bound, under the setting of both early and late schedulers. The approximation algorithm is based on a differential characterization of cost-bounded reachability probability which in turn is derived from the integral characterization. The error bound is obtained through the differential characterization and the Lipschitz property described. Moreover, the approximation algorithm runs in polynomial time in the size of the CTMDP and the reciprocal of the error bound, and exponential in the dimension of the unit-cost vector. An important missing part is the generation of an ϵ -optimal scheduler. However, we conjecture that an ϵ -optimal scheduler is not difficult to obtain given that the approximation scheme has been established.

A future direction is to determine an ϵ -optimal scheduler, under both early and late schedulers. Besides, we believe that the paradigms developed in this paper can also be applied to minimum cost-bounded reachability probability and even stochastic games [12] with multi-dimensional cost-bounded reachability objective.

Acknowledgement. I thank Prof. Joost-Pieter Katoen for his valuable advices on the writing of the paper, especially for the Introduction part. I also thank anonymous referees for valuable comments.

References

1. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Reachability in continuous-time Markov reward decision processes. In: Flum, J., Grädel, E., Wilke, T. (eds.) *Logic and Automata. Texts in Logic and Games*, vol. 2, pp. 53–72. Amsterdam University Press (2008)
2. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. *Commun. ACM* 53(9), 76–85 (2010)
3. Brázdil, T., Forejt, V., Kreál, J., Kretínský, J., Kučera, A.: Continuous-time stochastic games with time-bounded reachability. *Inf. Comput.* 224, 46–70 (2013)
4. Buchholz, P., Schulz, I.: Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers & OR* 38(3), 651–659 (2011)
5. Fearnley, J., Rabe, M., Schewe, S., Zhang, L.: Efficient approximation of optimal control for continuous-time Markov games. In: Chakraborty, S., Kumar, A. (eds.) *FSTTCS. LIPIcs*, vol. 13, pp. 399–410. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2011)
6. Feller, W.: *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, New York (1966)
7. Fu, H.: Maximal cost-bounded reachability probability on continuous-time Markov decision processes. *CoRR* abs/1310.2514 (2013)
8. Hatefi, H., Hermanns, H.: Improving time bounded reachability computations in interactive Markov chains. In: Arbab, F., Sirjani, M. (eds.) *FSEN 2013. LNCS*, vol. 8161, pp. 250–266. Springer, Heidelberg (2013)
9. Neuhäuser, M.R.: *Model checking nondeterministic and randomly timed systems*. Ph.D. thesis, RWTH Aachen (2010)

10. Neuhäüßer, M.R., Stoelinga, M., Katoen, J.-P.: Delayed nondeterminism in continuous-time Markov decision processes. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 364–379. Springer, Heidelberg (2009)
11. Neuhäüßer, M.R., Zhang, L.: Time-bounded reachability probabilities in continuous-time Markov decision processes. In: QEST, pp. 209–218. IEEE Computer Society (2010)
12. Prieto-Rumeau, T., Hernández-Lerma, O.: Selected Topics on Continuous-Time Controlled Markov Chains and Markov Games. Imperial College Press, London (2012)
13. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st edn. John Wiley & Sons, Inc., New York (1994)
14. Rabe, M.N., Schewe, S.: Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Inf.* 48(5-6), 291–315 (2011)
15. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Inc., New York (1986)
16. Wolovick, N., Johr, S.: A characterization of meaningful schedulers for continuous-time Markov decision processes. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 352–367. Springer, Heidelberg (2006)

Type Reconstruction for the Linear π -Calculus with Composite and Equi-Recursive Types

Luca Padovani

Dipartimento di Informatica, Università di Torino, Italy

Abstract. We extend the linear π -calculus with composite and equi-recursive types in a way that enables the sharing of data containing linear values, provided that there is no overlapping access on such values. We show that the extended type system admits a complete type reconstruction algorithm and, as a by-product, we solve the problem of reconstruction for equi-recursive session types.

1 Introduction

The linear π -calculus [11] is a formal model of communicating processes in which channels are either *unlimited* or *linear*. Unlimited channels can be used without restrictions, while linear channels can only be used once for an input/output. Linear channels occur frequently in actual systems, they allow optimisations and efficient implementations [6,5,11], and communications on linear channels enjoy important properties such as interference freedom and partial confluence [13,11].

Type reconstruction is the problem of *inferring* the type of entities – channels in our case – given a program using them. For the linear π -calculus this problem was addressed in [7], although that work did not consider composite or recursive types. The goal of this work is the definition of a type reconstruction algorithm for the linear π -calculus extended with pairs, disjoint sums, and equi-recursive types. These constructs, albeit standard, gain relevance and combine in non-trivial ways with the features of the linear π -calculus. We explain why this is the case in the rest of this section.

By definition, linear channels can only be used for one-shot communications. It is a known fact, however, that more sophisticated interactions can be implemented taking advantage of *channel mobility* using a continuation-passing style [9,2]. The basic idea is that, along with the proper payload of a communication, one can send another channel on which the rest of the conversation takes place. This technique is illustrated below

$$P(x,y) \stackrel{\text{def}}{=} (\nu a)(\bar{x}\langle y,a \rangle \mid P\langle a,y+1 \rangle) \quad C(x) \stackrel{\text{def}}{=} x(y,z).C\langle z \rangle \quad (1.1)$$

where a *producer* process P sends messages to a *consumer* process C . At each iteration, the producer creates a new channel a , sends the payload y to the consumer on x along with the continuation a on which subsequent communications will take place, and iterates. In parallel, the consumer waits for the payload and the continuation from the producer on x and then iterates. Explicit continuations are key to preserve the order of produced messages. Had we modelled (1.1) re-using the same channel x at each iteration, there would be no guarantee that messages were received in order.

Let us now assign types to the channels in (1.1) starting from x in the consumer. There, x is used once for receiving a pair made of an integer y and another channel z .

Say the type of z is t and note that z is used in $C\langle z \rangle$ in the same place as x , meaning that x and z must have the same type. Then, also x has type t and t must be a channel type satisfying the equation $t = [\mathbf{int} \times t]^{1,0}$ where $\mathbf{int} \times t$ is the type of messages received from the channel and the numbers 1 and 0, henceforth called *uses*, indicate that the channel of type t is used once for input and never for output. Channel x is used once in the producer for sending an integer y and a continuation a . Clearly, the continuation must have type t for that is how it is used in C . Therefore, x in P has type $s = [\mathbf{int} \times t]^{0,1}$ where the uses 0 and 1 indicate that x is never used for input and is used once for output. Finally, there are two (non-binding) occurrences of a in the producer: a in $\bar{x}\langle n, a \rangle$ has type t because that is how a will be used by the consumer; a in $P\langle a, y+1 \rangle$ has type s because that is how a will be used by the producer at the next iteration. Overall, the uses in the types of a say that a is a linear channel: it is used once by the producer for sending the payload and once by the consumer for receiving it.

Note that linear channels, like a in (1.1), may syntactically occur multiple times and that different occurrences of the same channel may have different yet *compatible* types. In the case of (1.1), the types t and s of the non-binding occurrences of a are compatible because corresponding uses in t and s are never 1 at the same time. The binding occurrence of a in (va) has type $[\mathbf{int} \times t]^{1,1}$, which can be seen as the *combination* of t and s as defined in [11,14].

One legitimate question is whether and how the notions of type compatibility and combination extend beyond channel types. In this respect, the existing literature lacks definitive and satisfactory answers: the works on (type reconstruction for) the linear π -calculus [11,7] do not consider composite or recursive types. Linear type systems with composite types have been discussed in [5,6] for the linear π -calculus and in [15] for a functional language. These works, however, see linearity as a “contagious” property: every structure that contains linear values becomes linear itself (there are a few exceptions for specific types [10] or relaxed notions of linearity [8]). Such interpretation may be appropriate in a sequential setting, but is not the only sensible one in a concurrent/parallel setting. In fact, it is acceptable (and desirable, for the sake of parallelism) that several processes share the *same* composite data structure, provided that they access *different* linear values stored therein. The problem is whether the type system is accurate enough to capture the fact that there are no overlapping accesses to the same linear values. To illustrate, consider the type t_{list} satisfying the equation

$$t_{list} = \mathbf{unit} \oplus ([\mathbf{int}]^{0,1} \times t_{list})$$

which is the disjoint sum between \mathbf{unit} and the product $[\mathbf{int}]^{0,1} \times t_{list}$ and which can be used for typing lists of linear channels with type $[\mathbf{int}]^{0,1}$. If we follow [15,5,6], an identifier l having type t_{list} can syntactically occur only once in a program, and a process like for instance $Odd\langle l \rangle | Even\langle l \rangle$ where l occurs twice is illegal. However, suppose that Odd and $Even$ are the two processes defined by

$$\begin{aligned} Odd(x) &\stackrel{\text{def}}{=} \mathbf{case } x \text{ of } [] \Rightarrow \mathbf{0} & Even(x) &\stackrel{\text{def}}{=} \mathbf{case } x \text{ of } [] \Rightarrow \mathbf{0} \\ y : z &\Rightarrow \bar{y}\langle 3 \rangle | Even\langle z \rangle & y : z &\Rightarrow Odd\langle z \rangle \end{aligned}$$

which walk through a list x : if x is the empty list $[]$ they do nothing; if x has head y and tail z , Odd sends 3 on y and continues as $Even\langle z \rangle$ while $Even$ ignores y and continues

as $Odd\langle z \rangle$. So, $Odd\langle l \rangle$ sends 3 on every odd-indexed channel in l and $Even\langle l \rangle$ sends 3 on every even-indexed channel in l . The fact that Odd and $Even$ access different linear values of a list is reflected in the two (mutually recursive) types of x in Odd and $Even$:

$$t_{odd} = \mathbf{unit} \oplus ([\mathbf{int}]^{0,1} \times t_{even}) \quad \text{and} \quad t_{even} = \mathbf{unit} \oplus ([\mathbf{int}]^{0,0} \times t_{odd}) \quad (1.2)$$

where the two 0's in t_{even} denote that $Even$ does not use at all the first (and more generally every odd-indexed) element of its parameter x . The key observation is that just like a was allowed to occur twice in (1.1) with two compatible types t and s whose combination was $[\mathbf{int} \times t]^{1,1}$, then we can allow l to occur twice in $Odd\langle l \rangle \mid Even\langle l \rangle$ given that the two occurrences of l are used according to two compatible types t_{odd} and t_{even} whose combination is t_{list} . The “only” difference is that, while t and s were channel types and their combination could be expressed simply by combining the uses in them, t_{odd} and t_{even} are recursive, structured types that combine to t_{list} *in the limit*.

To summarise, composite and recursive types are basic yet fundamental features whose smooth integration in the linear π -calculus requires some care. In this work we extend the linear π -calculus with composite and recursive types in such a way that multiple processes can safely share structured data containing linear values and we define a complete type reconstruction algorithm for the extended type system.

We proceed with the formal definition of the language and of the type system (Section 2). The type reconstruction algorithm consists of a constraint generation phase (Section 3) and a constraint solving phase (Section 4). Section 5 concludes. The full version of the paper (with proofs) and a Haskell implementation of the algorithm are available on the author's home page.

2 The Linear π -Calculus

We let m, n, \dots range over integer numbers; we use a countable set of *channels* a, b, \dots and a disjoint countable set of *variables* x, y, \dots ; *names* u, v, \dots are either channels or variables. We work with the asynchronous π -calculus extended with constants, pairs, and disjoint sums. The syntax of expressions and processes is defined below:

$$\begin{aligned} e &::= n \mid u \mid (e, e) \mid \mathbf{inl} \ e \mid \mathbf{inr} \ e \mid \dots \\ P &::= \mathbf{0} \mid u(x).P \mid \bar{u}\langle e \rangle \mid (P \mid Q) \mid *P \mid (va)P \mid \begin{array}{l} \mathbf{let} \ x, y = e \\ \mathbf{in} \ P \end{array} \mid \begin{array}{l} \mathbf{case} \ e \ \mathbf{of} \\ \{ \mathbf{inl} \ x \Rightarrow P, \\ \mathbf{inr} \ y \Rightarrow Q \} \end{array} \end{aligned}$$

Expressions e, \dots are either integers, names, pairs (e_1, e_2) of expressions, or the injection ($i \ e$) of an expression e using the constructor $i \in \{\mathbf{inl}, \mathbf{inr}\}$. *Values* v, w, \dots are expressions without variables.

Processes P, Q, \dots comprise the standard constructs of the asynchronous π -calculus. In addition to these, we include two process forms for deconstructing pairs and disjoint sums. In particular, the process $\mathbf{let} \ x_1, x_2 = e \ \mathbf{in} \ P$ evaluates e , which must result into a pair v_1, v_2 , binds each v_i to x_i , and continues as P .¹ The process $\mathbf{case} \ e \ \mathbf{of} \{i \ x_i \Rightarrow$

¹ As pointed out by one reviewer, this construct is superfluous, because the type system we are about to define allows linear pairs to be accessed more than once with the conventional projection operators. We have added support for pair projections in the implementation, but we keep the \mathbf{let} construct in the formal presentation.

Table 1. Reduction of processes

$\frac{}{[\text{R-COMM}] \quad \bar{a}\langle v \rangle \mid a(x).Q \xrightarrow{a} Q\{v/x\}}$	$\frac{}{[\text{R-LET}] \quad \text{let } x, y = (v, w) \text{ in } P \xrightarrow{\tau} P\{v, w/x, y\}}$	
$\frac{}{[\text{R-CASE}] \quad \text{case } (k \ v) \ \text{of } \{i \ x_i \Rightarrow P_i\}_{i=\text{inl}, \text{inr}} \xrightarrow{\tau} P_k\{v/x_k\}}$	$\frac{P \xrightarrow{\ell} P'}{[\text{R-PAR}] \quad P \mid Q \xrightarrow{\ell} P' \mid Q}$	
$\frac{P \xrightarrow{a} Q}{[\text{R-NEW 1}] \quad (va)P \xrightarrow{\tau} (va)Q}$	$\frac{P \xrightarrow{\ell} Q \quad \ell \neq a}{[\text{R-NEW 2}] \quad (va)P \xrightarrow{\ell} (va)Q}$	$\frac{P \equiv P' \quad P' \xrightarrow{\ell} Q' \quad Q' \equiv Q}{[\text{R-STRUCT}] \quad P \xrightarrow{\ell} Q}$

$P_i\}_{i=\text{inl}, \text{inr}}$ evaluates e , which must result into a value $(i \ v)$ for $i \in \{\text{inl}, \text{inr}\}$, binds v to x_i and continues as P_i . Notions of *free names* $\text{fn}(P)$ and *bound names* $\text{bn}(P)$ of P are as expected. We identify processes modulo renaming of bound names and we write $P\{v/x\}$ for the capture-avoiding substitution of v for the free occurrences of x in P .

The operational semantics of the language is defined in terms of a structural congruence relation and a reduction relation, as usual. *Structural congruence* \equiv is completely standard (in particular, it includes the law $*P \equiv *P \mid P$). *Reduction* is defined in Table 1 and is also conventional, except that, like in [11], we decorate the relation with *labels* ℓ that are either channels or the special symbol τ , denoting an unobservable action: in [R-COMM] the label is the channel a on which a message is exchanged, while in [R-LET] and [R-CASE] it is τ to denote the fact that these are internal computations not involving communications. Rules [R-PAR], [R-NEW 1], and [R-NEW 2] propagate labels through parallel compositions and restrictions. In [R-NEW 1], the label a becomes τ when it escapes the scope of a . Rule [R-STRUCT] closes reduction under structural congruence.

The type system makes use of a countable set of *type variables* α, β, \dots and of *uses* κ, \dots which are elements of the set $\{0, 1, \omega\}$. Types t, s, \dots are defined by

$$t ::= \text{int} \mid \alpha \mid t \times t \mid t \oplus t \mid [t]^{\kappa_1, \kappa_2} \mid \mu \alpha. t$$

and include the type of integers **int**, products $t_1 \times t_2$ typing values of the form (v_1, v_2) where v_i has type t_i for $i = 1, 2$, and disjoint sums $t_1 \oplus t_2$ typing values of the form $(\text{inl } v)$ where v has type t_1 or of the form $(\text{inr } v)$ where v has type t_2 . Throughout the paper we let \odot stand for either \times or \oplus . The type $[t]^{\kappa_1, \kappa_2}$ denotes channels for exchanging messages of type t . The uses κ_1 and κ_2 respectively denote how many input and output operations are allowed on the channel: 0 for none, 1 for a single use, and ω for any number of uses. For example: a channel with type $[t]^{1,1}$ must be used once for receiving a message of type t and once for sending a message of type t ; a channel with type $[t]^{0,0}$ cannot be used; a channel with type $[t]^{\omega, \omega}$ can be used any number of times for sending and/or receiving. Type variables and μ operators are used for building recursive types, as usual. Notions of free and bound type variables are as expected. We assume that every bound type variable is guarded by a constructor to avoid meaningless terms such as $\mu \alpha. \alpha$. We identify types modulo renaming of bound type variables and if their infinite

unfoldings are the same (regular) tree [1]. In particular, we let $\mu\alpha.t = t\{\mu\alpha.t/\alpha\}$ where $t\{s/\alpha\}$ is the capture-avoiding substitution of the free occurrences of α in t with s .

We now define some key relations on uses and types. In particular, \leq is the least *partial order* such that $0 \leq \kappa$ and *compatibility* \succsim is the least relation such that

$$0 \succsim \kappa \quad \kappa \succsim 0 \quad \omega \succsim \omega \quad (2.1)$$

In what follow we will write $\kappa_1 < \kappa_2$ if $\kappa_1 \leq \kappa_2$ and $\kappa_1 \neq \kappa_2$ and $\kappa_1 \vee \kappa_2$ for the *least upper bound* of κ_1 and κ_2 , when it is defined. Note that $1 \not\leq \omega$ does *not* hold. Compatibility determines whether the least upper bound of two uses expresses their combination without any loss of precision. For example, $0 \succsim 1$ because the combination of 0 uses and 1 use is $0 \vee 1 = 1$ use. On the contrary, $1 \not\succsim 1$ because there is no 2 use that expresses the combination of 1 and 1 and ω is less precise than 2. Similarly, $1 \not\succsim \omega$ because there is no use expressing the fact that a channel is used *at least* once.

Every binary relation \mathcal{R}_{use} on uses induces a corresponding relation \mathcal{R}_{type} on types, defined coinductively by the following rules:

$$\text{int } \mathcal{R}_{type} \text{ int} \quad \frac{\kappa_1 \mathcal{R}_{use} \kappa_3 \quad \kappa_2 \mathcal{R}_{use} \kappa_4}{[t]^{\kappa_1, \kappa_2} \mathcal{R}_{type} [t]^{\kappa_3, \kappa_4}} \quad \frac{t_1 \mathcal{R}_{type} s_1 \quad t_2 \mathcal{R}_{type} s_2}{t_1 \odot t_2 \mathcal{R}_{type} s_1 \odot s_2} \quad (2.2)$$

Similarly, the partial operation \vee on uses coinductively induces one on types so that $t \vee s$ is the least upper bound of t and s , if it is defined. Note that \leq is antisymmetric, in particular $t = s$ if and only if $t \leq s$ and $t \geq s$. Note also that $[t]^{\kappa_1, \kappa_2} \mathcal{R}_{type} [s]^{\kappa_3, \kappa_4}$ implies $t = s$ regardless of \mathcal{R}_{type} . The relation $t \succsim t$ is particularly interesting, because it characterises unlimited types, those typing values that must not or need not be used. Linear types, on the other hand, denote values that must be used:

Definition 2.1. *We say that t is unlimited if $t \succsim t$. We say that t is linear otherwise.*

Channel types are either limited or unlimited depending on their uses. So, $[t]^{1,0}$, $[t]^{0,1}$, and $[t]^{1,1}$ are all linear types whereas $[t]^{0,0}$ and $[t]^{\omega, \omega}$ are both unlimited. Other types are linear or unlimited according to the channel types occurring in them. For example, $[t]^{0,0} \times [t]^{1,0}$ is linear while $[t]^{0,0} \times [t]^{\omega, 0}$ is unlimited. Recursion does not affect the classification of types into linear and unlimited. For example, $\mu\alpha. [\text{int} \times \alpha]^{1,0}$ is a linear type that denotes a channel that must be used once for receiving a pair made of an integer and another channel with the same type.

We type check expressions and processes in *type environments* Γ, \dots , which are finite maps from names to types that we write as $u_1 : t_1, \dots, u_n : t_n$. We identify type environments modulo the order of their bindings, we denote the *empty environment* with \emptyset , we write $\text{dom}(\Gamma)$ for the *domain* of Γ , namely the set of names for which there is a binding in Γ , and Γ_1, Γ_2 for the *union* of Γ_1 and Γ_2 when $\text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset$. We extend the relation \succsim between types pointwise to type environments:

$$\begin{aligned} \Gamma_1 + \Gamma_2 &\stackrel{\text{def}}{=} \Gamma_1, \Gamma_2 && \text{if } \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset \\ (\Gamma_1, u : t) + (\Gamma_2, u : s) &\stackrel{\text{def}}{=} (\Gamma_1 + \Gamma_2), u : t \vee s && \text{if } t \succsim s \end{aligned} \quad (2.3)$$

The operator $+$ generalises type combination in [11] and the \uplus operator in [14]. Note that $\Gamma_1 + \Gamma_2$ is undefined if there is $u \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)$ and $\Gamma_1(u) \not\succsim \Gamma_2(u)$ and that

Table 2. Type rules for expressions and processes

Expressions				
$\frac{[\text{T-CONST}]}{\Gamma \succ \Gamma} \quad \Gamma \succ \Gamma$	$\frac{[\text{T-NAME}]}{\Gamma \succ \Gamma} \quad \Gamma \succ \Gamma$	$\frac{[\text{T-PAIR}]}{\Gamma \vdash e : t \quad \Gamma' \vdash e' : s} \quad \Gamma + \Gamma' \vdash (e, e') : t \times s$	$\frac{[\text{T-INL}]}{\Gamma \vdash e : t} \quad \Gamma \vdash e : t$	$\frac{[\text{T-INR}]}{\Gamma \vdash e : s} \quad \Gamma \vdash e : s$
$\Gamma \vdash n : \mathbf{int}$	$\Gamma, u : t \vdash u : t$	$\Gamma + \Gamma' \vdash (e, e') : t \times s$	$\Gamma \vdash \mathbf{inl} \ e : t \oplus s$	$\Gamma \vdash \mathbf{inr} \ e : t \oplus s$
Processes				
$\frac{[\text{T-IDLE}]}{\Gamma \succ \Gamma} \quad \Gamma \succ \Gamma$	$\frac{[\text{T-IN}]}{\Gamma, x : t \vdash P} \quad \Gamma, x : t \vdash P$	$\frac{[\text{T-OUT}]}{\Gamma \vdash e : t} \quad \Gamma \vdash e : t$	$\frac{[\text{T-PAR}]}{\Gamma_i \vdash P_i \ (i=1,2)} \quad \Gamma_1 + \Gamma_2 \vdash P_1 P_2$	$\frac{[\text{T-REP}]}{\Gamma \vdash P} \quad \Gamma \vdash P$
$\Gamma \vdash \mathbf{0}$	$\Gamma + u : [t]^{\kappa, 0} \vdash u(x).P$	$\Gamma + u : [t]^{0, \kappa} \vdash \bar{u}(e)$	$\Gamma_1 + \Gamma_2 \vdash P_1 P_2$	$\Gamma \vdash *P$
$\frac{[\text{T-NEW}]}{\Gamma, a : [t]^{\kappa, \kappa} \vdash P} \quad \Gamma, a : [t]^{\kappa, \kappa} \vdash P$	$\frac{[\text{T-LET}]}{\Gamma \vdash e : t \times s} \quad \Gamma \vdash e : t \times s$	$\frac{[\text{T-CASE}]}{\Gamma \vdash e : t \oplus s} \quad \Gamma \vdash e : t \oplus s$	$\frac{[\text{T-CASE}]}{\Gamma \vdash e : t \oplus s} \quad \Gamma', x_i : t \vdash P_i \ (i=\mathbf{inl}, \mathbf{inr})$	
$\Gamma \vdash (va)P$	$\Gamma + \Gamma' \vdash \mathbf{let} \ x, y = e \ \mathbf{in} \ P$	$\Gamma + \Gamma' \vdash \mathbf{case} \ e \ \mathbf{of} \ \{i \ x_i \Rightarrow P_i\}_{i=\mathbf{inl}, \mathbf{inr}}$	$\Gamma + \Gamma' \vdash \mathbf{case} \ e \ \mathbf{of} \ \{i \ x_i \Rightarrow P_i\}_{i=\mathbf{inl}, \mathbf{inr}}$	

$\text{dom}(\Gamma_1 + \Gamma_2) = \text{dom}(\Gamma_1) \cup \text{dom}(\Gamma_2)$. Thinking of type environments as of specifications of the resources used by expressions/processes, $\Gamma_1 + \Gamma_2$ expresses the combined use of the resources specified in Γ_1 and Γ_2 . Any resource occurring in only one of these environments occurs in $\Gamma_1 + \Gamma_2$; any resource occurring in both Γ_1 and Γ_2 must be used according to compatible types, and its type in $\Gamma_1 + \Gamma_2$ is their least upper bound. For example, if a channel is used by a process for sending a message of type \mathbf{int} , it has type $[\mathbf{int}]^{0,1}$ in that process; if it is used by another process for receiving a message of type \mathbf{int} , it has type $[\mathbf{int}]^{1,0}$ in that process. Overall, the two processes in parallel use the channel according to the type $[\mathbf{int}]^{0,1} \vee [\mathbf{int}]^{1,0} = [\mathbf{int}]^{1,1}$.

Type rules for expressions and processes are presented in Table 2. These rules are basically the same as those found in the literature [11,7], and the additional flexibility enabled by our typing discipline is actually a consequence of our notion of type combination \vee . The rules for expressions are unremarkable. Just observe that the part of the type environment that is not used in the expression must be unlimited ($\Gamma \succ \Gamma$). The idle process does nothing, so it is well typed only in an unlimited environment. Input and output processes require a strictly positive use of the corresponding operation in the type of the channel u used for communication. Rule [T-REP] states that a replicated process $*P$ is well typed in the environment Γ provided that P is well typed in Γ and that Γ is unlimited. The rationale for this is that $*P$ stands for an unbounded number of copies of P composed in parallel, hence it cannot contain (free) linear channels. The rules [T-PAR], [T-LET], and [T-CASE] are conventional. Finally, rule [T-NEW] states that $(va)P$ is well typed if so is P , where P has visibility of a . We require the type of a to have the same uses for input and output. This is not necessary for the soundness of the type system, although it is a sensible choice in practice.

The type system is sound and the results in Section 4.3 of [11] can be formulated in our setting. In particular, the operations on a channel never exceed the uses in its type. It is possible to establish other basic safety properties, for instance that closed, well-typed \mathbf{let} 's and \mathbf{case} 's always reduce. The long version of the paper gives more details.

Example 2.1. We model a recursive process definition using unlimited channels: a replicated input on the channel represents the definition, while an output on the channel represents an invocation of the definition. For example, for *Odd* and *Even* we have

$$\begin{array}{ll} *a(x).\mathbf{case} \ x \ \mathbf{of} & *b(x).\mathbf{case} \ x \ \mathbf{of} \\ \quad \mathbf{inl} \ y_1 \Rightarrow \mathbf{0} & \quad \mathbf{inl} \ y_1 \Rightarrow \mathbf{0} \\ \quad \mathbf{inr} \ y_2 \Rightarrow \mathbf{let} \ y, z = y_2 \ \mathbf{in} \ \bar{y}\langle 3 \rangle \mid \bar{b}\langle z \rangle & \quad \mathbf{inr} \ y_2 \Rightarrow \mathbf{let} \ y, z = y_2 \ \mathbf{in} \ \bar{a}\langle z \rangle \end{array}$$

and we assume that $\mathbf{inl} \ 0$ represents the empty list $[]$ and $\mathbf{inr} \ (y, z)$ represents the non-empty list $y : z$ with head y and tail z . Below is a derivation showing that *Odd* encoded as shown above is well typed, where we take t_{odd} and t_{even} defined in (1.2):

$$\frac{\frac{\frac{}{y_1 : \mathbf{int} \vdash \mathbf{0}}{\text{[T-IDLE]}} \quad \frac{\frac{\frac{}{y : [\mathbf{int}]^{0,1} \vdash \bar{y}\langle 3 \rangle}}{\text{[T-OUT]}} \quad \frac{\frac{}{b : [t_{even}]^{0,\omega}, z : t_{even} \vdash \bar{b}\langle z \rangle}}{\text{[T-OUT]}}}{\text{[T-PAR]}}}{\frac{b : [t_{even}]^{0,\omega}, y : [\mathbf{int}]^{0,1}, z : t_{even} \vdash \bar{y}\langle 3 \rangle \mid \bar{b}\langle z \rangle}{\text{[T-LET]}}} \quad \frac{}{b : [t_{even}]^{0,\omega}, y_2 : [\mathbf{int}]^{0,1} \times t_{even} \vdash \mathbf{let} \ y, z = y_2 \ \mathbf{in} \ \dots}}{\text{[T-CASE]}}}{\frac{b : [t_{even}]^{0,\omega}, x : t_{odd} \vdash \mathbf{case} \ x \ \mathbf{of} \ \dots}{\text{[T-IN]}}} \quad \frac{}{a : [t_{odd}]^{\omega,0}, b : [t_{even}]^{0,\omega} \vdash a(x).\mathbf{case} \ x \ \mathbf{of} \ \dots}}{\text{[T-REP]}}}{\frac{}{a : [t_{odd}]^{\omega,0}, b : [t_{even}]^{0,\omega} \vdash *a(x).\mathbf{case} \ x \ \mathbf{of} \ \dots}}$$

Note that a and b must be unlimited channels because they occur free in a replicated process, for which rule [T-REP] requires an unlimited environment. A similar derivation shows that *Even* is well typed in an environment where the types of a and b have swapped uses

$$a : [t_{odd}]^{0,\omega}, b : [t_{even}]^{\omega,0} \vdash *b(x).\mathbf{case} \ x \ \mathbf{of} \ \dots$$

so the combined types of a and b are $[t_{odd}]^{\omega,\omega}$ and $[t_{even}]^{\omega,\omega}$ respectively. We conclude

$$a : [t_{odd}]^{\omega,\omega}, b : [t_{even}]^{\omega,\omega}, l : t_{list} \vdash \bar{a}\langle l \rangle \mid \bar{b}\langle l \rangle$$

because $t_{odd} \asymp t_{even}$ and $t_{odd} \vee t_{even} = t_{list}$. ■

3 Constraint Generation

We can formalise the problem of type reconstruction as follows: given a process P , find a type environment Γ such that $\Gamma \vdash P$, provided there is one. In general we also want to maximise the number of linear types in Γ . The rules shown in Table 2 rely on a fair amount of guessing that concerns the structure of types in the type environment, how they are split/combined using \vee , and the uses occurring in them. So, these rules cannot be easily turned into a type reconstruction algorithm. The way we follow to define one is rather conventional: we give an alternative set of syntax-directed rules that compute *constraints* on types and uses and then we search for a solution of such constraints. The novelty is that we need constraints expressing not only the *equality* between types and uses, but also the order \leq and compatibility \asymp , which affect the solution phase in non-trivial ways.

To get started, we generalise uses to *use expressions*, which are either uses or *use variables* ρ, \dots that denote an unknown use. We also define *type expressions* as types without μ 's and where we admit use expressions wherever uses can occur. We keep κ and t for ranging over use and type expressions, respectively, and we say that t is *proper* if it is not a type variable. *Constraints* φ, \dots have one of these forms:

$$\varphi ::= \kappa_1 \mathcal{R}_c \kappa_2 \mid t_1 \mathcal{R}_c t_2$$

where $\mathcal{R} \in \{\leq, <, \succ, \sim\}$ and \sim is the trivial relation such that $\kappa_1 \sim \kappa_2$ holds for all κ_1 and κ_2 . We call $\kappa_1 \mathcal{R}_c \kappa_2$ a *use constraint* and $t_1 \mathcal{R}_c t_2$ a *type constraint*. The subscript \cdot_c reminds us that $\kappa_1 \mathcal{R}_c \kappa_2$ and $t_1 \mathcal{R}_c t_2$ are just *triples* made of two use or type expressions and a *symbol* \mathcal{R}_c denoting a relation. Equality constraints $=_c$ can be expressed as the conjunction of two constraints \leq_c and \geq_c , given that \leq is antisymmetric. Constraints with the strict order $<_c$ will be generated only for use expressions. Compatibility constraints \succ_c will be generated for ensuring type and use combination, as well as for expressing the assumption that a type/use is unlimited (see *e.g.* the premise of [T-IDLE]). Finally, \sim_c constraints relate types that must be *structurally coherent*. For example, $[t]^{\kappa_1, \kappa_2} \sim [t]^{\kappa_3, \kappa_4}$ holds regardless of $\kappa_1, \kappa_2, \kappa_3$, and κ_4 (but note that according to (2.2) there must be the same t in the two types). We will see in Section 4 the role of these constraints.

We let \mathcal{C}, \dots range over finite sets of constraints. The *domain* of \mathcal{C} , written $\text{dom}(\mathcal{C})$, is the (finite) set of use and type expressions occurring in the constraints in \mathcal{C} . We let σ range over finite maps from type variables to types and from use variables to uses. The *application* of σ replaces use variables ρ and type variables α with the corresponding uses $\sigma(\rho)$ and types $\sigma(\alpha)$. We write $\sigma\kappa$ and σt for the application of σ to κ and t , respectively. We say that σ is a *solution* of \mathcal{C} if $\sigma t \mathcal{R} \sigma s$ for every $t \mathcal{R}_c s \in \mathcal{C}$ and $\sigma\kappa_1 \mathcal{R} \sigma\kappa_2$ for every $\kappa_1 \mathcal{R}_c \kappa_2 \in \mathcal{C}$. We extend the \leq relation pointwise to solutions and we say that a solution σ for \mathcal{C} is *minimal* if every solution $\sigma' \leq \sigma$ for \mathcal{C} is such that $\sigma \leq \sigma'$. We say that \mathcal{C} is *satisfiable* if it has a solution. We say that \mathcal{C}_1 and \mathcal{C}_2 are *equivalent* if they have the same solutions.

We need two operators for combining and merging type environments in the reconstruction algorithm. They take two type environments Γ_1 and Γ_2 and produce a pair consisting of another type environment Γ and a set of constraints \mathcal{C} :

$$\frac{\text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset}{\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma_1, \Gamma_2; \emptyset} \quad \frac{\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C} \quad \alpha \text{ fresh}}{(\Gamma_1, u : t) \sqcup (\Gamma_2, u : s) \rightsquigarrow \Gamma, u : \alpha; \mathcal{C} \cup \{t \succ_c s, t \leq_c \alpha, s \leq_c \alpha\}}$$

$$\frac{\emptyset \sqcap \emptyset \rightsquigarrow \emptyset; \emptyset}{\Gamma_1 \sqcap \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}} \quad \frac{\Gamma_1 \sqcap \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}}{(\Gamma_1, u : t) \sqcap (\Gamma_2, u : s) \rightsquigarrow \Gamma, u : t; \mathcal{C} \cup \{t =_c s\}}$$

The relation $\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}$ combines the type environments Γ_1 and Γ_2 into Γ when the names in $\text{dom}(\Gamma_1) \cup \text{dom}(\Gamma_2)$ are used *both* as specified in Γ_1 *and also* in Γ_2 , so \sqcup is analogous to $+$ in (2.3). When Γ_1 and Γ_2 have disjoint domains, their combination is just their union and no constraints are generated. Any name u that occurs in $\text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)$ must be used according to compatible types $\Gamma_1(u) \succ \Gamma_2(u)$ and its type must be an upper bound of both $\Gamma_1(u)$ and $\Gamma_2(u)$. In general $\Gamma_1(u)$ and $\Gamma_2(u)$ are type expressions with free type variables, hence these relations cannot be checked right

Table 3. Constraint generation for expressions and processes

Expressions			
	$\text{[I-INT]} \quad n : \mathbf{int} \triangleright \emptyset; \emptyset$	$\text{[I-NAME]} \quad u : \alpha \triangleright u : \alpha; \emptyset$	
$\text{[I-PAIR]} \quad \frac{e_i : t_i \triangleright \Gamma_i; \mathcal{C}_i \quad (i=1,2) \quad \Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}_3}{(e_1, e_2) : t_1 \times t_2 \triangleright \Gamma; \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$	$\text{[I-INL]} \quad \frac{e : t \triangleright \Gamma; \mathcal{C}}{\mathbf{inl} \ e : t \oplus \alpha \triangleright \Gamma; \mathcal{C}}$	$\text{[I-INR]} \quad \frac{e : t \triangleright \Gamma; \mathcal{C}}{\mathbf{inr} \ e : \alpha \oplus t \triangleright \Gamma; \mathcal{C}}$	
Processes			
$\text{[I-IDLE]} \quad \frac{\mathbf{0} \triangleright \emptyset; \emptyset}{u(x).P \triangleright \Gamma'; \mathcal{C} \cup \mathcal{C}' \cup \{0 <_c \rho\}}$	$\text{[I-IN]} \quad \frac{P \triangleright \Gamma, x : t; \mathcal{C} \quad \Gamma \sqcup u : [t]^{p,0} \rightsquigarrow \Gamma'; \mathcal{C}'}{u(x).P \triangleright \Gamma'; \mathcal{C} \cup \mathcal{C}' \cup \{0 <_c \rho\}}$	$\text{[I-OUT]} \quad \frac{e : t \triangleright \Gamma; \mathcal{C} \quad \Gamma \sqcup u : [t]^{0,p} \rightsquigarrow \Gamma'; \mathcal{C}'}{\bar{u}(e) \triangleright \Gamma'; \mathcal{C}' \cup \{0 <_c \rho\}}$	
$\text{[I-PAR]} \quad \frac{P_1 \triangleright \Gamma_i; \mathcal{C}_i \quad (i=1,2) \quad \Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}_3}{P_1 P_2 \triangleright \Gamma; \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$	$\text{[I-REP]} \quad \frac{P \triangleright \Gamma; \mathcal{C} \quad \Gamma \sqcup \Gamma \rightsquigarrow \Gamma'; \mathcal{C}'}{*P \triangleright \Gamma'; \mathcal{C} \cup \mathcal{C}'}$	$\text{[I-WEAK]} \quad \frac{P \triangleright \Gamma; \mathcal{C}}{P \triangleright \Gamma, u : \alpha; \mathcal{C} \cup \{\alpha \succ_c \alpha\}}$	
$\text{[I-NEW]} \quad \frac{P \triangleright \Gamma, a : t; \mathcal{C}}{(va)P \triangleright \Gamma; \mathcal{C} \cup \{t =_c [\alpha]^{p,p}\}}$	$\text{[I-LET]} \quad \frac{e : t \triangleright \Gamma_1; \mathcal{C}_1 \quad P \triangleright \Gamma_2, x : t_1, y : t_2; \mathcal{C}_2 \quad \Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}_3}{\mathbf{let} \ x, y = e \ \mathbf{in} \ P \triangleright \Gamma; \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \{t =_c t_1 \times t_2\}}$		
$\text{[I-CASE]} \quad \frac{e : t \triangleright \Gamma_1; \mathcal{C}_1 \quad P_i \triangleright \Gamma_i, x_i : t_i; \mathcal{C}_i \quad (i=\mathbf{inl}, \mathbf{inr}) \quad \Gamma_{\mathbf{inl}} \sqcap \Gamma_{\mathbf{inr}} \rightsquigarrow \Gamma_2; \mathcal{C}_2 \quad \Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma_3; \mathcal{C}_3}{\mathbf{case} \ e \ \mathbf{of} \ \{i \ x_i \Rightarrow P_i\}_{i=\mathbf{inl}, \mathbf{inr}} \triangleright \Gamma_3; \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \mathcal{C}_{\mathbf{inl}} \cup \mathcal{C}_{\mathbf{inr}} \cup \{t =_c t_{\mathbf{inl}} \oplus t_{\mathbf{inr}}\}}$			

away. Rather, they are symbolically recorded in the set of constraints \mathcal{C} . Note in particular that the combined type of u is unknown and is represented by a fresh type variable that is an upper bound of $\Gamma_1(u)$ and $\Gamma_2(u)$. The relation $\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}$ merges the type environments Γ_1 and Γ_2 into Γ when the names in $\text{dom}(\Gamma_1) \cup \text{dom}(\Gamma_2)$ are used in alternative branches of a **case** construct. Note that $\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}$ holds if and only if $\Gamma_1(u) = \Gamma_2(u)$ for every $u \in \text{dom}(\Gamma_1) = \text{dom}(\Gamma_2)$. This corresponds to the fact that in [T-CASE] we use the *same* type environment Γ' for typing the two branches of the **case**.

The rules to reconstruct type environments and to generate constraints are presented in Table 3 and derive judgements of the form $e : t \triangleright \Gamma; \mathcal{C}$ for expressions and $P \triangleright \Gamma; \mathcal{C}$ for processes. They closely correspond to those in Table 2; for this and space reasons we will not describe them in detail. In general, unknown uses and types become fresh use and type variables (all variables introduced by the rules are assumed to be fresh), every application of $+$ in Table 2 becomes an application of \sqcup in Table 3, and every assumption on uses and types becomes a constraint. Constraints accumulate from the premises to the conclusion of each rule. In rules [I-INL] and [I-INR] the type of the disjoint sum which was guessed in [T-INL] and [T-INR] becomes a fresh type variable. In rules [I-IN] and [I-OUT] it is not known whether the used channel u is linear or unlimited, so the constraint $0 <_c \rho$ records the fact that ρ must be either 1 or ω . Rule [I-NEW]

requires a to have a channel type with equal uses by having the same use variable ρ twice. There is also a rule [I-WEAK] that has no correspondence in Table 2. It is necessary because [I-IN], [I-NEW], [I-LET], and [I-CASE], which correspond to the binding constructs of the calculus, *assume* that the bound names occur in the premises on these rules. This may not be the case if a bound name is never used. With rule [I-WEAK] we can introduce missing names in type environments wherever is convenient. Of course, an unused name must have a type α that is unlimited, which is recorded by the constraint $\alpha \asymp_c \alpha$. Strictly speaking, with [I-WEAK] this set of rules is not syntax directed, which in principle is a problem if we want to obtain an algorithm. In practice, the places where [I-WEAK] may be necessary are easy to spot (in the premises of all the aforementioned rules for the binding constructs). What we gain with [I-WEAK] is a simpler presentation of the rules for constraint generation.

There is a tight correspondence between the type system and constraint generation. Every satisfiable set of constraints generated from P corresponds to a typing for P .

Theorem 3.1. *If $P \triangleright \Gamma; \mathcal{C}$ and σ is a minimal solution for \mathcal{C} , then $\sigma \Gamma \vdash P$.*

In fact, when $P \triangleright \Gamma; \mathcal{C}$ we can think of $\Gamma; \mathcal{C}$ as the *principal typing* of P , because any type environment Γ' such that $\Gamma' \vdash P$ can be obtained by applying a solution for \mathcal{C} to Γ .

Theorem 3.2. *If $\Gamma' \vdash P$, then $P \triangleright \Gamma; \mathcal{C}$ for some Γ, \mathcal{C} and σ solution of \mathcal{C} and $\Gamma' = \sigma \Gamma$.*

Example 3.1. We show the constraint set generated by two processes accessing the same composite structure containing linear values. The *Odd* and *Even* processes in Section 1 are too large to be discussed in here, so we focus on a simpler, artificial process that exhibits the same phenomenon. We consider

$$a(x).(\text{let } y, z = x \text{ in } \bar{y}\langle 1 \rangle \mid \text{let } y, z = x \text{ in } \bar{z}\langle 2 \rangle)$$

which receives a pair x of channels from a and sends 1 and 2 on them. Note that the pair x is deconstructed twice, but every time only one of its components is used. We obtain

$$\frac{\frac{\frac{}{\bar{y}\langle 1 \rangle \triangleright y : [\text{int}]^{0, \rho_1}; \mathcal{C}_1} \text{[I-OUT]}}{\bar{y}\langle 1 \rangle \triangleright y : [\text{int}]^{0, \rho_1}, z : \gamma; \mathcal{C}_2} \text{[I-WEAK]}}{\text{let } y, z = x \text{ in } \bar{y}\langle 1 \rangle \triangleright x : \alpha_1; \mathcal{C}_3} \text{[I-LET]}} \quad \frac{\frac{\frac{}{\bar{z}\langle 2 \rangle \triangleright z : [\text{int}]^{0, \rho_2}; \mathcal{C}_4} \text{[I-OUT]}}{\bar{z}\langle 2 \rangle \triangleright y : \beta, z : [\text{int}]^{0, \rho_2}; \mathcal{C}_5} \text{[I-WEAK]}}{\text{let } y, z = x \text{ in } \bar{z}\langle 2 \rangle \triangleright x : \alpha_2; \mathcal{C}_6} \text{[I-LET]}}{\text{let } y, z = x \text{ in } \bar{y}\langle 1 \rangle \mid \text{let } y, z = x \text{ in } \bar{z}\langle 2 \rangle \triangleright x : \alpha; \mathcal{C}_7} \text{[I-PAR]}}{\frac{}{a(x).(\text{let } y, z = x \text{ in } \bar{y}\langle 1 \rangle \mid \text{let } y, z = x \text{ in } \bar{z}\langle 2 \rangle) \triangleright a : [\alpha]^{p_3, 0}; \mathcal{C}_8} \text{[I-IN]}}$$

where

$$\begin{aligned} \mathcal{C}_1 &\stackrel{\text{def}}{=} \{0 <_c \rho_1\} & \mathcal{C}_2 &\stackrel{\text{def}}{=} \mathcal{C}_1 \cup \{\gamma \asymp_c \gamma\} & \mathcal{C}_3 &\stackrel{\text{def}}{=} \mathcal{C}_2 \cup \{\alpha_1 =_c [\text{int}]^{0, \rho_1} \times \gamma\} \\ \mathcal{C}_4 &\stackrel{\text{def}}{=} \{0 <_c \rho_2\} & \mathcal{C}_5 &\stackrel{\text{def}}{=} \mathcal{C}_4 \cup \{\beta \asymp_c \beta\} & \mathcal{C}_6 &\stackrel{\text{def}}{=} \mathcal{C}_5 \cup \{\alpha_2 =_c \beta \times [\text{int}]^{0, \rho_2}\} \\ \mathcal{C}_7 &\stackrel{\text{def}}{=} \mathcal{C}_3 \cup \mathcal{C}_6 \cup \{\alpha_1 \asymp_c \alpha_2, \alpha_1 \leq_c \alpha, \alpha_2 \leq_c \alpha\} & \mathcal{C}_8 &\stackrel{\text{def}}{=} \mathcal{C}_7 \cup \{0 <_c \rho_3\} \end{aligned}$$

Within each **let** the variable x is assigned a distinct type variable α_i . Eventually, [I-PAR] finds out that x occurs twice, so it records in \mathcal{C}_7 the fact that the two types α_1 and α_2 must be compatible and that the overall type α of x must be an upper bound of both. \blacksquare

Table 4. Constraint solver algorithm

<p>Input: a set of constraints \mathcal{C}.</p> <p>Output: either fail or a solution of \mathcal{C}.</p> <ol style="list-style-type: none"> 1. Compute $\overline{\mathcal{C}}$; 2. Compute a minimal solution σ_{use} for the use constraints in $\overline{\mathcal{C}}$, or fail if there is none; 3. If $t \sim_c s \in \overline{\mathcal{C}}$ and t, s are proper and have different topmost type constructors, then fail; 4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\overline{\mathcal{C}}, \sigma_{use}}(\{\alpha\}) \mid \alpha \in \text{dom}(\overline{\mathcal{C}})\}$; 5. Return $\sigma_{use} \cup \sigma_{type}$.

4 Constraint Solving

In this section we define an algorithm that determines whether a given set of constraints \mathcal{C} is satisfiable and, if this is the case, computes a solution of \mathcal{C} . The algorithm, sketched in Table 4, comprises 5 steps that can be roughly grouped in three phases: *saturation*, *verification*, and *synthesis*. The phases are detailed in the rest of the section.

Saturation (step 1). The \leq_c and \succ_c constraints determined during constraint generation relate type expressions, but they are meant to affect the use variables occurring in these type expressions (recall from (2.2) that every relation \mathcal{R}_{type} between types is the extension of \mathcal{R}_{use} between uses). In order to find all constraints that must hold between use expressions, we *saturate* the set \mathcal{C} with all the constraints that are entailed by those already in \mathcal{C} . Entailment is expressed through a binary relation \vDash defined as follows:

[E-REFL]	$\{t \mathcal{R}_c s\} \vDash \{t \mathcal{R}_c t, s \mathcal{R}_c s\}$	$\mathcal{R} \in \{\leq, \sim\}$
[E-SYMM]	$\{t \mathcal{R}_c s\} \vDash \{s \mathcal{R}_c t\}$	$\mathcal{R} \in \{=, \sim\}$
[E-TRANS]	$\{t_1 \mathcal{R}_c t_2, t_2 \mathcal{R}_c t_3\} \vDash \{t_1 \mathcal{R}_c t_3\}$	$\mathcal{R} \in \{\leq, \sim\}$
[E-COMP 1]	$\{t_1 =_c t_2, t_2 \succ_c t_3\} \vDash \{t_1 \succ_c t_3\}$	
[E-COMP 2]	$\{t_1 \leq_c t_2, t_2 \succ_c t_3\} \vDash \{t_1 \succ_c t_3\}$	
[E-OPER]	$\{t_1 \odot t_2 \mathcal{R}_c s_1 \odot s_2\} \vDash \{t_1 \mathcal{R}_c s_1, t_2 \mathcal{R}_c s_2\}$	
[E-CHANNEL]	$\{[t]^{K_1, K_2} \mathcal{R}_c [s]^{K_3, K_4}\} \vDash \{t =_c s, K_1 \mathcal{R}_c K_3, K_2 \mathcal{R}_c K_4\}$	
[E-STRUCT]	$\{t \mathcal{R}_c s\} \vDash \{t \sim_c s\}$	

The first three rules [E-REFL], [E-SYMM], and [E-TRANS] compute the reflexive, symmetric, and transitive closures of those relations that enjoy such properties. Rule [E-COMP 1] propagates compatibility constraints across equivalent types, and [E-COMP 2] propagates compatibility constraints “downwards” from a type to a smaller one (indeed, it is the case that $\kappa_1 \leq \kappa_2 \succ \kappa_3$ implies $\kappa_1 \succ \kappa_3$). Rule [E-OPER] propagates constraints between composite types to their components. Rule [E-CHANNEL] propagates constraints from type to use expressions and imposes the equality of message types for related channel types. Finally, rule [E-STRUCT] generates \sim_c constraints between any pair of related type expressions. This is necessary to make sure that all message type equality constraints are generated by [E-CHANNEL], given that \succ is not transitive. We denote by $\overline{\mathcal{C}}$ the smallest set that includes \mathcal{C} and that is closed by the rules [E-*] above. Observe that every \mathcal{C} generated by the rules in Table 3 is finite, and that the entailment rules [E-*] do not

change the domain of the set \mathcal{C} being saturated. Therefore, $\overline{\mathcal{C}}$ is always finite and can be computed in finite time by a simple iterative algorithm that repeatedly applies the entailment rules until no new constraints are discovered. We have:

Proposition 4.1. *\mathcal{C} and $\overline{\mathcal{C}}$ are equivalent.*

Verification (steps 2 and 3). In this phase the algorithm verifies whether $\overline{\mathcal{C}}$ is satisfiable and fails if this is not the case. The key observation is that satisfiability of the type constraints does not depend upon *one particular* solution of the use constraints because the previous phase has computed all possible relations that must hold between use expressions. Therefore, we can independently verify the satisfiability of use and type constraints and fail if any of these checks fails.

Recall that there is a finite number of use constraints, which are relationships between use expressions made of a finite number of use variables ranging over a finite domain $\{0, 1, \omega\}$. Therefore, there exists a complete (albeit combinatorial) verification algorithm that determines whether or not the use constraints in $\overline{\mathcal{C}}$ are satisfiable. It is also possible to define an “optimal” algorithm that aims at maximising the number of use variables that are assigned value 1 as opposed to ω . We do not discuss the issues related to solving use constraints any further.

If the use constraints in $\overline{\mathcal{C}}$ are satisfiable, then satisfiability of the type constraints is granted provided that there are no constraints relating types built with different constructors. For example, $\text{int} \leq_c [\alpha]^{K_1, K_2}$ is clearly unsatisfiable. Because of [E-STRUCT] and [E-TRANS], for any pair of types that must be related there is a constraint $t \sim_c s$ in $\overline{\mathcal{C}}$. Therefore, if there is any such constraint where t and s are not type variables and are built using different topmost constructors, then $\overline{\mathcal{C}}$ is for sure unsatisfiable and the algorithm fails.

Proposition 4.2. *If the algorithm fails in this phase, then $\overline{\mathcal{C}}$ is not satisfiable.*

Synthesis (steps 4 and 5). The last phase computes a solution σ_{type} for the type constraints in $\overline{\mathcal{C}}$ given any minimal solution σ_{use} for the use constraints in $\overline{\mathcal{C}}$ determined at step 2 of the algorithm. To compute σ_{type} we need the definitions below:

$$\begin{aligned} \text{cls}_{\mathcal{R}, \mathcal{C}}(T) &\stackrel{\text{def}}{=} \{s \mid s \mathcal{R}_c t \in \mathcal{C} \text{ for some } t \in T \text{ and } s \text{ is proper}\} \\ \text{sup}_{\mathcal{C}, \sigma}(T) &\stackrel{\text{def}}{=} \begin{cases} [\text{sup}_{\mathcal{C}, \sigma}(\{t_i\}_{i \in I})]^{\forall i \in I \sigma K_i, \forall i \in I \sigma K'_i} & \text{if } \text{cls}_{\leq, \mathcal{C}}(T) = \{[t_i]^{K_i, K'_i}\}_{i \in I} \neq \emptyset \\ \text{sup}_{\mathcal{C}, \sigma}(\{t_i\}_{i \in I}) \odot \text{sup}_{\mathcal{C}, \sigma}(\{s_i\}_{i \in I}) & \text{if } \text{cls}_{\leq, \mathcal{C}}(T) = \{t_i \odot s_i\}_{i \in I} \neq \emptyset \\ \text{zero}_{\mathcal{C}, \sigma}(T) & \text{otherwise} \end{cases} \\ \text{zero}_{\mathcal{C}, \sigma}(T) &\stackrel{\text{def}}{=} \begin{cases} [\text{sup}_{\mathcal{C}, \sigma}(\{t_i\}_{i \in I})]^{0,0} & \text{if } \text{cls}_{\sim, \mathcal{C}}(T) = \{[t_i]^{K_i, K'_i}\}_{i \in I} \neq \emptyset \\ \text{zero}_{\mathcal{C}, \sigma}(\{t_i\}_{i \in I}) \odot \text{zero}_{\mathcal{C}, \sigma}(\{s_i\}_{i \in I}) & \text{if } \text{cls}_{\sim, \mathcal{C}}(T) = \{t_i \odot s_i\}_{i \in I} \neq \emptyset \\ \text{int} & \text{otherwise} \end{cases} \end{aligned}$$

The set $\text{cls}_{\mathcal{R}, \mathcal{C}}(T)$ is made of the proper type expressions s such that $s \mathcal{R}_c t \in \mathcal{C}$ for some $t \in T$. Note that not all \mathcal{R} 's are symmetric and that s is the type expression on the left hand side of \mathcal{R}_c . So, $\text{cls}_{\leq, \mathcal{C}}(T)$ is the set of type expressions that are lower

bounds of some $t \in T$, while $\text{cls}_{\sim, \mathcal{C}}(T)$ is the set of type expressions that share the same topmost type constructor with some $t \in T$ but have possibly different uses. Note also that $\text{cls}_{\leq, \mathcal{C}}(T) \subseteq \text{cls}_{\sim, \mathcal{C}}(T)$ because $\leq \subseteq \sim$. The algorithm (Table 4) resolves each variable α to $\text{sup}_{\overline{\mathcal{C}}, \sigma_{\text{use}}}(\{\alpha\})$ where, $\text{sup}_{\mathcal{C}, \sigma}(T)$ is, roughly speaking, the least upper bound of the types in T (even though the algorithm always invokes $\text{sup}_{\mathcal{C}, \sigma}$ with a singleton, in general we need to define $\text{sup}_{\mathcal{C}, \sigma}$ over a set of type expressions that are known to be equivalent). There are three cases that determine $\text{sup}_{\mathcal{C}, \sigma}(T)$: if there exists any lower bound for some of the types in T and these lower bounds are either channel or composite types, then $\text{sup}_{\mathcal{C}, \sigma}(T)$ is defined as the least upper bound of such lower bounds (first two cases in the definition of $\text{sup}_{\mathcal{C}, \sigma}$); if there is no lower bound but there exists at least one \sim_c constraint involving any of the types in T , then $\text{sup}_{\mathcal{C}, \sigma}(T)$ is defined as a type that is structurally coherent with such constraints but has use 0 for all of its topmost channel types (third case in the definition of $\text{sup}_{\mathcal{C}, \sigma}$ and first two cases in the definition of $\text{zero}_{\mathcal{C}, \sigma}$); if there are no \sim_c constraints involving any of the types in T , or if some of the types in T have been determined to be structurally coherent with **int**, then $\text{zero}_{\mathcal{C}, \sigma}(T)$ is defined to be **int** (third case in the definition of $\text{zero}_{\mathcal{C}, \sigma}$).

Interpreting $\text{sup}_{\mathcal{C}, \sigma}$ and $\text{zero}_{\mathcal{C}, \sigma}$ as functions is appropriate for presentation (and implementation) purposes, but formally tricky for two reasons: **(1)** the equations given above are mutually dependent and **(2)** they are undefined for some particular T 's (for instance, for $T = \{\mathbf{int}, [\mathbf{int}]^{0,0}\}$ which contains two types with incompatible structures or for $T = \{[\mathbf{int}]^{0,0}, [\mathbf{int}]^{1,0}\}$ which contains two types with incompatible uses). Concerning **(1)**, the formal interpretation of the equations above is as a set $\{\alpha_i = t_i\}$ where each α_i has the form $\text{sup}_{\mathcal{C}, \sigma}(T)$ or $\text{zero}_{\mathcal{C}, \sigma}(T)$, $T \subseteq \text{dom}(\mathcal{C})$, and t_i is determined by the right hand side of the equation. We know that this set is always finite because $\text{dom}(\mathcal{C})$ is finite and so is its powerset. Furthermore, $\text{zero}_{\mathcal{C}, \sigma}$ always yields a proper type when it is defined and so does $\text{sup}_{\mathcal{C}, \sigma}$ when it is not defined in terms of $\text{zero}_{\mathcal{C}, \sigma}$. Therefore, the equations in $\{\alpha_i = t_i\}$ are contractive in the sense that there is no infinite chain of equations involving type variables only. In this case, it is known [1] that these equations can be folded into a possibly recursive contractive term using μ 's. Concerning **(2)**, it turns out that, when σ is a solution of the use constraints in \mathcal{C} , $\text{sup}_{\mathcal{C}, \sigma}(T)$ is defined if T is \mathcal{C} -composable and that $\text{zero}_{\mathcal{C}, \sigma}(T)$ is defined if T is \mathcal{C} -compatible, where:

- T is \mathcal{C} -composable if $t \leq_c s \in \mathcal{C}$ or $s \leq_c t \in \mathcal{C}$ or $t \succ_c s \in \mathcal{C}$ for every $t, s \in T$;
- T is \mathcal{C} -compatible if $t \sim_c s \in \mathcal{C}$ for every $t, s \in T$.

Indeed observe that: $\text{cls}_{\leq, \mathcal{C}}(T)$ is \mathcal{C} -composable if so is T by [E-COMP 2]; $\text{cls}_{\sim, \mathcal{C}}(T)$ is \mathcal{C} -compatible if T is \mathcal{C} -composable by [E-STRUCT]; if $\{[t_i]^{k_i, k'_i}\}_{i \in I}$ is \mathcal{C} -composable then the least upper bounds $\bigvee_{i \in I} \sigma k_i$ and $\bigvee_{i \in I} \sigma k'_i$ are defined (consequence of the use constraints generated by [E-CHANNEL] and the hypothesis that σ is a solution of them); if $\{[t_i]^{k_i, k'_i}\}_{i \in I}$ is \mathcal{C} -compatible, then $\{t_i\}_{i \in I}$ is \mathcal{C} -composable (consequence of the $=_c$ constraints generated by [E-CHANNEL]); if $\{t_i \odot s_i\}_{i \in I}$ is \mathcal{C} -composable/compatible, then so are the sets $\{t_i\}_{i \in I}$ and $\{s_i\}_{i \in I}$ by [E-OPER]; the type expressions in \mathcal{C} -composable/compatible sets are built using the same topmost constructor (check in step 3 of the algorithm). Finally, observe that a singleton $\{\alpha\}$ is always \mathcal{C} -composable, so the invocations of $\text{sup}_{\overline{\mathcal{C}}, \sigma_{\text{use}}}$ in Table 4 regard well-defined equations. We conclude:

Theorem 4.1. *If the algorithm returns σ , then σ is a minimal solution for \mathcal{C} .*

Each step of the algorithm terminates and if the algorithm fails it is because \mathcal{C} has no solution (Proposition 4.2). Therefore:

Corollary 4.1 (completeness). *If \mathcal{C} is satisfiable, the algorithm returns a solution.*

Example 4.1. The saturation of the constraint set \mathcal{C} computed in Example 3.1 contains, among others, the constraints $[\mathbf{int}]^{0,\rho_1} \times \gamma \succ_c \beta \times [\mathbf{int}]^{0,\rho_2}$ and consequently $[\mathbf{int}]^{0,\rho_1} \succ_c \beta$ and $\gamma \succ_c [\mathbf{int}]^{0,\rho_2}$ by [E-COMP 1] and [E-COMP 2]. An optimal solution of the use constraints in $\overline{\mathcal{C}}$ is $\sigma_{use} \stackrel{\text{def}}{=} \{\rho_1 \mapsto 1, \rho_2 \mapsto 1, \rho_3 \mapsto 1\}$. From this we obtain

$$\sup_{\overline{\mathcal{C}}, \sigma_{use}}(\{\alpha\}) = [\mathbf{int}]^{0,1} \times [\mathbf{int}]^{0,1}$$

indicating that the pair of channels received from a is shared by the two `let` processes in such a way that each of the two channels contained therein is used exactly once. ■

5 Concluding Remarks

Previous works on the linear π -calculus either ignore composite types [11,7] or are based on an interpretation of linearity that limits data sharing and parallelism [5,6]. Recursive types have also been neglected, despite their prominent role for describing complex interactions occurring on linear channels [2]. In this work we extend the linear π -calculus with both composite and recursive types and we adopt a more relaxed attitude towards linearity that fosters data sharing and parallelism while preserving complete type reconstruction. The extension is a very natural one, as witnessed by the fact that our type system uses essentially the same rules of previous works, the main novelty being a different type composition operator. This small change has nonetheless non-trivial consequences on the reconstruction algorithm, which must reconcile the propagation of constraints across composite types with the impossibility to rely on plain type unification due to the fact that different occurrences of the same identifier may be assigned different types and because of recursive types. Technically, we tackle these problems by expressing type combination in previous works (which is a ternary relation $t_1 + t_2 = t_3$) in terms of two simpler binary relations, namely compatibility $t_1 \succ t_2$ and order $t_i \leq t_3$, and by seeking for *minimal* solutions of the constraint set. Our extension also gives renewed relevance to types like $[t]^{0,0}$. In previous works these types were admitted but essentially useless: channels with such types could only be passed around in messages without actually ever being used. That is, they could be erased without affecting processes. In our type system, it is the existence of these types that enables the sharing of structured data (see the decomposition of t_{list} into t_{even} and t_{odd} in Section 1).

Given that sessions [3,4] can be fully encoded into the linear π -calculus [9,2], we also indirectly provide a complete reconstruction algorithm for equi-recursive session types without subtyping. Interestingly, the concept of *duality*, which is a source of significant complication of the type reconstruction algorithm for *finite* session types [12], is simplified by the encoding, where it reduces to compatibility.

To assess the feasibility of the approach, we have developed a prototype based on the naïve constraint saturation and combinatorial verification of use expressions described in Section 4. Regarding the complexity of the reconstruction algorithm, the most critical

aspects are the solution of use constraints and the computation of the transitive closure of type constraints. While polynomial algorithms are known for the latter, the former problem entails, in principle, an exponential cost. Preliminary experiments with the prototype implementation of the algorithm have shown that, in both cases, constraints can often be partitioned into relatively small independent subsets that can be solved in isolation (the prototype already supports such partitioning for use constraints). This property paves the way for significant performance improvements.

Acknowledgements. The author is grateful to the FoSSaCS'14 reviewers for their detailed and insightful comments. This work has been partially supported by ICT COST Action IC1201 BETTY, MIUR project CINA, and Ateneo/CSP project SALT.

References

1. Courcelle, B.: Fundamental properties of infinite trees. *Theor. Comp. Sci.* 25, 95–169 (1983)
2. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: *PPDP 2012*, pp. 139–150. ACM (2012)
3. Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) *CONCUR 1993*. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993)
4. 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)
5. Igarashi, A.: Type-based analysis of usage of values for concurrent programming languages (1997), <http://www.sato.kuis.kyoto-u.ac.jp/~igarashi/papers/>
6. Igarashi, A., Kobayashi, N.: Type-based analysis of communication for concurrent programming languages. In: Van Hentenryck, P. (ed.) *SAS 1997*. LNCS, vol. 1302, pp. 187–201. Springer, Heidelberg (1997)
7. Igarashi, A., Kobayashi, N.: Type Reconstruction for Linear π -Calculus with I/O Subtyping. *Inf. and Comp.* 161(1), 1–44 (2000)
8. Kobayashi, N.: Quasi-linear types. In: *POPL 1999*, pp. 29–42. ACM (1999)
9. Kobayashi, N.: Type systems for concurrent programs. In: Aichernig, B.K. (ed.) *Formal Methods at the Crossroads. From Panacea to Foundational Support*. LNCS, vol. 2757, pp. 439–453. Springer, Heidelberg (2003), Extended version at <http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf>
10. Kobayashi, N.: A new type system for deadlock-free processes. In: Baier, C., Hermanns, H. (eds.) *CONCUR 2006*. LNCS, vol. 4137, pp. 233–247. Springer, Heidelberg (2006)
11. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.* 21(5), 914–947 (1999)
12. Mezzina, L.G.: How to infer finite session types in a calculus of services and sessions. In: Lea, D., Zavattaro, G. (eds.) *COORDINATION 2008*. LNCS, vol. 5052, pp. 216–231. Springer, Heidelberg (2008)
13. Nestmann, U., Steffen, M.: Typing confluence. In: *FMICS 1997*, pp. 77–101 (1997), Also available as report ERCIM-10/97-R052, European Research Consortium for Informatics and Mathematics (1997)
14. Sangiorgi, D., Walker, D.: *The Pi-Calculus - A theory of mobile processes*. Cambridge University Press (2001)
15. Turner, D.N., Wadler, P., Mossin, C.: Once upon a type. In: *FPCA 1995*, pp. 1–11 (1995)

A Semantical and Operational Account of Call-by-Value Solvability

Alberto Carraro^{1,2} and Giulio Guerrieri²

¹ DAIS, Università Ca' Foscari Venezia
alberto.carraro@unive.it

² PPS, Univ Paris Diderot, Sorbonne Paris Cité
giulio.guerrieri@pps.univ-paris-diderot.fr

Abstract. In Plotkin's call-by-value lambda-calculus, solvable terms are characterized syntactically by means of call-by-name reductions and there is no neat semantical characterization of such terms. Preserving confluence, we extend Plotkin's original reduction without adding extra syntactical constructors, and we get a call-by-value operational characterization of solvable terms. Moreover, we give a semantical characterization of solvable terms in a relational model, based on Linear Logic, satisfying the Taylor expansion formula. As a technical tool, we also use a resource-sensitive calculus (with tests) in which the elements of the model are definable.

Keywords: (resource) call-by-value lambda-calculus, tests, potential valuability, solvability, relational semantics, weak and stratified reductions.

1 Introduction

In the theory of ordinary (i.e. untyped call-by-name) λ -calculus, the notion of solvability plays a crucial role. A λ -term M is *solvable* if there is a *head context* H such that $H(M) \rightarrow_{\beta} \lambda x.x = \mathbf{I}$ (the identity); M is *unsolvable* if it is not solvable. Solvability (see [1]) underlies the fundamental notions of approximants, Böhm-trees and separability; moreover, it is possible to encode partial recursive functions in λ -calculus in such a way that undefinedness is represented by unsolvable λ -terms ([1, Ch. 8]). Enforcing the idea of unsolvable-as-meaningless, it is consistent to equate all unsolvable λ -terms (but not all λ -terms having no β -normal form, [1, Ch. 16]). A fundamental theorem for ordinary λ -calculus (see [2,3]) states that for every λ -term M the following are equivalent: (1) M is solvable; (2) the head reduction of M terminates; (3) the semantics of M in the Scott's model D_{∞} is not the least element. Equivalence (1) \Leftrightarrow (2) (resp. (1) \Leftrightarrow (3)) gives a *semantical* (resp. *syntactical* or *operational*) characterization of solvability in ordinary λ -calculus.

The most common parameter passing policy for programming languages is call-by-value (CBV). Plotkin [4] introduced the λ_v -calculus in order to grasp the CBV paradigm in a pure λ -calculus setting. The λ_v -calculus (without constants)

has the same syntax as ordinary λ -calculus but its β_v -reduction rule allows the contraction of a β -redex only if the argument is a λ -value, i.e. a variable or an abstraction. As argued in [5], a good CBV λ -calculus should enjoy an *internal* operational characterization (i.e. by using CBV reduction rules) of CBV-solvability. This is not the case for Plotkin's λ_v -calculus and the weakness of β_v -reduction is widely recognized and accepted. Following [6,7], a λ -term M is λ_v -solvable if there is a head context H such that $H(M) \rightarrow_{\beta_v} \mathbf{I}$. Let $\Delta = \lambda x.xx$: there is no head context sending (via β_v -reduction) $N = (\lambda y.\Delta)(x\mathbf{I})\Delta$ to \mathbf{I} , thus N is λ_v -unsolvable and hence it should be divergent, whereas it is β_v -normal. An operational characterization of λ_v -solvability has been provided in [6,7] but through a *call-by-name* reduction; this result is improved in [8] where the characterization is built upon strong normalization of the (call-by-name) lazy β -reduction.

There are many proposals of alternative CBV λ -calculi (see [9,10,11,12,5]) extending Plotkin's one by using explicit substitutions (constructors of the form `let . . . in`). In particular, Accattoli and Paolini [5] introduced recently the λ_{vsub} -calculus where the reduction rule acts at a distance by extending the notion of β_v -redex (with explicit substitutions). In this setting they give an internal operational characterization of solvability and this characterization lifts to Herbelin and Zimmermann's λ_{CBV} -calculus, another CBV λ -calculus with explicit substitutions introduced in [9] (without rules acting at a distance but with commutation rules for explicit substitutions).

Paolini and Ronchi Della Rocca [6,7] made major contributions to the study of CBV-solvability through denotational semantics. In [6] they showed an intersection type system that characterizes λ_v -potentially valuable¹ (Thm. 6.4) and λ_v -solvable λ -terms (Thm. 6.5). We quote from [6, p. 28]: “The type assignment system presented here is strongly related to the system presented in [13] for reasoning on the denotational semantics of the [Plotkin's] λ_v -calculus. [...] The two systems have the same typability power”. It is not shown whether this type system is “legal” (see [7, Def. 10.1.5]), which is substantially a sufficient condition to turn the type system into a *filter model* (i.e. a true domain model). In [7, Ch. 12] the same authors exhibit two models, \mathcal{V} (§ 12.1) and \mathcal{VV} (§ 12.2), both built from intersection type systems. The model \mathcal{V} comes from a legal type system and it is shown to be isomorphic to the one of [13]. All and only λ_v -potentially valuable λ -terms have non trivial interpretation in \mathcal{V} , but \mathcal{V} gives only a *partial* semantical characterization of λ_v -solvable λ -terms (Thm. 12.1.19). The model \mathcal{VV} characterizes observational equivalence (Thm. 12.2.14) but it is not a filter model. Recently, Ehrhard [14] used a relational model of the λ_v -calculus, based on Linear Logic, to show that if the semantics of a λ -term M is not empty, then M is strongly normalizing for the lazy β_v -reduction (which does not reduce under abstractions); the converse is false (the aforesaid λ -term N is a counterexample).

¹ Following [6,7], a λ -term is λ_v -potentially valuable if there is a substitution sending it (via \rightarrow_{β_v}) into a λ -value. This notion is important for a CBV λ -calculus because if we want to manipulate some subterms, we need first to transform them into λ -values.

The starting points of our work are [6,5,14]. We introduce the λ_v^σ -calculus, a CBV λ -calculus having the same syntax as ordinary (and hence Plotkin’s CBV) λ -calculus (there are no explicit substitutions) and extending the β_v -reduction by adding two reduction rules, σ_1 and σ_3 . For the λ_v^σ -calculus we give a semantical and an internal operational characterization of solvability and potential valuability. We use the relational model of [14], which can also be seen as a model of ordinary λ -calculus (unlike the model \mathcal{V} of [7]) and satisfies a version of the Taylor formula (see [14]). We also introduce a resource-sensitive calculus with tests in which the elements of the relational model are definable: this is a promising tool to face the CBV full abstraction problem, along the lines of [15].

Our λ_v^σ -calculus springs from Girard’s call-by-value “boring” translation $(\cdot)^v$ of λ -calculus into Intuitionistic Multiplicative Exponential Linear Logic (IMELL) proof-nets, identified by $(A \Rightarrow B)^v = !A^v \multimap !B^v$ (see [16]). The images of a σ_1 - or σ_3 -redex and its contractum under $(\cdot)^v$ are equal modulo some specified “immediate” steps of cut-elimination. Our σ -rules are related to (but partly different from) Regnier’s σ -reduction defined in [17,18] for the ordinary λ -calculus. Moreover, σ_1 and σ_3 correspond respectively to the commutation rules let_{app} and (a generalization of) let_{let} in λ_{CBV} -calculus (see [9,5]). In some sense, they can be seen as a finer (and local) decomposition of the reduction rules acting at a distance in λ_{vsub} -calculus (it is possible to simulate λ_{vsub} - and λ_{CBV} -calculus in our λ_v^σ -calculus), but the absence of explicit substitutions in λ_v^σ -calculus prevents from lifting the internal operational characterization of CBV-solvability from λ_{vsub} - or λ_{CBV} -calculus to our λ_v^σ -calculus.

Outline. In §2 we introduce our λ_v^σ -calculus. Then, §3, §4 and §5 are devoted to the technical notions which are necessary in order to state our main results: in §3 we present two sub-reductions in the λ_v^σ -calculus, called **w**- and **s**-reduction; in §4 and §5 we present a resource-sensitive version of the λ_v^σ -calculus and the relational model of the (resource) λ_v^σ -calculus. In §6 we state and prove our main theorems: the semantical (via the relational model) and syntactical (via **w**- and **s**-reductions) characterization of potential valuability and solvability; they say also that weak and strong normalizations coincide for both **w**- and **s**-reductions.

2 A CBV Lambda-Calculus with Sigma-Like-Reductions

In this section we introduce λ_v^σ , our version of CBV λ -calculus. The syntax of λ_v^σ is the same as the one of ordinary λ -calculus. Given a countable set of *variables* (denoted by x, y, z, \dots), the language of λ_v^σ is defined by the following grammar:

$$\begin{array}{ll} (A^v) & V, U ::= x \mid \lambda x.M \quad \lambda\text{-values} \\ (A) & M, N, L ::= V \mid MN \quad \lambda\text{-terms} \end{array}$$

All λ -terms are considered up to α -conversion. The set of free variables of a λ -term M is denoted by $\text{fv}(M)$. Given pairwise distinct variables x_1, \dots, x_n , we denote by $M\{V_1/x_1, \dots, V_n/x_n\}$ the λ -term obtained by the *capture-avoiding*

simultaneous substitution of each free occurrence of x_i in the λ -term M by the λ -value V_i (for $1 \leq i \leq n$). Notice that, for all λ -values V, V_1, \dots, V_n and pairwise distinct variables x_1, \dots, x_n , $V\{V_1/x_1, \dots, V_n/x_n\}$ is a λ -value.

Contexts (with exactly one hole) are defined as usual via the grammar:

$$\mathbf{C} ::= (\cdot) \mid \lambda x. \mathbf{C} \mid \mathbf{C}M \mid M\mathbf{C}.$$

We use $\mathbf{C}(M)$ for the λ -term obtained by the capture-allowing substitution of the λ -term M for (\cdot) in the context \mathbf{C} .

Definition 1. We define the following binary relations from Λ to Λ :

$$\begin{aligned} (\lambda x.M)V &\mapsto_{\beta_v} M\{V/x\} && \text{with } V \in \Lambda^v \\ (\lambda x.M)NL &\mapsto_{\sigma_1} (\lambda x.ML)N && \text{with } x \notin \text{fv}(L) \\ V((\lambda x.L)N) &\mapsto_{\sigma_3} (\lambda x.VL)N && \text{with } x \notin \text{fv}(V) \text{ and } V \in \Lambda^v \end{aligned}$$

For $R \in \{\beta_v, \sigma_1, \sigma_3\}$, if $M \mapsto_R M'$ then M is called R -redex.

We set $\mapsto_\sigma = \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3}$ and $\mapsto_v = \mapsto_{\beta_v} \cup \mapsto_\sigma$.

The side conditions on \mapsto_σ in Def. 1 can be always fulfilled by α -renaming.

Notation. Let $\mapsto_R \subseteq \Lambda \times \Lambda$. We use \rightarrow_R (called R -reduction) for the closure of \mapsto_R under all contexts; we denote by \rightarrow_R (resp. \rightarrow_R^+) the reflexive-transitive (resp. transitive) closure of \rightarrow_R . Let M be a λ -term: M is R -normal if there is no λ -term N such that $M \rightarrow_R N$; M is R -normalizable if there is a R -normal λ -term N such that $M \rightarrow_R N$; M is *strongly* R -normalizing if there is no sequence $(N_i)_{i \in \mathbf{N}}$ such that $M = N_0$ and $N_i \rightarrow_R N_{i+1}$ for every $i \in \mathbf{N}$.

Notice that, for any λ -value V , if $V \rightarrow_v M$, then M is a λ -value.

The λ_v^σ -calculus is the set Λ of λ -terms endowed with the v -reduction \rightarrow_v . The set Λ endowed with \rightarrow_{β_v} is Plotkin's CBV λ -calculus ([4]) without constants.

Informally, σ -rules unblock β_v -redexes which are hidden by the ‘‘hyper-sequential structure’’ of λ -terms. This approach is alternative to the one in [5] where hidden β_v -redexes are reduced thanks to a rule acting at a distance.

Example. $N = (\lambda y. \Delta)(x\mathbf{I})\Delta \rightarrow_{\sigma_1} (\lambda y. \Delta\Delta)(x\mathbf{I}) \rightarrow_{\beta_v} (\lambda y. \Delta\Delta)(x\mathbf{I}) \rightarrow_{\beta_v} \dots$ is the only possible v -reduction path from N : N is not v -normalizable but β_v -normal.

2.1 Confluence of Our CBV Lambda-Calculus

Our goal here is to prove that the v -reduction is confluent.

Proposition 2. *The reduction \rightarrow_σ is strongly normalizing.*

Proof. First, we define two sizes $\mathfrak{s}(M)$ and $\#M$ by induction on the λ -term M :

$$\begin{aligned} \mathfrak{s}(x) &= 2; & \#x &= 1; \\ \mathfrak{s}(\lambda x.M) &= \mathfrak{s}(M) + 1; & \#\lambda x.M &= \#M + \mathfrak{s}(M); \\ \mathfrak{s}(MN) &= \mathfrak{s}(M) + \mathfrak{s}(N). & \#MN &= \#M + \#N + 2\mathfrak{s}(M)\mathfrak{s}(N) - 1. \end{aligned}$$

It is sufficient to show that if $N \rightarrow_\sigma N'$ then $\mathfrak{s}(N) = \mathfrak{s}(N')$ and $\#N > \#N'$. \square

Proposition 3. *The reduction \rightarrow_σ is (not strongly) confluent.*

Proof. By Newman’s Lemma and Prop. 2, it is sufficient to show that \rightarrow_σ is locally confluent. The proof of local confluence is by induction on M . The λ -term $\Xi = (\lambda x.x')((\lambda y.y'\mathbf{I})(z\mathbf{I}))(z'\mathbf{I})$ is an objection to strong confluence of \rightarrow_σ . \square

Lemma 4 (Hindley–Rosen, [1, p. 64]). *Let $\rightarrow_1, \rightarrow_2 \subseteq X^2$ (for any set X). If they are both confluent and they commute, i.e. if $t \rightarrow_1 u_1$ and $t \rightarrow_2 u_2$ then there exists s such that $u_1 \rightarrow_2 s$ and $u_2 \rightarrow_1 s$, then $\rightarrow_1 \cup \rightarrow_2$ is confluent.*

Lemma 5. *Let $M, M' \in \Lambda$, $V, V', V_1, \dots, V_m \in \Lambda^v$ and $R \in \{\beta_v, \sigma, v\}$.*

- (i) *If $V \rightarrow_R V'$ then $M\{V/x\} \rightarrow_R M\{V'/x\}$.*
- (ii) *If $M \rightarrow_R M'$ then $M\{V_1/x_1, \dots, V_m/x_m\} \rightarrow_R M'\{V_1/x_1, \dots, V_m/x_m\}$.*

Lemma 6. *The reductions \rightarrow_{β_v} and \rightarrow_σ commute.*

Proof. It suffices to prove that if $M \rightarrow_\sigma N_1$ and $M \rightarrow_{\beta_v} N_2$ then there is L s.t. $N_2 \rightarrow_\sigma L$ and $N_1 \rightarrow_{\beta_v} L$. The proof of this statement is by induction on M . \square

By Lemmas 4 and 6, Prop. 3 and confluence of \rightarrow_{β_v} (see [4]), we conclude:

Theorem 7. *The reduction \rightarrow_v is (not strongly) confluent.*

The λ -term Ξ (see proof of Prop. 3) is an objection to strong confluence of \rightarrow_v .

If in the definition of \mapsto_{σ_3} (Def. 1) we replace the λ -value V with any λ -term M then \rightarrow_σ and \rightarrow_v are not (locally) confluent: consider $(\lambda x.x')(z\mathbf{I})((\lambda y.y')(z'\mathbf{I}))$.

3 Weak and Stratified CBV Reductions

In this section we introduce two sub-reductions of \rightarrow_v : *weak* (or **w**-) *reduction* and *stratified* (or **s**-) *reduction*. We will show in §6 that they give an operational characterization of potential valuability and solvability: they are the “CBV counterpart” of head reduction for ordinary λ -calculus. Whereas head reduction is strictly deterministic (any λ -term has at most one head redex), a λ -term might have several (overlapping) **w**- or **s**-redexes. Anyway, both **w**- and **s**-reductions are confluent (Prop. 10) and for them weak and strong normalization coincide (Thm. 24 and 25). We have gathered our definition of **w**- and **s**-reductions from [5].

Definition 8. *Weak and stratified contexts (denoted respectively by **W** and **S**) are contexts defined via the grammar:*

$$\mathbf{W} ::= (\cdot) \mid \mathbf{W}M \mid M\mathbf{W} \mid (\lambda x.\mathbf{W})M \qquad \mathbf{S} ::= \mathbf{W} \mid \lambda x.\mathbf{S} \mid SM$$

Notation. Let $\mapsto_R \subseteq \Lambda \times \Lambda$: we use $\rightarrow_{\mathbf{w}[R]}$ (resp. $\rightarrow_{\mathbf{s}[R]}$) for the closure under weak (resp. stratified) contexts of \mapsto_R . We set $\mathbf{w} = \mathbf{w}[v]$ and $\mathbf{s} = \mathbf{s}[v]$; for instance, $\rightarrow_{\mathbf{w}} = \rightarrow_{\mathbf{w}[v]}$ (called **w**-reduction) and $\rightarrow_{\mathbf{s}} = \rightarrow_{\mathbf{s}[v]}$ (called **s**-reduction).

Note that $\rightarrow_w \subsetneq \rightarrow_s \subsetneq \rightarrow_v$. In weak contexts, if the hole is under an abstraction then this abstraction is the left-hand side of an application. Stratified contexts never contain the hole under an abstraction which is in the right-hand side of some application, unless the abstraction is the left-hand side of an application.

Example. Let $\Omega = \Delta\Delta$: one has $\Omega \rightarrow_w \Omega \rightarrow_w \dots$, $\lambda y.\Omega \rightarrow_s \lambda y.\Omega \rightarrow_s \dots$, and $x(\lambda y.\Omega) \rightarrow_v x(\lambda y.\Omega) \rightarrow_v \dots$, whereas $\lambda y.\Omega$ (resp. $x(\lambda y.\Omega)$) is **w**- (resp. **s**-)normal.

We will now prove that the **w**- and **s**-reductions are confluent.

- Lemma 9.** (i) *The reductions $\rightarrow_{w[\beta_v]}$ and $\rightarrow_{s[\beta_v]}$ are strongly confluent.*
(ii) *The reductions $\rightarrow_{w[\sigma]}$ and $\rightarrow_{s[\sigma]}$ are confluent.*
(iii) *The reductions $\rightarrow_{w[\beta_v]}$ and $\rightarrow_{w[\sigma]}$ (resp. $\rightarrow_{s[\beta_v]}$ and $\rightarrow_{s[\sigma]}$) commute.*

By Lemmas 4 and 9 we can conclude:

Proposition 10. *The reductions \rightarrow_w and \rightarrow_s are (not strongly) confluent.*

The λ -term Ξ (see p. 107) is an objection to strong confluence of \rightarrow_w and \rightarrow_s .

3.1 Characterization of **w**- and **s**-Normal Forms

Our goal here is to characterize **w**- and **s**-normal forms. Having no explicit substitutions, our characterization appears more concise than the one in [5].

Definition 11. *We define the subsets \mathbf{a}_{nf} , \mathbf{s}_{nf} and \mathbf{w}_{nf} of Λ as follows:*

$$\begin{aligned} (\mathbf{a}_{\text{nf}}) \quad A_{\text{nf}} &::= xV \mid xA_{\text{nf}} \mid A_{\text{nf}}W_{\text{nf}} \\ (\mathbf{w}_{\text{nf}}) \quad W_{\text{nf}} &::= V \mid (\lambda x.W_{\text{nf}})A_{\text{nf}} \mid A_{\text{nf}} \\ (\mathbf{s}_{\text{nf}}) \quad S_{\text{nf}} &::= x \mid \lambda x.S_{\text{nf}} \mid (\lambda x.S_{\text{nf}})A_{\text{nf}} \mid A_{\text{nf}} \end{aligned}$$

A β -redex is a λ -term of shape $(\lambda x.M)L$. Notice that $\mathbf{a}_{\text{nf}} \subsetneq \mathbf{s}_{\text{nf}} \subsetneq \mathbf{w}_{\text{nf}}$ and if $N \in \mathbf{a}_{\text{nf}}$ then N has a free “head variable” and it is neither a value nor a β -redex.

Proposition 12. *Let M be a λ -term.*

- (i) *M is **w**-normal iff $M \in \mathbf{w}_{\text{nf}}$.*
(ii) *M is **s**-normal iff $M \in \mathbf{s}_{\text{nf}}$.*
(iii) *M is **w**- (resp. **s**-)normal and is neither a value nor a β -redex iff $M \in \mathbf{a}_{\text{nf}}$.*

4 A Resource CBV Lambda-Calculus

We now introduce the *resource λ_v^σ -calculus*, a valuable tool to prove some parts of our main results. It is an extension of the resource CBV λ -calculus introduced in [14, §5.2]. Its syntax is defined by the following grammar (the same as in [14]):

$$\begin{aligned} (rA^v) \quad u, v &::= x \mid \lambda x.t && \text{resource values} \\ (rA^t) \quad s, t &::= st \mid [v_1, \dots, v_k] \quad (k \geq 0) && \text{resource terms} \\ (rA) \quad e, f &::= v \mid t && \text{expressions} \end{aligned}$$

A resource term like $[v_1, \dots, v_k]$ is a multiset of resource values (called *bag*).

The resource-version of the β_v -rule makes use of *linear substitution*, which requires to enrich the syntax of the calculus with finite sets of resource terms.

Notation. Since the set $\mathcal{P}_f(A)$ of all finite subsets of a set A is the free module $\mathbf{2}\langle A \rangle$ generated by A over the boolean semiring $\{0, 1\}$ with $1 + 1 = 1$, we will use algebraic notations for operations on its elements ($+$ for set unions, 0 for the empty set), as done in [15,14].

We denote by $\text{deg}_x(e)$ the number of free occurrences of the variable x in the expression e . Given $e \in rA$, $v_1, \dots, v_k \in rA^v$ and an enumeration of the free occurrences of variable x in e , if $\text{deg}_x(e) = k$ then by $\sum_{f \in \mathfrak{S}_k} e\{v_{f(1)}/x^1, \dots, v_{f(k)}/x^k\}$ we mean the sum of all expressions obtained by substituting $v_{f(i)}$ for the i -th free occurrence of x in e , as f varies over all elements of the set \mathfrak{S}_k of permutations of $\{1, \dots, k\}$. Finally, the linear substitution of $[v_1, \dots, v_k]$ for x in e is

$$e\langle [v_1, \dots, v_k] / x \rangle = \begin{cases} \sum_{f \in \mathfrak{S}_k} e\{v_{f(1)}/x^1, \dots, v_{f(k)}/x^k\} & \text{if } \text{deg}_x(e) = k \\ 0 & \text{otherwise} \end{cases}$$

Notice that, for $\mathbf{n} \in \{\mathbf{v}, \mathbf{t}\}$, if $e \in rA^{\mathbf{n}}$ then $e\langle [v_1, \dots, v_k] / x \rangle \in \mathbf{2}\langle rA^{\mathbf{n}} \rangle$.

Resource contexts (with exactly one hole) are defined via the grammar:

$$\mathbf{R} ::= (\cdot) \mid \mathbf{R}t \mid t\mathbf{R} \mid [\lambda x. \mathbf{R}, v_1, \dots, v_k] \quad (k \geq 0)$$

Let \mathbf{R} be a resource context. We use $\mathbf{R}(t)$ for the resource term obtained by the capture-allowing substitution of the resource term t for the hole (\cdot) in \mathbf{R} . If $\mathbb{T} = \sum_{i=1}^n t_i$ (with $t_1, \dots, t_n \in rA^{\mathbf{t}}$), then $\mathbf{R}(\mathbb{T}) = \sum_{i=1}^n \mathbf{R}(t_i) \in \mathbf{2}\langle rA^{\mathbf{t}} \rangle$ (see also [14, §5.2] and [15, §2.1]). For example, $\mathbf{R}(0) = 0$ and $[\lambda x. [x]([y][z] + [z][y]), y] = [\lambda x. [x]([y][z]), y] + [\lambda x. [x]([z][y]), y]$.

Definition 13. We define the following binary relations from $rA^{\mathbf{t}}$ to $\mathbf{2}\langle rA^{\mathbf{t}} \rangle$:

$$\begin{aligned} [\lambda x. t][v_1, \dots, v_k] &\mapsto_{\beta_v} t\langle [v_1, \dots, v_k] / x \rangle & [\lambda x. t]ss' &\mapsto_{\sigma_1} [\lambda x. ts']s \quad \text{if } x \notin \text{fv}(s') \\ [v_1, \dots, v_n]t &\mapsto_0 0 \quad \text{if } n \neq 1 & [v]([\lambda x. t]s) &\mapsto_{\sigma_3} [\lambda x. [v]t]s \quad \text{if } x \notin \text{fv}(v) \end{aligned}$$

We set $\mapsto_v = \mapsto_{\beta_v} \cup \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3} \cup \mapsto_0$.

According to the convention of §2, $\rightarrow_v \subseteq rA^{\mathbf{t}} \times \mathbf{2}\langle rA^{\mathbf{t}} \rangle$ is the reduction obtained by resource-contextual closure of \mapsto_v .

The *resource λ_v^σ -calculus* consists of the language $rA^{\mathbf{t}}$ and the reduction \rightarrow_v : it is the resource CBV λ -calculus of [14] plus the σ_1 - and σ_3 -rules.

As a technical simplification, we extend \rightarrow_v to a binary relation on $\mathbf{2}\langle rA^{\mathbf{t}} \rangle$ by linearity, i.e. $(\sum_{i=1}^n t_i) + \mathbb{S} \rightarrow_v (\sum_{i=1}^n \mathbb{T}_i) + \mathbb{S}$ iff $t_i \rightarrow_v \mathbb{T}_i$ for every $i = 1, \dots, n$ ($n \geq 1$). With this extension we can concisely state the following theorem:

Theorem 14. *Reduction \rightarrow_v on $\mathbf{2}\langle rA^{\mathbf{t}} \rangle$ is strongly normalizing and confluent.*

We omit the proof of Thm. 14. Strong normalization is evident (see [14] for a proof for the resource-contextual closure of $\mapsto_{\beta_v} \cup \mapsto_0$). The proof of local confluence for the resource λ_v^σ -calculus is analogous to the one for v -reduction on λ -terms (see §2). Finally, confluence is obtained by Newman's Lemma.

5 A Relational Model of (Resource) CBV Lambda-Calculus

In this section we present a relational model for both the λ_V^σ -calculus and the resource λ_V^σ -calculus. This model is to be found in the category **Rel** of sets and relations (i.e. $\mathbf{Rel}(X, Y) = \mathcal{P}(X \times Y)$). In **Rel** identities are diagonal relations and composition of morphisms is the standard composition of relations. This category has a symmetric monoidal structure given by $\mathbf{1} = \{1\}$ (arbitrary singleton set) and $X \otimes Y = X \times Y$. This symmetric monoidal category is closed, with $X \multimap Y = X \times Y$, and $*$ -autonomous with dualizing object $\perp = \mathbf{1}$. Category **Rel** is cartesian, with $X \& Y = (\{1\} \times X) \cup (\{2\} \times Y)$, and has an exponential functor $!$ defined by $!X = \mathcal{M}_f(X)$ (the set of finite multisets on X) and $!f = \{([\alpha_1, \dots, \alpha_n], [\beta_1, \dots, \beta_n]) : n \geq 0, (\alpha_i, \beta_i) \in f \forall 1 \leq i \leq n\}$ for $f \in \mathbf{Rel}(X, Y)$.

All this structure makes **Rel** a new-Seely category and hence a categorical model of Linear Logic (LL). For more details we refer the reader to [19,14].

The model. We build inductively a family of sets $(U_n)_{n \in \mathbf{N}}$ given by $U_0 = \emptyset$ and $U_{n+1} = \mathcal{M}_f(U_n) \times \mathcal{M}_f(U_n)$. Finally, we set $U = \bigcup_{n \in \mathbf{N}} U_n$. Notice that $U_n \subsetneq U_{n+1}$ for all $n \in \mathbf{N}$, and $U = \mathcal{M}_f(U) \times \mathcal{M}_f(U) = !U \multimap !U$.

5.1 Interpreting the CBV Lambda-Calculus

Using the fact that **Rel** has the structure of a LL model, we can give a concrete interpretation of λ -terms as morphisms from $\mathcal{M}_f(U)^n$ to $\mathcal{M}_f(U)$ in **Rel** (where $\mathcal{M}_f(U)^n$ is the n -fold set-theoretic power of $\mathcal{M}_f(U)$). This semantics can also be described by type judgements (see [14]). With $a \uplus b$ we indicate the union of the multisets a and b (accounting for repetitions); if \vec{a} and \vec{b} are two finite sequences (of the same length) of multisets, $\vec{a} \uplus \vec{b}$ is their component-wise union.

Definition 15. For every λ -term M and repetition-free list $\vec{x} \supseteq \text{fv}(M)$, we define, by induction on M , its interpretation $\llbracket M \rrbracket_{\vec{x}} \subseteq \mathcal{M}_f(U)^n \times \mathcal{M}_f(U)$ (where n is the length of \vec{x}), as follows:

$$\begin{aligned} \llbracket x_i \rrbracket_{\vec{x}} &= \{(\vec{a}, a_i) : a_i \in \mathcal{M}_f(U), a_j = [] \text{ for all } 1 \leq j \leq n \text{ with } j \neq i\} \\ \llbracket \lambda y. N \rrbracket_{\vec{x}} &= \{(\biguplus_{i=1}^k \vec{a}_i, \biguplus_{i=1}^k [(b_i, c_i)]) : k \geq 0, \forall i = 1, \dots, k. ((\vec{a}_i, b_i), c_i) \in \llbracket N \rrbracket_{\vec{x}, y}\} \\ \llbracket MN \rrbracket_{\vec{x}} &= \{(\vec{a}_0 \uplus \vec{a}_1, c) : \exists b \in \mathcal{M}_f(U). (\vec{a}_0, [(b, c)]) \in \llbracket M \rrbracket_{\vec{x}}, (\vec{a}_1, b) \in \llbracket N \rrbracket_{\vec{x}}\}. \end{aligned}$$

Notation. Hereafter, whenever we write $\llbracket M \rrbracket_{\vec{x}}$ we suppose that \vec{x} is a repetition-free list of variables containing $\text{fv}(M)$. Moreover, we will sometimes silently use the fact that $\llbracket M \rrbracket_{\vec{x}, y} = \{((\vec{a}, []), b) : (\vec{a}, b) \in \llbracket M \rrbracket_{\vec{x}}\}$ whenever $y \notin \vec{x}$.

Theorem 16 (soundness). Let $M, N \in \Lambda$. If $M \rightarrow_V N$, then $\llbracket M \rrbracket_{\vec{x}} = \llbracket N \rrbracket_{\vec{x}}$.

5.2 Interpreting the Resource CBV Lambda-Calculus

In addition to the structure mentioned above, **Rel** is additive, and more precisely its hom-sets are enriched over the category of complete lattices, with set-theoretic

union as join operation. The category **Rel** is a *weak differential LL model* (see [14]). Using this structure we can give the concrete interpretation of expressions as morphisms from $\mathcal{M}_f(U)^n$ to $\mathcal{M}_f(U)$ in **Rel**.

Definition 17. *For every expression e and repetition-free list $\vec{x} \supseteq \text{fv}(e)$, we define, by induction on e , its interpretation $\llbracket e \rrbracket_{\vec{x}} \subseteq \mathcal{M}_f(U)^n \times \mathcal{M}_f(U)$ (where n is the length of \vec{x}), as follows:*

$$\begin{aligned} \llbracket x_i \rrbracket_{\vec{x}} &= \{(\vec{a}, [\alpha]) : \alpha \in U, a_i = [\alpha], a_j = [] \text{ for all } 1 \leq j \leq n \text{ with } j \neq i\} \\ \llbracket \lambda z.t \rrbracket_{\vec{x}} &= \{(\vec{a}, [(b, c)]) : ((\vec{a}, b), c) \in \llbracket t \rrbracket_{\vec{y}, z}\} \\ \llbracket st \rrbracket_{\vec{x}} &= \{(\vec{a}_0 \uplus \vec{a}_1, c) : \exists b \in \mathcal{M}_f(U). (\vec{a}_0, [(b, c)]) \in \llbracket s \rrbracket_{\vec{x}}, (\vec{a}_1, b) \in \llbracket t \rrbracket_{\vec{x}}\} \\ \llbracket [v_1, \dots, v_k] \rrbracket_{\vec{x}} &= \{(\biguplus_{i=1}^k \vec{a}_i, \biguplus_{i=1}^k b_i) : k \geq 0, \forall i = 1, \dots, k. (\vec{a}_i, b_i) \in \llbracket v_i \rrbracket_{\vec{x}}\}. \end{aligned}$$

Finally, sums of expressions are interpreted by setting $\llbracket \sum_{i=1}^n e_i \rrbracket_{\vec{x}} = \bigcup_{i=1}^n \llbracket e_i \rrbracket_{\vec{x}}$.

Notation. As for λ -terms, whenever we write $\llbracket e \rrbracket_{\vec{x}}$ we suppose that \vec{x} is a repetition-free list of variables containing $\text{fv}(e)$, and similarly for the sums. Note that $\llbracket [] \rrbracket_{\vec{x}} = \{([]^n, [])\} \subseteq \mathcal{M}_f(U)^n \times \mathcal{M}_f(U)$, where $[]^n = \underbrace{([], \dots, [])}_{n \text{ times}}$.

Theorem 18 (soundness). *Let $\mathbb{S}, \mathbb{T} \in \mathbf{2}\langle \text{rA}^t \rangle$. If $\mathbb{S} \rightarrow_{\vee} \mathbb{T}$, then $\llbracket \mathbb{S} \rrbracket_{\vec{x}} = \llbracket \mathbb{T} \rrbracket_{\vec{x}}$.*

The following notion of CBV Taylor expansion has been introduced in [14].

Definition 19 ([14], Taylor expansion). *Given a λ -term M , we inductively define a set $\mathcal{T}(M)$ of resource terms, called the Taylor expansion of M , as follows:*

$$\begin{aligned} \mathcal{T}(x) &= \{[x^n] : n \geq 0\} \quad \text{where } [x^n] = \overbrace{[x, \dots, x]}^{n \text{ times}} \\ \mathcal{T}(\lambda x.M) &= \{[\lambda x.t_1, \dots, \lambda x.t_n] : n \geq 0, \forall i. t_i \in \mathcal{T}(M)\} \\ \mathcal{T}(MN) &= \{st : s \in \mathcal{T}(M), t \in \mathcal{T}(N)\}. \end{aligned}$$

Theorem 20 ([14]). *Let M be a λ -term. Then $\llbracket M \rrbracket_{\vec{x}} = \bigcup_{t \in \mathcal{T}(M)} \llbracket t \rrbracket_{\vec{x}}$.*

Thm. 20 shows the semantical connection between λ -terms and their Taylor expansion. In the next section (§6) it will be applied in Thm. 39.1, which is in turn a fundamental part of one of our main results Thm. 24.

Definition 21. *For every expression e we define by induction the set $\text{strat}(e)$ of multisets of resource values that occur in e in stratified position, as follows:*

$$\begin{aligned} \text{strat}(x) &= \emptyset; & \text{strat}([v_1, \dots, v_n]) &= \{[v_1, \dots, v_n]\} \cup \bigcup_{i=1}^n \text{strat}(v_i) \quad (n \geq 0); \\ \text{strat}(st) &= \text{strat}(s); & \text{strat}(\lambda x.t) &= \text{strat}(t). \end{aligned}$$

We set $\text{Strat} = \{t \in \text{rA}^t : [] \notin \text{strat}(t)\}$, whose elements are called stratified resource terms.

A stratified resource term t does not contain any $[]$ in stratified position, i.e. every occurrence of $[]$ in t is a subterm of some subterm of t in argument position. For instance: $[x][], [x]([\lambda z.[]]) \in \mathbf{Strat}$ but $[], [][z], [\lambda z.[][x, y]] \notin \mathbf{Strat}$.

Stratified resource terms are not closed under ν -reduction. For example, the stratified resource term $[\lambda x.[x]][\lambda y.[]]$ ν -reduces to the non-stratified $[\lambda y.[]]$.

Definition 22 (stratified Taylor expansion). *Given a λ -term M , we define its stratified Taylor expansion $\mathcal{T}_s(M) = \{t \in \mathcal{T}(M) : \text{if } t \twoheadrightarrow_\nu \mathbb{T}, \text{ then } \mathbb{T} \subseteq \mathbf{Strat}\}$.*

Example. The λ -term $M = (\lambda xy.x)\Omega$ is neither \mathbf{w} - nor \mathbf{s} -normalizable and every resource term in $\mathcal{T}(M)$ ν -reduces to 0. Instead the non- \mathbf{s} -normalizable (but \mathbf{w} -normal) λ -term $N = (\lambda xy.\Omega)(zz')$ has infinitely many resource terms in $\mathcal{T}(N)$ that do not ν -reduce to 0, like $t = [\lambda x.[]]([z][z'])$ for example. However $t \notin \mathcal{T}_s(N)$ and $\mathcal{T}_s(N)$ contains only resource terms that ν -reduce to 0, because all resource terms in $\mathcal{T}(N)$ not ν -reducing to 0 contain at least one $[]$ in stratified position.

The semantical connection between λ -terms and their stratified Taylor expansion is illustrated in one of our main results, Thm. 25. In particular, Thm. 39.2 is the step in which it is proved that the interpretation of $\mathcal{T}_s(M)$ actually witnesses the strong \mathbf{s} -normalization of M . Intuitively, if $t \in \mathcal{T}_s(M)$ then the ν -normal form of t is a sum $\sum_{i=1}^n t_i$ ($n \geq 0$) of stratified resource terms, each of which does not contain $[]$ in stratified position: a subterm $[]$ inside a t_i does not “hide” a non- \mathbf{s} -normalizable λ -term N such that $M = \mathbf{S}(N)$. So, by Lemma 38.ii one can prove that if $t \neq 0$ then M is strongly \mathbf{s} -normalizing.

6 The Main Theorems

In this section we will present our main results: the semantical and internal operational characterization of *potential valuability* (Thm. 24) and *solvability* (Thm. 25) for the $\lambda_{\mathcal{V}}^c$ -calculus. See §1 for an overview of these notions.

Definition 23 (Potential valuability, solvability). *Let M be a λ -term:*

- M is potentially valuable if there exist variables x_1, \dots, x_m and λ -values V, V_1, \dots, V_m (with $m \geq 0$) such that $M\{V_1/x_1, \dots, V_m/x_m\} \twoheadrightarrow_\nu V$;
- M is solvable if there exist variables x_1, \dots, x_m and λ -terms N_1, \dots, N_n (for some $n, m \geq 0$) such that $(\lambda x_1 \dots x_m.M)N_1 \dots N_n \twoheadrightarrow_\nu \mathbf{I}$.

We state now the two main theorems. In particular, Thm. 24 says that \mathbf{w} -normalizability (i.e. potential valuability) plays a role analogous to that of head-normalizability for many call-by-name models, like Scott’s \mathbf{D}_∞ .

Theorem 24. *Let M be a λ -term with $\vec{x} \supseteq \mathbf{fv}(M)$. The following are equivalent:*

- (i) M is \mathbf{w} -normalizable;
- (ii) M is potentially valuable;
- (iii) $[[M]]_{\vec{x}} \neq \emptyset$;
- (iv) M is strongly \mathbf{w} -normalizing.

Theorem 25. *Let M be a λ -term with $\vec{x} \supseteq \mathbf{fv}(M)$. The following are equivalent:*

- (i) M is \mathbf{s} -normalizable; (iii) $\bigcup_{t \in \mathcal{T}_s(M)} \llbracket t \rrbracket_{\bar{x}} \neq \emptyset$;
(ii) M is solvable; (iv) M is strongly \mathbf{s} -normalizing.

An immediate corollary of Thm. 24 and 25 is that every solvable (i.e. \mathbf{s} -normalizable) λ -term is also potentially valuable (i.e. \mathbf{w} -normalizable).

The proofs of Thm. 24 and 25 are divided into parts, which are detailed separately in the next subsections, due to the different techniques used for each one of them. The splitting of the two proofs follows the same pattern. The implications (i) \Rightarrow (ii) of both theorems are proved in §6.1 by purely syntactical means. The implication (ii) \Rightarrow (iii) of Thm. 24 is shown in §6.2 using the resource λ_V^σ -calculus of §4; for this implication of Thm. 25 we use an extension of the resource λ_V^σ -calculus presented in §6.3. The implication (iii) \Rightarrow (iv) of both theorems is proved in §6.4 using simulations of \mathbf{w} - and \mathbf{s} -reductions in λ_V^σ -calculus by the \mathbf{v} -reduction of the resource λ_V^σ -calculus. Finally, (iv) \Rightarrow (i) are trivial in both cases.

6.1 From Weak and Stratified Normalization to Solvability and Potential Valuability

Our goal here is to prove the implication (i) \Rightarrow (ii) of Thm. 24 and 25. Our approach is largely inspired by [6,7,5].

For every $n \in \mathbf{N}$, we set $\mathbf{o}^n = \lambda x_n \dots x_0.x_0$. Notice that $\mathbf{o}^0 = \mathbf{I}$ and \mathbf{o}^n is a closed value for any $n \in \mathbf{N}$. Moreover, $\mathbf{o}^n V \mapsto_{\beta_v} \mathbf{o}^{n-1}$ for any $n > 0$ and $V \in \Lambda^v$.

Lemma 26. *Let $M \in \mathbf{w}_{\text{nf}}$ with $\text{fv}(M) \subseteq \{x_1, \dots, x_m\}$ and let $j \in \mathbf{N}$. Then there exists $h > 0$ such that for all $n_1, \dots, n_m \geq j + h$ there exists a λ -term N such that $M\{\mathbf{o}^{n_1}/x_1, \dots, \mathbf{o}^{n_m}/x_m\} \rightarrow_v \lambda x.N$ and $\lambda x.N$ is closed.*

Lemma 27. *Let $M \in \mathbf{s}_{\text{nf}}$ with $\text{fv}(M) \subseteq \{x_1, \dots, x_m\}$ and let $j \in \mathbf{N}$. Then there exist $h, k \in \mathbf{N}$ such that for all $n_1, \dots, n_{m+k} \geq j + h$ there exists $n \geq j$ such that $M\{\mathbf{o}^{n_1}/x_1, \dots, \mathbf{o}^{n_m}/x_m\} \mathbf{o}^{n_{m+1}} \dots \mathbf{o}^{n_{m+k}} \rightarrow_v \mathbf{o}^n$.*

Theorem 28. *Let M be a λ -term.*

1. [(i) \Rightarrow (ii) of Thm. 24] *If M is \mathbf{w} -normalizable then M is potentially valuable.*
2. [(i) \Rightarrow (ii) of Thm. 25] *If M is \mathbf{s} -normalizable then M is solvable.*

Proof. For point 1 (resp. 2), hypothesis means that there is a \mathbf{w} - (resp. \mathbf{s} -) normal form M' such that $M \rightarrow_w M'$ (resp. $M \rightarrow_s M'$), moreover $M' \in \mathbf{w}_{\text{nf}}$ (resp. $M' \in \mathbf{s}_{\text{nf}}$) by Prop. 12. Let $\text{fv}(M) = \{x_1, \dots, x_m\}$ and thus $\text{fv}(M') \subseteq \{x_1, \dots, x_m\}$.

1. By Lemma 26 (taking $j = 0$) there exists $h > 0$ such that:

$M'\{\mathbf{o}^h/x_1, \dots, \mathbf{o}^h/x_m\} \rightarrow_v \lambda x.N$, for some closed λ -value $\lambda x.N$. One has $M\{\mathbf{o}^h/x_1, \dots, \mathbf{o}^h/x_m\} \rightarrow_v M'\{\mathbf{o}^h/x_1, \dots, \mathbf{o}^h/x_m\}$ by Lemma 5.ii, so that M is potentially valuable because $\lambda x.N$ is a closed λ -value.

2. By Lemma 27 (taking $j = 0$), there exist $h, k, n \in \mathbf{N}$ such that:

$(M'\{\mathbf{o}^h/x_1, \dots, \mathbf{o}^h/x_m\}) \mathbf{o}^h \dots \mathbf{o}^h \rightarrow_v \mathbf{o}^n$ (\mathbf{o}^h is applied k times). We conclude that M is solvable because if we set $\mathbf{H} = (\lambda x_1 \dots x_m. \underbrace{\mathbf{o}^h \dots \mathbf{o}^h}_{m+k \text{ times}} \underbrace{\mathbf{I} \dots \mathbf{I}}_n)$, then

$$\begin{aligned} \mathbf{H}(M) &\rightarrow_v \mathbf{H}(M') \\ &\rightarrow_v (M'\{\mathbf{o}^h/x_1, \dots, \mathbf{o}^h/x_m\}) \mathbf{o}^h \dots \mathbf{o}^h \mathbf{I} \dots \mathbf{I} \rightarrow_v \mathbf{o}^n \mathbf{I} \dots \mathbf{I} \rightarrow_v \mathbf{I}. \quad \square \end{aligned}$$

6.2 From Potential Valuability to Non-emptiness

The following theorem proves the implication (ii) \Rightarrow (iii) of Thm. 24.

Theorem 29. *Let M be a λ -term with $\vec{x} \supseteq \text{fv}(M)$. If M is potentially valuable, then $\llbracket M \rrbracket_{\vec{x}} \neq \emptyset$.*

Proof. If M is potentially valuable (see Def. 23) there exist variables x_1, \dots, x_m and λ -values V, V_1, \dots, V_m (for some $m \geq 0$) s.t. $M\{V_1/x_1, \dots, V_m/x_m\} \rightarrow_{\nu} V$. Since variables are λ -values, we can suppose without loss of generality that $\vec{x} = (x_1, \dots, x_m) \supseteq \text{fv}(M)$. Let $\vec{y} = \text{fv}(V) \cup \bigcup_{i=1}^m \text{fv}(V_i)$. One can prove by induction on M that

$$\begin{aligned} \llbracket M\{V_1/x_1, \dots, V_m/x_m\} \rrbracket_{\vec{y}} = & \{ (\bigoplus_{i=1}^m \bar{a}_i, c) : \exists b_1, \dots, b_m \in \mathcal{M}_f(U) : \\ & ((b_1, \dots, b_m), c) \in \llbracket M \rrbracket_{\vec{x}}, (\bar{a}_i, b_i) \in \llbracket V_i \rrbracket_{\vec{y}} \text{ for all } 1 \leq i \leq m \} . \end{aligned}$$

Since $\llbracket V \rrbracket_{\vec{y}} \neq \emptyset$ (this can be proved by simple inspection), by Thm. 16 we obtain that $\llbracket M\{V_1/x_1, \dots, V_m/x_m\} \rrbracket_{\vec{y}} \neq \emptyset$ also holds, so that $\llbracket M \rrbracket_{\vec{x}} \neq \emptyset$. \square

6.3 From Solvability to Non-emptiness of Stratified Taylor Expansion

The implication (ii) \Rightarrow (iii) of Thm. 25 seems much more difficult to prove. To accomplish this task we introduce the *resource λ_{ν}^{σ} -calculus with tests*, a CBV version of the resource calculus with tests defined in [15]. In this syntax all elements of the relational model are definable (see Def. 34).

The language extends that of resource λ_{ν}^{σ} -calculus (see §4, p. 108) as follows:

$$\begin{array}{lll} (rA^{\nu}) & u, v ::= x \mid \lambda x.t & \text{resource values} \\ (rA^{\natural}) & s, t ::= t * p \mid st \mid [v_1, \dots, v_k] \quad (k \geq 0) & \text{resource terms} \\ (rA^{\tau}) & p, q ::= \tau[t_1, \dots, t_k] \quad (k \geq 0) & \text{tests} \end{array}$$

Note the overloaded use of rA^{ν} and rA^{\natural} , which now (and until Lemma 36) indicate larger sets than those introduced in §4. We will use this extension to prove Lemma 36 (whose statement concerns only resource terms without tests).

Tests are – formally – multisets of resource terms, the “ τ ” being a tag for distinguishing them from bags of values. Intuitively, they are constructions which can produce either *success*, represented by $\tau[\]$, or *failure*, represented by 0.

Notation. We set $\varepsilon = \tau[\]$ and $\tau[t_1, \dots, t_k] \parallel \tau[t_{k+1}, \dots, t_n] = \tau[t_1, \dots, t_n]$ ($k \leq n$).

The test $p \parallel q$ represents the (must-)parallel composition of p and q (i.e., $p \parallel q$ succeeds iff both p and q succeed). The composition is parallel in the sense that the order of evaluation is inessential (remember that they are multisets). The binary operator $*$ allows to build a resource term out of a resource term and a test: intuitively, the resource term $t * p$ may be thought of as something that

outputs the result of t only if p succeeds. Dually, the “cork construction” $\tau[t]$ may be thought of as a check that tests whether or not t ν -reduces to $[\]$.

Resource, *test-resource* and *test-test contexts* (with exactly one hole), denoted resp. by \mathbf{R} , \mathbf{Q} and \mathbf{P} , are defined by mutual induction via the grammar ($k \geq 0$):

$$\begin{aligned} \mathbf{R} &::= (\cdot) \mid \mathbf{R}t \mid t\mathbf{R} \mid t * \mathbf{Q} \mid [\lambda x. \mathbf{R}, v_1, \dots, v_k] \quad (\text{resource contexts}); \\ \mathbf{Q} &::= \tau[\mathbf{R}, t_1, \dots, t_k] \quad (\text{test-resource c.}); \quad \mathbf{P} ::= (\cdot) \parallel \tau[t_1, \dots, t_k] \quad (\text{test-test c.}). \end{aligned}$$

Let $t, t_1, \dots, t_n \in r\Lambda^t$ (resp. $p, p_1, \dots, p_n \in r\Lambda^\tau$). We use $\mathbf{Q}(t)$ (resp. $\mathbf{P}(p)$) for the test obtained by the capture-allowing substitution of t (resp. p) for the hole (\cdot) in \mathbf{Q} (resp. \mathbf{P}); similarly for $\mathbf{R}(t)$ (see p. 109). As usual, $\mathbf{R}(\sum_i t_i) = \sum_i \mathbf{R}(t_i)$, $\mathbf{Q}(\sum_i t_i) = \sum_i \mathbf{Q}(t_i)$ and $\mathbf{P}(\sum_i p_i) = \sum_i \mathbf{P}(p_i)$. E.g., $t * 0 = t * \mathbf{Q}(0) = \mathbf{R}(0) = 0$.

Definition 30. *The operational semantics of the resource λ^σ -calculus with tests extends the set of rules listed in Def. 13 with the following ones:*

$$\begin{array}{ll} t(s * p) \mapsto_{\tau_1} ts * p & \tau[t * p] \mapsto_{\tau_4} \tau[t] \parallel p \\ (t * p)s \mapsto_{\tau_2} ts * p & \tau[[v_1, \dots, v_n]] \mapsto_{\tau_5} \begin{cases} \varepsilon & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} \\ (t * p) * q \mapsto_{\tau_3} t * (p \parallel q) & \end{array}$$

We set $\mapsto_{\nu\tau} = \mapsto_\nu \cup (\bigcup_{i=1}^5 \mapsto_{\tau_i}) \subseteq (r\Lambda^t \times \mathbf{2}\langle r\Lambda^t \rangle) \cup (r\Lambda^\tau \times \mathbf{2}\langle r\Lambda^\tau \rangle)$. Then, according to the convention of §2, $\rightarrow_{\nu\tau} \subseteq r\Lambda^\tau \times \mathbf{2}\langle r\Lambda^\tau \rangle$ is the reduction obtained by test-contextual closure⁴ of $\mapsto_{\nu\tau}$. The *resource λ^σ -calculus with tests* consists of the language $r\Lambda^\tau$ and the reduction $\rightarrow_{\nu\tau}$.

As a technical simplification, we extend $\rightarrow_{\nu\tau}$ to a binary relation on $\mathbf{2}\langle r\Lambda^\tau \rangle$ by linearity, i.e., $(\sum_{i=1}^n q_i) + \mathbb{P} \rightarrow_{\nu\tau} (\sum_{i=1}^n \mathbb{Q}_i) + \mathbb{P}$ iff $q_i \rightarrow_{\nu\tau} \mathbb{Q}_i$ for every $i = 1, \dots, n$ ($n \geq 1$). With this extension we can concisely state the following theorem:

Theorem 31. *Reduction $\rightarrow_{\nu\tau}$ on $\mathbf{2}\langle r\Lambda^\tau \rangle$ is strongly normalizing and confluent.*

Definition 32. *For every test p and repetition-free list $\vec{x} \supseteq \text{fv}(p)$, we define the interpretation $\llbracket p \rrbracket_{\vec{x}} \subseteq \mathcal{M}_f(U)^n \times \mathbf{1}$ of p , where n is the length of \vec{x} , by mutual induction with Def. 17 as follows:*

$$\begin{aligned} \llbracket \varepsilon \rrbracket_{\vec{x}} &= \{([\]^n, 1)\} & \llbracket p \parallel q \rrbracket_{\vec{x}} &= \{(\vec{a} \uplus \vec{b}, 1) : (\vec{a}, 1) \in \llbracket p \rrbracket_{\vec{x}}, (\vec{b}, 1) \in \llbracket q \rrbracket_{\vec{x}}\} \\ \llbracket \tau[t] \rrbracket_{\vec{x}} &= \{(\vec{a}, 1) : (\vec{a}, [\]) \in \llbracket t \rrbracket_{\vec{x}}\} & \llbracket t * p \rrbracket_{\vec{x}} &= \{(\vec{a} \uplus \vec{b}, c) : (\vec{a}, c) \in \llbracket t \rrbracket_{\vec{x}}, (\vec{b}, 1) \in \llbracket p \rrbracket_{\vec{x}}\}. \end{aligned}$$

Finally, sums of tests are interpreted by setting $\llbracket \sum_{i=1}^n p_i \rrbracket_{\vec{x}} = \bigcup_{i=1}^n \llbracket p_i \rrbracket_{\vec{x}}$.

Theorem 33 (soundness). *Let $\mathbb{P}, \mathbb{Q} \in \mathbf{2}\langle r\Lambda^\tau \rangle$. If $\mathbb{P} \rightarrow_{\nu\tau} \mathbb{Q}$, then $\llbracket \mathbb{P} \rrbracket_{\vec{x}} = \llbracket \mathbb{Q} \rrbracket_{\vec{x}}$.*

A key tool to connect the semantics with the $\nu\tau$ -reduction is the following transformation of elements of $\mathcal{M}_f(U)$ into resource terms and test contexts. The role of this transformation is made clear in Lemma 35, used to prove Lemma 36.

⁴ This means that, for every $p \in r\Lambda^\tau$ and $p' \in \mathbf{2}\langle r\Lambda^\tau \rangle$, if $p \rightarrow_{\nu\tau} p'$ then either there exist a test-test context $\mathbf{P}, q \in r\Lambda^\tau$ and $q' \in \mathbf{2}\langle r\Lambda^\tau \rangle$ such that $p = \mathbf{P}(q)$, $p' = \mathbf{P}(q')$ and $q \mapsto_{\tau_i} q'$ with $i \in \{4, 5\}$; or there exist a test-resource context $\mathbf{Q}, t \in r\Lambda^t$ and $t' \in \mathbf{2}\langle r\Lambda^t \rangle$ such that $p = \mathbf{Q}(t)$, $p' = \mathbf{Q}(t')$ and $t \mapsto_{\nu\tau'} t'$ with $\mapsto_{\nu\tau'} = \mapsto_\nu \cup (\bigcup_{i=1}^3 \mapsto_{\tau_i})$.

Definition 34. Let $c = [(a_1, b_1), \dots, (a_n, b_n)] \in \mathcal{M}_f(U)$ ($n \geq 0$). We define:

- the closed resource term $c^- = [\lambda y_1. b_1^- * a_1^+ (\llbracket y_1^{m_1} \rrbracket), \dots, \lambda y_n. b_n^- * a_n^+ (\llbracket y_n^{m_n} \rrbracket)]$, where m_i is the cardinality of the multiset a_i (for $i = 1, \dots, n$);
- the test-resource context $c^+ = \tau[\llbracket \lambda x. [\cdot] * \prod_{i=1}^n \tau[\llbracket \lambda y. [\cdot] * b_i^+ (\llbracket y^{k_i} \rrbracket) \rrbracket] (\llbracket x \rrbracket a_i^-) \rrbracket] (\cdot)$, where k_i is the cardinality of the multiset b_i (for $i = 1, \dots, n$).

Notation. For any $a \in \mathcal{M}_f(U)$, $\#a$ indicates its cardinality. For $\vec{a} = (a_1, \dots, a_n) \in \mathcal{M}_f(U)^n$ and $t \in \mathbf{r}\Lambda^\dagger$, we write $t\langle \vec{a}^- / \vec{x} \rangle$ as a shorthand for $t\langle a_1^- / x_1 \rangle \cdots \langle a_n^- / x_n \rangle$.

Lemma 35. Let $(\vec{a}, b) \in \mathcal{M}_f(U)^n \times \mathcal{M}_f(U)$, $k = \#b$ and $t \in \mathbf{r}\Lambda^\dagger$ with $\vec{x} \supseteq \mathbf{fv}(t)$. Then $(\vec{a}, b) \in \llbracket t \rrbracket_{\vec{x}}$ iff $\tau[\llbracket \lambda y. [\cdot] * b^+ (\llbracket y^k \rrbracket) \rrbracket] (t\langle \vec{a}^- / \vec{x} \rangle) \rightarrow_{\mathbf{v}\tau} \varepsilon$.

Lemma 36. Let s and t be \mathbf{v} -normal resource terms without tests (i.e., generated by the grammar on §4, p. 108). If $s \in \mathbf{Strat}$ and $t \notin \mathbf{Strat}$, then $\llbracket s \rrbracket_{\vec{x}} \cap \llbracket t \rrbracket_{\vec{x}} = \emptyset$.

Proof. Let $(\vec{a}, b) \in \mathcal{M}_f(U)^n \times \mathcal{M}_f(U)$ and $\mathbf{Q}(\cdot) = \tau[\llbracket \lambda y. [\cdot] * b^+ (\llbracket y^k \rrbracket) \rrbracket] (\llbracket \cdot \rrbracket \langle \vec{a}^- / \vec{x} \rangle)$, with $k = \#b$. One can prove by induction on the \mathbf{v} -normal resource terms (without tests) that: either $\mathbf{Q}(t) \rightarrow_{\mathbf{v}\tau} \varepsilon$ and $\mathbf{Q}(s) \rightarrow_{\mathbf{v}\tau} 0$; or $\mathbf{Q}(s) \rightarrow_{\mathbf{v}\tau} \varepsilon$ and $\mathbf{Q}(t) \rightarrow_{\mathbf{v}\tau} 0$; or $\mathbf{Q}(s) \rightarrow_{\mathbf{v}\tau} 0$ and $\mathbf{Q}(t) \rightarrow_{\mathbf{v}\tau} 0$. Hence, by Lemma 35, $(\vec{a}, b) \notin \llbracket s \rrbracket_{\vec{x}} \cap \llbracket t \rrbracket_{\vec{x}}$. \square

Hereafter, when we will mention resource terms, we will refer to the ones without test (i.e., generated by the grammar on §4, p. 108).

The following theorem proves the implication (ii) \Rightarrow (iii) of Thm. 25.

Theorem 37. Let M be a λ -term and let $\vec{x} \supseteq \mathbf{fv}(M)$. If M is solvable, then $\bigcup_{t \in \mathcal{T}_s(M)} \llbracket t \rrbracket_{\vec{x}} \neq \emptyset$.

Proof. If M is solvable then there exists a context $\mathbf{C} = (\lambda x_1 \dots x_m. (\cdot)) N_1 \cdots N_n$ (for some $n, m \geq 0$) such that $\mathbf{C}(M) \rightarrow_{\mathbf{v}} \mathbf{I}$. By Thm. 16 and 20, $\bigcup_{t \in \mathcal{T}(\mathbf{C}(M))} \llbracket t \rrbracket_{\vec{x}} = \llbracket \mathbf{C}(M) \rrbracket_{\vec{x}} = \llbracket \mathbf{I} \rrbracket_{\vec{x}} = \bigcup_{t \in \mathcal{T}(\mathbf{I})} \llbracket t \rrbracket_{\vec{x}}$. Using Lemma 36 we infer that $\bigcup_{t \in \mathcal{T}_s(\mathbf{C}(M))} \llbracket t \rrbracket_{\vec{x}} = \bigcup_{t \in \mathcal{T}_s(\mathbf{I})} \llbracket t \rrbracket_{\vec{x}}$. Therefore $\bigcup_{t \in \mathcal{T}_s(\mathbf{C}(M))} \llbracket t \rrbracket_{\vec{x}} \neq \emptyset$ because it is easy to check that $\bigcup_{t \in \mathcal{T}_s(\mathbf{I})} \llbracket t \rrbracket_{\vec{x}} \neq \emptyset$. By Thm. 18 and 14, $\bigcup_{t \in \mathcal{T}_s(\mathbf{C}(M))} \llbracket t \rrbracket_{\vec{x}} \neq \emptyset$ implies that there is a resource term in $\mathcal{T}_s(\mathbf{C}(M))$ that \mathbf{v} -reduces to a non-zero \mathbf{v} -normal form. Now all resource terms in $\mathcal{T}_s(\mathbf{C}(M))$ are of the shape $\mathbf{R}(s)$ for some $s \in \mathcal{T}_s(M)$ (because the hole of \mathbf{C} is in stratified position), so that if all resource terms in $\mathcal{T}_s(M)$ \mathbf{v} -reduced to 0, then all resource terms in $\mathcal{T}_s(\mathbf{C}(M))$ would \mathbf{v} -reduce to 0. Thus, there is $t \in \mathcal{T}_s(M)$ that \mathbf{v} -reduces to a \mathbf{v} -normal form $\mathbb{T} \neq 0$. It is easy to prove that $\llbracket t' \rrbracket_{\vec{x}} \neq \emptyset$ for every \mathbf{v} -normal form t' , hence $\llbracket t \rrbracket_{\vec{x}} = \llbracket \mathbb{T} \rrbracket_{\vec{x}} \neq \emptyset$ by Thm. 18. \square

6.4 From Non-emptiness to Strong Normalization

Our goal here is to prove the implication (iii) \Rightarrow (iv) of Thm. 24 and 25.

Lemma 38. *Let M, M' be λ -terms.*

- (i) *If $M \rightarrow_w M'$ and $t \in \mathcal{T}(M)$, then there exists $\mathbb{T} \subseteq \mathcal{T}(M')$ such that $t \rightarrow_v \mathbb{T}$.*
- (ii) *If $M \rightarrow_s M'$ and $s \in \mathcal{T}_s(M)$, then there exists $\mathbb{S} \subseteq \mathcal{T}_s(M')$ such that $s \rightarrow_v^+ \mathbb{S}$.*

Lemma 38.i is false if we replace the hypothesis $M \rightarrow_w M'$ with $M \rightarrow_s M'$. For instance, take $M = \lambda x.\Omega$: then $[\] \in \mathcal{T}(M)$ and $M \rightarrow_s M$, but $[\]$ is v -normal.

Theorem 39. *Let M be a λ -term and let $\vec{x} \supseteq \text{fv}(M)$.*

- 1. [(iii) \Rightarrow (iv) of Thm. 24] *If $\llbracket M \rrbracket_{\vec{x}} \neq \emptyset$ then M is strongly w -normalizing.*
- 2. [(iii) \Rightarrow (iv) of Thm. 25] *If $\bigcup_{t \in \mathcal{T}_s(M)} \llbracket t \rrbracket_{\vec{x}} \neq \emptyset$, then M is strongly s -normalizing.*

Proof. Let $(\vec{a}, b) \in \llbracket M \rrbracket_{\vec{x}}$ (resp. $(\vec{a}, b) \in \bigcup_{t \in \mathcal{T}_s(M)} \llbracket t \rrbracket_{\vec{x}}$). By Thm. 20 (resp. Then) there exists $t \in \mathcal{T}(M)$ (resp. $t \in \mathcal{T}_s(M)$) such that $(\vec{a}, b) \in \llbracket t \rrbracket_{\vec{x}}$. If $M \rightarrow_w M'$ (resp. $M \rightarrow_s M'$), then by Lemma 38.i (resp. Lemma 38.ii) there exists $\mathbb{T} \subseteq \mathcal{T}(M')$ (resp. $\mathbb{T} \subseteq \mathcal{T}_s(M')$) such that $t \rightarrow_v^+ \mathbb{T}$. According to Thm. 18, $(\vec{a}, b) \in \llbracket \mathbb{T} \rrbracket_{\vec{x}}$, hence $\mathbb{T} \neq \emptyset$ and so there exists $t' \in \mathbb{T}$ such that $(\vec{a}, b) \in \llbracket t' \rrbracket_{\vec{x}}$. Therefore, if there was an infinite reduction $M \rightarrow_w M_1 \rightarrow_w M_2 \rightarrow_w \dots$ (resp. $M \rightarrow_s M_1 \rightarrow_s M_2 \rightarrow_s \dots$) then there would also be an infinite reduction $t \rightarrow_v^+ \mathbb{T}_1 \rightarrow_v^+ \mathbb{T}_2 \rightarrow_v^+ \dots$, which is impossible by Thm. 14. \square

Conclusions and Future Work

Our approach, that exploits the validity of the Taylor formula for a resource CBV λ -calculus, makes use of purely combinatorial proofs, rather than more standard approaches based on reducibility or some specific machines. The interesting feature of this approach is that it can be used for many different calculi always using a similar relational model and a suitable resource calculus.

We think that using the ordinary syntax of λ -calculus with our reduction will allow to develop a reasonable theory of CBV Böhm trees, never defined before (Paolini's separability result in [20] for λ_v -calculus does not use Böhm trees), together with connections between equivalence of Böhm trees and observational equivalence. A future challenge is that of finding other fully abstract denotational models, in view of Paolini and Ronchi Della Rocca's proof of absence of fully abstract filter models (see [7, Thm. 12.1.25]) built from legal type systems.

Another direction is relating two equivalence relations on λ -terms, the one generated by our σ -rules and the one induced by Girard's CBV "boring" translation $(\cdot)^v$ of λ -calculus into IMELL proof-nets (along the lines of [17,18,21]).

References

- 1. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics. North-Holland, Amsterdam (1984)
- 2. Barendregt, H.: Solvability in lambda-calculi. J. Symb. Logic 75(3), 191–231 (1975)

3. Wadsworth, C.: The Relation Between Computational and Denotational Properties for Scott's D_∞ -Models of the λ -Calculus. *SIAM J. Comput.* 5(3), 488–521 (1976)
4. Plotkin, G.: Call-by-Name, Call-by-Value and the λ -Calculus. *Theor. Comput. Sci.* 1(2), 125–159 (1975)
5. Accattoli, B., Paolini, L.: Call-by-Value Solvability, Revisited. In: Schrijvers, T., Thiemann, P. (eds.) *FLOPS 2012*. LNCS, vol. 7294, pp. 4–16. Springer, Heidelberg (2012)
6. Paolini, L., Ronchi Della Rocca, S.: Call-by-value Solvability. *ITA* 33(6), 507–534 (1999)
7. Paolini, L., Ronchi Della Rocca, S.: The Parametric λ -calculus: a Metamodel for Computation. *Texts in Theoretical Computer Science*. Springer, Berlin (2004)
8. Paolini, L., Pimentel, E., Ronchi Della Rocca, S.: Strong Normalization from an unusual point of view. *Theor. Comput. Sci.* 412(20), 1903–1915 (2011)
9. Herbelin, H., Zimmermann, S.: An operational account of Call-by-Value Minimal and Classical λ -calculus in “Natural Deduction” form. In: Curien, P.-L. (ed.) *TLCA 2009*. LNCS, vol. 5608, pp. 142–156. Springer, Heidelberg (2009)
10. Hofmann, M.: Sound and Complete Axiomatisations of Call-by-Value Control Operators. *Mathematical Structures in Computer Science* 5(4), 461–482 (1995)
11. Moggi, E.: Computational Lambda-Calculus and Monads. In: *LICS*, pp. 14–23 (1989)
12. Dyckhoff, R., Lengrand, S.: Call-by-Value lambda-calculus and LJQ. *J. Log. Comput.* 17(6), 1109–1134 (2007)
13. Egidi, L., Honsell, F., Ronchi Della Rocca, S.: Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticæ* 16(2), 149–169 (1992)
14. Ehrhard, T.: Collapsing non-idempotent intersection types. In: *CSL*, pp. 259–273 (2012)
15. Bucciarelli, A., Carraro, A., Ehrhard, T., Manzonetto, G.: Full Abstraction for the Resource Lambda Calculus with Tests, through Taylor Expansion. *Logical Methods in Computer Science* 8(4) (2012)
16. Girard, J.: Linear Logic. *Theor. Comput. Sci.* 50(1), 1–102 (1987)
17. Regnier, L.: Lambda calcul et réseaux. PhD thesis, Université Paris 7 (1992)
18. Regnier, L.: Une équivalence sur les lambda-termes. *Theor. Comput. Sci.* 126(2), 281–292 (1994)
19. Melliès, P.: Categorical semantics of Linear Logic. In: *Interactive Models of Computation and Program Behaviour*. Panoramas et Synthèses, vol. 27, pp. 1–196. Société Mathématique de France (2009)
20. Paolini, L.: Call-by-Value Separability and Computability. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) *ICTCS 2001*. LNCS, vol. 2202, pp. 74–89. Springer, Heidelberg (2001)
21. Accattoli, B., Kesner, D.: The Permutative λ -Calculus. In: Bjørner, N., Voronkov, A. (eds.) *LPAR-18*. LNCS, vol. 7180, pp. 23–36. Springer, Heidelberg (2012)

Network-Formation Games with Regular Objectives

Guy Avni¹, Orna Kupferman¹, and Tami Tamir²

¹ School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

² School of Computer Science, The Interdisciplinary Center, Herzliya, Israel

Abstract. Classical network-formation games are played on a directed graph. Players have reachability objectives, and each player has to select a path satisfying his objective. Edges are associated with costs, and when several players use the same edge, they evenly share its cost. The theoretical and practical aspects of network-formation games have been extensively studied and are well understood. We introduce and study *network-formation games with regular objectives*. In our setting, the edges are labeled by alphabet letters and the objective of each player is a regular language over the alphabet of labels, given by means of an automaton or a temporal-logic formula. Thus, beyond reachability properties, a player may restrict attention to paths that satisfy certain properties, referring, for example, to the providers of the traversed edges, the actions associated with them, their quality of service, security, etc.

Unlike the case of network-formation games with reachability objectives, here the paths selected by the players need not be simple, thus a player may traverse some transitions several times. Edge costs are shared by the players with the share being proportional to the number of times the transition is traversed. We study the existence of a pure Nash equilibrium (NE), convergence of best-response-dynamics, the complexity of finding the social optimum, and the inefficiency of a NE compared to a social-optimum solution. We examine several classes of networks (for example, networks with uniform edge costs, or alphabet of size 1) and several classes of regular objectives. We show that many properties of classical network-formation games are no longer valid in our game. In particular, a pure NE might not exist and the Price of Stability equals the number of players (as opposed to logarithmic in the number of players in the classic setting, where a pure NE always exists). In light of these results, we also present special cases for which the resulting game is more stable.

1 Introduction

Network design and formation is a fundamental well-studied problem that involves many interesting combinatorial optimization problems. In practice, network design is often conducted by multiple strategic users whose individual costs are affected by the decisions made by others. Early works on network design focus on analyzing the efficiency and fairness properties associated with different sharing rules (e.g., [23,30]). Following the emergence of the Internet, there has been an explosion of studies employing game-theoretic analysis to explore Internet applications, such as routing in computer networks and network formation [17,1,12,2]. In network-formation games (for a survey, see [35]), the network is modeled by a weighted graph. The weight of an edge

indicates the cost of activating the transition it models, which is independent of the number of times the edge is used. Players have reachability objectives, each given by sets of possible source and target nodes. Players share the cost of edges used in order to fulfill their objectives. Since the costs are positive, the runs traversed by the players are simple. Under the common Shapley cost-sharing mechanism, the cost of an edge is shared evenly by the players that use it.

The players are selfish agents who attempt to minimize their own costs, rather than to optimize some global objective. In network-design settings, this would mean that the players selfishly select a path instead of being assigned one by a central authority. The focus in game theory is on the *stable* outcomes of a given setting, or the *equilibrium* points. A Nash equilibrium (NE) is a profile of the players' strategies such that no player can decrease his cost by an unilateral deviation from his current strategy, that is, assuming that the strategies of the other players do not change.¹

Reachability objectives enable the players to specify possible sources and targets. Often, however, it is desirable to refer also to other properties of the selected paths. For example, in a *communication* setting, edges may belong to different providers, and a user may like to specify requirements like "all edges are operated by the same provider" or "no edge operated by AT&T is followed by an edge operated by Verizon". Edges may also have different quality or security levels (e.g., "noisy channel", "high-bandwidth channel", or "encrypted channel"), and again, users may like to specify their preferences with respect to these properties. In *planning* or in *production systems*, nodes of the network correspond to configurations, and edges correspond to the application of actions. The objectives of the players are sequences of actions that fulfill a certain plan, which is often more involved than just reachability [13]; for example "once the arm is up, do not put it down until the block is placed".

The challenge of reasoning about behaviors has been extensively studied in the context of formal verification. While early research concerned the input-output relations of terminating programs, current research focuses on on-going behaviors of reactive systems [22]. The interaction between the components of a reactive system correspond to a multi-agent game, and indeed in recent years we see an exciting transfer of concepts and ideas between the areas of game theory and formal verification: logics for specifying multi-agent systems [3,9], studies of equilibria in games that correspond to the synthesis problem [8,7,16], an extension of mechanism design to on-going behaviors [25], studies of non-zero-sum games in formal methods [10,6], and more.

In this paper we extend network-formation games to a setting in which the players can specify regular objectives. This involves two changes of the underlying setting: First, the edges in the network are labeled by letters from a designated alphabet. Second, the objective of each player is specified by a *language* over this alphabet. Each player should select a path labeled by a word in his objective language. Thus, if we view the network as a *nondeterministic weighted finite automaton* (WFA) \mathcal{A} , then the set of strategies for a player with objective L is the set of accepting runs of \mathcal{A} on some word in L . Accordingly, we refer to our extension as *automaton-formation games*. As in classical network-formation games, players share the cost of edges they use. Unlike

¹ Throughout this paper, we focus on pure strategies and pure deviations, as is the case for the vast literature on cost-sharing games.

the classical game, the runs selected by the players need not be simple, thus a player may traverse some edges several times. Edge costs are shared by the players, with the share being proportional to the number of times the edge is traversed. This latter issue is the main technical difference between automaton-formation and network-formation games, and as we shall see, it is very significant.

Many variants of cost-sharing games and congestion games have been studied. A generalization of the network-formation game of [2] in which players are weighted and a player's share in an edge cost is proportional to its weight is considered in [11], where it is shown that the weighted game does not necessarily have a pure NE. In a different type of congestion games, players' payments depend on the resource they choose to use, the set of players using this resource, or both [29,26,27,19]. In some of these variants a NE is guaranteed to exist while in others it is not. All these variants are different from automaton-formation games, where a player needs to select a *multiset* of resources (namely, the edges he is going to traverse) rather than a single one.

We study the theoretical and practical aspects of automaton-formation games. In addition to the general game, we consider classes of instances that have to do with the network, the specifications, or to their combination. Recall that the network can be viewed as a WFA \mathcal{A} . We consider the following classes of WFAs: (1) *all-accepting*, in which all the states of \mathcal{A} are accepting, thus its language is prefix closed (2) *uniform costs*, in which all edges have the same cost, and (3) *single letter*, in which \mathcal{A} is over a single-letter alphabet. We consider the following classes of specifications: (1) *single word*, where the language of each player is a single word, (2) *symmetric*, where all players have the same objective. We also consider classes of instances that are intersections of the above classes.

Each of the restricted classes we consider corresponds to a real-life variant of the general setting. Let us elaborate below on single-letter instances. The language of an automaton over a single letter $\{a\}$ induces a subset of \mathbb{N} , namely the numbers $k \in \mathbb{N}$ such that the automaton accepts a^k . Accordingly, single-letter instances correspond to settings in which a player specifies possible lengths of paths. Several communication protocols are based on the fact that a message must pass a pre-defined length before reaching its destination. This includes *onion routing*, where the message is encrypted in layers [33], or *proof-of-work* protocols that are used to deter denial of service attacks and other service abuses such as spam (e.g., [15]).

We provide a complete picture of the following questions for various classes of the game (for formal definitions, see Section 2): (i) Existence of a *pure Nash equilibrium*. That is, whether each instance of the game has a profile of pure strategies that constitutes a NE. As we show, unlike the case of classical network design games, a pure NE might not exist in general automaton-formation games and even in very restricted instances of it. (ii) The complexity of finding the *social optimum* (SO). The SO is a profile that minimizes the total cost of the edges used by all players; thus the one obtained when the players obey some centralized authority. We show that for some restricted instances finding the SO can be done efficiently, while for other restricted instances, the complexity agrees with the NP-completeness of classical network-formation games. (iii) An analysis of *equilibrium inefficiency*. It is well known that decentralized decision-making may lead to solutions that are sub-optimal from the point of view of society

as a whole. We quantify the inefficiency incurred due to selfish behavior according to the *price of anarchy* (PoA) [24,31] and *price of stability* (PoS) [2] measures. The PoA is the worst-case inefficiency of a Nash equilibrium (that is, the ratio between the worst NE and the SO). The PoS is the best-case inefficiency of a Nash equilibrium (that is, the ratio between the best NE and the SO). We show that while the PoA in automaton-formation games agrees with the one in classical network-formation games and is equal to the number of players, the PoS also equals the number of players, again already in very restricted instances. This is in contrast with classical network-formation games, where the PoS tends to *log* the number of players. Thus, the fact that players may choose to use edges several times significantly increases the challenge of finding a stable solution as well as the inefficiency incurred due to selfish behavior. We find this as the most technically challenging result of this work. We do manage to find structural restrictions on the network with which the social optimum is a NE.

The technical challenge of our setting is demonstrated in the seemingly easy instance in which all players have the same objective. Such *symmetric* instances are known to be the simplest to handle in all cost-sharing and congestion games studied so far. Specifically, in network-formation games, the social optimum in symmetric instances is also a NE and the PoS is 1. Moreover, in some games [18], computing a NE is PLS-complete in general, but solvable in polynomial time for symmetric instances. Indeed, once all players have the same objective, it is not conceivable that a player would want to deviate from the social-optimum solution, where each of the k players pays $\frac{1}{k}$ of the cost of the optimal solution. We show that, surprisingly, symmetric instances in AF-games are not simple at all. Specifically, the social optimum might not be a NE, and the PoS is at least $\frac{k}{k-1}$. In particular, for symmetric two-player AF games, we have that $PoS = PoA = 2$. We also show that the PoA equals the number of players already for very restricted instances.

Due to lack of space, some proofs are omitted and can be found in the full version, in the authors' homepages.

2 Preliminaries

2.1 Automaton-Formation Games

A *nondeterministic finite weighted automaton* on finite words (WFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, F, c \rangle$, where Σ is an alphabet, Q is a set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of accepting states, and $c : \Delta \rightarrow \mathbb{R}$ is a function that maps each transition to the cost of its formation [28]. A *run* of \mathcal{A} on a word $w = w_1, \dots, w_n \in \Sigma^*$ is a sequence of states $\pi = \pi^0, \pi^1, \dots, \pi^n$ such that $\pi^0 = q_0$ and for every $0 \leq i < n$ we have $\Delta(\pi^i, w_{i+1}, \pi^{i+1})$. The run π is *accepting* iff $\pi^n \in F$. The *length* of π is n , whereas its *size*, denoted $|\pi|$, is the number of different transitions in it. Note that $|\pi| \leq n$.

An *automaton-formation game* (AF game, for short) between k selfish players is a pair $\langle \mathcal{A}, O \rangle$, where \mathcal{A} is a WFA over some alphabet Σ and O is a k -tuple of regular languages over Σ . Thus, the objective of Player i is a regular language L_i , and he needs to choose a word $w_i \in L_i$ and an accepting run of \mathcal{A} on w_i in a way that minimizes his payments. The cost of each transition is shared by the players that use it in their

selected runs, where the share of a player in the cost of a transition e is proportional to the number of times e is used by the player. Formally, The set of strategies for Player i is $\mathcal{S}_i = \{\pi : \pi \text{ is an accepting run of } \mathcal{A} \text{ on some word in } L_i\}$. We assume that \mathcal{S}_i is not empty. We refer to the set $\mathcal{S} = \mathcal{S}_1 \times \dots \times \mathcal{S}_k$ as the set of *profiles* of the game.

Consider a profile $P = \langle \pi_1, \pi_2, \dots, \pi_k \rangle$. We refer to π_i as a sequence of transitions. Let $\pi_i = e_i^1, \dots, e_i^{\ell_i}$, and let $\eta_P : \Delta \rightarrow \mathbf{N}$ be a function that maps each transition in Δ to the number of times it is traversed by all the strategies in P , taking into an account several traversals in a single strategy. Denote by $\eta_i(e)$ the number of times e is traversed in π_i , that is, $\eta_i(e) = |\{1 \leq j \leq \ell_i : e_i^j = e\}|$. Then, $\eta_P(e) = \sum_{i=1 \dots k} \eta_i(e)$. The *cost of player i in the profile P* is

$$\text{cost}_i(P) = \sum_{e \in \pi_i} \frac{\eta_i(e)}{\eta_P(e)} c(e). \quad (1)$$

For example, consider the WFA \mathcal{A} depicted in Fig. 1. The label $e_1 : a, 1$ on the transition from q_0 to q_1 indicates that this transition, which we refer to as e_1 , traverses the letter a and its cost is 1. We consider a game between two players. Player 1's objective is the language is $L_1 = \{ab^i : i \geq 2\}$ and Player 2's language is $\{ab, ba\}$. Thus, $\mathcal{S}_1 = \{\{e_1, e_2, e_2\}, \{e_1, e_2, e_2, e_2\}, \dots\}$ and $\mathcal{S}_2 = \{\{e_3, e_4\}, \{e_1, e_2\}\}$. Consider the profile $P = \langle \{e_1, e_2, e_2\}, \{e_3, e_4\} \rangle$, the strategies in P are disjoint, and we have $\text{cost}_1(P) = 2 + 2 = 4$, $\text{cost}_2(P) = 1 + 3 = 4$. For the profile $P' = \langle \{e_1, e_2, e_2\}, \{e_1, e_2\} \rangle$, it holds that $\eta_1(e_1) = \eta_2(e_1)$ and $\eta_1(e_2) = 2 \cdot \eta_2(e_2)$. Therefore, $\text{cost}_1(P') = \frac{1}{2} + 2 = 2\frac{1}{2}$ and $\text{cost}_2(P') = \frac{1}{2} + 1 = 1\frac{1}{2}$.

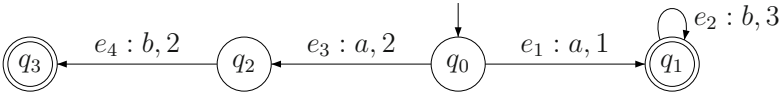


Fig. 1. An example of a WFA

We consider the following instances of AF games. Let $G = \langle \mathcal{A}, O \rangle$. We start with instances obtained by imposing restrictions on the WFA \mathcal{A} . In *one-letter* instances, \mathcal{A} is over a singleton alphabet, i.e., $|\Sigma| = 1$. When depicting such WFAs, we omit the letters on the transitions. In *all-accepting* instances, all the states in \mathcal{A} are accepting; i.e., $F = Q$. In *uniform-costs* instances, all the transitions in the WFA have the same cost, which we normalize to 1. Formally, for every $e \in \Delta$, we have $c(e) = 1$. We continue to restrictions on the objectives in O . In *single-word* instances, each of the languages in O consists of a single word. In *symmetric* instances, the languages in O coincide, thus the players all have the same objective. We also consider combinations on the restrictions. In particular, we say that $\langle \mathcal{A}, O \rangle$ is *weak* if it is one-letter, all states are accepting, costs are uniform, and objectives are single words. Weak instances are simple indeed – each player only specifies a length of a path he should patrol, ending anywhere in the WFA, where the cost of all transitions is the same. As we shall see, many of our hardness results and lower bounds hold already for the class of weak instances.

2.2 Nash Equilibrium, Social Optimum, and Equilibrium Inefficiency

For a profile P , a strategy π_i for Player i , and a strategy π , let $P[\pi_i \leftarrow \pi]$ denote the profile obtained from P by replacing the strategy for Player i by π . A profile $P \in \mathcal{S}$ is a *pure Nash equilibrium* (NE) if no player i can benefit from unilaterally deviating from his run in P to another run; i.e., for every player i and every run $\pi \in \mathcal{S}_i$ it holds that $\text{cost}_i(P[\pi_i \leftarrow \pi]) \geq \text{cost}_i(P)$. In our example, the profile P is not a NE, since Player 2 can reduce his payments by deviating to profile P' .

The (social) cost of a profile P , denoted $\text{cost}(P)$, is the sum of costs of the players in P . Thus, $\text{cost}(P) = \sum_{1 \leq i \leq k} \text{cost}_i(P)$. Equivalently, if we view P as a set of transitions, with $e \in P$ iff there is $\pi \in P$ for which $e \in \pi$, then $\text{cost}(P) = \sum_{e \in P} c(e)$. We denote by OPT the cost of an optimal solution; i.e., $OPT = \min_{P \in \mathcal{S}} \text{cost}(P)$. It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of society as a whole. We quantify the inefficiency incurred due to self-interested behavior according to the *price of anarchy* (PoA) [24,31] and *price of stability* (PoS) [2] measures. The PoA is the worst-case inefficiency of a Nash equilibrium, while the PoS measures the best-case inefficiency of a Nash equilibrium. Formally,

Definition 1. Let \mathcal{G} be a family of games, and let $G \in \mathcal{G}$ be a game in \mathcal{G} . Let $\Upsilon(G)$ be the set of Nash equilibria of the game G . Assume that $\Upsilon(G) \neq \emptyset$.

- The price of anarchy of G is the ratio between the maximal cost of a NE and the social optimum of G . That is, $PoA(G) = \max_{P \in \Upsilon(G)} \text{cost}(P)/OPT(G)$. The price of anarchy of the family of games \mathcal{G} is $PoA(\mathcal{G}) = \sup_{G \in \mathcal{G}} PoA(G)$.
- The price of stability of G is the ratio between the minimal cost of a NE and the social optimum of G . That is, $PoS(G) = \min_{P \in \Upsilon(G)} \text{cost}(P)/OPT(G)$. The price of stability of the family of games \mathcal{G} is $PoS(\mathcal{G}) = \sup_{G \in \mathcal{G}} PoS(G)$.

Uniform Sharing Rule: A different cost-sharing rule that could be adopted for automaton-formation games is the uniform sharing rule, according to which the cost of a transition e is equally shared by the players that traverse e , independent of the number of times e is traversed by each player. Formally, let $\kappa_P(e)$ be the number of runs that use the transition e at least once in a profile P . Then, the cost of including a transition e at least once in a run is $c(e)/\kappa_P(e)$. This sharing rule induces a potential game, where the potential function is identical to the one used in the analysis of the classical network design game [2]. Specifically, let $\Phi(P) = \sum_{e \in E} c(e) \cdot H(\kappa_P(e))$, where $H_0 = 0$, and $H_k = 1 + 1/2 + \dots + 1/k$. Then, $\Phi(P)$ is a potential function whose value reduces with every improving step of a player, thus a pure NE exists and BRD is guaranteed to converge². The similarity with classical network-formation games makes the study of this setting straightforward. Thus, throughout this paper we only consider the proportional sharing rule as defined in (1) above.

² Best-response-dynamics (BRD) is a local-search method where in each step some player is chosen and plays his best-response strategy, given that the strategies of the other players do not change.

3 Properties of Automaton-Formation Games

In this section we study the theoretical properties of AF games: existence of NE and equilibrium inefficiency. We show that AF games need not have a pure Nash equilibrium. This holds already in the very restricted class of weak instances, and is in contrast with network-formation games. There, BRD converges and a pure NE always exists. We then analyze the PoS in AF games and show that there too, the situation is significantly less stable than in network-formation games.

Theorem 1. *Automaton-formation games need not have a pure NE. This holds already for the class of weak instances.*

Proof. Consider the WFA \mathcal{A} depicted in Fig. 2 and consider a game with $k = 2$ players. The language of each player consists of a single word. Recall that in one-letter instances we care only about the lengths of the objective words. Let these be ℓ_1 and ℓ_2 , with $\ell_1 \gg \ell_2 \gg 0$ that are multiples of 12. For example, $\ell_1 = 30000, \ell_2 = 300$. Let C_3 and C_4 denote the cycles of length 3 and 4 in \mathcal{A} , respectively. Let D_3 denote the path of length 3 from q_0 to q_1 . Every run of \mathcal{A} consists of some repetitions of these cycles possibly with one pass on D_3 .

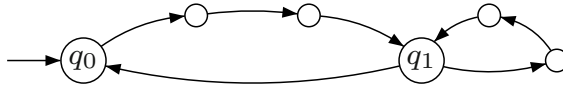


Fig. 2. A weak instance of AF games with no NE

We claim that no pure NE exists in this instance. Since we consider long runs, the fact that the last cycle might be partial is ignored in the calculations below. We first show that the only candidate runs for Player 1 that might be part of a NE profile are $\pi_1 = (C_4)^{\frac{\ell_1}{4}}$ and $\pi'_1 = D_3 \cdot (C_3)^{\frac{\ell_1}{3}-1}$. If Player 1 uses both C_3 and C_4 multiple times, then, given that $\ell_1 \gg \ell_2$, he must almost fully pay for at least one of these cycles, thus, deviating to the run that repeats this fully-paid cycle is beneficial.

When Player 1 plays π_1 , Player 2's best response is $\pi_2 = (C_4)^{\frac{\ell_2}{4}}$. In the profile $\langle \pi_1, \pi_2 \rangle$, Player 1 pays almost all the cost of C_4 , so the players' costs are $(4 - \varepsilon, \varepsilon)$. This is not a NE. Indeed, since $\ell_2 \gg 0$, then by deviating to π'_1 , the share of Player 1 in D_3 reduces to almost 0, and the players' costs in $\langle \pi'_1, \pi_2 \rangle$, are $(3 + \varepsilon, 4 - \varepsilon)$. This profile is not a NE as Player 2's best response is $\pi'_2 = D_3 \cdot (C_3)^{\frac{\ell_2}{3}-1}$. Indeed, in the profile $\langle \pi'_1, \pi'_2 \rangle$, the players' costs are $(4.5 - \varepsilon, 1.5 + \varepsilon)$ as they share the cost of D_3 and Player 1 pays almost all the cost of C_3 . This is not a NE either, as Player 1 would deviate to the profile $\langle \pi_1, \pi'_2 \rangle$, in which the players' costs are $(4 - \varepsilon, 3 + \varepsilon)$. The latter is still not a NE, as Player 2 would head back to $\langle \pi_1, \pi_2 \rangle$. We conclude that no NE exists in this game. \square

The fact that a pure NE may not exist is a significant difference between standard cost-sharing games and AF games. The bad news do not end here and extend to equilibrium inefficiency. We first note that the cost of any NE is at most k times the social

optimum (as otherwise, some player pays more than the cost of the SO and can benefit from migrating to his strategy in the SO). Thus, it holds that $PoS \leq PoA \leq k$. The following theorem shows that this is tight already for highly restricted instances.

Theorem 2. *The PoS in AF games equals the number of players. This holds already for the class of weak instances.*

Proof. We show that for every $k, \delta > 0$ there exists a simple game with k players for which the PoS is more than $k - \delta$. Given k and δ , let r be an integer such that $r > \max\{k, \frac{k-1}{\delta} - 1\}$. Consider the WFA \mathcal{A} depicted in Fig. 3. Let $L = (\ell_1, \ell_2, \dots, \ell_k)$ for $\ell_2 = \dots = \ell_k$ and $\ell_1 \gg \ell_2 \gg r$ denote the lengths of the objective words. Thus, Player 1 has an ‘extra-long word’ and the other $k - 1$ players have words of the same, long, length. Let C_r and C_{r+1} denote, respectively, the cycles of length r and $r + 1$ to the right of q_0 . Let D_r denote the path of length r from q_0 to q_1 , and let D_{kr} denote the ‘lasso’ consisting of the kr -path and the single-edge loop to the left of q_0 .

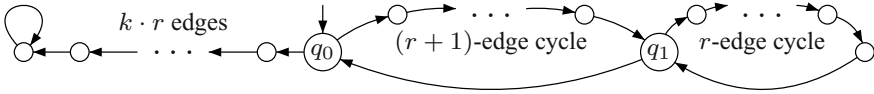


Fig. 3. A weak instance of AF games for which $PoS = k$

The social optimum of this game is to buy C_{r+1} . Its cost is $r + 1$. However, as we show, the profile P in which all players use D_{kr} is the only NE in this game. We first show that P is a NE. In this profile, Player 1 pays $r + 1 - \varepsilon$ and each other player pays $r + \varepsilon/(k - 1)$. No player will deviate to a run that includes edges from the right side of \mathcal{A} . Next, we show that P is the only NE of this game: Every run on the right side of \mathcal{A} consists of some repetitions of C_{r+1} and C_r , possibly with one traversal of D_r . Since we consider long runs, the fact that the last cycle might be partial is ignored in the calculations below.

In the social optimum profile, Player 1 pays $r + 1 - \varepsilon$ and each of the other players pays $\varepsilon/(k - 1)$. The social optimum is not a NE as Player 1 would deviate to $D_r \cdot C_r^*$ and will reduce his cost to $r + \varepsilon'$. The other players, in turn, will also deviate to $D_r \cdot C_r^*$. In the profile in which they are all selecting a run of the form $D_r \cdot C_r^*$, Player 1 pays $r + r/k - \varepsilon > r + 1$ and prefers to return to C_{r+1}^* . The other players will join him sequentially, until the non-stable social optimum is reached. Thus, no NE that uses the right part of \mathcal{A} exists. Finally, it is easy to see that no run that involves edges from both the left and right sides of \mathcal{A} or includes both C_{r+1} and C_r can be part of a NE.

The cost of the NE profile is $kr + 1$ and the PoS is therefore $\frac{kr+1}{r+1} = k - \frac{k-1}{r+1} > k - \delta$. \square

4 Computational Complexity Issues in AF Games

In this section we study the computational complexity of two problems: finding the cost of the social optimum and finding the best-response of a player. Recall that the

social optimum (SO) is a profile that minimizes the total cost the players pay. It is well-known that finding the social optimum in a network-formation game is NP-complete. We show that this hardness is carried over to simple instances of AF games. On the positive side, we identify non-trivial classes of instances, for which it is possible to compute the SO efficiently. The other issue we consider is the complexity of finding the best strategy of a single player, given the current profile, namely, the best-response of a player. In network-formation games, computing the best-response reduces to a shortest-path problem, which can be solved efficiently. We show that in AF games, the problem is NP-complete.

The proofs of the following theorems can be found in the full version. The reductions we use are from the set-cover problem, where choice of sets are related to choice of transitions.

Theorem 3. *Finding the value of the social optimum in AF games is NP-complete. Moreover, finding the social optimum is NP-complete already in single-worded instances that are also uniform-cost and are either single-lettered or all-accepting.*

The hardness results in Theorem 3 for single-word specification use one of two properties: either there is more than one letter, or not all states are accepting. We show that finding the SO in instances that have both properties can be done efficiently, even for specifications with arbitrary number of words.

For a language L_i over $\Sigma = \{a\}$, let $short(i) = \min_j \{a^j \in L_i\}$ denote the length of the shortest word in L_i . For a set O of languages over $\Sigma = \{a\}$, let $\ell_{max}(O) = \max_i short(i)$ denote the length of the longest shortest word in O . Clearly, any solution, in particular the social optimum, must include a run of length $\ell_{max}(O)$. Thus the cost of the social optimum is at least the cost of the cheapest run of length $\ell_{max}(O)$. Moreover, since the WFA is single-letter and all-accepting, the other players can choose runs that are prefixes of this cheapest run, and no additional transitions should be acquired. We show that finding the cheapest such run can be done efficiently.

Theorem 4. *The cost of the social optimum in a single-letter all-accepting instance $\langle \mathcal{A}, O \rangle$ is the cost of the cheapest run of length $\ell_{max}(O)$. Moreover, this cost can be found in polynomial time.*

We turn to prove the hardness of finding the best-response of a player. Our proof is valid already for a single player that needs to select a strategy on a WFA that is not used by other players (one-player game).

Theorem 5. *Finding the best-response of a player in AF games is NP-complete.*

5 Tractable Instances of AF Games

In the example in Theorem 1, Player 1 deviates from a run on the shortest (and cheapest) possible path to a run that uses a longer path. By doing so, most of the cost of the original path, which is a prefix of the new path and accounts to most of its cost, goes to Player 2. We consider *semi-weak* games in which the WFA is uniform-cost, all-accepting, and single-letter, but the objectives need not be a single word. We identify a

property of such games that prevents this type of deviation and which guarantees that the social optimum is a NE. Thus, we identify a family of AF games in which a NE exists, finding the SO is easy, and the PoS is 1.

Definition 2. Consider a semi-weak game $\langle \mathcal{A}, O \rangle$. A lasso is a path $u \cdot v$, where u is a simple path that starts from the initial state and v is a simple cycle. A lasso ν is minimal in \mathcal{A} if \mathcal{A} does not have shorter lassos. Note that for minimal lassos $u \cdot v$, we have that $u \cap v = \emptyset$. We say that \mathcal{A} is resistant if it has no cycles or there is a minimal lasso $\nu = u \cdot v$ such that for every other lasso ν' we have $|u \setminus \nu'| + |v| \leq |\nu' \setminus \nu|$.

Consider a resistant weak game $\langle \mathcal{A}, O \rangle$. In order to prove that the social optimum is a NE, we proceed as follows. Let ν be the lasso that is the witness for the resistance of \mathcal{A} . We show that the profile S^* in which all players choose runs that use only the lasso ν or a prefix of it, is a NE. The proof is technical and we go over all the possible types of deviations for a player and use the weak properties of the network along with its resistance. By Theorem 4, the cost of the profile is the SO. Hence the following. The full proof can be found in full version.

Theorem 6. For resistant semi-weak games, the social optimum is a NE.

A corollary of Theorem 6 is the following:

Corollary 1. For resistant semi-weak games, we have $PoS = 1$.

We note that resistance can be defined also in WFAs with non-uniform costs, with $cost(\nu)$ replacing $|\nu|$. Resistance, however, is not sufficient in the *slightly* stronger model where the WFA is single-letter and all-accepting but not uniform-cost. Indeed, given k , we show a such a game in which the PoS is kx , for a parameter x that can be arbitrarily close to 1. Consider the WFA A in Fig. 5. Note that \mathcal{A} has a single lasso and is thus a resistant WFA. The parameter ℓ_1 is a function of x , and the players' objectives are single words of lengths $\ell_1 \gg \ell_2 \gg \dots \gg \ell_k \gg 0$. Similar to the proof of Theorem 2, there is only one NE in the game, which is when all players choose the left chain. The social optimum is attained when all players use the self-loop, and thus for a game in this family, $PoS = \frac{k \cdot x}{1}$. Since x tends to 1, we have $PoS = k$ for resistant all-accepting single-letter games. The proof can be found in the full version.

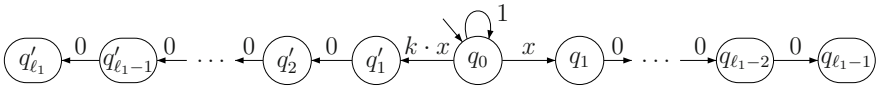


Fig. 4. A resistant all-accepting single-letter game in which the PoS tend to k

6 Surprises in Symmetric Instances

In this section we consider the class of symmetric instances, where all players share the same objective, that is, there exists a language L , such that for all $1 \leq i \leq k$, we have $L_i = L$. In such instances it is tempting to believe that the social optimum is also a NE, as all players evenly share the cost of the solution that optimizes their common

objective. While this is indeed the case in all known symmetric games, we show that, surprisingly, this is not valid for AF-games, in fact already for the class of one-letter, all accepting, unit-cost and single-word instances.

Before we show that the PoS can be larger than 1, let us elaborate on the PoA. It is easy to see that in symmetric AF games, we have $PoA = k$. This bound is achieved, as in the classic network-formation game, by a network with two parallel edges labeled by a and having costs k and 1. The players all have the same specification $L = \{a\}$. The profile in which all players select the expensive path is a NE. We show that $PoA = k$ is achieved even for weak symmetric instances.

Theorem 7. *The PoA equals the number of players, already for weak symmetric instances.*

Proof. We show a lower bound of k . The example is a generalization of the PoA in cost sharing games [2]. For k players, consider the weak instance depicted in Fig. 6, where all players have the length k . Intuitively, the social optimum is attained when all players use the loop $\langle q_0, q_0 \rangle$ and thus $OPT = 1$. The worst NE is when all players use the run $q_0 q_1 \dots q_k$, and its cost is clearly k . Formally, there are two NEs in the game:

- The cheap NE is when all players use the loop $\langle q_0, q_0 \rangle$. This is indeed a NE because if a player deviates, he must buy at least the transition $\langle q_0, q_1 \rangle$. Thus, he pays at least 1, which is higher than $\frac{1}{k}$, which is what he pays when all players use the loop.
- The expensive NE is when all players use the run q_0, q_1, \dots, q_k . This is a NE because a player has two options to deviate. Either to the run that uses only the loop, which costs 1, or to a run that uses the loop and some prefix of q_0, q_1, \dots, q_k , which costs at least $1 + \frac{1}{k}$. Since he currently pays 1, he has no intention of deviating to either runs.

Since the cheap NE costs 1 and the expensive one costs k , we get $PoA = k$. □

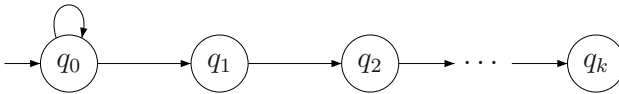


Fig. 5. The WFA \mathcal{A} for which a symmetric game with $|L| = 1$ achieves $PoA = k$

We now turn to the PoS analysis. We first demonstrate the anomaly of having $PoS > 1$ with the two-player game appearing in Fig. 6. All the states in the WFA \mathcal{A} are accepting, and the objectives of both players is a single long word. The social optimum is when both players traverse the loop q_0, q_1, q_0 . Its cost is $2 + \epsilon$, so each player pays $1 + \frac{\epsilon}{2}$. This, however, is not a NE, as Player 1 (or, symmetrically, Player 2) prefers to deviate to the run $q_0, q_1, q_1, q_1, \dots$, where he pays the cost of the loop q_1, q_1 and his share in the transition from q_0 to q_1 . We can choose the length of the objective word and ϵ so that this share is smaller than $\frac{\epsilon}{2}$, justifying his deviation. Note that the new situation is not a NE either, as Player 2, who now pays 2, is going to join Player 1, resulting in an unfortunate NE in which both players pay 1.5.

It is not hard to extend the example from Fig. 6 to $k > 2$ players by changing the 2-valued transition to k , and adjusting ϵ and the lengths of the players accordingly. The

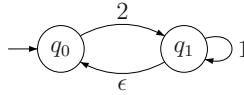


Fig. 6. The WFA \mathcal{A} for which the SO in a symmetric game is not a NE

social optimum and the only NE are as in the two-player example. Thus, the PoS in the resulting game is $1 + \frac{1}{k}$.

A higher lower bound of $1 + \frac{1}{k-1}$ is shown in the following theorem. Although both bounds tend to 1 as k grows to infinity, this bound is clearly stronger. Also, for $k = 2$, the bound $PoS = 1 + \frac{1}{k-1} = 2$ is tight. We conjecture that $\frac{k}{k-1}$ is tight for every $k > 2$.

Theorem 8. *In a symmetric k -player game, the PoS is at least $\frac{k}{k-1}$.*

Proof. For $k \geq 2$, we describe a family of symmetric games for which the PoS tends to $\frac{k}{k-1}$. For $n \geq 1$, the game $G_{\epsilon,n}$ uses the WFA that is depicted in Figure 7. Note that this is a one-letter instance in which all states are accepting. The players have an identical specification, consisting of a single word w of length $\ell \gg 0$. We choose ℓ and $\epsilon = \epsilon_0 > \dots > \epsilon_{n-1}$ as follows. Let C_0, \dots, C_n denote, respectively, the cycles with costs $(k^n + \epsilon_0), (k^{n-1} + \epsilon_1), \dots, (k + \epsilon_{n-1}), 1$. Let r_0, \dots, r_n be lasso-runs on w that end in C_0, \dots, C_n , respectively. Consider $0 \leq i \leq n - 1$ and let P_i be the profile in which all players choose the run r_i . We choose ℓ and ϵ_i so that Player 1 benefits from deviating from P_i to the run r_{i+1} , thus P_i is not a NE. Note that by deviating from r_i to r_{i+1} , Player 1 pays the same amount for the path leading to C_i . However, his share of the loop C_i decreases drastically as he uses the k^{n-i} -valued transition only once whereas the other players use it close to ℓ times. On the other hand, he now buys the loop C_{i+1} by himself. Thus, the change in his payment change is $\frac{1}{k} \cdot (k^{n-i} + \epsilon_i) - (\epsilon' + k^{n-(i+1)} + \epsilon_{i+1})$. We choose ϵ_{i+1} and ℓ so that $\frac{\epsilon_i}{k} > \epsilon' + \epsilon_{i+1}$, thus the deviation is beneficial.

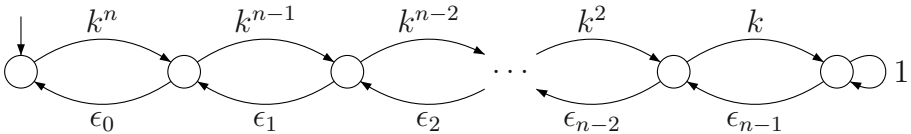


Fig. 7. The network of the identical-specification game $G_{\epsilon,n}$, in which PoS tends to $\frac{k}{k-1}$

We claim that the only NE is when all players use the run r_n . Indeed, it is not hard to see that every profile in which a player selects a run that is not from r_0, \dots, r_n cannot be a NE. Also, a profile in which two players select runs r_i and r_j , for $1 \leq i < j \leq n$, cannot be a NE as the player using r_i can decrease his payment by joining the other player in r_j . Finally, by our selection of $\epsilon_1, \dots, \epsilon_n$, and ℓ , every profile in which all the players choose the run r_i , for $0 \leq i \leq n - 1$, is not a NE.

Clearly, the social optimum is attained when all players choose the run r_0 , and its cost is $k^n + \epsilon$. Since the cost of the only NE in the game is $\sum_{0 \leq i \leq n} k^{n-i}$, the PoS in this family of games tends to $\frac{k}{k-1}$ as n grows to infinity and ϵ to 0. \square

Finally, we note that our hardness result in Theorem 5 implies that finding the social optimum in a symmetric AF-game is NP-complete. Indeed, since the social optimum is the cheapest run on some word in L , finding the best-response in a one-player game is equivalent to finding the social optimum in a symmetric game. This is contrast with other cost-sharing and congestion game (e.g. [18], where the social optimum in symmetric games can be computed using a reduction to max-flow).

7 Conclusions and Future Work

Our results on the stability of AF games are mostly negative. We identified some stable cases and we believe that additional positive results can be derived for restricted classes of instances. As we suggest below, these restrictions can be characterized by the structure of the automaton or by the set of players' objectives.

Ordinary open problems include the study of approximate-NE, networks with profits, capacitated networks, and coordinated deviation. We highlight below several interesting directions for future work that are specific to the study of AF games.

1. Our lower bounds use WFAs with cycles. We believe that for acyclic all-accepting one-letter instances, the PoS for can be bounded by a constant. Specifically, for k players, we conjecture that $PoS = \sum_{i=1}^k \frac{1}{2^{i-1}}$, which is bounded by 2. In the full version we present a lower bound of this value that is valid already for automata consisting of disjoint paths. Such an analysis will provide a nice distinction between the classical network-formation game, for which $PoS = \Theta(\log k)$, and our game, even when all players use a *simple* path for their run. We note that it is possible to restrict the class of languages in the objectives so that the players have no incentive not to use simple paths for their runs. For example, when the languages are *closed under infix disposal* (that is, if $x \cdot y \cdot z \in L$, for $x, y, z \in \Sigma^*$, then $x \cdot z \in L$).
2. Other presumably more stable games are those in which the range of costs or the ratio between the maximal and the minimal transition costs is bounded, or when the ratio between the longest and the shortest word in the objective languages is bounded. Indeed, bounding these ratios also bounds the proportion in which costs are shared, making the game closer to one with a uniform sharing rule.
3. AF-games are an example of cost-sharing games in which players' strategies are *multisets* of resources. In such games, a player may need multiple uses of the same resource, and his share in the resource cost is proportional to the number of times he uses the resource. Our results imply that, in general, such games are less stable than classical cost-sharing games. It is desirable to study more settings of such games, and to characterize non-trivial instances that arise in practice and for which the existence of pure NE can be shown, and its inefficiency can be bounded. In the context of formal methods, an appealing application is that of *synthesis from components*, where the resources are functions from a library, and agents need to synthesize their objectives using such functions, possibly by a repeated use of some functions.

4. For symmetric AF games, we leave open the problem of NE existence as well as the problem of finding an upper-bound for the PoS for $k > 2$.

Recall that in planning, the WFA models a production system in which transitions correspond to actions. In such cases, the objectives of the players may be languages of *infinite* words, describing desired on-going behaviors. The objectives can be specified by linear temporal logic or nondeterministic Büchi automata, and each player has to select a lasso computation or accepting run for a word in his language. The setting of infinite words involves transitions that are taken infinitely often and calls for new sharing rules. When the sharing rule refers to the frequency in which transitions are taken, we obtain a proportional sharing rule that is similar to the one studied here. One can also follow a sharing rule in which all players that traverse a transition infinitely often share its cost evenly, perhaps with some favorable proportion towards players that use it only finitely often. This gives rise to simpler sharing rules, which seem more stable.

Acknowledgments. We thank Michal Feldman, Noam Nisan, and Michael Schapira for helpful discussions.

References

1. Albers, S., Elits, S., Even-Dar, E., Mansour, Y., Roditty, L.: On Nash Equilibria for a Network Creation Game. In: Proc. 17th SODA, pp. 89–98 (2006)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, É., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. *SIAM J. Comput.* 38(4), 1602–1623 (2008)
3. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. *Journal of the ACM* 49(5), 672–713 (2002)
4. Aminof, B., Kupferman, O., Lampert, R.: Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms* 6(2) (2010)
5. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. *Distributed Computing* 2, 117–126 (1987)
6. Brihaye, T., Bruyère, V., De Pril, J., Gimbert, H.: On subgame perfection in quantitative reachability games. *Logical Methods in Computer Science* 9(1) (2012)
7. Chatterjee, K.: Nash equilibrium for upward-closed objectives. In: Ésik, Z. (ed.) *CSL 2006*. LNCS, vol. 4207, pp. 271–286. Springer, Heidelberg (2006)
8. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Games with secure equilibria. *Theoretical Computer Science* 365(1-2), 67–82 (2006)
9. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. In: Caires, L., Vasconcelos, V.T. (eds.) *CONCUR 2007*. LNCS, vol. 4703, pp. 59–73. Springer, Heidelberg (2007)
10. 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)
11. Chen, H., Roughgarden, T.: Network Design with Weighted Players. *Theory of Computing Systems* 45(2), 302–324 (2009)
12. Correa, J.R., Schulz, A.S., Stier-Moses, N.E.: Selfish Routing in Capacitated Networks. *Mathematics of Operations Research* 29, 961–976 (2004)

13. Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)
14. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer (2009)
15. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
16. Fisman, D., Kupferman, O., Lustig, Y.: Rational synthesis. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 190–204. Springer, Heidelberg (2010)
17. Fabrikant, A., Luthra, A., Maneva, E., Papadimitriou, C., Shenker, S.: On a network creation game. In: Proc. 22nd PODC, pp. 347–351 (2003)
18. Fabrikant, A., Papadimitriou, C., Talwar, K.: The Complexity of Pure Nash Equilibria. In: Proc. 36th STOC, pp. 604–612 (2004)
19. Feldman, M., Tamir, T.: Conflicting Congestion Effects in Resource Allocation Games. *Journal of Operations Research* 60(3), 529–540 (2012)
20. von Falkenhausen, P., Harks, T.: Optimal Cost Sharing Protocols for Scheduling Games. In: Proc. 12th EC, pp. 285–294 (2011)
21. de Giacomo, G., Vardi, M.Y.: Automata-Theoretic Approach to Planning for Temporally Extended Goals. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS, vol. 1809, pp. 226–238. Springer, Heidelberg (2000)
22. Harel, D., Pnueli, A.: On the development of reactive systems. In: LMCS. NATO Advanced Summer Institutes, vol. F-13, pp. 477–498. Springer (1985)
23. Herzog, S., Shenker, S., Estrin, D.: Sharing the “Cost” of Multicast Trees: An Axiomatic Analysis. *IEEE/ACM Transactions on Networking* (1997)
24. Koutsoupias, E., Papadimitriou, C.: Worst-case Equilibria. *Computer Science Review* 3(2), 65–69 (2009)
25. Kupferman, O., Tamir, T.: Coping with selfish on-going behaviors. *Information and Computation* 210, 1–12 (2012)
26. Mavronicolas, M., Milchtaich, I., Monien, B., Tiemann, K.: Congestion Games with Player-Specific Constants. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 633–644. Springer, Heidelberg (2007)
27. Milchtaich, I.: Weighted Congestion Games With Separable Preferences. *Games and Economic Behavior* 67, 750–757 (2009)
28. Mohri, M.: Finite-state transducers in language and speech processing. *Computational Linguistics* 23(2), 269–311 (1997)
29. Monderer, D., Shapley, L.: Potential Games. *Games and Economic Behavior* 14, 124–143 (1996)
30. Moulin, H., Shenker, S.: Strategyproof Sharing of Submodular Costs: Budget Balance Versus Efficiency. *Journal of Economic Theory* 18, 511–533 (2001)
31. Papadimitriou, C.: Algorithms, Games, and the Internet. In: Proc 33rd STOC, pp. 749–753 (2001)
32. Paes Leme, R., Syrgkanis, V., Tardos, E.: The curse of simultaneity. In: Innovations in Theoretical Computer Science (ITCS), pp. 60–67 (2012)
33. Reed, M.G., Syverson, P.F., Goldschlag, D.M.: Anonymous Connections and Onion Routing. *IEEE J. on Selected Areas in Communication*, Issue on Copyright and Privacy Protection (1998)
34. Rosenthal, R.W.: A Class of Games Possessing Pure-Strategy Nash Equilibria. *International Journal of Game Theory* 2, 65–67 (1973)
35. Tardos, E., Wexler, T.: Network Formation Games and the Potential Function Method. In: *Algorithmic Game Theory*. Cambridge University Press (2007)
36. Vöcking, B.: Selfish Load Balancing. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. (eds.) *Algorithmic Game Theory*, ch. 20. Cambridge University Press (2007)

Playing with Probabilities in Reconfigurable Broadcast Networks

Nathalie Bertrand¹, Paulin Fournier², and Arnaud Sangnier³

¹ Inria Rennes Bretagne Atlantique

² ENS Cachan Antenne de Bretagne

³ LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, CNRS

Abstract. We study verification problems for a model of network with the following characteristics: the number of entities is parametric, communication is performed through broadcast with adjacent neighbors, entities can change their internal state probabilistically and reconfiguration of the communication topology can happen at any time. The semantics of such a model is given in terms of an infinite state system with both non-deterministic and probabilistic choices. We are interested in qualitative problems like whether there exists an initial topology and a resolution of the non-determinism such that a configuration exhibiting an error state is almost surely reached. We show that all the qualitative reachability problems are decidable and some proofs are based on solving a 2-player game played on the graphs of a reconfigurable network with broadcast with parity and safety objectives.

1 Introduction

Providing methods to analyze and verify distributed systems is a complex task and this for several reasons. First there are different families of distributed systems depending on the communication means (shared memory or message passing), on the computing power of the involved entities, on the knowledge of the system provided to the entities (full knowledge, or local knowledge of their neighbors, or no knowledge at all) or on the type of communication topology that is considered (ring, tree, arbitrary graph, etc). Second, most of the protocols developed for distributed systems are supposed to work for an unbounded number of participants, hence in order to verify that a system behaves correctly, one needs to develop methods which allow to deal with such a parameter.

In [12], the authors propose a model which allows to take into account the main features of a family of distributed networks, namely ad-hoc networks. It characterizes the following aspects of such systems: the nodes in the network can only communicate with their neighbors using broadcast communication and the number of participants is unbounded. In this model, each entity behaves similarly following a protocol which is represented by a finite state machine performing three kinds of actions (1) broadcast of a message, (2) reception of a message and (3) internal action. Furthermore, the communication topology does not change during an execution and no entity is deleted or added during an execution.

The *control state reachability problem* consists then in determining whether there exists an initial number of entities in a communication topology such that it is possible to reach a configuration where at least one process is in a specific control state (considered for instance as an error state). The main difficulty in solving such a problem lies in the fact that both the number of processes and the initial communication topology are parameters, for which one wishes to find an instantiation. In [12], it is proven that this problem is undecidable but becomes decidable when considering non-deterministic *reconfiguration* of the communication topology, i.e. when at any moment the nodes can move and change their neighborhood. In [11] this latter problem is shown to be P-complete. An other way to gain decidability in such so called broadcast networks consists in restricting the set of communication topologies to complete graphs (aka cliques) or bounded depth graphs [13] or acyclic directed graphs [1].

We propose here to extend the model of reconfigurable broadcast networks studied in [11] by allowing probabilistic internal actions, that is, a process can change its internal state according to a probabilistic distribution. Whereas the semantics of reconfigurable broadcast networks was given in term of an infinite state system with non-determinism (due to the different possibility of sending messages from different nodes and also to the non-determinism of the protocol itself), we obtain here an infinite state system with probabilistic and non-deterministic choices. On such a system we study the probabilistic version of the *control state reachability* by seeking for the existence of a scheduler resolving non-determinism which minimizes or maximizes the probability to reach a configuration exhibiting a specific state. We focus on the qualitative aspects of this problem by comparing probabilities only with 0 and 1. Note that another model of broadcast networks with probabilistic protocols was defined in [6]; it was however different: the communication topologies were necessarily cliques and decidability of qualitative probabilistic reachability only holds when the network size evolves randomly over time.

For finite state systems with non-determinism and probabilities (like finite state Markov Decision Processes), most verification problems are decidable [5], but when the number of states is infinite, they are much harder to tackle. The introduction of probabilities might even lead to the undecidability, for problems that are decidable in the non-probabilistic case. For instance for extensions of pushdown systems with non-deterministic and probabilistic choices, the model-checking problems of linear time or branching time logic are undecidable [14,8]. On the other hand, it is not always the case that the introduction of probabilistic transitions leads to undecidability but then dedicated verification methods have to be invented, as it is the case for instance for nondeterministic probabilistic lossy channel systems [4]. Even if for well-structured infinite state systems [2,15] (where a monotonic well-quasi order is associated to the set of configurations), a class to which belong the broadcast reconfigurable networks of [12], a general framework for the extension to purely probabilistic transitions has been proposed in [3], it seems hard to adapt such a framework to the case with probabilistic and non-deterministic choices.

In this paper, we prove that the qualitative versions of the *control state reachability problem* for reconfigurable broadcast networks with probabilistic internal choices are all decidable. For some of these problems, like finding a scheduler such that the probability of reaching a control state is equal to 1, our proof technique is based on a reduction to a 2 player game played on infinite graphs with safety and parity objectives. This translation is inspired by a similar translation for finite state systems (see for instance [9]). However when moving to infinite state systems, two problems raise: first whether the translation is correct when the system has an infinite number of states, and then whether we can solve the game. In our translation, we answer the first question in Section 3 and the second one in Section 4. We also believe that the parity game we define on broadcast reconfigurable networks could be used to verify other properties on such systems. Due to lack of space, omitted details and proofs can be found in [7].

2 Networks of Probabilistic Reconfigurable Protocols

2.1 Preliminary Definitions

For a finite or denumerable set E , we write $\text{Dist}(E)$, for the set of discrete probability distributions over E , that is the set of functions $\delta : E \mapsto [0, 1]$ such that $\sum_{e \in E} \delta(e) = 1$. We now give the definition of a $1 - \frac{1}{2}$ player game, which will be later used to provide the semantics of our model.

Definition 1 ($1 - \frac{1}{2}$ player game). *A $1 - \frac{1}{2}$ player game is a tuple $\mathcal{M} = (\Gamma, \Gamma^{(1)}, \Gamma^{(p)}, \rightarrow, \text{prob})$ where Γ is a denumerable set of configurations (or vertices) partitioned into the configurations of Player 1 $\Gamma^{(1)}$ and the probabilistic configurations $\Gamma^{(p)}$; $\rightarrow : \Gamma^{(1)} \mapsto \Gamma$ is the non deterministic transition relation; $\text{prob} : \Gamma^{(p)} \mapsto \text{Dist}(\Gamma^{(1)})$ is the probabilistic transition relation.*

For a tuple $(\gamma, \gamma') \in \rightarrow$, we will sometimes use the notations $\gamma \rightarrow \gamma'$. A *finite path* in the game $\mathcal{M} = (\Gamma, \Gamma^{(1)}, \Gamma^{(p)}, \rightarrow, \text{prob})$ is a finite sequence of configurations $\gamma_0 \gamma_1 \dots \gamma_k$ such that for all $0 \leq i \leq k - 1$, if $\gamma_i \in \Gamma^{(1)}$ then $\gamma_i \rightarrow \gamma_{i+1}$ and otherwise $\text{prob}(\gamma_i)(\gamma_{i+1}) > 0$; moreover we will say that such a path starts from the configuration γ_0 . An *infinite path* is an infinite sequence $\rho \in \Gamma^\omega$ such that any finite prefix of ρ is a finite path. Furthermore we will say that a path ρ is *maximal* if it is infinite or it is finite and there does not exist a configuration γ such that $\rho\gamma$ is a finite path (in other words a finite maximal path ends up in a deadlock configuration). The set of maximal paths is denoted Ω .

A *scheduler* in the game $\mathcal{M} = (\Gamma, \Gamma^{(1)}, \Gamma^{(p)}, \rightarrow, \text{prob})$ is a function $\pi : \Gamma^* \cdot \Gamma^{(1)} \mapsto \Gamma$ that assigns, to a finite sequence of configurations ending with a configuration in $\Gamma^{(1)}$, a successor configuration such that for all $\rho \in \Gamma^*$, $\gamma \in \Gamma^{(1)}$ and $\gamma' \in \Gamma$, if $\pi(\rho \cdot \gamma) = \gamma'$ then $\gamma \rightarrow \gamma'$. We denote by Π the set of schedulers for \mathcal{M} . Given a scheduler $\pi \in \Pi$, we say that a finite path $\gamma_0 \gamma_1 \dots \gamma_n$ *respects* the scheduler π if for every $i \in \{0 \dots n - 1\}$, we have that if $\gamma_i \in \Gamma^{(1)}$ then $\pi(\gamma_0 \dots \gamma_i) = \gamma_{i+1}$. Similarly we say that an infinite path $\rho = \gamma_0 \gamma_1 \dots$ *respects* the scheduler π if every finite prefix of ρ respects π .

Remark 1. Alternatively, a scheduler in the game $\mathcal{M} = (\Gamma, \Gamma^{(1)}, \Gamma^{(p)}, \rightarrow, prob)$ can be defined as what is often called a *scheduler with memory*. It is given by a set M called the *memory* together with a *strategic function* $\pi_M : \Gamma^{(1)} \times M \rightarrow \Gamma$, an *update function* $U_M : \Gamma^{(1)} \times M \times \Gamma \rightarrow M$, and an initialization function $I_M : \Gamma^{(0)} \rightarrow M$. Intuitively, the update function updates the memory state given the previous configuration, the current memory state and the current configuration. The two definitions for schedulers coincide, and we will use one or the other, depending on what is more convenient.

The set of paths starting from a configuration and respecting a scheduler represents a stochastic process. Given a measurable set of paths $\mathcal{A} \subseteq \Omega$, we denote by $\mathbb{P}(\mathcal{M}, \gamma, \pi, \mathcal{A})$ the probability of event \mathcal{A} for the infinite paths starting from the configuration $\gamma \in \Gamma$ and respecting the scheduler π . We define then extremal probabilities of the event \mathcal{A} starting from configuration γ as follows:

$$\mathbb{P}_{\inf}(\mathcal{M}, \gamma, \mathcal{A}) = \inf_{\pi \in \Pi} \mathbb{P}(\mathcal{M}, \gamma, \pi, \mathcal{A}) \text{ and } \mathbb{P}_{\sup}(\mathcal{M}, \gamma, \mathcal{A}) = \sup_{\pi \in \Pi} \mathbb{P}(\mathcal{M}, \gamma, \pi, \mathcal{A})$$

2.2 Networks of Probabilistic Reconfigurable Protocols

We introduce in this section our model to represent the behavior of a communication protocol in a network. This model has three main features : the communication in the network is performed via broadcast communication, each node in the network can change its internal state probabilistically and the communication topology can change dynamically. This model extends the one proposed in [11] with probability and can be defined in two steps. First, a configuration of the network is represented by a labelled graph in which the edges characterize the communication topology and the label of the nodes give the state and whether they are the next node which will perform an action or not.

Definition 2 (\mathcal{L} -graph). *Given \mathcal{L} a set of labels, an \mathcal{L} -graph is a labelled undirected graph $G = (V, E, L)$ where: V is a finite set of nodes, $E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$ is a finite set of edges such that $(v, v') \in E$ iff $(v', v) \in E$, and $L : V \mapsto \mathcal{L}$ is a labelling function.*

We denote by $\mathcal{G}_{\mathcal{L}}$ the infinite set of \mathcal{L} -graphs and for a graph $G = (V, E, L)$, let $L(G) \subseteq \mathcal{L}$ be the set of all the labels present in G , i.e. $L(G) = \{L(v) \mid v \in V\}$. For an edge $(v, v') \in E$, we use the notation $v \sim_G v'$ to denote that the two vertices v and v' are adjacent in G . When the considered graph G is made clear from the context, we may omit G and write simply $v \sim v'$.

Then, in our model, each node of the network behaves similarly following a protocol whose description is given by what can be seen as a finite $1 - \frac{1}{2}$ player game labelled with a communication alphabet.

Definition 3 (Probabilistic protocol). *A probabilistic protocol is a tuple $\mathcal{P} = (Q, Q^{(1)}, Q^{(P)}, q_0, \Sigma, \Delta, \Delta^{int})$ where Q is a finite set of control states partitioned into $Q^{(1)}$, the states of Player 1, and $Q^{(P)}$ the probabilistic states; $q_0 \in Q^{(1)}$ is the initial control state; Σ is a finite message alphabet; $\Delta \subseteq (Q^{(1)} \times \{!!a, ??a \mid a \in$*

$\Sigma\} \times Q^{(1)} \cup (Q^{(1)} \times \{\varepsilon\} \times Q)$ is the transition relation; $\Delta^{int} : Q^{(P)} \mapsto \text{Dist}(Q^{(1)})$ is the internal probabilistic transition relation.

The label $!!a$ [resp. $??a$] represents the broadcast [resp. reception] of the message $a \in \Sigma$, whereas ε represents an internal action. Given a state $q \in Q$ and a message $a \in \Sigma$, we define the set $R_a(q) = \{q' \in Q \mid (q, ??a, q') \in \Delta\}$ containing the control states that can be reached in \mathcal{P} from the state q after receiving the message a . We also denote by $ActStates$ the set of states $\{q \in Q \mid \exists(q, !!a, q') \in \Delta \text{ or } \exists(q, \varepsilon, q') \in \Delta\}$ from which a broadcast or an internal action can be performed.

The semantics associated to a protocol $\mathcal{P} = (Q, Q^{(1)}, Q^{(P)}, q_0, \Sigma, \Delta, \Delta^{int})$ is given in terms of an infinite state $1 - \frac{1}{2}$ player game. We will represent the network by labelled graphs. The intuition is that each node of the graph runs the protocol and the semantics respect the following rules: first the Player 1 chooses non deterministically a communication topology (i.e. the edge relation) and a node which will then perform either a broadcast or an internal change; if the node broadcasts a message, all the adjacent nodes able to receive it will change their states, and if the node performs an internal move, then it will be the only one to change its state to a new state, if it is a probabilistic state a probabilistic move will then follow. Observe that the topology can hence possibly change at each step of the Player 1. Finally, in our model, there is no creation neither deletion of nodes, hence along a path in the associated game the number of nodes in the graphs is fixed. We now formalize this intuition.

Let $\mathcal{P} = (Q, Q^{(1)}, Q^{(P)}, q_0, \Sigma, \Delta, \Delta^{int})$ be a probabilistic protocol. The set of configurations $\Gamma_{\mathcal{P}}$ of the network built over \mathcal{P} is a set of $(Q \times \{\perp, \top\})$ -graphs formally defined as follows: $\Gamma_{\mathcal{P}}^{(1)} = \{(V, E, L) \in \mathcal{G}_{Q^{(1)} \times \{\perp, \top\}} \mid \text{card}(\{v \in V \mid L(v) \in Q^{(1)} \times \{\top\}\}) \leq 1\}$ and $\Gamma_{\mathcal{P}}^{(p)} = \{(V, E, L) \in \mathcal{G}_{Q \times \{\perp\}} \mid \text{card}(\{v \in V \mid L(v) \in Q^{(P)} \times \{\perp\}\}) = 1\}$ and $\Gamma_{\mathcal{P}} = \Gamma^{(1)} \cup \Gamma^{(p)}$. Hence in the configurations of Player 1, there is no node labelled with probabilistic state and at most one node labelled with \top (it is the chosen node for the action to be performed) and in the probabilistic configurations no node is labelled with \top and exactly one node is labelled with a probabilistic state. For this last set of configurations, the intuition is that when in the network one node changes its state to a probabilistic one then the network goes in a configuration in $\Gamma_{\mathcal{P}}^{(p)}$ from which it performs a probabilistic choice for the next possible state of the considered node.

The semantics of the network built over \mathcal{P} is then given in terms of the $1 - \frac{1}{2}$ player game $\mathcal{M}_{\mathcal{P}} = (\Gamma_{\mathcal{P}}, \Gamma_{\mathcal{P}}^{(1)}, \Gamma_{\mathcal{P}}^{(p)}, \rightarrow_{\mathcal{P}}, \text{prob}_{\mathcal{P}})$ where:

- $\rightarrow_{\mathcal{P}} \subseteq \Gamma_{\mathcal{P}}^{(1)} \times \Gamma_{\mathcal{P}}$ is defined as follows, for all $\gamma = (V, E, L)$ in $\Gamma_{\mathcal{P}}^{(1)}$, all $\gamma' = (V', E', L')$ in $\Gamma_{\mathcal{P}}$, we have $\gamma \rightarrow_{\mathcal{P}} \gamma'$ iff one of the following conditions hold:

Reconfiguration and process choice: $\gamma \in \mathcal{G}_{Q^{(1)} \times \{\perp, \top\}}$, $V' = V$ and there exists a vertex $v \in V$ and a state $q \in ActStates$ such that $L(v) = (q, \perp)$ and $L'(v) = (q, \top)$ and for all $v' \in V \setminus \{v\}$, $L(v') = L'(v')$ (in this step E' is arbitrarily defined and this is what induces reconfiguration);

Internal: $\gamma \in \Gamma_{\mathcal{P}}^{(1)}$, $V' = V$, $E' = E$ and there exists $v \in V$, $q \in Q^{(1)}$ and $q' \in Q$ such that $L(v) = (q, \top)$, $L'(v) = (q', \perp)$ and $(q, \varepsilon, q') \in \Delta$, and for all $v' \in V \setminus \{v\}$, $L'(v') = L(v')$;

Communication: $\gamma' \in \Gamma_{\mathcal{P}}^{(1)}$, $V' = V$, $E' = E$ and there exists $v \in V$, $q, q' \in Q^{(1)}$ and $a \in \Sigma$ such that $L(v) = (q, \top)$, $L'(v) = (q', \perp)$, $(q, !!a, q') \in \Delta$ and for every $v' \in V \setminus \{v\}$ with $L(v') = (q'', \perp)$, if $v \sim v'$ and $R_a(q'') \neq \emptyset$ then $L'(v') = (q''', \perp)$ with $q''' \in R_a(q'')$ and otherwise $L'(v') = L(v')$;

– $prob_{\mathcal{P}} : \Gamma_{\mathcal{P}}^{(p)} \mapsto \text{Dist}(\Gamma_{\mathcal{P}}^{(1)})$ is defined as follows, for all $\gamma = (V, E, L) \in \Gamma_{\mathcal{P}}^{(p)}$, we have : if $v \in V$ is the unique vertex such that $L(v) \in Q^{(P)} \times \{\perp\}$ and if $\Delta^{\text{int}}(L(v)) = \mu$, then for all $\gamma' = (V', E', L') \in \Gamma_{\mathcal{P}}$, if $V' = V$ and $E' = E$ and for all $v' \in V \setminus \{v\}$, $L'(v') = L(v)$ and then $prob_{\mathcal{P}}(\gamma)(\gamma') = \mu(q')$ where $(q, \perp) = L(v)$ and $(q', \perp) = L'(v)$, and otherwise $prob_{\mathcal{P}}(\gamma)(\gamma') = 0$.

Finally we will denote by $\Gamma_{\mathcal{P},0}$ the set of initial configurations in which all the vertices are labelled with (q_0, \perp) . We point out the fact that since we do not impose any restriction on the size of the Q -graphs, the $1 - \frac{1}{2}$ player game $\mathcal{M}_{\mathcal{P}}$ has hence an infinite number of configurations. However the number of configurations reachable from an initial configuration $\gamma \in \Gamma_{\mathcal{P},0}$ since the number of states in a probabilistic protocol is finite. Furthermore, note that since the topology can change arbitrarily at any reconfiguration step, we could have considered an equivalent semantics without topology but with a set of possible receivers for each emitted message.

A simple example of probabilistic protocol is represented on Figure 1. The initial state is q_0 and the only probabilistic state is q_p . From q_r the broadcast of a message a leads back to q_0 , and this message can be received from q_l to reach the target q_f .

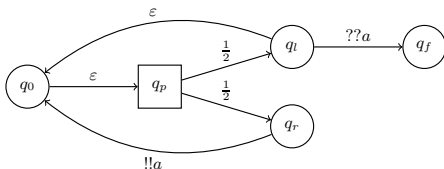


Fig. 1. Simple example of probabilistic protocol

2.3 Qualitative Reachability Problems

The problems we propose to investigate are qualitative ones where we will compare the probability of reaching a particular state in a network built over a probabilistic protocol with 0 or 1. Given a probabilistic protocol $\mathcal{P} = (Q, Q^{(1)}, Q^{(P)}, q_0, \Sigma, \Delta, \Delta^{\text{int}})$ and a state $q_f \in Q$, we denote by $\diamond q_f$ the set of all maximal paths of $\mathcal{M}_{\mathcal{P}}$ of the form $\gamma_0 \cdot \gamma_1 \cdots$ such that there exists $i \in \mathbb{N}$ verifying $(q_f, \perp) \in L(\gamma_i)$, i.e. the set of paths which eventually reach a graph where a node is labelled with the state q_f . It is well known that such a set of paths is measurable (see [5] for instance). We are now ready to provide the definition of the different qualitative

reachability problems that we will study. Given $\text{opt} \in \{\min, \max\}$, $\mathbf{b} \in \{0, 1\}$ and $\sim \in \{<, =, >\}$, let $\text{REACH}_{\text{opt}}^{\sim \mathbf{b}}$ be the following problem:

Input: A probabilistic protocol \mathcal{P} , and a control state $q_f \in Q$.
Question: Does there exist an initial configuration γ_0 such that $\mathbb{P}_{\text{opt}}(\mathcal{M}_{\mathcal{P}}, \gamma_0, \diamond q_f) \sim \mathbf{b}$?

Remark that this problem is parameterized by the initial configuration and this is the point that make this problem difficult to solve (and that leads to undecidability in the case with no probabilistic choice and no reconfiguration in the network [12]). However for a fixed given initial configuration, the problem boils down to the analysis of a finite $1 - \frac{1}{2}$ player game as already mentioned. As a consequence, the minimum and maximum (rather than infimum and supremum) probabilities are well-defined when an initial configuration γ_0 is fixed; moreover, these extremal values are met for memoryless schedulers.

3 Networks of Parity Reconfigurable Protocols

3.1 Parity, Safety and Safety/parity Games

We first introduce 2 player turn-based zero-sum games with various winning objectives. For technical reasons, our definition differs from the classical one: colors (or parities) label the edges rather than the vertices.

Definition 4 (2 player game). A 2 player game is a tuple $\mathbf{G} = (\Lambda, \Lambda^{(1)}, \Lambda^{(2)}, T, \text{col}, \text{safe})$ where Λ is a denumerable set of configurations, partitioned into $\Lambda^{(1)}$ and $\Lambda^{(2)}$, configurations of Player 1 and 2, respectively; $T \subseteq \Lambda \times \Lambda$ is a set of directed edges; $\text{col} : T \rightarrow \mathbb{N}$ is the coloring function such that $\text{col}(T)$ is finite; $\text{safe} \subseteq T$ is a subset of safe edges.

As in the case of $1 - \frac{1}{2}$ player game, we define the notions of paths and the equivalent to schedulers: strategies. A *finite path* ρ is a finite sequence of configurations $\lambda_0 \lambda_1 \cdots \lambda_n \in \Lambda^*$ such that $(\lambda_i, \lambda_{i+1}) \in T$ for all $0 \leq i < n$. Such a path is said to start at configuration λ_0 . An *infinite path* is an infinite sequence $\rho \in \Lambda^\omega$ such that any finite prefix of ρ is a finite path. Similarly to paths in $1 - \frac{1}{2}$ player game, *maximal paths* in \mathbf{G} are infinite paths or finite paths ending in a deadlock configuration.

A *strategy* for Player 1 dictates its choices in configurations of $\Lambda^{(1)}$. More precisely, a strategy for Player 1 in the game $\mathbf{G} = (\Lambda, \Lambda^{(1)}, \Lambda^{(2)}, T, \text{col}, \text{safe})$ is a function $\sigma : \Lambda^* \Lambda^{(1)} \mapsto \Lambda$ such that for every finite path ρ and $\lambda \in \Lambda^{(1)}$, $(\lambda, \sigma(\rho\lambda)) \in T$. Strategies $\tau : \Lambda^* \Lambda^{(2)} \rightarrow \Lambda$ for Player 2 are defined symmetrically, and we write $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$ for the set of strategies for each player. A *strategy profile* is a pair of strategies, one for each player. Given a strategy profile (σ, τ) and an initial configuration λ_0 , the game \mathbf{G} gives rise to the following maximal path, aka the *play*, $\rho(\mathbf{G}, \lambda_0, \sigma, \tau) = \lambda_0 \lambda_1 \cdots$ such that for all $i \in \mathbb{N}$, if $\lambda_i \in \Lambda^{(1)}$ then $\lambda_{i+1} = \sigma(\lambda_0 \dots \lambda_i)$, otherwise $\lambda_{i+1} = \tau(\lambda_0 \dots \lambda_i)$.

Remark 2. Similarly to the case of schedulers in $1 - \frac{1}{2}$ player game, (see Remark 1), when convenient the players' strategies can be alternatively defined as strategies with memory. In this case, a strategy for Player 1 with memory M is given by means of a *strategic function* $\sigma_M : \Lambda^{(1)} \times M \rightarrow \Lambda$, an *update function* $U_M : \Lambda^{(1)} \times M \times \Lambda \rightarrow M$, and an initialization function $I_M : \Lambda \rightarrow M$.

The winning condition for Player 1 is a subset of plays $\text{Win} \subseteq \Lambda^* \cup \Lambda^\omega$. In this paper, we characterize winning conditions through safety, parity objectives and combinations of these two objectives, respectively denoted by Win_s , Win_p and Win_{sp} , and defined as follows:

$$\begin{aligned} \text{Win}_s &= \{\rho \in \Lambda^* \cup \Lambda^\omega \mid \forall 0 \leq i < |\rho| - 1. (\rho(i), \rho(i+1)) \in \text{safe} \text{ and } \rho \text{ is maximal}\} \\ \text{Win}_p &= \{\rho \in \Lambda^\omega \mid \max\{n \in \mathbb{N} \mid \forall i \geq 0. \exists j \geq i. \text{col}((\rho(j), \rho(j+1))) = n\} \text{ is even}\} \\ \text{Win}_{sp} &= (\text{Win}_p \cap \text{Win}_s) \cup (\Lambda^* \cap \text{Win}_s) \end{aligned}$$

The safety objective denotes the maximal path that use only edges in **safe**, the parity objective denotes the infinite paths for which the maximum color visited infinitely often is even and the safety-parity objective denotes the set of safe maximal paths which have to respect the parity objectives when they are infinite. Note that in the context of games played over a finite graph the safety-parity objective can easily be turned into a parity objective, by removing the unsafe edges and by adding an even parity self-loop on deadlock states; However when the game is played on an infinite graph, this transformation is difficult because one first has to be able to detect deadlock configurations. Finally, we say that a play ρ is *winning for Player 1* for an objective $\text{Win} \subseteq \Lambda^* \cup \Lambda^\omega$ if $\rho \in \text{Win}$, in the other case it is winning for Player 2. Last, a strategy σ for Player 1 is a *winning strategy* from configuration λ_0 if for every strategy τ of Player 2, the play $\rho(\mathbb{G}, \lambda_0, \sigma, \tau)$ is winning for Player 1.

3.2 Networks of Parity Reconfigurable Protocols

We now come to the definition of networks of parity reconfigurable protocols, introducing their syntax and semantics. The main differences with the probabilistic protocol introduced previously lies in the introduction of states for Player 2, the use of colors associated to the transition relation and the removal of the probabilistic transitions.

Definition 5 (Parity protocol). A parity protocol is as a tuple $\mathbf{P} = (Q, Q^{(1)}, Q^{(2)}, q_0, \Sigma, \Delta, \text{col}, \text{safe})$ where Q is a finite set of control states partitioned into $Q^{(1)}$ and $Q^{(2)}$; $q_0 \in Q^{(1)}$ is the initial control state; Σ is a finite message alphabet; $\Delta \subseteq (Q^{(1)} \times (\{!!a, ??a \mid a \in \Sigma\} \cup \{\varepsilon\}) \times Q) \cup (Q^{(2)} \times \{\varepsilon\} \times Q)$ is the transition relation; $\text{col} : \Delta \rightarrow \mathbb{N}$ is the coloring function; $\text{safe} \subseteq \Delta$ is a set of safe edges.

Note that the roles of Player 1 and Player 2 are not symmetric: only Player 1 can initiate a communication, and Player 2 performs only internal actions. The semantics associated to a parity protocol is given in term of a 2 player game whose definition is similar to the $1 - \frac{1}{2}$ player game associated to a probabilistic

protocol (the complete definition can be found in [7]). Here also the Player 1 has the ability to choose a communication topology and a node which will perform an action, and according to the control state labelling this node either Player 1 or Player 2 will then perform the next move. The set of configurations $\Lambda_{\mathbf{P}}$ of the network built over a parity protocol $\mathbf{P} = (Q, Q^{(1)}, Q^{(2)}, q_0, \Sigma, \Delta, \text{col}, \text{safe})$ is defined as follows: $\Lambda_{\mathbf{P}} = \{(V, E, L) \in \mathcal{G}_{Q \times \{\perp, \top\}} \mid \text{card}(\{v \in V \mid L(v) \in Q \times \{\top\}\}) \leq 1\}$ and then we have $\Lambda_{\mathbf{P}}^{(1)} = \mathcal{G}_{Q \times \{\perp\}} \cup \{(V, E, L) \in \Lambda_{\mathbf{P}} \mid \text{card}(\{v \in V \mid L(v) \in Q^{(1)} \times \{\top\}\}) = 1\}$ and $\Lambda_{\mathbf{P}}^{(2)} = \{(V, E, L) \in \Lambda_{\mathbf{P}} \mid \text{card}(\{v \in V \mid L(v) \in Q^{(2)} \times \{\top\}\}) = 1\}$. We observe that Player 1 owns vertices where no node is tagged \top , and Player i owns the vertices where the node tagged \top is in a Player i control state. The semantics of the network built over \mathbf{P} is then given in term of the 2 player game $\mathbf{G}_{\mathbf{P}} = (\Lambda_{\mathbf{P}}, \Lambda_{\mathbf{P}}^{(1)}, \Lambda_{\mathbf{P}}^{(2)}, T_{\mathbf{P}}, \text{col}_{\mathbf{P}}, \text{safe}_{\mathbf{P}})$ where $T_{\mathbf{P}} \subseteq \Lambda_{\mathbf{P}} \times \Lambda_{\mathbf{P}}$ is defined using reconfiguration and process choices for Player 1 and internal and communication rules as the one defined in the case of probabilistic protocols, whereas $\text{col}_{\mathbf{P}} : T_{\mathbf{P}} \rightarrow \mathbb{N}$ and $\text{safe}_{\mathbf{P}} \subseteq T_{\mathbf{P}}$ are defined following col and safe lifting the definition from states to configurations. Finally, we will say that a configuration $\lambda = (V, E, L)$ is initial if $L(v) = (q_0, \perp)$ for all $v \in V$ and we will write $\Lambda_{\mathbf{P},0}$ the set of initial configurations. Note that here also the number of initial configuration is infinite. We are now able to define the game problem for parity protocol as follows:

Input: A parity protocol \mathbf{P} , and a winning condition Win .

Question: Does there exists an initial configuration $\lambda_0 \in \Lambda_{\mathbf{P},0}$ such that Player 1 has a winning strategy in $\mathbf{G}_{\mathbf{P}}$ from λ_0 ?

3.3 Restricting the Strategies of Player 2

In order to solve the game problem for parity protocols, we first show that we can restrict the strategies of Player 2 to strategies that always choose from a given control state the same successor, independently of the configuration, or the history in the game.

We now consider a parity protocol $\mathbf{P} = (Q, Q^{(1)}, Q^{(2)}, q_0, \Sigma, \Delta, \text{col}, \text{safe})$. We begin by defining what are the local positional strategies for Player 2 in $\mathbf{G}_{\mathbf{P}}$. A *local behavior* for Player 2 in $\mathbf{G}_{\mathbf{P}}$ is a function $b : (Q^{(2)} \cap \text{ActStates}) \mapsto \Delta$ such that for all $q \in Q^{(2)} \cap \text{ActStates}$, $b(q) \in \{(q, \varepsilon, q') \mid (q, \varepsilon, q') \in \Delta\}$. Such a local behavior induces what we will call a *local strategy* τ_b for Player 2 in $\mathbf{G}_{\mathbf{P}}$ defined as follows: let ρ be a finite path in $\Lambda_{\mathbf{P}}^*$ and $\lambda = (V, E, L) \in \Lambda^{(2)}$, if v is the unique vertex in V such that $L(v) \in Q^{(2)} \times \{\top\}$ and if $L(v) = (q, \top)$, we have $\tau_b(\rho\lambda) = \lambda'$ where λ' is the unique configuration obtained from λ by applying accordingly to the definition of $\mathbf{G}_{\mathbf{P}}$ the rule corresponding to $b(q)$ (i.e. the internal action initiated from vertex v). We denote by $\mathcal{S}_1^{(2)}$ the set of local strategies for Player 2. Note that there are a finite number of states and of edges in \mathbf{P} , the set $\mathcal{S}_1^{(2)}$ is thus finite and contains at most $\text{card}(\Delta)$ strategies. The next lemma shows that we can restrict Player 2 to follow only local strategies in order to solve the game problem for \mathbf{P} when considering the previously introduced winning objectives.

Lemma 1. *For $\text{Win} \in \{\text{Win}_s, \text{Win}_p, \text{Win}_{sp}\}$, we have $\exists \lambda_0 \in \Lambda_{\mathbf{P},0}. \exists \sigma \in \mathcal{S}^{(1)}. \forall \tau \in \mathcal{S}^{(2)}, \rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau) \in \text{Win} \iff \forall \tau \in \mathcal{S}_l^{(2)}. \exists \lambda_0 \in \Lambda_{\mathbf{P},0}. \exists \sigma \in \mathcal{S}^{(1)}, \rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau) \in \text{Win}$*

The proof of this lemma shares some similarities with the one to establish that memoryless strategies are sufficient for Player 2 in energy parity games [10]. It is performed by induction on the number of states of Player 2 in the parity protocol. In the induction step, a configuration is split into several sub-configurations (one for each local strategy of Player 2) and Player 1 navigates among the sub-configurations each time Player 2 changes strategy. For instance if Player 2 has two choices, say left and right, then at the beginning Player 1 plays in the “left”-sub-configuration and when Player 2 decides to choose right instead of left, then the associated node is moved to the “right”-sub-configuration and the game proceeds in this sub-configuration, and so on. It can be shown that if Player 1 wins against the strategy which chooses always left and against the one which chooses always right, then it wins against any strategy of Player 2.

3.4 Solving the Game against Local Strategies

In this section, we explain how to decide whether there exists an initial configuration and a strategy for Player 1 which is winning against a fixed local strategy. We consider a parity protocol $\mathbf{P} = (Q, Q_1, Q_2, q_0, \Sigma, \Delta, \text{col}, \text{safe})$ and a local behavior b . From this parity protocol we build a parity protocol $\mathbf{P}' = (Q, q_0, \Sigma, \Delta', \text{col}', \text{safe}')$ by removing the choices of Player 2 not corresponding to b and by merging states of Player 1 and states of Player 2; this protocol is formally defined as follows: $\Delta' \subseteq \Delta$ and $(q, \mathbf{a}, q') \in \Delta'$ iff $q \in Q^{(1)}$ and $(q, \mathbf{a}, q') \in \Delta'$, or, $q \in Q^{(2)}$ and $b(q)$ is defined and equal to (q, \mathbf{a}, q') , furthermore col' is the restriction of col to Δ' and $\text{safe}' = \Delta' \cap \text{safe}$. The following lemma states the relation between \mathbf{P} and \mathbf{P}' .

Lemma 2. *For $\text{Win} \in \{\text{Win}_s, \text{Win}_p, \text{Win}_{sp}\}$, there exists a path ρ in $\mathbf{G}_{\mathbf{P}'}$ starting from an initial configuration and such that $\rho \in \text{Win}$ iff $\exists \lambda_0 \in \Lambda_{\mathbf{P},0}. \exists \sigma \in \mathcal{S}^{(1)}, \rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau_b) \in \text{Win}$.*

We will now show how to decide the two following properties on $\mathbf{G}_{\mathbf{P}'}$: whether there exists a maximal finite path in Win_s starting from an initial configuration in $\mathbf{G}_{\mathbf{P}'}$ and whether there exists an infinite path $\in \text{Win}_p \cap \text{Win}_s$ starting from an initial configuration. Once, we will have shown how to solve these two problems, this will provide us, for each winning condition, an algorithm to decide whether there exists a winning path in $\mathbf{G}_{\mathbf{P}'}$.

We now provide the idea to solve the first problem. By definition, a finite path $\rho = \lambda_0 \lambda_1 \cdots \lambda_n$ in the game $\mathbf{G}_{\mathbf{P}'}$ is maximal if there does not exist a configuration $\lambda \in \Lambda_{\mathbf{P}'}$ such that $(\lambda_n, \lambda) \in T'_{\mathbf{P}}$ and according to the semantics of the parity protocol \mathbf{P}' , this can be the case if and only if $\lambda_n = (V, E, L)$ where $L(\lambda_n) \subseteq (Q \times \{\perp\}) \setminus (\text{ActStates} \times \{\perp\})$. In [11], it is shown that, for reconfigurable broadcast protocol, one can decide in NP whether, given a set of protocol states,

there exists a path starting from an initial configuration reaching a configuration in which no vertices are labelled by the given states. We deduce the next lemma.

Lemma 3. *The problem of deciding whether there exists in $\mathbf{G}_{\mathbf{P}'}$ a finite maximal path belonging to Win_s starting from an initial configuration is in NP.*

We now show how to decide in polynomial time whether there exists an infinite path in $\text{Win}_p \cap \text{Win}_s$ starting from an initial configuration. The idea is the following. We begin by removing in \mathbf{P}' the unsafe edges. Then we compute in polynomial time all the reachable control states using an algorithm of [11]. Then from [11] we also know that there exists a reachable configuration exhibiting as many reachable states as we want. Finally, we look for an infinite loop respecting the parity condition from such a configuration. This is done by using a counting abstraction method which translates the system into a Vector Addition System with States (VASS) and then by looking in this VASS for a cycle whose effect on each of the manipulated values is 0 (i.e. a cycle whose edge's labels sum to 0) and this is can be done in polynomial time thanks to [16].

Lemma 4. *The problem of deciding whether there exists an infinite path ρ starting from an initial configuration in $\mathbf{G}_{\mathbf{P}'}$ such that $\rho \in \text{Win}_p \cap \text{Win}_s$ is in PTIME.*

By Lemma 2 we know hence that: there is an NP algorithm to decide whether $\exists \lambda_0 \in A_{\mathbf{P},0}. \exists \sigma \in \mathcal{S}^{(1)}, \rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau_b) \in \text{Win}_s$ (in fact this reduces to looking for a finite maximal path belonging to Win_s and use Lemma 3 or an infinite safe path, in this case we put all the colors to 0 and we use Lemma 4); there is a polynomial time algorithm to decide the same problem with Win_p instead of Win_s (use Lemma 3 with all the transitions considered as safe) and there is an NP algorithm for the same problem with Win_{sp} (here again we look either for a finite maximal safe path and use Lemma 3 or for an infinite safe path satisfying the parity condition and we use Lemma 4).

So now since the number of local strategies is finite, this gives us non deterministic algorithms to solve whether $\exists \tau \in \mathcal{S}_1^{(2)}. \forall \lambda_0 \in A_{\mathbf{P},0}. \forall \sigma \in \mathcal{S}^{(1)}, \rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau) \notin \text{Win}$ with $\text{Win} \in \{\text{Win}_s, \text{Win}_p, \text{Win}_{sp}\}$. Note that for Win_p we will have an NP algorithm and for Win_s and Win_{sp} , an NP algorithm using an NP oracle (i.e. an algorithm in $\text{NP}^{\text{NP}} = \Sigma_2^P$). Hence thanks to Lemma 1, we are able to state the main result of this section.

Theorem 1. *For safety and safety-parity objectives, the game problem for parity protocol is decidable and in Π_2^P ($=\text{co-NP}^{\text{NP}}$), and in co-NP for parity objectives.*

4 Solving Probabilistic Networks

In this section we solve the qualitative reachability problems for probabilistic reconfigurable broadcast networks. The most involved case is REACH_{\max}^1 for which we reduce to games on parity protocols with a parity winning condition.

4.1 $\text{Reach}_{\max}^=1$

Let us now discuss the most involved case, $\text{REACH}_{\max}^=1$, and show how to reduce it to the game problem for parity protocols with a parity winning condition. From $\mathcal{P} = (Q, Q^{(1)}, Q^{(P)}, q_0, \Sigma, \Delta, \Delta^{\text{int}})$ a probabilistic protocol and $q_f \in Q$ a control state, we derive the parity protocol $\mathbf{P} = (Q_{\mathbf{P}}, Q_{\mathbf{P}}^{(1)}, Q_{\mathbf{P}}^{(2)}, q_{0\mathbf{P}}, \Sigma_{\mathbf{P}}, \Delta_{\mathbf{P}}, \text{col}, \text{safe})$ as follows: $Q_{\mathbf{P}} = Q_{\mathbf{P}}^{(1)} \cup Q_{\mathbf{P}}^{(2)}$, $Q_{\mathbf{P}}^{(1)} = Q^{(1)} \cup Q^{(P)} \times \{1\}$, $Q_{\mathbf{P}}^{(2)} = Q^{(P)} \times \{2\}$, and $q_{0\mathbf{P}} = q_0$; $\Sigma_{\mathbf{P}} = \Sigma$; $\Delta_{\mathbf{P}} = (Q^{(1)} \times \{!!a, ??a \mid a \in \Sigma\} \times Q^{(1)} \cap \Delta) \cup \{(q_f, \varepsilon, q_f)\} \cup \{(q, \varepsilon, (q', 2)), ((q', i), \varepsilon, q'), ((q, 2), \varepsilon, (q, 1)) \mid (q, \varepsilon, q') \in \Delta, i \in \{1, 2\}\} \cup \{((q, i), \varepsilon, q') \mid \Delta^{\text{int}}(q)(q') > 0, i \in \{2, 3\}\}$; and last $\text{col}((q_f, \varepsilon, q_f)) = 2$, $\text{col}((q, 2), \varepsilon, q') = 2$ and otherwise $\text{col}(\delta) = 1$.

Intuitively, all random choices corresponding to internal actions in \mathcal{P} are replaced in \mathbf{P} with choices for Player 2, where either he decides the outcome of the probabilistic choice, or he lets Player 1 choose. Only transitions where Player 2 makes the decision corresponding to a probabilistic choice and the self loop on the state q_f have parity 2. Figure 2 illustrates this reduction on the example probabilistic protocol from Figure 1. This construction ensures:

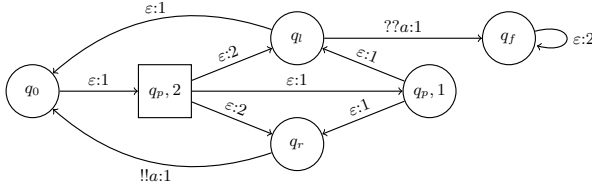


Fig. 2. Parity protocol for the probabilistic protocol from Figure 1

Proposition 1. $\exists \lambda_0 \in \Lambda_{\mathbf{P},0}$. $\exists \sigma \in \mathcal{S}^{(1)}$. $\forall \tau \in \mathcal{S}^{(2)}$, $\rho(\mathbf{G}_{\mathbf{P}}, \lambda_0, \sigma, \tau) \in \text{Win}_{\mathbf{P}}$ if and only if $\exists \gamma_0 \in \Gamma_{\mathcal{P},0}$. $\mathbb{P}_{\max}(\mathcal{M}_{\mathcal{P}}, \gamma_0, \diamond q_f) = 1$.

Proof (sketch). The easiest direction is from left to right. Assuming that some scheduler π ensures to reach q_f with probability 1, one builds a winning strategy σ for the parity objective as follows. When Player 2 makes a decision corresponding to a probabilistic choice in \mathcal{P} , the strategy chooses to play this probabilistic transition. Now, when Player 1 needs to make a decision in some configuration λ where there is a vertex v labelled by $((q, 1), \top) \in Q^{(P)} \times \{1\} \times \{\top\}$, the strategy is to play along a shortest path respecting π from γ to a configuration containing q_f , where γ is defined as λ but the label of v is q . Assuming that π reaches q_f with probability 1, such a path must exist for every reachable configuration in the game. This definition of σ ensures to eventually reach q_f under the assumption that Player 2, from some point on, always lets Player 1 decide in configurations corresponding to probabilistic states of \mathcal{P} .

Let us now briefly explain how the right to left implication works. Notice that if Player 2 always chooses transitions with parity 1 (thus letting Player 1 decide the outcome of probabilistic choices), the only way for Player 1 to win

is to reach q_f , and from there use the self loop to ensure the parity condition. As a consequence, from any reachable configuration, the target state q_f must be reachable.

From a winning strategy σ , we define a scheduler π that mimics the choices of σ on several copies of the network. The difficulty comes from the transformation of choices of Player 1 in states of the form $(q, 1) \in Q^{(P)} \times \{1\}$ into probabilistic choices. Indeed, the outcome of these random choices cannot surely match the decision of Player 1. The idea is the following: when a probabilistic choice in \mathcal{P} does not agree with the decision of Player 1 in \mathbf{P} , this “wrong choice” is attributed to Player 2. The multiple copies thus account for memories of the “wrong choices”, and a process performing such a choice is moved to a copy where the choice was made by Player 2. With probability 1, eventually a “good choice” is made, and the 1-1/2 player game can continue in the original copy of the network. Therefore, almost-surely the play will end in a given copy, where Player 1 always decides, and thus q_f is reached. \square

Theorem 2. $\text{REACH}_{\max}^=1$ is co-NP-complete.

Proof (sketch). The co-NP membership is a consequence of Proposition 1 and Theorem 1, and we now establish the matching lower-bound. To establish the CONP-hardness we reduce the unsatisfiability problem to $\text{REACH}_{\max}^=1$. From φ a formula in conjunctive normal form, we define a probabilistic protocol \mathcal{P}_φ and a control state q_f such that φ is unsatisfiable if and only if there exists an initial configuration $\gamma_0 \in \Gamma_{\mathcal{P},0}$ and a scheduler π such that $\mathbb{P}(\mathcal{M}_{\mathcal{P},\gamma_0,\pi}, \diamond q_f) = 1$.

We provide here the construction on an example in Figure 3, the general definition is given in Appendix. For simplicity, the initial state q_0 of the probabilistic protocol is duplicated in the picture. The idea, if φ is unsatisfiable, is to generate a random assignment of the variables (using the gadgets represented bottom of the Figure), which will necessarily violate a clause of φ . Choosing then this clause in the above part of the protocol allows to reach state r_1 , and from there to reach q_f with probability half. Iterating this process, the target can be almost-surely reached. The converse implication relies on the fact that if φ is satisfiable, there is a positive probability to generate a valuation satisfying it, and then not to be able to reach r_1 , a necessary condition to reach q_f . Therefore, the maximum probability to reach the target is smaller than 1 in this case. \square

4.2 Other Cases

The decision problems $\text{REACH}_{\min}^=0$ [resp. $\text{REACH}_{\min}^{<1}$] can be reduced to a game problem for parity protocols with a safety [resp. safety/parity] winning condition. From a probabilistic protocol \mathcal{P} , for $\text{REACH}_{\min}^=0$, we build a parity protocol \mathbf{P} where all random choices in \mathcal{P} are replaced in \mathbf{P} with choices for Player 2. The transitions with target q_f are the only ones that do not belong to the safe set *safe*. For $\text{REACH}_{\min}^{<1}$, \mathbf{P} consists of two copies of \mathcal{P} . In the first copy, all random choices are replaced with choices of Player 1, whereas in the second copy they are replaced with choices of Player 2. Also, at any time, one can move from the

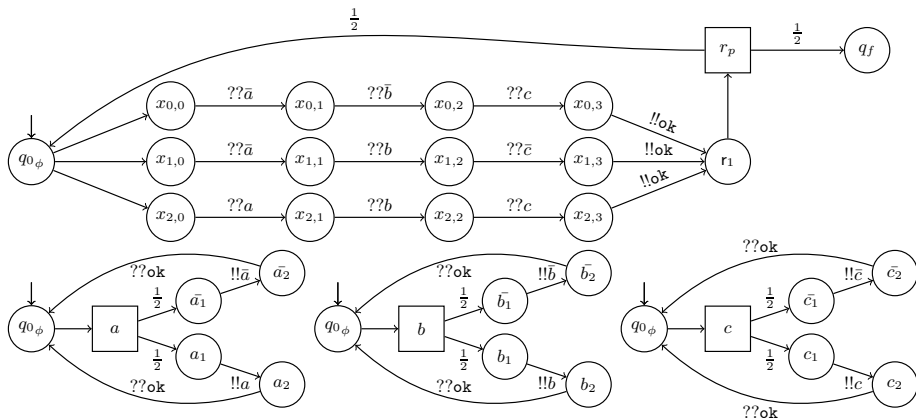


Fig. 3. Probabilistic protocol for the formula $\varphi = (a \vee b \vee \bar{c}) \wedge (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$

first to the second copy. The parity of transitions with target in the second copy is 2, and otherwise it is 1. Moreover, the only unsafe transitions are those with target q_f . In these two cases, using Theorem 1, we obtain:

Theorem 3. $REACH_{\min}^=0$ and $REACH_{\min}^{<1}$ are in Π_2^P .

The decidability and complexity of the remaining cases are established directly, without reducing to games on parity protocols. First of all, $REACH_{\max}^>0$ is interreducible to the reachability problem in *non-probabilistic* reconfigurable broadcast networks, known to be P-complete [11]. For the other decision problems we use a monotonicity property: intuitively, with more nodes, the probability to reach the target can only increase. The problems are then reduced to qualitative reachability problems in the finite state MDP for the network with a single process, and thus belong to PTIME.

Theorem 4. $REACH_{\max}^>0$, $REACH_{\max}^=0$, $REACH_{\max}^{<1}$, $REACH_{\min}^=1$ and $REACH_{\min}^>0$ are in PTIME.

5 Conclusion

In this paper we introduced probabilistic reconfigurable broadcast networks and studied parameterized qualitative reachability questions. The decidability of these verification questions are proved by a reduction to a 2-player games played on an infinite graphs, for which we provide decision algorithms. The complexities range from PTIME to $CONP^{NP}$, as summarized in the table below.

Problem	$REACH_{\min}^=0$	$REACH_{\min}^{<1}$	$REACH_{\max}^=1$	others
Complexity	Π_2^P	Π_2^P	CONP-complete	PTIME

In the future, we would like to find the precise complexity for $\text{REACH}_{\min}^=0$ and $\text{REACH}_{\min}^{<1}$ either by determining matching lower bounds or by improving the decision procedures. We will also study quantitative versions of the reachability problem. Finally we also believe that we could use our games played over reconfigurable broadcast protocols either to decide other properties on this family of systems or to analyze new models.

References

1. Abdulla, P.A., Atig, M.F., Rezine, O.: Verification of directed acyclic ad hoc networks. In: Beyer, D., Boreale, M. (eds.) FMOODS/FORTE 2013. LNCS, vol. 7892, pp. 193–208. Springer, Heidelberg (2013)
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321. IEEE Computer Society (1996)
3. Abdulla, P.A., Henda, N.B., Mayr, R.: Decisive Markov chains. Logical Methods in Computer Science 3(4) (2007)
4. Baier, C., Bertrand, N., Schnoebelen, P.: Verifying nondeterministic probabilistic channel systems against ω -regular linear-time properties. ACM Transactions on Computational Logic 9(1) (2007)
5. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press (2008)
6. Bertrand, N., Fournier, P.: Parameterized verification of many identical probabilistic timed processes. In: FSTTCS 2013. LIPIcs, vol. 24, pp. 501–513. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
7. Bertrand, N., Fournier, P., Sangnier, A.: Playing with probabilities in Reconfigurable Broadcast Networks. Research Report HAL-00929857, HAL, CNRS, France (2014)
8. Brázdil, T., Kučera, A., Stražovský, O.: On the decidability of temporal properties of probabilistic pushdown automata. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 145–157. Springer, Heidelberg (2005)
9. Chatterjee, K., de Alfaro, L., Faella, M., Legay, A.: Qualitative logics and equivalences for probabilistic systems. Logical Methods in Computer Science 5(2) (2009)
10. Chatterjee, K., Doyen, L.: Energy parity games. Theoretical Computer Science 458, 49–60 (2012)
11. Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the complexity of parameterized reachability in reconfigurable broadcast networks. In: FSTTCS 2012. LIPIcs, vol. 18, pp. 289–300. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012)
12. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 313–327. Springer, Heidelberg (2010)
13. Delzanno, G., Sangnier, A., Zavattaro, G.: On the power of cliques in the parameterized verification of ad hoc networks. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 441–455. Springer, Heidelberg (2011)
14. Etessami, K., Yannakakis, M.: Recursive Markov decision processes and recursive stochastic games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 891–903. Springer, Heidelberg (2005)
15. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
16. Kosaraju, S.R., Sullivan, G.F.: Detecting cycles in dynamic graphs in polynomial time (preliminary version). In: STOC 1988, pp. 398–406. ACM (1988)

Unsafe Order-2 Tree Languages Are Context-Sensitive

Naoki Kobayashi¹, Kazuhiro Inaba², and Takeshi Tsukada³

¹ The University of Tokyo

² Google Inc.

³ University of Oxford and JSPS Postdoctoral Fellow for Research Abroad

Abstract. Higher-order grammars have been extensively studied in 1980's and interests in them have revived recently in the context of higher-order model checking and program verification, where higher-order grammars are used as models of higher-order functional programs. A lot of theoretical questions remain open, however, for *unsafe* higher-order grammars (grammars without the so-called safety condition). In this paper, we show that any tree languages generated by order-2 unsafe grammars are context-sensitive. This also implies that any unsafe order-3 word languages are context-sensitive. The proof involves novel technique based on typed lambda-calculus, such as type-based grammar transformation.

1 Introduction

Higher-order (or high-level) grammars, where non-terminal symbols may take higher-order functions as arguments, have been introduced in 1970's [19,20,15] and extensively studied in 1980's [3]. They form a natural extension of Chomsky hierarchy [20], in the sense that they form an infinite language hierarchy, where the order-0 and order-1 word languages are exactly regular languages and context-free languages respectively. Recently, higher-order grammars have been studied as models of higher-order programs [8,16], and applied to automated verification of higher-order programs [9,13,17].

Earlier theoretical results on higher-order grammars [3,8,6] have been for those with the so-called *safety* restriction [8] (or, with the condition on *derived types* [3]). Although some of the analogous results have recently been obtained for *unsafe* grammars (those without the safety restriction) [16,7,14], many problems still remain open, such as the context-sensitiveness of higher-order languages. This is a pity, as many of the recent applications of higher-order grammars make use of unsafe ones.

In the present paper, we are interested in the open problem mentioned above: whether the languages generated by higher-order grammars are context-sensitive. As a solution to a special case of the open problem, we show that the tree languages (or more precisely, the word languages obtained by preorder traversal of trees, because the context-sensitiveness is usually the terminology for word languages) generated by any order-2 grammars are also context-sensitive. Since the

order- $(n + 1)$ word languages can be obtained as the leaf languages of trees generated by order- n grammars [11], the result also implies that the word languages generated by order-3 grammars are context-sensitive.¹

Our techniques to prove the context-sensitiveness of order-2 tree languages are quite different from those used in Inaba and Maneth’s proof for context-sensitiveness of safe languages [6]. Recall that the context-sensitiveness is equivalent to the membership problem being NLIN-SPACE (non-deterministic linear space). To show that, Inaba and Maneth decomposed higher-order (safe) transducers (whose image is the set of higher-order safe languages) into macro tree transducers, and transformed the transducers so that the size of intermediate trees increases monotonically. For the unsafe case, similar decomposition appears to be extremely difficult.

Instead of going through transducers or automata, we directly reason about grammars with a help of techniques of typed λ -calculus (intersection types, in particular). The high-level structure of our proof is actually similar to that of the (straightforward) proof of the context-sensitivity of context-free languages. For a context-free grammar (say, $\{S \rightarrow \mathbf{a}AA, A \rightarrow \epsilon \mid \mathbf{a}Ab\}$), one can eliminate ϵ -rules ($A \rightarrow \epsilon$ in the above example) to ensure that the size of intermediate phrases occurring in a production of a final word w is bounded by the size of w . For example, the above grammar can be transformed to $\{S \rightarrow \mathbf{a}AA \mid \mathbf{a} \mid \mathbf{a}A, A \rightarrow \mathbf{a}Ab \mid \mathbf{ab}\}$, by propagating information that A may be replaced by ϵ . The first part of our proof shows that intersection types can be used to achieve a similar (but more elaborate) transformation of higher-order grammars to exclude out certain rewriting rules. More precisely, given a finite set \mathcal{C} of functions, one can exclude out rules that allow non-terminals to behave like one of the functions in \mathcal{C} . The second part of the proof shows that for the order-2 case, if we choose as \mathcal{C} a set of “permutator [2]-like” terms, then the size of intermediate terms occurring in a production of a tree π is linearly bounded by the size of π . Thus, given an order-2 grammar \mathcal{G} , one can first transform \mathcal{G} to an equivalent grammar \mathcal{G}' that satisfies the property above, and then the membership of a tree π in the tree language of \mathcal{G}' can be decided in space linear in π . This implies that the language of (word representation of) trees generated by \mathcal{G} is context-sensitive.

From a practical viewpoint, the result may be applicable to the following problem: given a program P and a possible execution trace (or an execution tree) π , is π a real trace of P ? If P is a simply-typed program with recursion and finite base types, one can use the technique of [9] to construct a grammar that represents all the possible traces of P . One can then use the above algorithm to decide the membership problem in linear space with respect to the size of π . If one asks many questions for a fixed P and different π , using the above algorithm is theoretically more efficient than using higher-order model checking [9].

The rest of the paper is structured as follows. Section 2 defines higher-order grammars and the languages generated by grammars. Section 3 describes the

¹ The order-2 word languages are known to be context-sensitive. The result follows from context-sensitiveness of *safe* word languages [6] and the equivalence of safe and unsafe word languages for the order-2 case [1].

type-based grammar transformation that removes certain rewriting rules. Section 4 focuses on order-2 grammars and shows that after the grammar transformation, the size of intermediate terms is linearly bounded by the size of the produced tree. Section 5 discusses related work and Section 6 concludes. For the space limitation, we omit some details and proofs, which are found in an extended version of this paper, available from the first author's web page.

2 Preliminaries

This section defines higher-order grammars and the languages generated by them. When f is a map, we write $\text{dom}(f)$ and $\text{codom}(f)$ for the domain and codomain of f .

Definition 1 (types). *The set of **simple types**, ranged over by κ , is defined by: $\kappa ::= \circ \mid \kappa_1 \rightarrow \kappa_2$. The order and arity of a simple type κ , written $\text{order}(\kappa)$ and $\text{ar}(\kappa)$, are defined by:*

$$\begin{aligned} \text{order}(\circ) &= 0 & \text{order}(\kappa_1 \rightarrow \kappa_2) &= \max(\text{order}(\kappa_1) + 1, \text{order}(\kappa_2)) \\ \text{ar}(\circ) &= 0 & \text{ar}(\kappa_1 \rightarrow \kappa_2) &= 1 + \text{ar}(\kappa_2) \end{aligned}$$

Intuitively, \circ is the type of trees. We assume a ranked alphabet Σ , which is a map from a finite set of symbols (called **terminals**) to their arities. We use each terminal a as a tree constructor of arity $\Sigma(a)$. We assume a finite set of symbols called **non-terminals**, ranged over by A .

Definition 2 (λ -terms). *The set of λ -terms, ranged over by t , is defined by: $t ::= x \mid A \mid a \mid t_1 t_2 \mid \lambda x : \kappa. t$. A term t is called an **applicative term** (or simply a **term**) if it does not contain λ -abstractions.*

We often omit the type annotation and just write $\lambda x. t$ for $\lambda x : \kappa. t$. We consider only well-typed terms; the type judgment relation $\mathcal{K} \vdash_{\text{ST}} t : \kappa$ (where non-terminals are treated as variables) is defined inductively by:

$$\begin{array}{c} \frac{}{\mathcal{K} \cup \{x : \kappa\} \vdash_{\text{ST}} x : \kappa} \\ \frac{\mathcal{K} \vdash_{\text{ST}} t_1 : \kappa_2 \rightarrow \kappa \quad \mathcal{K} \vdash_{\text{ST}} t_2 : \kappa_2}{\mathcal{K} \vdash_{\text{ST}} t_1 t_2 : \kappa} \quad \frac{\mathcal{K} \vdash_{\text{ST}} a : \underbrace{\circ \rightarrow \cdots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ}{\mathcal{K} \cup \{x : \kappa_1\} \vdash_{\text{ST}} t : \kappa_2} \\ \frac{}{\mathcal{K} \vdash_{\text{ST}} \lambda x : \kappa_1. t : \kappa_1 \rightarrow \kappa_2} \end{array}$$

We call t a (finite, Σ -ranked) **tree** if t consists of only terminals and applications, and $\emptyset \vdash_{\text{ST}} t : \circ$ holds. We write \mathbf{Tree}_{Σ} for the set of Σ -ranked trees, and use the meta-variable π for a tree.

Definition 3 (higher-order grammar). *A **higher-order grammar** (called simply a **grammar**) is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where (i) Σ is a ranked alphabet; (ii) \mathcal{N} is a map from a finite set of non-terminals to their types; (iii) \mathcal{R} is a finite set of **rewriting rules** of the form $A x_1 \cdots x_\ell \rightarrow t$, where $A \in \text{dom}(\mathcal{N})$*

and t is an applicative term. We require that $\mathcal{N}(A)$ must be of the form $\kappa_1 \rightarrow \dots \rightarrow \kappa_\ell \rightarrow \circ$ and $\mathcal{N}, x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell \vdash_{\text{ST}} t : \circ$ must hold. (iv) S is a non-terminal called **the start symbol**, and $\mathcal{N}(S) = \circ$. The **order (arity, resp.)** of a grammar \mathcal{G} , written $\text{order}(\mathcal{G})$ ($\text{ar}(\mathcal{G})$, resp.), is the largest order (arity, resp.) of the types of non-terminals. We sometimes write $\Sigma_{\mathcal{G}}, \mathcal{N}_{\mathcal{G}}, \mathcal{R}_{\mathcal{G}}, S_{\mathcal{G}}$ for the four components of \mathcal{G} .

For a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, the rewriting relation $\rightarrow_{\mathcal{G}}$ is defined by:

$$\frac{A x_1 \cdots x_k \rightarrow t \in \mathcal{R}}{A t_1 \cdots t_k \rightarrow_{\mathcal{G}} [t_1/x_1, \dots, t_k/x_k]t} \quad \frac{t_i \rightarrow_{\mathcal{G}} t'_i \quad i \in \{1, \dots, k\} \quad \Sigma(a) = k}{a t_1 \cdots t_k \rightarrow_{\mathcal{G}} a t_1 \cdots t_{i-1} t'_i t_{i+1} \cdots t_k}$$

Here, $[t_1/x_1, \dots, t_k/x_k]t$ is the term obtained by substituting t_i for the free occurrences of x_i in t . We write $\rightarrow_{\mathcal{G}}^*$ for the reflexive transitive closure of $\rightarrow_{\mathcal{G}}$.

The **tree language generated by \mathcal{G}** , written $\mathcal{L}(\mathcal{G})$, is the set $\{\pi \in \text{Tree}_{\Sigma_{\mathcal{G}}} \mid S \rightarrow_{\mathcal{G}}^* \pi\}$. When the arity of every symbol in Σ is at most 1, the **word language generated by \mathcal{G}** is $\{a_1 \cdots a_n \mid a_1(\cdots(a_n e)\cdots) \in \mathcal{L}(\mathcal{G})\}$. The **leaf language generated by \mathcal{G}** , written $\mathcal{L}_{\text{leaf}}(\mathcal{G})$, is the set: $\{\text{leaves}(\pi) \mid S \rightarrow_{\mathcal{G}}^* \pi \in \text{Tree}_{\Sigma_{\mathcal{G}}}\}$, where $\text{leaves}(\pi)$ is the sequence of symbols in the leaves of π , defined inductively by: $\text{leaves}(a) = a$, and $\text{leaves}(a \pi_1 \pi_2) = \text{leaves}(\pi_1) \text{leaves}(\pi_2)$. The **order of a tree language** is the smallest order of a grammar that generates the language.

A grammar is **safe** if for the type $\kappa_1 \rightarrow \dots \rightarrow \dots \rightarrow \kappa_\ell \rightarrow \circ$ of each term t , (i) $\text{order}(\kappa_1) \geq \dots \geq \text{order}(\kappa_\ell)$ holds, and (ii) if $\text{order}(\kappa_i) = \text{order}(\kappa_{i+1})$, the i -th and $(i+1)$ -th arguments of t are passed always together. Grammars without the safety restriction are sometimes called **unsafe**, to emphasize the fact that there is no safety restriction. (Thus, the set of unsafe grammars include safe grammars.) A language is called **safe** if it is generated by some safe grammar.

In the rest of this paper, we assume that every terminal has arity 0 or 2. This does not lose generality, because every tree can be represented by a corresponding binary tree with linear size increase.

Example 1. Consider the order-2 grammar $\mathcal{G}_0 = (\{\mathbf{a}:2, \mathbf{b}:2, \mathbf{e}:0\}, \{S:\circ, F:(\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ, C:(\circ \rightarrow \circ \rightarrow \circ) \rightarrow (\circ \rightarrow \circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ, T:(\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ\}, \mathcal{R}, S)$ where \mathcal{R} consists of the rules:

$$\begin{array}{lll} S \rightarrow F(C \mathbf{a} \mathbf{b}) \mathbf{e} & F g x \rightarrow g x, & F g x \rightarrow F(T g) x \\ C g h x \rightarrow g x x & C g h x \rightarrow h x x & T g x \rightarrow g(g x). \end{array}$$

Then, the following is a possible reduction sequence:

$$\begin{aligned} S &\longrightarrow F(C \mathbf{a} \mathbf{b}) \mathbf{e} \longrightarrow F(T(C \mathbf{a} \mathbf{b})) \mathbf{e} \longrightarrow T(C \mathbf{a} \mathbf{b}) \mathbf{e} \\ &\longrightarrow (C \mathbf{a} \mathbf{b})(C \mathbf{a} \mathbf{b} \mathbf{e}) \longrightarrow \mathbf{a}(C \mathbf{a} \mathbf{b} \mathbf{e})(C \mathbf{a} \mathbf{b} \mathbf{e}) \longrightarrow^* \mathbf{a}(\mathbf{b} \mathbf{e} \mathbf{e})(\mathbf{a} \mathbf{e} \mathbf{e}). \end{aligned}$$

$\mathcal{L}(\mathcal{G}_0)$ is the set of perfect finite trees of height 2^n (where all the leaves have the same depth). $\mathcal{L}_{\text{leaf}}(\mathcal{G}_0) = \{\mathbf{e}^{2^{2^n}} \mid n \geq 0\}$.

Example 2. Consider the grammar $\mathcal{G}_1 = (\{\mathbf{f}:2, \mathbf{g}:2, \mathbf{a}:0, \mathbf{b}:0, \mathbf{e}:0\}, \{S:\mathbf{o}, F:(\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}, G:\mathbf{o} \rightarrow \mathbf{o}, H:\mathbf{o} \rightarrow \mathbf{o}\}, \mathcal{R}, S)$ where \mathcal{R} consists of:

$$\begin{array}{lll} S \rightarrow F G \mathbf{a} \mathbf{b} & F \varphi x y \rightarrow \mathbf{f}(F(F \varphi x) y (H y))(\mathbf{f}(\varphi y) x) & F \varphi x y \rightarrow \mathbf{e} \\ G x \rightarrow \mathbf{g} x \mathbf{e} & H x \rightarrow \mathbf{g} \mathbf{e} x. & \end{array}$$

This has been obtained from the grammar conjectured to be inherently unsafe ([8], p.213), by adding the rule $F \varphi x y \rightarrow \mathbf{e}$ (so that the grammar generates a set of finite trees, instead of an infinite tree) and encoding unary tree constructors \mathbf{g} and \mathbf{h} in their grammar as G and H (so that $\mathbf{h}(\pi)$ and $\mathbf{g}(\pi)$ are represented by $\mathbf{g} \mathbf{e} \pi'$ and $\mathbf{g} \pi' \mathbf{e}$ respectively). The following is a possible reduction sequence:

$$\begin{aligned} S &\longrightarrow F G \mathbf{a} \mathbf{b} \longrightarrow \mathbf{f}(F(F G \mathbf{a}) \mathbf{b} (H \mathbf{b}))(\mathbf{f}(G \mathbf{b}) \mathbf{a}) \longrightarrow \mathbf{f} \mathbf{e} (\mathbf{f}(G \mathbf{b}) \mathbf{a}) \\ &\longrightarrow \mathbf{f} \mathbf{e} (\mathbf{f}(\mathbf{g} \mathbf{b} \mathbf{e}) \mathbf{a}). \end{aligned}$$

3 Type-Based Grammar Transformation

As mentioned in Section 1, a key idea of our proof is to first transform a grammar to an equivalent grammar, so that the size of intermediate terms in a production sequence of tree π is linearly bound by the size of π . Note that the size of intermediate terms is not bounded for arbitrary grammars. For example, for the rewriting rules $\{S \rightarrow F \mathbf{e}, F x \rightarrow \mathbf{e}, F x \rightarrow F(F x)\}$, an arbitrarily large intermediate term $F^n \mathbf{e}$ may occur in a production of \mathbf{e} . As another example, replace the rule $F x \rightarrow \mathbf{e}$ above with $F x \rightarrow x$. Again, an arbitrarily large intermediate term $F^n \mathbf{e}$ may occur in a production of \mathbf{e} .

The problems above are attributed to the rules $F x \rightarrow \mathbf{e}$ and $F x \rightarrow x$, which respectively allow F to ignore arguments and to behave like an identity function. This section formalizes a type-based transformation that can remove such “non-productive” behaviors of non-terminals. A complication arises because (i) the grammars must actually be extended to enable such transformation, and (ii) the kinds of non-productive behaviors that should be removed depends on the order of grammars (more need to be eliminated with the increase of the order) and we have not yet obtained a general characterization of non-productive behaviors. We thus first present extended grammars in Section 3.1, and formalize the transformation by parametrizing it with a set of prohibited behaviors in Section 3.2. In the next section, we provide a sufficient characterization of prohibited behaviors for the order-2 case, and show that the removal of those behaviors indeed guarantee that the size of intermediate terms is linearly bounded by a generated tree.

3.1 Extended Grammars

This section introduces extended grammars, which are used as the target of the transformation.

Definition 4 (extended terms). The set of **extended terms**, ranged over by e , is defined by:

$$e ::= a \mid x \mid A \mid e E \mid \langle f \rangle E \quad E ::= \{e_1, \dots, e_k\} \quad f ::= e \mid \lambda x : \kappa. f$$

Here, A ranges over non-terminals, and $k > 0$ in $\{e_1, \dots, e_k\}$. We require that f in $\langle f \rangle$ contains no non-terminals, terminals, nor free variables.

Intuitively, $e\{e_1, \dots, e_k\}$ applies the function e to the argument $\{e_1, \dots, e_k\}$, which non-deterministically evaluate to e_i for some i ; however, e must use each e_1, \dots, e_k at least once. Thus, if we have a rule $Ax \rightarrow \mathbf{a} x x$, then $A\{e_1, e_2\}$ may be reduced to $\mathbf{a} e_1 e_2$ or $\mathbf{a} e_2 e_1$ but not to $\mathbf{a} e_1 e_1$. We often write $e e_1$ for $e\{e_1\}$. The term $\langle f \rangle$ is semantically the same as the (extended) λ -term f . Note that $\langle f \rangle$ cannot occur in an argument position; for example, $A\langle \lambda x. x \rangle$ is disallowed. (To save the number of rules, however, we allow e to be instantiated to $\langle f \rangle$ in the definitions of the type judgment and substitutions below.) We later restrict the set of terms f that may occur in the form of $\langle f \rangle$.

The type judgment relation $\mathcal{K} \vdash_E e : \kappa$ is defined inductively by:

$$\begin{array}{c} \frac{}{\overline{x : \kappa} \vdash_E x : \kappa} \quad \frac{}{\overline{A : \kappa} \vdash_E A : \kappa} \quad \frac{}{\emptyset \vdash_E a : \underbrace{\circ \rightarrow \dots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ} \\ \\ \frac{\{x_1 : \kappa_1, \dots, x_k : \kappa_k\} \vdash_E e : \circ}{\vdash_E \langle \lambda x_1 : \kappa_1. \dots \lambda x_k : \kappa_k. e \rangle : \kappa_1 \rightarrow \dots \rightarrow \kappa_k \rightarrow \circ} \\ \\ \frac{\mathcal{K}_1 \vdash_E e_1 : \kappa_2 \rightarrow \kappa \quad \mathcal{K}_2 \vdash_E E_2 : \kappa_2}{\mathcal{K}_1 \cup \mathcal{K}_2 \vdash_E e_1 E_2 : \kappa} \quad \frac{\mathcal{K}_i \vdash_E e_i : \kappa \text{ for each } i \in I}{\bigcup_{i \in I} \mathcal{K}_i \vdash_E \{e_i \mid i \in I\} : \kappa} \end{array}$$

Please notice that weakening is not allowed in the above rules. Therefore, if $\mathcal{K} \vdash_E e : \kappa$, then every variable in \mathcal{K} must occur at least once in e .

Definition 5 (extended grammars). A **combinator** is an extended λ -term f such that $\emptyset \vdash_E f : \kappa$ for some κ . Let \mathcal{C} be a finite set of combinators. An **extended grammar** over \mathcal{C} is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where: (i) Σ is a ranked alphabet; (ii) \mathcal{N} is a map from a finite set of non-terminals to their types; (iii) \mathcal{R} is a finite set of **extended rewriting rules** of the form $Ax_1 \dots x_\ell \rightarrow e$, where $A \in \text{dom}(\mathcal{N})$, and $f \in \mathcal{C}$ for every $\langle f \rangle$ in e . We require that $\mathcal{N}(A)$ must be of the form $\kappa_1 \rightarrow \dots \rightarrow \kappa_\ell \rightarrow \circ$ and $\Gamma \cup \{x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell\} \vdash_E e : \circ$ must hold for some $\Gamma \subseteq \mathcal{N}$. Furthermore, $\lambda x_1. \dots \lambda x_\ell. e \notin \mathcal{C}$, and e must not contain a subterm of the form $\langle \lambda x_1 \dots x_k. e' \rangle E_1 \dots E_k$. (iv) S is a non-terminal called **the start symbol**, and $\mathcal{N}(S) = \circ$. As before, the order and arity of \mathcal{G} , written $\text{order}(\mathcal{G})$ and $\text{ar}(\mathcal{G})$, are the largest order and arity of the types of non-terminals.

To define the rewriting relation for extended grammars, we need to extend the ordinary notion of substitutions. An (extended) **substitution** is a map from variables to sets of terms. We write $[E_1/x_1, \dots, E_k/x_k]$ for the substitution that

maps x_i to E_i , and use the meta-variable θ . The operation $[E/x]e$ replaces each occurrence of x in e with an element of E in a non-deterministic manner. Thus, we define the substitution operation as a relation $\theta \models e \rightsquigarrow e'$, which means that e' is the term obtained by applying the substitution θ to e . The relations $\theta \models e \rightsquigarrow e'$ and $\theta \models E \rightsquigarrow E'$ are defined inductively by:

$$\frac{\overline{[] \models a \rightsquigarrow a} \quad \overline{[] \models A \rightsquigarrow A} \quad \overline{[] \models \langle f \rangle \rightsquigarrow \langle f \rangle} \quad \overline{[\{e\}/x] \models x \rightsquigarrow e}}{\frac{\theta_1 \models e_1 \rightsquigarrow e'_1 \quad \theta_2 \models E_2 \rightsquigarrow E'_2}{\theta_1 \cup \theta_2 \models e_1 E_2 \rightsquigarrow e'_1 E'_2} \quad \frac{\theta_{i,j} \models e_i \rightsquigarrow e_{i,j} \text{ for each } i \in I, j \in J_i}{\bigcup_{i \in I, j \in J_i} \theta_{i,j} \models \{e_i \mid i \in I\} \rightsquigarrow \{e_{i,j} \mid i \in I, j \in J_i\}}}$$

Here, the operation $\theta_0 \cup \theta_1$ on substitutions is defined by: (i) $\text{dom}(\theta_0 \cup \theta_1) = \text{dom}(\theta_0) \cup \text{dom}(\theta_1)$; (ii) $(\theta_0 \cup \theta_1)(x) = \theta_0(x) \cup \theta_1(x)$ if $x \in \text{dom}(\theta_0) \cap \text{dom}(\theta_1)$, and (iii) $(\theta_0 \cup \theta_1)(x) = \theta_i(x)$ if $x \in \text{dom}(\theta_i) \setminus \text{dom}(\theta_{1-i})$.

Example 3. Let $\theta = [\{\mathbf{b}, \mathbf{c}\}/x]$ and $e = \mathbf{a} x x$. Then $\theta \models e \rightsquigarrow \mathbf{a} \mathbf{b} \mathbf{c}$ and $\theta \models e \rightsquigarrow \mathbf{a} \mathbf{c} \mathbf{b}$ hold, but neither $\theta \models e \rightsquigarrow \mathbf{a} \mathbf{b} \mathbf{b}$ nor $\theta \models e \rightsquigarrow \mathbf{a} \mathbf{c} \mathbf{c}$ does.

For $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, the rewriting relation $\longrightarrow_{\mathcal{G}}$ on terms is defined by:

$$\frac{A x_1 \cdots x_k \rightarrow e \in \mathcal{R} \quad [E_1/x_1, \dots, E_k/x_k] \models e \rightsquigarrow e'}{A E_1 \cdots E_k \longrightarrow_{\mathcal{G}} e'} \quad \frac{[E_1/x_1, \dots, E_k/x_k] \models e \rightsquigarrow e'}{\langle \lambda x_1 \cdots x_k. e \rangle E_1 \cdots E_k \longrightarrow_{\mathcal{G}} e'} \quad \text{(ER-COMB)}$$

$$\text{(ER-NT)}$$

$$\frac{e_i \longrightarrow_{\mathcal{G}} e'_i \quad i \in \{1, \dots, \Sigma(a)\}}{a \{e_1\} \cdots \{e_{\Sigma(a)}\} \longrightarrow_{\mathcal{G}} a \{e_1\} \cdots \{e_{i-1}\} \{e'_i\} \{e_{i+1}\} \cdots \{e_{\Sigma(a)}\}} \quad \text{(ER-CONG)}$$

We often omit the subscript \mathcal{G} . The **tree language generated** by an extended grammar \mathcal{G} , written $\mathcal{L}(\mathcal{G})$, is the set $\{\pi \in \mathbf{Tree}_{\Sigma_{\mathcal{G}}} \mid S \longrightarrow_{\mathcal{G}}^* \pi\}$ (where we identify a singleton set $\{e\}$ with e ; for example, the extended term $\mathbf{a}\{\mathbf{e}\}\{\mathbf{e}\}$ is interpreted as the tree $\mathbf{a} \mathbf{e} \mathbf{e}$).

Example 4. Consider the extended grammar $\mathcal{G}_2 = (\{\mathbf{a}:2, \mathbf{b}:0, \mathbf{c}:0\}, \{S:\circ, F:\circ \rightarrow \circ\}, \mathcal{R}, S)$ where $\mathcal{R} = \{S \rightarrow F\{\mathbf{b}, \mathbf{c}\}, Fx \rightarrow \mathbf{a}\{F\{x\}\}\{F\{x\}\}, Fx \rightarrow x\}$, then:

$$S \longrightarrow F\{\mathbf{b}, \mathbf{c}\} \longrightarrow \mathbf{a}(F\{\mathbf{b}\})(F\{\mathbf{b}, \mathbf{c}\}) \longrightarrow^* \mathbf{a} \mathbf{b} (\mathbf{a}(F\{\mathbf{c}\})(F\{\mathbf{b}\})) \longrightarrow^* \mathbf{a} \mathbf{b} (\mathbf{a} \mathbf{c} \mathbf{b}).$$

$\mathcal{L}(\mathcal{G}_2)$ is the set of all binary trees that contain at least one \mathbf{b} and one \mathbf{c} .

Reduction with Eager Normalization. We define $e \longrightarrow_{\lambda} e'$ inductively by: (i) $e \longrightarrow_{\lambda} e'$ if $e \longrightarrow_{\mathcal{G}} e'$ is derivable by using rule ER-COMB, (ii) $eE \longrightarrow_{\lambda} e'E$ if $e \longrightarrow_{\lambda} e'$; (iii) $e_0(E \uplus \{e\}) \longrightarrow_{\lambda} e_0(E \cup \{e_1, \dots, e_k\})$ if $e \longrightarrow_{\lambda} e_i$ for each $i \in \{1, \dots, k\}$ with $k \geq 1$; (iv) $(\lambda x.e)E \longrightarrow_{\lambda} e'$ if $[E/x] \models e \rightsquigarrow e'$; (v) $\lambda x.e \longrightarrow_{\lambda} \lambda x.e'$ if $e \longrightarrow_{\lambda} e'$; and (vi) $\langle f_1 \rangle \langle f_2 \rangle E \longrightarrow_{\lambda} \langle f \rangle E$ if $\lambda x.f_1(f_2 x) \longrightarrow_{\lambda} f \in \mathcal{C}$. In the above definition, we have extended the syntax of extended terms and allowed λ -abstractions to occur outside $\langle \cdot \rangle$, but ordinary extended terms are closed under $\longrightarrow_{\lambda}$. In $e \longrightarrow_{\lambda} e'$, we implicitly require that every argument of a terminal symbol must be a singleton set both in e and e' .

Henceforth, we assume that the set \mathcal{C} is closed under composition, in the sense that if $f_1, f_2 \in \mathcal{C}$ and $\lambda x.f_1(f_2 x) \rightarrow_{\lambda}^* e$, then $e \rightarrow_{\lambda}^* f$ for some $f \in \mathcal{C}$. We write $e \downarrow_{\lambda} e'$ if $e \rightarrow_{\lambda}^* e' \not\rightarrow_{\lambda}$, and write $e \Rightarrow_{\mathcal{G}} e'$ if $e(\downarrow_{\lambda} \cdot \rightarrow_{\mathcal{G}} \downarrow_{\lambda})e'$. For every term e of type \circ and tree π , $e \rightarrow_{\mathcal{G}}^* \pi$ if and only if $e \Rightarrow_{\mathcal{G}}^* \pi$. In Section 4, we bound the size of intermediate terms in a rewriting sequence $S \Rightarrow_{\mathcal{G}}^* \pi$.

3.2 From Grammars to Extended Grammars

This section presents a translation from (ordinary) grammars to extended grammars over a finite set \mathcal{C} of combinators, and shows that the translation preserves the tree language. We use type-based transformation techniques to eliminate useless arguments and (non-applied) combinators in \mathcal{C} .

Definition 6 (intersection types). *The set of intersection types over \mathcal{C} , ranged over by τ , is given by:*

$$\tau ::= \circ \mid (\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \circ, \eta) \quad \sigma ::= \bigwedge \{\tau_1, \dots, \tau_\ell\} \quad \eta \text{ (flag)} ::= \mathbf{nc} \mid \langle f \rangle$$

Here, f ranges over \mathcal{C} . We define $\text{flag}(\tau)$ by $\text{flag}(\circ) = \mathbf{nc}$ and $\text{flag}(\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \circ, \eta) = \eta$.

We often write $\tau_1 \wedge \cdots \wedge \tau_k$ and \top for $\bigwedge \{\tau_1, \dots, \tau_k\}$ and $\bigwedge \emptyset$ respectively. We assume a certain total order $<$ on the intersection types. Intuitively, the type \circ describes trees. The type $\bigwedge \{\tau_1, \dots, \tau_\ell\}$ describes terms that behave like a value of type τ_i for every $i \in \{1, \dots, \ell\}$. The type $(\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \circ, \eta)$ describes functions that take arguments of types $\sigma_1, \dots, \sigma_k$ and return a tree of type \circ . The flag η describes how the term behaves *after the transformation* for removing unused arguments. If $\eta = \langle f \rangle$, then the term behaves like f after the transformation, and if $\eta = \mathbf{nc}$, the term does not behave like any of the combinators in \mathcal{C} . For example, the term $\lambda x.\lambda y.y$ has type $(\top \rightarrow \circ \rightarrow \circ, \langle \lambda y.y \rangle)$, because after removing the redundant argument x , the term behaves like the identity function $\lambda y.y$.

We consider only types that respect underlying sorts. The operation $\llbracket \cdot \rrbracket$ given below maps an intersection type to the simple type obtained by the grammar transformation.

$$\begin{aligned} \llbracket (\tilde{\sigma} \rightarrow \circ, \eta) \rrbracket &= \llbracket \tilde{\sigma} \rightarrow \circ \rrbracket & \llbracket \circ \rrbracket &= \circ \\ \llbracket \bigwedge \{\tau_1, \dots, \tau_\ell, \tau'_1, \dots, \tau'_\ell\} \rightarrow \tilde{\sigma} \rightarrow \circ \rrbracket &= \llbracket \tau_1 \rrbracket \rightarrow \cdots \rightarrow \llbracket \tau_\ell \rrbracket \rightarrow \llbracket \tilde{\sigma} \rightarrow \circ \rrbracket \\ &\text{if } \text{flag}(\tau'_j) \neq \mathbf{nc} \text{ and } \text{flag}(\tau_j) = \mathbf{nc} \text{ and } j < j' \text{ implies } \tau_j < \tau_{j'} \end{aligned}$$

Here, $\tilde{\sigma} \rightarrow \circ$ is an abbreviation of $\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \circ$. The type τ is called a **refinement** of κ , if $\tau :: \kappa$ is derivable by the following rules.

$$\frac{}{\circ :: \circ} \quad \frac{\sigma_i :: \kappa_i \text{ for each } i \in \{1, \dots, k\} \quad \emptyset \vdash \langle f \rangle : \llbracket (\tilde{\sigma} \rightarrow \circ, f) \rrbracket}{(\tilde{\sigma} \rightarrow \circ, \langle f \rangle) :: \tilde{\kappa} \rightarrow \circ} \\ \frac{\tau_i :: \kappa \text{ for each } i \in \{1, \dots, k\}}{\bigwedge \{\tau_1, \dots, \tau_k\} :: \kappa} \quad \frac{}{(\tilde{\sigma} \rightarrow \circ, \mathbf{nc}) :: \tilde{\kappa} \rightarrow \circ}$$

Henceforth we consider only intersection types that are refinement of some simple types. For example, intersection types like $\bigwedge \{\circ, (\circ \rightarrow \circ, \mathbf{nc})\} \rightarrow \circ$ and $(\circ \rightarrow \circ, \langle \lambda f.\lambda x.f(x) \rangle)$ are excluded out.

Transformation Rules. We define the term transformation relation $\Gamma \vdash t : \tau \Rightarrow e$, where: (i) Γ is an (intersection) type environment, i.e., a set of type bindings of the form $\{x_1:\tau_1, \dots, x_k:\tau_k\}$, where each variable may occur more than once (we often omit curly brackets and just write $x_1:\tau_1, \dots, x_k:\tau_k$); (ii) t is a term; (iii) τ is the type of t ; and (iv) e is an extended term. When $\sigma = \bigwedge\{\tau_1, \dots, \tau_k\}$, we sometimes write $x : \sigma$ for $x : \tau_1, \dots, x : \tau_k$. Intuitively, $\Gamma \vdash t : \tau \Rightarrow e$ means that the term t corresponds to e , when t behaves as specified by τ . For example, if $\Gamma = \{g : (\circ \rightarrow \circ, \langle \lambda x.x \rangle)\}$, then $\Gamma \vdash g \mathbf{e} : \tau \Rightarrow \langle \lambda x.x \rangle \mathbf{e}$ should hold, since Γ says that g will be transformed to a term that behaves like $\lambda x.x$.

The transformation relation is inductively defined by the following rules:

$$\frac{\text{flag}(\tau) = \langle f \rangle}{x : \tau \vdash x : \tau \Rightarrow \langle f \rangle} \quad (\text{X-VARC}) \qquad \frac{\text{flag}(\tau) = \mathbf{nc}}{x : \tau \vdash x : \tau \Rightarrow x_\tau} \quad (\text{X-VAR})$$

$$\frac{\emptyset \vdash a : (\underbrace{\circ \rightarrow \dots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ, \mathbf{nc}) \Rightarrow a}{\emptyset \vdash A : \tau \Rightarrow A_\tau} \quad (\text{X-T}) \qquad \frac{\text{flag}(\tau) = \mathbf{nc}}{\emptyset \vdash A : \tau \Rightarrow A_\tau} \quad (\text{X-NT})$$

$$\frac{A x_1 \dots x_k \rightarrow t \in \mathcal{R} \quad f = \lambda \mathbf{Vars}(\{x_1 : \sigma_1, \dots, x_k : \sigma_k\}, x_1 \dots x_k). e \in \mathcal{C} \quad \tau = (\sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \circ, \langle f \rangle) \quad x_1 : \sigma_1, \dots, x_k : \sigma_k \vdash t : \circ \Rightarrow e}{\emptyset \vdash A : \tau \Rightarrow \langle f \rangle} \quad (\text{X-NTC})$$

$$\frac{\Gamma_0 \vdash t_0 : (\bigwedge\{\tau_1, \dots, \tau_\ell\} \rightarrow \rho, \eta) \Rightarrow e_0 \quad \eta' = \begin{cases} \eta & \text{if } k = 0 \\ \mathbf{nc} & \text{if } k > 0 \end{cases} \quad \Gamma_i \vdash t_1 : \tau_i \Rightarrow E_i \quad \text{flag}(\tau_i) = \mathbf{nc} \text{ for } i \in \{1, \dots, k\} \quad \tau_i < \tau_j \text{ if } i < j \leq k \quad \Gamma_i \vdash t_1 : \tau_i \Rightarrow e_{1,i} \quad \text{flag}(\tau_i) \neq \mathbf{nc} \text{ for } i \in \{k+1, \dots, \ell\}}{\Gamma_0 \cup \bigcup_{i \in \{1, \dots, \ell\}} \Gamma_i \vdash t_0 t_1 : (\rho, \eta') \Rightarrow e_0 E_1 \dots E_k} \quad (\text{X-APP})$$

$$\frac{\Gamma \vdash t : \tau \Rightarrow \langle \lambda x_1. \dots \lambda x_k. e_0 \rangle E_1 \dots E_k \quad [E_1/x_1, \dots, E_k/x_k] \models e_0 \rightsquigarrow e}{\Gamma \vdash t : \tau \Rightarrow e} \quad (\text{X-RED})$$

$$\frac{\Gamma_i \vdash t : \tau \Rightarrow e_i \text{ for each } i \in \{1, \dots, k\} \quad k \geq 1}{\Gamma_1 \cup \dots \cup \Gamma_k \vdash t : \tau \Rightarrow \{e_1, \dots, e_k\}} \quad (\text{X-SET})$$

In the rule X-NTC above, $\mathbf{Vars}(\Gamma, \tilde{x})$ (where \tilde{x} is a possibly empty sequence of variables) is a sequence of type bindings defined by (recall that $<$ is the total order on intersection types):

$$\mathbf{Vars}(\Gamma, \epsilon) = \epsilon \quad \mathbf{Vars}(\Gamma, x\tilde{y}) = (x_{\tau_1} : \llbracket \tau_1 \rrbracket) \dots (x_{\tau_k} : \llbracket \tau_k \rrbracket) \mathbf{Vars}(\Gamma, \tilde{y})$$

where $\{\tau_1, \dots, \tau_k\} = \{\tau \mid x : \tau \in \Gamma, \text{flag}(\tau) = \mathbf{nc}\}$ and $\tau_1 < \dots < \tau_k$.

Here is some explanation of the transformation rules. The rule X-VARC ensures that if x behaves like f , then x is replaced with $\langle f \rangle$; this allows us to propagate information about elements of \mathcal{C} during the transformation, and avoid

passing them around as function arguments. The rule X-VAR says that if x does not behave like an element of \mathcal{C} , then the variable is replicated for each type τ . (Here, we assume that x_τ and x'_τ are different variables if $x \neq x'$ or $\tau \neq \tau'$.) Similarly, there are two rules for non-terminals, depending on whether the body of a rule behaves like an element of \mathcal{C} . The rule X-APP is for applications. We ensure that only terms with **nc** flags remain as arguments, so that terms behaving like elements of \mathcal{C} are not passed around. Each argument is now a *set* of terms; this is because the output of transformation may not be unique. For example, if F has both types $(\circ \rightarrow \top \rightarrow \circ, \mathbf{nc})$ and $(\top \rightarrow \circ \rightarrow \circ, \mathbf{nc})$ (which means that F may use either the first or second argument), then $F \mathbf{b} \mathbf{c}$ in an argument position would be replaced by $\{F_{(\circ \rightarrow \top \rightarrow \circ, \mathbf{nc})} \mathbf{b}, F_{(\top \rightarrow \circ \rightarrow \circ, \mathbf{nc})} \mathbf{c}\}$.

For a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an extended one $\mathcal{G}' = (\Sigma, \mathcal{N}', \mathcal{R}', S_\circ)$, we write $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$ if (i) $\mathcal{N}' = \{F_\tau \mapsto \llbracket \tau \rrbracket \mid \tau :: \mathcal{N}(F)\}$ and (ii) \mathcal{R}' is the set:

$$\begin{aligned} & \{F_{(\sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \circ, \mathbf{nc})} y_1 \cdots y_m \rightarrow e \mid \\ & (F x_1 \cdots x_k \rightarrow t) \in \mathcal{R} \wedge x_1 : \sigma_1, \dots, x_k : \sigma_k \vdash t : \circ \Rightarrow e \\ & \wedge \mathbf{Vars}(\{x_1 : \sigma_1, \dots, x_k : \sigma_k\}, x_1 \cdots x_k) = (y_1 : \kappa_1) \cdots (y_m : \kappa_m) \\ & \wedge (\sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \circ, \mathbf{nc}) :: \mathcal{N}(F) \wedge \lambda y_1 : \kappa_1. \cdots \lambda y_m : \kappa_m. e \notin \mathcal{C} \wedge e \not\rightarrow_\lambda \}. \end{aligned}$$

So far we have implicitly assumed the set \mathcal{C} is fixed when we write $\Gamma \vdash t : \tau \Rightarrow e$ and $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$. We write $\Gamma \vdash_{\mathcal{C}} t : \tau \Rightarrow e$ and $\vdash_{\mathcal{C}} \mathcal{G} \Rightarrow \mathcal{G}'$ if we wish to make the set \mathcal{C} explicit.

Example 5. Recall \mathcal{G}_0 in Example 1. Let $\mathcal{C} = \{\lambda g. \lambda x. g x, \lambda g. \lambda x. g x x\}$. By applying the transformation and removing redundant rules, we obtain the grammar $\mathcal{G}'_0 = (\Sigma, \mathcal{N}', \mathcal{R}', S_\circ)$, where $f = \lambda g. \lambda x. g x x$ and $\tau = ((\circ \rightarrow \circ, \mathbf{nc}) \rightarrow \circ \rightarrow \circ, \mathbf{nc})$ with:

$$\begin{aligned} \mathcal{N}' &= \{S_\circ : \circ, F_\tau : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ, T_\tau : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ\} \\ \mathcal{R}' &= \{S_\circ \rightarrow \mathbf{a} \mathbf{e} \mathbf{e}, S_\circ \rightarrow \mathbf{b} \mathbf{e} \mathbf{e}, S_\circ \rightarrow F_\tau \{\langle f \rangle \mathbf{a}\} \mathbf{e}, \\ & S_\circ \rightarrow F_\tau \{\langle f \rangle \mathbf{b}\} \mathbf{e}, S_\circ \rightarrow F_\tau \{\langle f \rangle \mathbf{a}, \langle f \rangle \mathbf{b}\} \mathbf{e}, \\ & F_\tau g x \rightarrow T_\tau g x, F_\tau g x \rightarrow F_\tau (T_\tau g) x, T_\tau g x \rightarrow g(g x)\}. \end{aligned}$$

The tree $\mathbf{a}(\mathbf{b} \mathbf{e} \mathbf{e})(\mathbf{a} \mathbf{e} \mathbf{e})$ is obtained as follows. (We omit the subscripts of non-terminals, as they happen to be the same for each original non-terminal.)

$$\begin{aligned} S &\longrightarrow F \{\langle f \rangle \mathbf{a}, \langle f \rangle \mathbf{b}\} \mathbf{e} \longrightarrow T \{\langle f \rangle \mathbf{a}, \langle f \rangle \mathbf{b}\} \mathbf{e} \longrightarrow \langle f \rangle \mathbf{a} \{\langle f \rangle \mathbf{a} \mathbf{e}, \langle f \rangle \mathbf{b} \mathbf{e}\} \\ &\longrightarrow \mathbf{a}(\langle f \rangle \mathbf{b} \mathbf{e})(\langle f \rangle \mathbf{a} \mathbf{e}) \longrightarrow^* \mathbf{a}(\mathbf{b} \mathbf{e} \mathbf{e})(\mathbf{a} \mathbf{e} \mathbf{e}). \end{aligned}$$

The following theorem states that the transformation preserves the language.

Theorem 1. *If \mathcal{G} is an order- n grammar and $\vdash \mathcal{G} \Rightarrow \mathcal{G}'$, then \mathcal{G}' is a valid order- n extended grammar and $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.*

4 Bounding the Size of Intermediate Terms

In this section, we restrict the order of grammars to 2, and let \mathcal{C} be the following set \mathcal{C}_m :

$$\begin{aligned} & \{\lambda x : \circ. x\} \cup \\ & \{\lambda y_1 \cdots y_k. y_i E_1 \cdots E_\ell \mid k, \ell \leq m, \text{ and } E_1 \cup \dots \cup E_\ell = \{y_1, \dots, y_k\} \setminus \{y_i\}\}. \end{aligned}$$

We shall show that for an extended order-2 grammar over \mathcal{C} , the size of intermediate terms occurring in a production of a tree π is linearly bounded by the size of π . The **size** $|e|$ of an extended term e is defined by:

$$\begin{aligned} |a| &= |x| = |A| = 1 \\ |e\{e_1, \dots, e_k\}| &= |e| + |e_1| + \dots + |e_k| \quad |\langle f \rangle\{e_1, \dots, e_k\}| = 1 + |e_1| + \dots + |e_k|. \end{aligned}$$

Here, e_1, \dots, e_k are different from each other. The size $|\pi|$ of a tree π is the size of π as an extended term, which is the same as the number of nodes and leaves of π . The property mentioned above is stated more formally as follows.

Theorem 2. *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be an order-2 extended grammar over \mathcal{C}_m with $\mathbf{ar}(\mathcal{G}) \leq m$. Then there exists an (effectively computable) constant c such that for every tree $\pi \in \mathbf{Tree}_\Sigma$, if $S_o \Longrightarrow_{\mathcal{G}}^* e$, $e \Longrightarrow_{\mathcal{G}}^* \pi$, then $|e| \leq c|\pi|$.*

The following main result of this paper is obtained as a corollary:

Corollary 1. *Fix an order-2 grammar \mathcal{G} , Then the membership problem $\pi \stackrel{?}{\in} \mathcal{L}(\mathcal{G})$ can be decided in a non-deterministic Turing machine in $O(|\pi|)$ space.*

Proof. Suppose $\mathbf{ar}(\mathcal{G}) = k'$ and $k = \max(k', 2)$. We first determine m of \mathcal{C}_m . For each order-1 type κ of arity k , the number of intersection types such that $\tau :: \kappa$ and $\mathbf{flag}(\tau) = \mathbf{nc}$ is 2^k . Thus, for each order-2 type $\kappa = \kappa_1 \rightarrow \dots \rightarrow \kappa_j \rightarrow \circ$ (with $j \leq k$), the arity of $\llbracket \sigma \rrbracket$ for σ such that $\sigma :: \kappa$ is at most $k \times 2^k$. Let $m = k \times 2^k$ and $\mathcal{C} = \mathcal{C}_m$. By Theorem 1, we can effectively construct an order-2 extended grammar \mathcal{G}' over \mathcal{C}_m such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$. By the above reasoning, $\mathbf{ar}(\mathcal{G}') \leq m$. Compute the constant c of Theorem 2. Since \mathcal{G} is fixed, those steps can be performed offline. Given π , one can non-deterministically apply reductions by $\Longrightarrow_{\mathcal{G}}$ either until π is obtained (and answer yes only in this case), the size of a term exceeds $c|\pi|$, or the reduction gets stuck. By Theorem 2, there is an execution sequence that outputs yes if and only if $\pi \in \mathcal{L}(\mathcal{G}')$. Since \mathcal{G} is fixed (therefore non-terminals, terminals, and \mathcal{C}_m are also fixed), the actual space required for storing each intermediate term e is also linearly bounded by $|e| \leq c|\pi|$; hence this computation can be simulated by a non-deterministic Turing machine with $O(|\pi|)$ space. \square

We sketch the proof of Theorem 2 in the rest of this section. We call a λ -term of the form $\lambda x_1 \dots \lambda x_k. x_{\theta(1)} x_{\theta(2)} \dots x_{\theta(k)}$ (where $k \geq 1$ and θ is a permutation on $\{1, \dots, k\}$) an **extended permutator**. The proof consists of two steps. In the first step, from the reduction sequence $e \Longrightarrow_{\mathcal{G}}^* \pi$, we construct a term M of the linear λ -calculus that simulates the behavior of e in $e \Longrightarrow_{\mathcal{G}}^* \pi$, such that $|e|$ is bounded by the measure $\mathit{asize}(M)$ defined below (which is the number of top-level abstractions and variables), and M contains extended permutators only in restricted positions. In the second step, we show that any linear λ -term M that satisfies the conditions above is linearly bounded by $|\pi|$. For space restriction, we discuss only the first step below. Details about the first step and the second step are found in the extended version.

We first define a translation from extended grammars to linear λ -calculus with product types.

Definition 7. The set of **linear types**, ranged over by γ , is given by:

$$\gamma ::= \circ \mid \gamma \times \cdots \times \gamma \rightarrow \gamma$$

We assume a total order \leq on linear types. The **refinement relation** $\gamma :: \kappa$ on types is defined by:

$$\frac{}{\circ :: \circ} \quad \frac{\gamma_{1,i} :: \kappa_1 \text{ for each } i \in \{1, \dots, k\} \quad \gamma_2 :: \kappa_2 \quad \gamma_{1,1} \leq \cdots \leq \gamma_{1,k}}{(\gamma_{1,1} \times \cdots \times \gamma_{1,k} \rightarrow \gamma_2) :: (\kappa_1 \rightarrow \kappa_2)}$$

Henceforth we consider only types γ such that $\gamma :: \kappa$ for some κ .

Definition 8. The set of **linear λ -terms**, ranged over by u , is given by:

$$u ::= x \mid uU \mid \lambda(x_1 : \gamma_1, \dots, x_k : \gamma_k).u \quad U ::= (u_1, \dots, u_k)$$

A linear λ -term u is called a **pure linear λ -term** if the size of every tuple in u is 1 (i.e., $k = 1$ for every subterm of the form $\lambda(x_1, \dots, x_k).u'$ or (u_1, \dots, u_k) and every type $\gamma_1 \times \cdots \times \gamma_k \rightarrow \gamma$). We define $asize(u)$ by:

$$\begin{aligned} asize(x) &= asize(\lambda(x_1, \dots, x_k).u) = 1 \\ asize(u_0(u_1, \dots, u_k)) &= asize(u_0) + asize(u_1) + \cdots + asize(u_k). \end{aligned}$$

We use a meta-variable M for pure linear λ -terms. We often omit parentheses for unary tuples, and write $\lambda x.u$ for $\lambda(x).u$, and u for (u) .

The type judgment relation $\Delta \vdash_L u : \gamma$ for linear λ -terms is given by:

$$\frac{}{\{x : \gamma\} \vdash_L x : \gamma} \quad \frac{\Delta \uplus \{x_1 : \gamma_1, \dots, x_k : \gamma_k\} \vdash_L u : \gamma}{\Delta \vdash_L \lambda(x_1, \dots, x_k).u : \gamma_1 \times \cdots \times \gamma_k \rightarrow \gamma}$$

$$\frac{\Delta_0 \vdash_L u_0 : \gamma_1 \times \cdots \times \gamma_k \rightarrow \gamma \quad \Delta_i \vdash_L u_i : \gamma_i \text{ for each } i \in \{1, \dots, k\}}{\Delta_0 \uplus \cdots \uplus \Delta_k \vdash_L u_0(u_1, \dots, u_k) : \gamma}$$

Here, $\Delta_0 \uplus \Delta_1$ is defined to be $\Delta_0 \cup \Delta_1$ only if $dom(\Delta_0) \cap dom(\Delta_1) = \emptyset$.

The transformation relations $\mathcal{K} \vdash e : \kappa \Rightarrow u : \gamma \dashv \Delta$ and $\mathcal{K} \vdash E : \kappa \Rightarrow U : \gamma_1 \times \cdots \times \gamma_k \dashv \Delta$ are defined by the rules below.

$$\frac{i \text{ fresh}}{\emptyset \vdash a : \underbrace{\circ \rightarrow \cdots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ \Rightarrow a^{(i)} : \underbrace{\circ \rightarrow \cdots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ \dashv a^{(i)} : \underbrace{\circ \rightarrow \cdots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ} \quad (\text{LX-CONST})$$

$$\frac{i \text{ fresh} \quad \gamma :: \kappa}{\{x : \kappa\} \vdash x : \kappa \Rightarrow x^{(i)} : \gamma \dashv x^{(i)} : \gamma} \quad (\text{LX-V}) \quad \frac{\emptyset \vdash f : \kappa \Rightarrow u : \gamma \dashv \emptyset}{\emptyset \vdash \langle f \rangle : \kappa \Rightarrow u : \gamma \dashv \emptyset} \quad (\text{LX-COM})$$

$$\frac{\emptyset \vdash \lambda x_1. \cdots \lambda x_k. e : \kappa \Rightarrow u : \gamma \dashv \Delta \quad F x_1 \cdots x_k \rightarrow e \in \mathcal{R}}{\emptyset \vdash F : \kappa \Rightarrow u : \gamma \dashv \Delta} \quad (\text{LX-NT})$$

$$\frac{\mathcal{K}_i \vdash e_i : \kappa \Rightarrow u_i : \gamma_i \dashv \Delta_i \text{ for each } i \in \{1, \dots, \ell\} \quad \gamma_1 \leq \dots \leq \gamma_\ell}{\mathcal{K}_1 \cup \dots \cup \mathcal{K}_\ell \vdash \{e_1, \dots, e_\ell\} : \kappa \Rightarrow (u_1, \dots, u_\ell) : \gamma_1 \times \dots \times \gamma_\ell \dashv \Delta_1 \uplus \dots \uplus \Delta_\ell} \quad (\text{LX-TSET})$$

$$\frac{\mathcal{K}_0 \vdash e_0 : \kappa_0 \rightarrow \kappa \Rightarrow u_0 : \gamma_1 \times \dots \times \gamma_k \rightarrow \gamma \dashv \Delta_0 \quad \mathcal{K}_1 \vdash E : \kappa_0 \Rightarrow U : \gamma_1 \times \dots \times \gamma_k \dashv \Delta_1}{\mathcal{K}_0 \cup \mathcal{K}_1 \vdash e_0 E : \kappa \Rightarrow u_0 U : \gamma \dashv \Delta_0 \uplus \Delta_1} \quad (\text{LX-APP})$$

$$\frac{\mathcal{K} \cup \{x : \kappa_0\} \vdash e : \kappa \Rightarrow u : \gamma \dashv \Delta, x^{(i_1)} : \gamma_1, \dots, x^{(i_\ell)} : \gamma_\ell \quad \gamma_1 \leq \dots \leq \gamma_\ell \quad x \notin \text{dom}(\mathcal{K})}{\mathcal{K} \vdash \lambda x. e : \kappa_0 \rightarrow \kappa \Rightarrow \lambda(x^{(i_1)}, \dots, x^{(i_\ell)}) . u : \gamma_1 \times \dots \times \gamma_\ell \rightarrow \gamma \dashv \Delta} \quad (\text{LX-AB})$$

The idea is to replicate each variable and terminal for each use in a rewriting sequence $e \rightarrow_{\mathcal{G}}^* \pi$. In rule LX-CONST, $a^{(i)}$ obtained by the translation is treated as a variable. In LX-NT, a non-terminal is (non-deterministically) expanded, and then transformed to a linear λ -term. In LX-TSET, we allow $e_i = e_j$ even if $i \neq j$.

Example 6. Recall \mathcal{G}'_0 in Example 5. The term $T\{\langle f \rangle \mathbf{a}, \langle f \rangle \mathbf{b}\} \{\mathbf{e}\}$ occurring in the production of $\mathbf{a}(\mathbf{b} \mathbf{e} \mathbf{e})(\mathbf{a} \mathbf{e} \mathbf{e})$ is transformed to:

$$\begin{aligned} & \left(\lambda(g^{(1)}, g^{(2)}, g^{(3)}) . \lambda(x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}) . g^{(1)}(g^{(2)}(x^{(1)}, x^{(2)}), (g^{(3)}(x^{(3)}, x^{(4)}))) \right) \\ & \left((\lambda g . \lambda(y^{(1)}, y^{(2)}) . g(y^{(1)}, y^{(2)})) \mathbf{a}^{(1)}, (\lambda g . \lambda(y^{(1)}, y^{(2)}) . g(y^{(1)}, y^{(2)})) \mathbf{b}^{(2)}, \right. \\ & \quad \left. (\lambda g . \lambda(y^{(1)}, y^{(2)}) . g(y^{(1)}, y^{(2)})) \mathbf{a}^{(3)} \right) \\ & (\mathbf{e}^{(1)}, \mathbf{e}^{(2)}, \mathbf{e}^{(3)}, \mathbf{e}^{(4)}) \end{aligned}$$

with $\Delta = \mathbf{a}^{(1)} : \circ \rightarrow \circ \rightarrow \circ, \mathbf{b}^{(2)} : \circ \rightarrow \circ \rightarrow \circ, \mathbf{a}^{(3)} : \circ \rightarrow \circ \rightarrow \circ, \mathbf{e}^{(1)} : \circ, \mathbf{e}^{(2)} : \circ, \mathbf{e}^{(3)} : \circ, \mathbf{e}^{(4)} : \circ$. Here we have reused labels (for i in LX-V) when there is no danger of variable confusion.

The transformation satisfies the following property.

Theorem 3. *If $e \rightarrow_{\mathcal{G}}^* \pi$, then there exists u such that $\emptyset \vdash e : \circ \Rightarrow u : \circ \dashv \Delta$ where for each terminal symbol a , the number of bindings of the form $a^{(i)}$ in Δ is the same as the number of occurrences of a in π .*

We can obtain the following property from the above theorem.

Theorem 4. *Let \mathcal{G} be an order-2 extended grammar over \mathcal{C}_m with $\text{ar}(\mathcal{G}) \leq m$. If $S \Rightarrow_{\mathcal{G}}^* e \Rightarrow_{\mathcal{G}}^* \pi$, then there exists a pure linear λ -term M that satisfies: (i) $\Delta \vdash_{\mathcal{L}} M : \circ$; (ii) $\text{codom}(\Delta) \subseteq \{\circ, \circ \rightarrow \circ \rightarrow \circ\}$ and $|\{x \mid \Delta(x) = \circ\}|$ equals the number of leaves of π ; (iii) $\text{asize}(M) \geq |e|$; (iv) M contains only top-level β -redexes; and (v) M does not contain any extended permutator in an argument position, nor any consecutive application of extended permutators.*

Proof Sketch. Since $e \Longrightarrow_{\mathcal{G}}^* \pi$, we also have $e \longrightarrow_{\mathcal{G}}^* \pi$. Thus, one can construct a term u that satisfies the condition of Theorem 3. Let M be the pure linear λ -term obtained from u by applying the currying transformation, and then normalizing all the redexes under λ -abstraction. Then M satisfies the required conditions. \square

In the second step, we show that $asize(M) \leq 28|\{x \mid x : \circ \in \Delta\}|$ holds for any pure linear λ -term M and type environment Δ that satisfy the conditions (i), (ii), (iv), and (v), from which Theorem 2 follows.

5 Related Work

As mentioned in Section 1, higher-order (formal) languages have been introduced in 1970's and actively studied since then, but a number of problems remain open especially about *unsafe* higher-order languages. Inaba and Maneth [6] proved that any *safe* higher-order (word) languages are context-sensitive; they actually proved the stronger result that the membership is in the intersection of *deterministic* linear space and NP. Context-sensitiveness of *unsafe* higher-order languages has been open (for order-2 or higher for the tree language case, and for order-3 or higher for the word language case).

Type-based techniques for reasoning about higher-order grammars have been recently applied to obtain simpler proofs for the decidability of higher-order (local) model checking [9,12], and the strictness of tree hierarchy [10]. Haddad [4] developed a type-based transformation to eliminate non-productive OI derivations in deterministic higher-order tree grammars. He has also recently developed a type-based method for logical reflection and selection (which is a kind of grammar transformation) [5]. There is some similarity between the resource λ -calculus [18] and extended terms. In the resource λ -calculus, a function may be applied to a multiset consisting of *linear* terms (which must be used *exactly* once) and *reusable* terms (which may be used *an arbitrary number of times*). In our extended terms, each element of a set must be used *at least* once.

6 Conclusion

We have shown that order-2 unsafe tree languages are context-sensitive, by using novel type-based grammar transformation. It is not yet clear whether this approach can be extended to show context-sensitiveness of languages of arbitrary orders. For the general case, we need to find an appropriate set \mathcal{C} of combinators, and generalize the arguments in Section 4, which are currently specific to the order-2 case. We expect that the grammar transformation in Section 3 is also useful for reasoning about other properties of higher-order languages, such as pumping lemmas for higher-order languages.

Acknowledgments. We thank anonymous reviewers for useful comments. This work was partially supported by JSPS KAKENHI 23220001 and the Mitsubishi Foundation.

References

1. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: Safety is not a restriction at level 2 for string languages. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 490–504. Springer, Heidelberg (2005)
2. Curry, H.B., Feys, R.: *Combinatory Logic*, vol. 1. North-Holland (1958)
3. Damm, W.: The IO- and OI-hierarchies. *Theor. Comput. Sci.* 20, 95–207 (1982)
4. Haddad, A.: IO vs OI in higher-order recursion schemes. In: Proceedings of FICS 2012. EPTCS, vol. 77, pp. 23–30 (2012)
5. Haddad, A.: Model checking and functional program transformations. In: Proceedings of FSTTCS 2013. LIPIcs, vol. 24, pp. 115–126 (2013)
6. Inaba, K., Maneth, S.: The complexity of tree transducer output languages. In: Proceedings of FSTTCS 2008. LIPIcs, vol. 2, pp. 244–255 (2008)
7. Kartzow, A., Parys, P.: Strictness of the collapsible pushdown hierarchy. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 566–577. Springer, Heidelberg (2012)
8. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
9. Kobayashi, N.: Model checking higher-order programs. *Journal of the ACM* 60(3) (2013)
10. Kobayashi, N.: Pumping by typing. In: Proceedings of LICS 2013, pp. 398–407. IEEE Computer Society (2013)
11. Kobayashi, N., Inaba, K., Tsukada, T.: On unsafe tree and leaf languages (2014) (in preparation)
12. Kobayashi, N., Ong, C.-H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: Proceedings of LICS 2009, pp. 179–188. IEEE Computer Society (2009)
13. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Proceedings of PLDI 2011, pp. 222–233 (2011)
14. Kobele, G.M., Salvati, S.: The IO and OI hierarchies revisited. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 336–348. Springer, Heidelberg (2013)
15. Maslov, A.N.: The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.* 15, 1170–1174 (1974)
16. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: Proceedings of LICS 2006, pp. 81–90. IEEE Computer Society (2006)
17. Ong, C.-H.L., Ramsay, S.: Verifying higher-order programs with pattern-matching algebraic data types. In: Proceedings of POPL 2011, pp. 587–598 (2011)
18. Pagani, M., della Rocca, S.R.: Solvability in resource lambda-calculus. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 358–373. Springer, Heidelberg (2010)
19. Turner, R.: An infinite hierarchy of term languages - an approach to mathematical complexity. In: Proceedings of ICALP, pp. 593–608 (1972)
20. Wand, M.: An algebraic formulation of the Chomsky hierarchy. In: Manes, E.G. (ed.) *Category Theory Applied to Computation and Control*. LNCS, vol. 25, pp. 209–213. Springer, Heidelberg (1975)

Game Semantics for Nominal Exceptions^{*}

Andrzej S. Murawski¹ and Nikos Tzevelekos²

¹ DIMAP and Department of Computer Science, University of Warwick

² School of Electronic Engineering and Computer Science, Queen Mary University of London

Abstract. We present a fully abstract denotational model for a higher-order programming language combining call-by-value evaluation and local exceptions. The model is built using nominal game semantics and is the first one to achieve both effective presentability and freedom from “bad exception” constructs.

1 Introduction

Exceptions are a standard programming effect for raising and handling eccentric program behaviour, and more generally for manipulating the flow of control. They are a key feature, for example, of ML, Java and C++. The raising of an exception forces a program to escape out of its context and to the nearest applicable exception-handler. Thus, exceptions provide a means of overriding nested behaviour of pure functional programs. The mechanism that allows handlers to recognise the exceptions to be handled usually relies on the use of names. In the paper we shall focus on modelling such *nominal* exceptions.

The difficulties in modelling (even soundly) nominal exceptions stem from the combination of name-locality and name-mobility with non-local control flow. In particular, traditional approaches do not cope with locality and examine global exceptions only via the exception monad [9]. On the other hand, existing game models [5] rely on Reynolds’ principle of modelling references as objects with read/write methods [13], extended to the modelling of exceptions as objects with raise/handle methods. The main defect of this principle is that, in order to achieve full abstraction, “bad” constructors have to be included in the syntax, which means that the language examined will include “bad exceptions”, which are terms of exception type that do not correspond to genuine exceptions, but rather to couplings of arbitrary raise/handle methods. These constructs, while solving the full-abstraction problem, distance the languages from the programming features they were set out to capture; in particular, term-equivalence is not conservative with respect to bad constructors. For example, “handle x in (raise x) with skip” is not equivalent to “skip”.

Nominal game semantics advocates a departure from Reynolds’ modelling rule and stipulates that “nameful” types be modelled by names rather than objects. Nominal games were introduced in [1] and [6] in order to provide the first fully abstract models of the ν -calculus and its extension with pointers (i.e. storable names) respectively. They constitute a ‘nominalised’ version of game semantics, in which names may appear in

^{*} Research funded by the Engineering and Physical Sciences Research Council (EP/J019577/1) and a Royal Academy of Engineering Research Fellowship (Tzevelekos).

```

class MyExn extends Exception {}

public class Trap {
  public static void main(String [] argv)
  throws Exception {
    Exception e1 = new MyExn();
    Exception e2 = new MyExn();
    try { foo(e1); }
    catch ( Exception x ) {
      System.out.printf("%b, %b",
                        x==e1, x==e2);
    }
  }
  static void foo(Exception e)
  throws Exception {
    throw(e);
  }
}

exception MyExn
let e1 = MyExn
let e2 = MyExn
let foo(x) = raise x;;

try foo(e1) with x -> (x==e1, x==e2)

-----

fun new_exn() =
  let exception MyExn
  fun eq(x) = case x of
    MyExn => true
  | _ => false
  in (MyExn, eq) end
val e1 = new_exn()
val e2 = new_exn()
fun foo(x) = raise x;

foo(#1 e1) handle x => (#2 e1 x, #2 e2 x)

```

Fig. 1. Code samples. Clockwise, from upper-left corner: Java, OCaml and SML. In the Java example, the catch clause in method main is able to trap the exception e1 raised by foo, extract its name and pass it to x. As a result, the program prints true,false. In OCaml, the same effect is achieved by pattern matching the handled expression. We instigate analogous behaviour in SML, using the generativity of the exception constructor to produce local exceptions.

plays as atomic moves. Put differently, they are ordinary games constructed within the universe of *nominal sets* [2]. A first attempt to model exceptions using nominal games was made in [15]. However, the close reliance on the monadic approach led to a model which was too intensional to yield an explicit characterisation of contextual equivalence and the full abstraction result had to be obtained through the intrinsic quotient construction ([15, Proposition 5.23]). The development of a direct model was left as a major challenge for future work in [15]. In the present paper, we meet that challenge by producing two fully abstract and effectively presentable models for higher-order languages with references and exceptions. The fact that our models are not quotiented yields a direct approach to proving program equivalences, with scope for future automation (cf. [11]). In particular, we prove new non-trivial equivalences (cf. Example 28).

We consider two kinds of exception-handling mechanisms, in Sections 2-4 and 5-6 respectively. In the first one, illustrated by the code samples in Figure 1, the handler is given explicit access to the exception names that it encounters. Another, less invasive approach, is to require the handler to specify which exception is to be intercepted, under the assumption that all the others will be propagated by default. This approach respects privacy of exceptions in that no handler may react to a freshly generated exception. The latter kind of exceptions turns out to lead to a slightly more complicated game model.¹

At the technical level our full abstraction results are obtained by modelling the exception type by an arena whose moves belong to a countable set of *names*. Additionally, players are allowed use moves of the form $e!$ (where e is an exception name) as answers to *arbitrary* questions. Uses of $e!$ can be taken to correspond to raising an exception. These two relatively simple enrichments, along with standard game semantic conditions such as alternation and well-bracketing, already give rise to a fully abstract model of the

¹ This is a common pattern in game semantics: fewer conditions are needed to describe models of richer languages, because the corresponding interactions are less constrained.

$$\begin{array}{c}
\frac{}{\mathbf{u}, \Gamma \vdash (), \Omega : \text{unit}} \quad \frac{i \in \mathbb{Z}}{\mathbf{u}, \Gamma \vdash i : \text{int}} \quad \frac{l \in \mathbf{u} \cap \mathcal{L}_\beta}{\mathbf{u}, \Gamma \vdash l : \text{ref } \beta} \quad \frac{e \in \mathbf{u} \cap \mathcal{E}}{\mathbf{u}, \Gamma \vdash e : \text{exn}} \\
\frac{\mathbf{u}, \Gamma \vdash M : \text{int} \quad \mathbf{u}, \Gamma \vdash N_0, N_1 : \theta}{\mathbf{u}, \Gamma \vdash \text{if0 } M \text{ then } N_0 \text{ else } N_1 : \theta} \quad \frac{\mathbf{u}, \Gamma \vdash M, N : \text{int}}{\mathbf{u}, \Gamma \vdash M \oplus N : \text{int}} \quad \frac{\mathbf{u}, \Gamma \vdash M, N : \text{exn}, \text{ref } \beta}{\mathbf{u}, \Gamma \vdash M = N : \text{int}} \\
\frac{(x : \theta) \in \Gamma}{\mathbf{u}, \Gamma \vdash x : \theta} \quad \frac{\mathbf{u}, \Gamma, x : \theta \vdash M : \theta'}{\mathbf{u}, \Gamma \vdash \lambda x^\theta. M : \theta \rightarrow \theta'} \quad \frac{\mathbf{u}, \Gamma \vdash M : \theta \rightarrow \theta' \quad \mathbf{u}, \Gamma \vdash N : \theta}{\mathbf{u}, \Gamma \vdash MN : \theta'} \\
\frac{\mathbf{u}, \Gamma \vdash M : \beta}{\mathbf{u}, \Gamma \vdash \text{ref}_\beta(M) : \text{ref } \beta} \quad \frac{\mathbf{u}, \Gamma \vdash M : \text{ref } \beta}{\mathbf{u}, \Gamma \vdash !M : \beta} \quad \frac{\mathbf{u}, \Gamma \vdash M : \text{ref } \beta \quad \mathbf{u}, \Gamma \vdash N : \beta}{\mathbf{u}, \Gamma \vdash M := N : \text{unit}} \\
\frac{}{\mathbf{u}, \Gamma \vdash \text{exn}() : \text{exn}} \quad \frac{\mathbf{u}, \Gamma \vdash M : \text{exn}}{\mathbf{u}, \Gamma \vdash \text{raise } M : \theta} \quad \frac{\mathbf{u}, \Gamma \vdash M : \theta \quad \mathbf{u}, \Gamma, x : \text{exn} \vdash N : \theta}{\mathbf{u}, \Gamma \vdash M \text{ handle } x \Rightarrow N : \theta}
\end{array}$$

Fig. 2. Syntax of ExnML

first kind of exceptions, i.e. handlers have direct access to exception names. To model the other type of handlers, we identify a compositional subclass of strategies that must propagate any exceptions unless they were revealed to the program by the environment. In both cases, we obtain an explicit characterisation of contextual equivalence through the induced sets of *complete* plays, ones in which all questions are answered. In the setting where handling of private exceptions is not available, the latter set needs to be appropriately trimmed, so as to reflect the handling restrictions on environments.

2 A Language with Local Exceptions and Ground References

We introduce the language ExnML, which is a fragment of ML with full ground references² augmented with nominal exceptions. Its types θ are generated according to the following grammar.

$$\theta ::= \beta \mid \theta \rightarrow \theta \quad \beta ::= \text{unit} \mid \text{int} \mid \text{exn} \mid \text{ref } \beta$$

Note that reference types are available for each type of the shape β , including the exception type (full ground storage). We assume disjoint denumerable sets \mathcal{L} and \mathcal{E} of *locations* and *exceptions* respectively, such that $\mathcal{L} = \biguplus_\beta \mathcal{L}_\beta$. We range over location names by l, l' and over exception names by e, e' . Terms are typed in contexts (\mathbf{u}, Γ) , where \mathbf{u} a finite subset of $\mathcal{L} \cup \mathcal{E}$ and Γ is a variable context. Moreover, we assume a fixed set of binary integer operators ranged over by \oplus . The terms of the language are given by the following grammar (all $i \in \mathbb{Z}$), while its typing rules are in Figure 2.

$$\begin{array}{l}
M ::= () \mid \Omega \mid i \mid l \mid e \mid x \mid \lambda x^\theta. M \mid MM \mid \text{if0 } M \text{ then } M \text{ else } M \mid M \oplus M \\
\quad \mid M = M \mid \text{ref}_\beta(M) \mid !M \mid M := M \mid \text{exn}() \mid \text{raise } M \mid M \text{ handle } x \Rightarrow M
\end{array}$$

² Elements of all ground types are storable. We omit higher-order references in order not to complicate the exposition. The game model of higher-order references from [10] can be extended to exceptions by following Section 3.

We shall write $\Gamma \vdash M : \theta$ iff $\emptyset, \Gamma \vdash M : \theta$ can be derived using the rules of Figure 2. Similarly, $\vdash M : \theta$ is shorthand for $\emptyset, \emptyset \vdash M : \theta$. In what follows, we write $M; N$ for the term $(\lambda z^\theta. N)M$, where z does not occur in N and θ matches the type of M . let $x = M$ in N will stand for $(\lambda x^\theta. N)M$ in general. *Value terms* v , are given by:

$$v ::= () \mid i \mid e \mid l \mid \lambda x^\theta. M$$

The operational semantics of the language utilises finite *stores*, which record generated exceptions and assign to locations atomic values of compatible type:

$$\text{Sto} = \{s : \mathcal{L} \rightarrow_{\text{fin}} \text{Val} \mid l \in \text{dom}(s) \cap \mathcal{L}_\beta \implies s(l) \in \text{Val}_\beta\} \times \mathcal{P}_{\text{fin}}(\mathcal{E}),$$

where $\text{Val} = \text{Val}_{\text{unit}} \cup \text{Val}_{\text{int}} \cup \text{Val}_{\text{exn}} \cup \biguplus_\beta \text{Val}_{\text{ref}\beta}$, $\text{Val}_{\text{unit}} = \{*\}$, $\text{Val}_{\text{int}} = \mathbb{Z}$, $\text{Val}_{\text{exn}} = \mathcal{E}$, $\text{Val}_{\text{ref}\beta} = \mathcal{L}_\beta$. We range over Sto by Σ, T (and variants). Given $\Sigma \in \text{Sto}$ we write Σ_1, Σ_2 to refer to its respective components. Stores must be *closed* in the following sense: for all $\Sigma \in \text{Sto}$ and $l \in \text{dom}(\Sigma_1)$,

$$(\Sigma_1(l) \in \mathcal{L} \implies \Sigma_1(l) \in \text{dom}(\Sigma_1)) \wedge (\Sigma_1(l) \in \mathcal{E} \implies \Sigma_1(l) \in \Sigma_2).$$

Finally, we let *evaluation contexts* be given by the syntax:

$$\begin{aligned} E ::= & [-] \mid E N \mid (\lambda x. M) E \mid \text{if } 0 E \text{ then } N_0 \text{ else } N_1 \mid E \oplus N \mid i \oplus E \mid \text{ref}_\gamma(E) \mid E := N \\ & \mid l := E \mid !E \mid E = N \mid e = E \mid l = E \mid E \text{ handle } x \Rightarrow N \mid \text{raise } E. \end{aligned}$$

We write E_{-H} for contexts E derived from the above grammar applying any of the rules apart from $E \text{ handle } x \Rightarrow N$. In Figure 3 we give a small-step reduction relation for terms in contexts from Sto . Given $\vdash M : \text{unit}$ we write $M \Downarrow$ iff $(\emptyset, \emptyset), M \longrightarrow \Sigma, ()$ for some Σ .

Definition 1. We say that the term-in-context $u, \Gamma \vdash M_1 : \theta$ *approximates* $u, \Gamma \vdash M_2 : \theta$ (written $u, \Gamma \vdash M_1 \sqsubseteq M_2$) if $C[M_1] \Downarrow$ implies $C[M_2] \Downarrow$ for any context $C[-]$ such that $u, \emptyset \vdash C[M_1], C[M_2] : \text{unit}$. Two terms-in-context are *equivalent* if one approximates the other (written $u, \Gamma \vdash M_1 \cong M_2$).

Example 2. Take the terms $\vdash M_1, M_2 : \text{unit} \rightarrow \text{unit}$ to be respectively

$$M_1 \equiv \text{let } y = \text{exn}() \text{ in } \lambda x^{\text{unit}}. \text{raise } y \quad \text{and} \quad M_2 \equiv \lambda x^{\text{unit}}. \text{raise} (\text{exn}()).$$

Their game semantics will contain the following plays respectively

$$q_0^{\Sigma_0} \star_0^{\Sigma_0} q^{\Sigma_0} e_1!^{\Sigma_1} q^{\Sigma_1} e_1!^{\Sigma_1} \dots q^{\Sigma_1} e_1!^{\Sigma_1} \quad q_0^{\Sigma_0} \star_0^{\Sigma_0} q^{\Sigma_0} e_1!^{\Sigma_1} \dots q^{\Sigma_{k-1}} e_k!^{\Sigma_k}$$

where $\Sigma_i = (\emptyset, \{e_1, \dots, e_i\})$. Handlers of ExnML can extract the name of an exception and remember it for future comparisons. Accordingly, we have $\vdash M_1 \not\cong M_2$ (cf. Example 19).

$\Sigma, (\lambda x.M)v \longrightarrow \Sigma, M[v/x]$	$\Sigma, \text{if}0 \text{ then } N_0 \text{ else } N_1 \longrightarrow \Sigma, N_0$
$\Sigma, i_1 \oplus i_2 \longrightarrow \Sigma, (i_1 \oplus i_2)$	$\Sigma, \text{if}0 \text{ } i \text{ then } N_0 \text{ else } N_1 \longrightarrow \Sigma, N_1 \quad (i \neq 0)$
$\Sigma, !l \longrightarrow \Sigma, s(l)$	$\Sigma, \text{ref}_\gamma(v) \longrightarrow \Sigma[l \mapsto v], l \quad (l \notin \text{dom}(\Sigma_1))$
$\Sigma, l := v \longrightarrow \Sigma[l \mapsto v], ()$	$\Sigma, \text{exn}() \longrightarrow \Sigma \cup \{e\}, e \quad (e \notin \Sigma_2)$
$\Sigma, e = e \longrightarrow \Sigma, 1$	$\Sigma, e = e' \longrightarrow \Sigma, 0 \quad (e \neq e')$
$\Sigma, l = l \longrightarrow \Sigma, 1$	$\Sigma, l = l' \longrightarrow \Sigma, 0 \quad (l \neq l')$
$\Sigma, v \text{ handle } x \Rightarrow N \longrightarrow \Sigma, v$	$\Sigma, (\text{raise } e) \text{ handle } x \Rightarrow N \longrightarrow \Sigma, N[e/x]$
$\Sigma, E_{-H}[\text{raise } e] \longrightarrow \Sigma, \text{raise } e$	$\frac{\Sigma, M \longrightarrow \Sigma', M'}{\Sigma, E[M] \longrightarrow \Sigma', E[M']}$

Fig. 3. Small-step operational semantics of ExnML

3 Game Semantics

We construct a game model for ExnML by extending the fully abstract model of *Ground ML* [11] so as to incorporate nominal exceptional effects. Let \mathbb{A} be a countably infinite collection of *names*, corresponding to reference and exception names:

$$\mathbb{A} = \bigsqcup_{\beta} \mathbb{A}_{\beta} \uplus \mathbb{A}_e \quad \text{where } \mathbb{A}_{\beta} = \mathcal{L}_{\beta}, \mathbb{A}_e = \mathcal{E}.$$

We range over names with $a, b, \text{etc.}$, and also l, e when we want to be specific about their kind. The model is constructed using mathematical objects (moves, plays, strategies) that will feature names drawn from \mathbb{A} . Although names underpin various elements of our model, their precise nature is irrelevant. Hence, all of our definitions preserve name-invariance, i.e. our objects are (strong) *nominal sets* [2,16]. Note that we do not need the full power of the theory but mainly the basic notion of name-permutation. Here permutations are bijections $\pi : \mathbb{A} \rightarrow \mathbb{A}$ with finite support which respects the indexing of name-sets. For an element x belonging to a (nominal) set X , we write $\nu(x)$ for its name-support, i.e. the set of names occurring in x . Moreover, for any $x, y \in X$, we write $x \sim y$ if x and y are the same up to a permutation of \mathbb{A} . We let $[x] = \{y \in X \mid x \sim y\}$.

Our model is couched in the Honda-Yoshida style of modelling call-by-value computation [3]. Before we define what it means to play our games, let us introduce the auxiliary concept of an arena.

Definition 3. An *arena* $A = \langle M_A, I_A, \lambda_A, \vdash_A, M_e \rangle$ is given by:

- a set M_A of ordinary moves, a set $I_A \subseteq M_A$ of initial moves,
- a labelling function $\lambda_A : M_A \cup M_e \rightarrow \{O, P\} \times \{Q, A\}$,
- a justification relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A) \cup M_e$,
- and a fixed set $M_e = \{e!_O \mid e \in \mathbb{A}_e\} \cup \{e!_P \mid e \in \mathbb{A}_e\}$ of exceptional moves;

such that $M_A \cap M_e = \emptyset$ and, for all $m, m' \in M_A$ and $e \in \mathbb{A}_e$:

- $m \in I_A \implies \lambda_A(m) = (P, A)$,
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q$,

$$\begin{array}{ll}
 M_{A \otimes B} = (I_A \times I_B) \uplus \bar{I}_A \uplus \bar{I}_B & M_{A+B} = M_A \uplus M_B \\
 I_{A \otimes B} = I_A \times I_B & I_{A+B} = I_A \cup I_B \\
 \lambda_{A \otimes B} = [(\mathbf{i}_A, \mathbf{i}_B) \mapsto PA, \lambda_A \upharpoonright \bar{I}_A, \lambda_B \upharpoonright \bar{I}_B] & \lambda_{A+B} = [\lambda_A, \lambda_B] \\
 \vdash_{A \otimes B} = \{((\mathbf{i}_A, \mathbf{i}_B), m) \mid \mathbf{i}_A \vdash_A m \vee \mathbf{i}_B \vdash_B m\} \cup \bar{F}_A \cup \bar{F}_B & \vdash_{A+B} = \vdash_A \cup \vdash_B \\
 \\
 M_{A \Rightarrow B} = \{\star\} \uplus M_A \uplus M_B & M_{A \rightarrow B} = M_A \uplus M_B \\
 I_{A \Rightarrow B} = \{\star\} & I_{A \rightarrow B} = I_A \\
 \lambda_{A \Rightarrow B} = [\star \mapsto PA, \bar{\lambda}_A[\mathbf{i}_A \mapsto OQ], \lambda_B] & \lambda_{A \rightarrow B} = [\bar{\lambda}_A[\mathbf{i}_A \mapsto OQ], \lambda_B] \\
 \vdash_{A \Rightarrow B} = \{(\star, \mathbf{i}_A), (\mathbf{i}_A, \mathbf{i}_B)\} \cup \vdash_A \cup \vdash_B & \vdash_{A \rightarrow B} = \{(\mathbf{i}_A, \mathbf{i}_B)\} \cup \vdash_A \cup \vdash_B
 \end{array}$$

Fig. 4. Basic arena and prearena constructions

- $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$,
- $\lambda_A(e!_O) = (O, A) \wedge (\lambda_A(m) = (P, Q)) \implies m \vdash_A e!_O$,
- $\lambda_A(e!_P) = (P, A) \wedge (\lambda_A(m) = (O, Q)) \implies m \vdash_A e!_P$.

We write λ_A^{OP} (resp. λ_A^{QA}) for λ_A post-composed with the first (second) projection.

Note that, as M_e is fixed for all arenas A and so are the parts of λ_A, \vdash_A concerning moves from it, we will not be specifying them explicitly in definitions. We shall refer to moves from $M_A \cup M_e$ collectively as *moves of A* , we shall use i to range over initial moves (which are necessarily ordinary moves), and we shall range over exceptional moves via $e!$. Let $\bar{\lambda}_A$ be the *OP*-complement of λ_A . We define the basic (flat) arenas:

$$\begin{array}{ll}
 1 = \langle \{\star\}, \{\star\}, \{(\star, PA)\}, \emptyset \rangle & \mathbb{A}_\beta = \langle \mathbb{A}_\beta, \mathbb{A}_\beta, \{(a, PA) \mid a \in \mathbb{A}_\beta\}, \emptyset \rangle \\
 \mathbb{Z} = \langle \mathbb{Z}, \mathbb{Z}, \{(i, PA) \mid i \in \mathbb{Z}\}, \emptyset \rangle & \mathbb{A}_e = \langle \mathbb{A}_e, \mathbb{A}_e, \{(a, PA) \mid a \in \mathbb{A}_e\}, \emptyset \rangle
 \end{array}$$

Given arenas A, B , the arenas $A \otimes B$ and $A \Rightarrow B$ are constructed as in Figure 4, where $\bar{I}_A = M_A \setminus I_A, \bar{F}_A = (\vdash_A \upharpoonright \bar{I}_A \times \bar{I}_A)$ (and similarly for B). For each type θ we can now define the corresponding arena $\llbracket \theta \rrbracket$ by setting:

$$\llbracket \text{unit} \rrbracket = 1 \quad \llbracket \text{ref } \beta \rrbracket = \mathbb{A}_\beta \quad \llbracket \text{int} \rrbracket = \mathbb{Z} \quad \llbracket \text{exn} \rrbracket = \mathbb{A}_e \quad \llbracket \theta \rightarrow \theta' \rrbracket = \llbracket \theta \rrbracket \Rightarrow \llbracket \theta' \rrbracket$$

Although types are interpreted by arenas, the actual games will be played in *prearenas*, which are defined in the same way as arenas with the exception that initial moves are *O*-questions. Given arenas A, B we define the prearena $A \rightarrow B$ as in Figure 4. The moves will be accompanied by an explicit store component Σ . A *move-with-store* on a prearena A is thus a pair m^Σ with $m \in M_A \cup M_e$ and $\Sigma \in \text{Sto}$.

Definition 4. A *justified sequence* on a prearena A is a sequence of moves-with-store s on A such that, apart from the first move, which must be of the form i^Σ with $i \in I_A$, every move $n^{\Sigma'}$ in s is equipped with a pointer to an earlier move m^Σ such that $m \vdash_A n$. We then call m the *justifier* of n and, if $\lambda_A^{QA}(n) = A$, we also say that n *answers* m .

Remark 5. Note that, by definition, any exceptional move $e!$ can answer any question move in a play, as long as the latter has not already be answered. Thus, exceptional moves will model situations when evaluation of some term leads to raising an exception.

We shall write $s \sqsubseteq s'$ to mean that s is a prefix of s' . For each $S \subseteq \mathbb{A}$ and Σ we define $\Sigma^0(S) = S$ and $\Sigma^{i+1}(S) = \Sigma_1(\Sigma^i(S)) \cap \mathbb{A}$ ($i \geq 0$). Let $\Sigma^*(S) = \bigcup_i \Sigma^i(S)$. The set of *available names* of a justified sequence is defined inductively by $\text{Av}(\epsilon) = \emptyset$ and $\text{Av}(sm^\Sigma) = \Sigma^*(\text{Av}(s) \cup \nu(m))$. The *view* of a justified sequence s is defined as follows: $\text{view}(\epsilon) = \epsilon$, $\text{view}(m^\Sigma) = m^\Sigma$ and $\text{view}(s \overbrace{m^\Sigma t}^{n^{\Sigma'}}) = \text{view}(s) m^\Sigma n^{\Sigma'}$.

Definition 6. Let A be a prearena. A justified sequence s on A is called a **play**, if it satisfies the conditions below.

- No adjacent moves belong to the same player (*Alternation*).
- The justifier of each answer is the most recent unanswered question (*Bracketing*).
- For any $s'm^\Sigma \sqsubseteq s$ with non-empty s' , the justifier of m occurs in $\text{view}(s')$ (*Visibility*).
- For any $s'm^\Sigma \sqsubseteq s$, $\nu(\Sigma) = \text{Av}(s'm^\Sigma)$ (*Frugality*).

We write P_A for the set of plays on A .

We say that a name a is a *P-name* of a play s if there is $s'm^\Sigma \sqsubseteq s$ such that $a \in \nu(m^\Sigma) \setminus \nu(s')$ and m is a P-move. We write $P(s)$ for the set of all P-names of s . The set $O(s)$ is defined dually. We moreover define a partial function on alternating justified sequences which imposes the frugality condition by restricting the stores in moves to available names. More precisely, we define $\gamma(s)$ inductively by $\gamma(\epsilon) = \epsilon$ and:

$$\begin{aligned} \gamma(sm^\Sigma) &= \gamma(s) m^{\Sigma \upharpoonright \text{Av}(sm^\Sigma)} && \text{if } m \text{ an O-move} \\ \gamma(sm^\Sigma) &= \gamma(s) m^{\Sigma \upharpoonright \text{Av}(sm^\Sigma)} && \text{if } m \text{ a P-move, } \text{Av}(sm^\Sigma) \cap \nu(s) \subseteq \text{Av}(s) \\ &&& \text{and } \forall a \in \text{dom}(\Sigma) \setminus \text{Av}(sm^\Sigma). \Sigma(a) = \Sigma'(a) \end{aligned}$$

where, in the last clause above, the last move of s has store Σ' and, for each store Σ and set $S \subseteq \mathbb{A}$, $\Sigma \upharpoonright S = (\{(a, v) \in \Sigma_1 \mid a \in S\}, \Sigma_2 \cap S)$. Note that partiality arises from sequences breaking the conditions of the last clause.

Definition 7. A **strategy** σ on a prearena A is a set of even-length plays of A satisfying:

- If $so^\Sigma p^{\Sigma'} \in \sigma$ then $s \in \sigma$ (*Even-prefix closure*).
- If $s \in \sigma$ and $s \sim t$ then $t \in \sigma$ (*Equivariance*).
- If $s_1 p_1^{\Sigma_1}, s_2 p_2^{\Sigma_2} \in \sigma$ and $s_1 \sim s_2$ then $s_1 p_1^{\Sigma_1} \sim s_2 p_2^{\Sigma_2}$ (*Nominal determinacy*).

We write $\sigma : A$ for σ being a strategy on A .

Example 8. For each arena A , the strategy $\text{id}_A : A \rightarrow A$, is defined by

$$\text{id}_A = \{ s \in P_{A \rightarrow A}^{\text{even}} \mid \forall s' \sqsubseteq^{\text{even}} s. s' \upharpoonright A_l = s' \upharpoonright A_r \},$$

where the indices l, r distinguish the two copies of A , and $s' \upharpoonright A_x$ is the subsequence of s' containing only moves from the x -copy, along with any exceptional moves justified from them. For each arena A , let us write TA for the arena $1 \Rightarrow A$, i.e. $M_{TA} = \{\star_1, \star_2\} \uplus M_A$. Next we define the following exception-related strategies:

- $\text{raiz}_A : \mathbb{A}_e \rightarrow A = \{e^{\{e\}}e!^{\{e\}} \mid e \in \mathbb{A}_e\}$
- $\text{trap}_A : TA \rightarrow (A + \mathbb{A}_e) = \{\star_1 \star_2 s \mid s \in \text{id}_A\} \cup \{\star_1 \star_2 e!^{\{e\}}e^{\{e\}} \mid e \in \mathbb{A}_e\}$
- $\text{new}_e : 1 \rightarrow \mathbb{A}_e = \{\star e^{(\emptyset, \{e\})} \mid e \in \mathbb{A}_e\}$

Note that, in definitions like the above, we implicitly assume that we close the constructed set of plays under even-prefix closure. Thus, in raiz_A the play starts with O providing an exception name e , to which P answers by raising the exception $e!$ (thus, $e^{\{e\}}$ justifies $e!^{\{e\}}$); note that the play never opens in arena A . On the other hand, in trap_A , O starts the play by opening the initial move \star_1 under TA , to which P responds with a question \star_2 . At this point, O is given two choices: (a) to answer with an initial move of A , so the play will transform into a copycat between the A components of TA and $A + \mathbb{A}_e$; (b) to answer with an exceptional move $e!$, in which case P will ‘trap’ the name e and return it in the \mathbb{A}_e component of $A + \mathbb{A}_e$. Finally, in new_e P answers the initial move by playing a fresh exception name and adding it to the store. We will see below that the above mechanisms give us all the structure we need for modelling exceptional behaviours.

We proceed to strategy composition. Given arenas A, B, C , we define the prearena $A \rightarrow B \rightarrow C$ by setting $M_{A \rightarrow B \rightarrow C} = M_{A \rightarrow B} \uplus M_C$, $I_{A \rightarrow B \rightarrow C} = I_A$ and:

$$\lambda_{A \rightarrow B \rightarrow C} = [\lambda_{A \rightarrow B}[i_B \mapsto PQ], \bar{\lambda}_C] \quad \vdash_{A \rightarrow B \rightarrow C} = \vdash_{A \rightarrow B} \cup \{(i_B, i_C)\} \cup \vdash_C$$

Let u be a justified sequence on $A \rightarrow B \rightarrow C$. We define $u \upharpoonright BC$ to be u in which all A -moves and all exceptional moves justified by A -moves are suppressed. $u \upharpoonright AB$ is defined analogously, only that we also remove any exceptional move justified by an initial move of B . $u \upharpoonright AC$ is defined similarly with the caveat that, if there was a pointer from an initial C -move (resp. an exceptional move) to an initial B -move, which in turn had a pointer to an A -move, we add a pointer from the C -move (the exceptional move) to the A -move. Let us write $u \upharpoonright_\gamma X$ for $\gamma(u \upharpoonright X)$ with $X \in \{AB, BC, AC\}$. Below we shall often say that a move is an O - or a P -move in X meaning ownership in the associated prearena ($A \rightarrow B$, $B \rightarrow C$ or $A \rightarrow C$).

Definition 9. A justified sequence u on $A \rightarrow B \rightarrow C$ is an *interaction sequence* on A, B, C if it satisfies bracketing and frugality and, for all $X \in \{AB, BC, AC\}$, we have $(u \upharpoonright_\gamma X) \in P_X$ and the following conditions hold.

- $P(u \upharpoonright_\gamma AB) \cap P(u \upharpoonright_\gamma BC) = \emptyset$;
- $O(u \upharpoonright_\gamma AC) \cap (P(u \upharpoonright_\gamma AB) \cup P(u \upharpoonright_\gamma BC)) = \emptyset$;
- For each $u' \sqsubseteq u$ ending in $m^\Sigma m'^{\Sigma'}$ and $a \in \text{dom}(\Sigma')$ if
 - m' is a P -move in AB and $a \notin \text{Av}(u' \upharpoonright AB)$,
 - or m' is a P -move in BC and $a \notin \text{Av}(u' \upharpoonright BC)$,
 - or m' is an O -move in AC and $a \notin \text{Av}(u' \upharpoonright AC)$,

then $\Sigma(a) = \Sigma'(a)$.

We write $\text{Int}(A, B, C)$ for the set of interaction sequences on A, B, C , and $\sigma \parallel \tau$ for the set of interactions between strategies $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$:

$$\sigma \parallel \tau = \{u \in \text{Int}(A, B, C) \mid (u \upharpoonright_\gamma AB) \in \sigma \wedge (u \upharpoonright_\gamma BC) \in \tau\}.$$

and let $\sigma; \tau : A \rightarrow C = \{u \upharpoonright_\gamma AC \mid u \in \sigma \parallel \tau\}$.

The following result is deduced by translating our strategies into [7,10].

Lemma 10. *Strategy composition is associative and identity strategies are neutral elements. Thus, arenas and strategies yield a category of games \mathcal{G} .*

A first property of \mathcal{G} is that it has coproducts, given by $+$ and copairings $[\sigma, \tau] : (A + B) \rightarrow C = \sigma \cup \tau$ (for $A \xrightarrow{\sigma} C \xleftarrow{\tau} B$). Richer structure is highlighted below.

Remark 11. \mathcal{G} can be shown to host a lluf subcategory \mathcal{G}' , consisting of a variant of *single-threaded* strategies [7], where $(1, \otimes)$ yield finite products. Moreover, the operation T on arenas extends to a strong monad in \mathcal{G}' with T -exponentials, i.e. for all A, B, C there is a bijection $\Lambda^T : \mathcal{G}'(A \otimes B, TC) \cong \mathcal{G}'(A, B \Rightarrow C)$ natural in A, C . Then one can show that there exists a bijection $\Phi : \mathcal{G}(A, B) \cong \mathcal{G}'(A, TB)$, which establishes equivalence of \mathcal{G} and the Kleisli category on \mathcal{G}' determined by T (\mathcal{G}'_T). We write $\langle -, - \rangle$ for the left-pairing obtained in \mathcal{G} from pairing in \mathcal{G}'_T , and $\Lambda(-)$ for the weak exponential structure.

The above provides a canonical interpretation of application and λ -abstraction in \mathcal{G} . To interpret the remaining constructs of ExnML in \mathcal{G} , we need to define special morphisms for reference manipulation (cf. [14]) while for exceptions we shall use the morphisms from Example 8.

$$\begin{aligned} \text{get}_\beta &: \mathbb{A}_\beta \rightarrow \llbracket \beta \rrbracket = \{l^\Sigma \Sigma(l)^\Sigma \in P_{\mathbb{A}_\beta \rightarrow \llbracket \beta \rrbracket}\} \\ \text{set}_\beta &: \mathbb{A}_\beta \otimes \llbracket \beta \rrbracket \rightarrow 1 = \{(l, v)^\Sigma \star^\Sigma [l \rightarrow v] \in P_{\mathbb{A}_\beta \otimes \llbracket \beta \rrbracket \rightarrow 1}\} \\ \text{new}_\beta &: \llbracket \beta \rrbracket \rightarrow \mathbb{A}_\beta = \{v^\Sigma l^\Sigma [l \rightarrow v] \in P_{\llbracket \beta \rrbracket \rightarrow \mathbb{A}_\beta} \mid l \notin \text{dom}(\Sigma)\} \end{aligned}$$

We interpret any term-in-context $u, \Gamma \vdash M : \theta$ with a strategy $\llbracket u, \Gamma \vdash M : \theta \rrbracket : \llbracket u, \Gamma \vdash \theta \rrbracket$, denoted also as $\llbracket M \rrbracket : \llbracket u, \Gamma \vdash \theta \rrbracket$. The interpretation is given explicitly below. Suppose that $u = \{a_1, \dots, a_n\}$ and $\Gamma = \{x_1 : \theta_1, \dots, x_k : \theta_k\}$. We write $\llbracket u, \Gamma \rrbracket$ for the arena $\llbracket u \rrbracket \otimes \llbracket \theta_1 \rrbracket \otimes \dots \otimes \llbracket \theta_k \rrbracket$, where $\llbracket u \rrbracket$ is the flat arena $\langle M_u, I_u, \lambda_u, \vdash_u \rangle$ with $M_u = I_u = \langle (a_1, \dots, a_n) \rangle$.

- $\llbracket u, \Gamma \vdash () : \text{unit} \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{!} 1$, where $! = \{(\bar{a}, i_\Gamma)^\Sigma \star^\Sigma\}$.
- $\llbracket u, \Gamma \vdash \Omega : \text{unit} \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\perp} 1$, where $\perp = \{\epsilon\}$.
- $\llbracket u, \Gamma \vdash i : \text{int} \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{!} 1 \xrightarrow{\hat{i}} \mathbb{Z}$, where $\hat{i} = \{\star i\}$.
- $\llbracket u, \Gamma \vdash x_j : \theta_j \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\pi_{n+j}} \llbracket \theta_j \rrbracket$.
- $\llbracket u, \Gamma \vdash M_1 \oplus M_2 : \text{int} \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\langle \llbracket M_1 \rrbracket, \llbracket M_2 \rrbracket \rangle} \mathbb{Z} \otimes \mathbb{Z} \xrightarrow{\sigma_\oplus} \mathbb{Z}$, where $\sigma_\oplus = \{(i_1, i_2) (i_1 \oplus i_2)\}$.
- $\llbracket u, \Gamma \vdash \text{if0 } M \text{ then } N_0 \text{ else } N_1 : \theta \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\langle \llbracket M \rrbracket, \text{id} \rangle} \mathbb{Z} \otimes \llbracket u, \Gamma \rrbracket \xrightarrow{\text{if0} \otimes \text{id}} (1 + 1) \otimes \llbracket u, \Gamma \rrbracket \xrightarrow{\cong} \llbracket u, \Gamma \rrbracket + \llbracket u, \Gamma \rrbracket \xrightarrow{\llbracket \llbracket N_0 \rrbracket, \llbracket N_1 \rrbracket \rrbracket} \llbracket \theta \rrbracket$, where $\text{if0} = \{0 \star_l, i \star_r \mid i \neq 0\}$.
- $\llbracket u, \Gamma \vdash \text{ref}_\beta(M) : \text{ref } \beta \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket \beta \rrbracket \xrightarrow{\text{new}_\beta} \mathbb{A}_\beta$.
- $\llbracket u, \Gamma \vdash !M : \beta \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket} \mathbb{A}_\beta \xrightarrow{\text{get}} \llbracket \beta \rrbracket$.
- $\llbracket u, \Gamma \vdash M := N : \text{unit} \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\langle \llbracket M \rrbracket, \llbracket N \rrbracket \rangle} \mathbb{A}_\beta \otimes \llbracket \beta \rrbracket \xrightarrow{\text{set}_\beta} 1$.
- $\llbracket u, \Gamma \vdash MN : \theta' \rrbracket = \llbracket u, \Gamma \rrbracket \xrightarrow{\langle \llbracket M \rrbracket, \llbracket N \rrbracket \rangle} (\llbracket \theta \rrbracket \Rightarrow \llbracket \theta' \rrbracket) \otimes \llbracket \theta \rrbracket \xrightarrow{\text{ev}} \llbracket \theta' \rrbracket$.
- $\llbracket u, \Gamma \vdash \lambda x.M : \theta \rightarrow \theta' \rrbracket = \Lambda(\llbracket M \rrbracket : \llbracket u, \Gamma \rrbracket \otimes \llbracket \theta \rrbracket \rightarrow \llbracket \theta' \rrbracket)$.

- $\llbracket \mathbf{u}, \Gamma \vdash \text{exn}() : \text{exn} \rrbracket = \llbracket \mathbf{u}, \Gamma \rrbracket \xrightarrow{\mathbf{t}} 1 \xrightarrow{\text{new}_e} \mathbb{A}_e.$
- $\llbracket \mathbf{u}, \Gamma \vdash \text{raise } M : \theta \rrbracket = \llbracket \mathbf{u}, \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket} \mathbb{A}_e \xrightarrow{\text{raiz}_{\llbracket \theta \rrbracket}} \llbracket \theta \rrbracket.$
- $\llbracket \mathbf{u}, \Gamma \vdash M \text{ handle } x \Rightarrow N : \theta \rrbracket = \llbracket \mathbf{u}, \Gamma \rrbracket \xrightarrow{\langle \text{id}, \Phi(\llbracket M \rrbracket) \rangle} \llbracket \mathbf{u}, \Gamma \rrbracket \otimes T[\llbracket \theta \rrbracket] \xrightarrow{\text{id} \otimes \text{trap}_{\llbracket \theta \rrbracket}} \llbracket \mathbf{u}, \Gamma \rrbracket \otimes (\llbracket \theta \rrbracket + \mathbb{A}_e) \xrightarrow{\cong} (\llbracket \mathbf{u}, \Gamma \rrbracket \otimes \llbracket \theta \rrbracket) + (\llbracket \mathbf{u}, \Gamma \rrbracket \otimes \mathbb{A}_e) \xrightarrow{[\pi_2, \llbracket N \rrbracket]} \llbracket \theta \rrbracket.$

We can demonstrate that the game model is sound for contextual approximation (Proposition 12) by following the traditional route through Computational Soundness and Adequacy. For the former we show that we work in a modified version of a $\nu\epsilon\rho$ -model [15, Def. 5.13]. Recall that a play is *complete* if each question occurring in it justifies an answer. Given a set of plays X , let us write $\text{comp}(X)$ for the set of complete plays in X .

Proposition 12. *Let $\Gamma \vdash M_1, M_2 : \theta$ be terms of ExnML . $\text{comp}(\llbracket \Gamma \vdash M_1 : \theta \rrbracket) \subseteq \text{comp}(\llbracket \Gamma \vdash M_2 : \theta \rrbracket)$ implies $\Gamma \vdash M_1 \sqsubseteq M_2$.*

4 Full Abstraction

We prove full abstraction by showing that all finitary behaviours in the model are definable in ExnML . For the latter we use a factorisation argument which decomposes, in three steps, a strategy from \mathcal{G} into an *exception-free* strategy and strategies managing handling, raising and creation of exceptions respectively. Then, for the class of exception-free strategies we show that finitary members can be expressed in the fragment of ExnML corresponding to Ground ML.

We call a strategy σ *finitary* if the set $[\sigma] = \{[s] \mid s \in \sigma\}$ is finite (i.e. σ is *orbit-finite* in the nominal sense). For the first factorisation, we restrict strategies in the following manner. First, for each even-length play s , we let $\phi(s)$ be the justified sequence obtained from s by deleting all its O-moves of the form $e!^\Sigma$ (any e, Σ), as well as the moves following these. That is, $\phi(\epsilon) = \epsilon$ and

$$\phi(s m^\Sigma n^T) = \begin{cases} \phi(s) & \text{if } m^\Sigma = e!^T \\ \phi(s) m^\Sigma n^T & \text{otherwise} \end{cases}$$

We say that a play $s \in P_A$ is *exception-propagating* if $\gamma(\phi(s))$ is defined and, for all $s' e!^\Sigma m^T \sqsubseteq_{\text{even}} s, m^T = e!^\Sigma$. We write P_A^{prop} for the set of exception-propagating plays on A . We say that a strategy $\sigma : A$ is *exception-propagating* if $\sigma \subseteq P_A^{\text{prop}}$ and, for all $s \in \sigma$,

- for all $s e!^\Sigma \in P_A$, we have $s e!^\Sigma e!^\Sigma \in \sigma$;
- for all $s' \in P_A^{\text{prop}}$ with $\gamma(\phi(s)) = \gamma(\phi(s'))$, we have $s' \in \sigma$.

The former condition says that P always copycats raised exceptions, and the latter ensures that P cannot register moves that raise exceptions. We say that an exception-propagating strategy σ is *ϕ -finitary* if the set $\{[\gamma(\phi(s))] \mid s \in \sigma\}$ is finite.

Lemma 13. *Let $\sigma : A \rightarrow B$ be a strategy in \mathcal{G} . There is an exception-propagating strategy $\hat{\sigma} : \mathbb{A}_e \otimes A \otimes ((1 \Rightarrow 1) \Rightarrow \mathbb{A}_e) \rightarrow B$ such that³*

$$\sigma = \langle \langle !; [\text{exn}()] , \text{id} \rangle, !; [\lambda f. f() \text{ handle } x \Rightarrow x] \rangle; \hat{\sigma}.$$

Moreover, if σ is finitary then $\hat{\sigma}$ is ϕ -finitary.

Proof. Let $\tau = \langle \langle !; [\text{exn}()] , \text{id} \rangle, !; [\lambda f. f() \text{ handle } x \Rightarrow x] \rangle$ and $C = A_e \otimes A \otimes ((1 \Rightarrow 1) \Rightarrow \mathbb{A}_e) \rightarrow B$. We construct $\hat{\sigma} : C$ as follows. For each $s \in \sigma$, build \hat{s} in two stages. In the first stage, perform the following move replacements in s , from left to right.

- Replace the initial move i^Σ with $(h, i, \star)^\Sigma$, for some fresh $h \in \mathbb{A}_e$.
- Replace each P-question q^Σ with a sequence $q_1^\Sigma q_2^\Sigma q'^\Sigma$, where q_1 a question justified by the (newly added) initial \star , and q_2 justified by q_1 .
- Replace each exceptional move $e!^T$ of O, answering some previous q^Σ , with $e!^T e!^T e^T$, where the first (resp. second) $e!$ is justified by q (q_2), and e is justified by q_1 . Diagrammatically:

$$i \dots q^\Sigma \dots e!^T \dots \mapsto (h, i, \star) \dots q_1^\Sigma q_2^\Sigma q'^\Sigma \dots e!^T e!^T e^T \dots$$

- Replace each P-answer m^T (to some previous q'^Σ) with $h^T h^T \dots h^T h^T m^T$, where m is justified by q , and the h^T 's answer all open q_1 and q_2 moves that were added in the second step above and appear after q' . Note that these q_i 's are visible at the corresponding h because they are, in each such case, the pending question.

In the second stage, replace each store Σ in the resulting play with $(\Sigma_1, \Sigma_2 \uplus \{h\})$ (h is chosen fresh for s). We take $\hat{\sigma} = \{t \in P_C^{\text{PROP}} \mid \exists s \in \sigma. \gamma(\phi(t)) = \gamma(\phi(\hat{s}))\}$. Note first that $\hat{\sigma}$ includes the strategy $\sigma' = \{s \mid s \in \sigma\}$, as $\hat{s} \in P_C^{\text{PROP}}$ for all $s \in \sigma$, and $\tau; \sigma' = \sigma$. Hence, $\sigma = \tau; \hat{\sigma}$. By construction, $\hat{\sigma}$ is exception-propagating, and $[\gamma(\phi(\hat{\sigma}))]$ is finite if $[\sigma]$ is finite. Finally, note that the passage from σ' to $\hat{\sigma}$ does not break determinacy, as the moves deleted by ϕ are pairs of identical O/P moves. \square

The next factorisation eliminates from strategies the capability of raising exceptions. We say that an exception-propagating strategy σ is *handle/raise-free* if, for all $sm^T e!^\Sigma \in \sigma$, we have $m = e!$.

Lemma 14. *Let $\sigma : \mathbb{A}_e \otimes A \rightarrow B$ be an exception-propagating strategy. There is a handle/raise-free $\hat{\sigma} : \mathbb{A}_e \otimes A \otimes (\mathbb{A}_e \Rightarrow 1) \rightarrow B$ such that $\sigma = \langle \text{id}, !; [\vdash \lambda x. \text{raise } x] \rangle; \hat{\sigma}$. Moreover, if σ is ϕ -finitary then so is $\hat{\sigma}$.*

Proof. Let $\tau = \langle \text{id}, !; [\vdash \lambda x. \text{raise } x] \rangle$ and $C = \mathbb{A}_e \otimes A \otimes (\mathbb{A}_e \Rightarrow 1) \rightarrow B$. For each $s \in \sigma$ we construct \hat{s} by replacing each initial move $(h, i)^\Sigma$ with $(h, i, \star)^\Sigma$, and each P-move $e!^T$ breaking handle/raise-freeness with a sequence $e^T e!^T e!^T$. Diagrammatically ($m \neq e!$ and we omit some stores for brevity):

$$(h, i) \dots q \dots m^\Sigma e!^T \dots \mapsto (h, i, \star) \dots q \dots m^\Sigma e^T e!^T e!^T \dots$$

³ The role of the leftmost \mathbb{A}_e in $\hat{\sigma}$ is purely technical and fulfils two functions: (a) it supplies the default return value of $f() \text{ handle } x \Rightarrow x$; (b) it provides a default exception name to be used in subsequent factorisations removing reference generation (the name will be used as initial value for external generators of names in \mathbb{A}_{exn}).

We let $\hat{\sigma} = \{t \in P_C^{\text{PROP}} \mid \exists s \in \sigma. \gamma(\phi(t)) = \gamma(\phi(\hat{s}))\}$. As above, we have that $\tau; \hat{\sigma} = \sigma$. Since σ is exception-propagating, the move m above cannot be of the form $e'!$ (any $e' \in \mathbb{A}_e$), and therefore $\hat{\sigma}$ preserves the exception-propagating conditions. Moreover, by construction, $\hat{\sigma}$ is handle/raise-free, and $[\gamma(\phi(\hat{\sigma}))]$ is finite if $[\gamma(\phi(\sigma))]$ is. \square

Our final factorisation concerns removing any exception-name generation capability from our strategies. The technique is similar to the one used in the factorisations above and amounts to delegating all fresh exception-name creation to an external generator. Formally, a handle/raise-free strategy is called *exception-free* if, for all $s \in \sigma$, $P(s) \cap \mathbb{A}_e = \emptyset$.

Lemma 15. *Let $\sigma : \mathbb{A}_e \otimes A \rightarrow B$ be a handle/raise-free strategy. There is an exception-free $\hat{\sigma} : \mathbb{A}_e \otimes A \otimes (1 \Rightarrow \mathbb{A}_e) \rightarrow B$ such that $\sigma = \langle \text{id}, !; \llbracket \lambda z. \text{exn}() \rrbracket \rangle; \hat{\sigma}$. Moreover, if σ is ϕ -finitary then so is $\hat{\sigma}$.*

Let us call ExnML_{-e} the fragment of ExnML obtained by suppressing the constructors `handle`, `raise` and `exn()`. We can show that exception-freeness is captured by ExnML_{-e} in the following sense.

Lemma 16. *Let $\sigma : \mathbb{A}_e \otimes A \rightarrow B$ a ϕ -finitary exception-free strategy over a denotable prearena. There is an ExnML_{-e} term $u, \Gamma \vdash M : \theta$ such that $\llbracket M \rrbracket = \sigma$.*

Combining the four previous lemmas we obtain the following.

Proposition 17. *Let $\mathbb{A}_e \otimes A \rightarrow B$ be a denotable prearena and $\sigma : A \rightarrow B$ a finitary strategy. There is an ExnML term $u, \Gamma \vdash M : \theta$ such that $\llbracket M \rrbracket = \sigma$.*

Theorem 18. *For all ExnML -terms $\Gamma \vdash M_1, M_2 : \theta$, we have $\text{comp}(\llbracket \Gamma \vdash M_1 : \theta \rrbracket) \subseteq \text{comp}(\llbracket \Gamma \vdash M_2 : \theta \rrbracket)$ if, and only if, $\Gamma \vdash M_1 \sqsubset M_2$.*

5 Idealised Exceptions

The design of exception handling in ExnML was guided by common practice. In an idealised world, private exceptions should not be amenable to handling. This can be achieved by the alternative handling construct:

$$\frac{u, \Gamma \vdash M, N' : \theta \quad u, \Gamma \vdash N : \text{exn}}{u, \Gamma \vdash M \text{ handle } N \rightarrow N' : \theta}$$

We call ExnML_{\S} the language which differs from ExnML in featuring the above construct instead of “ $M \text{ handle } x \Rightarrow N$ ”. The new language has additional reduction rules:

$$\begin{aligned} \Sigma, (\text{raise } e) \text{ handle } e \rightarrow N &\longrightarrow \Sigma, N \\ \Sigma, E_{-e}[\text{raise } e] &\longrightarrow \Sigma, \text{raise } e \end{aligned}$$

Evaluation contexts are now given by:

$$\begin{aligned} E ::= [_] \mid E N \mid (\lambda x. M) E \mid \text{if0 } E \text{ then } N_0 \text{ else } N_1 \mid E \oplus N \mid i \oplus E \mid \text{ref}_{\gamma}(E) \mid E := N \\ \mid l := E \mid !E \mid E = N \mid e = E \mid M \text{ handle } E \rightarrow N \mid E \text{ handle } e \rightarrow N \mid \text{raise } E \end{aligned}$$

and, for each $e \in \mathcal{E}$, we write E_{-e} for contexts E derived by the above grammar applying any of the rules apart from E handle $e \rightarrow N$. Note that the new handler is easily definable in ExnML by:

$$M \text{ handle } N \rightarrow N' \equiv \text{let } z = N \text{ in } (M \text{ handle } x \Rightarrow (\text{if0 } x = z \text{ then raise } x \text{ else } N'))$$

Thus, ExnML_{\S} is a sublanguage of ExnML in terms of expressivity.

Example 19. Recall the terms M_1 and M_2 from Example 2. They will turn out equivalent in ExnML_{\S} , because in either case the private exceptions raised by the terms can only be propagated. Next we shall develop game-semantic constraints that reflect such scenarios.

6 Games Propagating Private Exceptions

We derive the game model of ExnML_{\S} by restricting the category \mathcal{G} with an additional condition on strategies. We need to depict semantically that terms in ExnML_{\S} are only able to handle exception names that are ‘known’ to them. In particular, fresh exceptions cannot be handled and will break through any evaluation context. Moreover, such exceptions cannot be remembered and neither can their accompanying stores. We therefore define the following notion of available subplay. For any even-length play s over some prearena A , we define the justified sequence $\$(s)$ inductively by $\$(\epsilon) = \epsilon$ and:

$$\$(s m^{\Sigma} n^T) = \begin{cases} \$(s) & \text{if } m = e! \text{ and } e \notin \text{Av}(\$(s)) \\ \$(s) m^{\Sigma} n^T & \text{otherwise} \end{cases}$$

We let $\text{Av}_{\S}(s) = \text{Av}(\$(s))$. The above definition disregards not only fresh exceptions raised by O , but also the P -moves succeeding them. This is due to the fact that the terms (and strategies) we consider simply propagate such exceptions.

Definition 20. We say that a play $s \in P_A$ is **\S -propagating** if $\gamma(\$(s))$ is defined and, for all $s' e!^{\Sigma} m^T \sqsubseteq_{\text{even}} s$ with $e \notin \text{Av}_{\S}(s)$, $m^T = e!^{\Sigma}$.

We say that a strategy $\sigma : A$ is **\S -propagating** if $\sigma \subseteq P_A^{\S\text{prop}}$ and, for all $s \in \sigma$,

- for all $s e!^{\Sigma} \in P_A$ and $e \notin \text{Av}_{\S}(s)$, we have $s e!^{\Sigma} e!^{\Sigma} \in \sigma$;
- for all $s' \in P_A^{\S\text{prop}}$ with $\gamma(\$(s)) = \gamma(\$(s'))$, we have $s' \in \sigma$.

We write $P_A^{\S\text{prop}}$ for the set of \S -propagating plays of A .

Thus, the former condition stipulates that strategies propagate raised exceptions if these feature fresh exception names. The latter ensures that strategies do not depend on these raised exceptions or their stores. We can show that these conditions are compositional. Suppose we compose \S -propagating strategies $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$. Exceptional moves suppressed by \S are O -moves, carrying O -names. Thus, by the name-ownership conditions of strategy-composition, if a move is suppressed in a composite play in AC , then it is also suppressed in its constituent plays in AB and BC . As a result, suppressed exceptions are propagated in σ and τ , resulting in propagation by $\sigma; \tau$. Similarly, saturation under $\gamma(\$(_))$ of $\sigma; \tau$ is ensured by componentwise saturation of σ and τ respectively.

Lemma 21. *If $\sigma : A \rightarrow B, \tau : B \rightarrow C$ are $\$$ -propagating then so is $\sigma; \tau$.*

Identity strategies are $\$$ -propagating by construction. We therefore obtain a lluf category $\mathcal{G}_\$$ of $\$$ -propagating strategies. Terms from $\text{ExnML}_\$$ are given denotations in $\mathcal{G}_\$$ as before, only that now we use the strategies $\text{hd}\$\!_A : TA \otimes \mathbb{A}_e \rightarrow A + 1 =$

$$\{(\star_1, e)^{\{e\}} \star_2^{\{e\}} s \mid s \in \text{id}_A, e \in \mathbb{A}_e\} \cup \{(\star_1, e)^{\{e\}} \star_2^{\{e\}} e!^{\{e\}} \star^{\{e\}} \mid e \in \mathbb{A}_e\}$$

instead of trap_A , which break $\$$ -propagation. With these constructs we obtain a $\nu\epsilon\rho$ -model [15] with references restricted to ground types. We thus have soundness.

Lemma 22. *For all $\text{ExnML}_\$$ -terms $\Gamma \vdash M_1, M_2 : \theta$, if $\llbracket \Gamma \vdash M_1 : \theta \rrbracket \subseteq \llbracket \Gamma \vdash M_2 : \theta \rrbracket$ then $\Gamma \vdash M_1 \sqsubseteq M_2$.*

Remark 23. In previous game models of control, the control-manipulating effect of privately-propagating exceptions was captured by relaxing the bracketing condition [4,8]. The latter was achieved in the expense of adding an additional pointer structure, called *control (or contingency) pointers*, to mark violations of bracketing. Here we took a different approach by exposing the private-exception mechanism that caters for such violations (cf. [15,8]). As a result, our model consists of plays that still satisfy bracketing, albeit in this extended setting. As in the case of control pointers, to avoid being overly intentional, we need to hide access to private exceptions via the propagation conditions.

While previously term approximation was characterised by inclusion of complete plays, now we have to restrict the set of plays to take into account $\$$ -propagation on the part of the environment. Given a complete play s on a prearena $1 \rightarrow A$ with final store Σ , we let $\hat{s} \equiv \star \overleftarrow{s} \star^{\Sigma} \in P_{(1 \Rightarrow A) \rightarrow 1}$. For each $\$$ -propagating strategy $\sigma : 1 \rightarrow A$, we then define $\text{comp}_\$(\sigma) = \{\gamma(\$(\hat{s})) \mid s \in \text{comp}(\sigma), \hat{s} \in P_{(1 \Rightarrow A) \rightarrow 1}^{\text{prop}}\}$.

Proposition 24. *For all $\text{ExnML}_\$$ -terms $\vdash M_1, M_2 : \theta$, if $\text{comp}_\$(\llbracket M_1 \rrbracket) \subseteq \text{comp}_\$(\llbracket M_2 \rrbracket)$ then $\vdash M_1 \sqsubseteq M_2$.*

Proof. Suppose the inclusion holds and let $C[M_1] \Downarrow$ for some context C . Then, by Lemma 22, $\llbracket C[M_1] \rrbracket = \{\star\star\}$, that is, $\llbracket \lambda z. M_1 \rrbracket; \llbracket f \vdash C[f()] \rrbracket = \{\star\star\}$. Let us write M'_i for $\lambda z. M_i$ ($i = 1, 2$), N for $\llbracket f \vdash C[f()] \rrbracket$, $\gamma_\$(\cdot)$ for $\gamma(\$(\cdot))$ and let $A = \llbracket \theta \rrbracket$. Then, $\llbracket M'_1 \rrbracket; \llbracket N \rrbracket = \{\star\star\}$, the latter due to composing some complete play $\star\star s \in \llbracket M'_1 \rrbracket$ with some $\star s \star^{\Sigma} \in \llbracket N \rrbracket$. Since $M'_1 \equiv \lambda \vec{x}. M_1$, s must be an interleaving of complete plays $s_1, \dots, s_k \in \llbracket M_1 \rrbracket$. For each i , we have $\hat{s}_i \in P_{(1 \Rightarrow A) \rightarrow 1}^{\text{prop}}$, because $\star s \star^{\Sigma} \in \llbracket N \rrbracket \subseteq P_{(1 \Rightarrow A) \rightarrow 1}^{\text{prop}}$, and therefore $\gamma_\$(\hat{s}_i) \in \text{comp}_\$(\llbracket M_1 \rrbracket)$. By hypothesis, $\gamma_\$(\hat{s}_i) \in \text{comp}_\$(\llbracket M_2 \rrbracket)$, and so there is $s'_i \in \text{comp}(\llbracket M_2 \rrbracket)$ such that $\gamma_\$(\hat{s}_i) = \gamma_\(\hat{s}'_i) . Let $s' \in P_{1 \rightarrow (1 \Rightarrow A)}$ be the interleaving of s'_1, \dots, s'_k obtained by simulating the interleaving pattern of s . Note that, for each i , since $\gamma_\$(\hat{s}_i) = \gamma_\(\hat{s}'_i) , s_i and s'_i share the same structure apart from P/O pairs of exceptional moves deleted by $\$$, which do not affect simulating the interleaving of s (change of thread can only occur in O-moves). We thus obtain some $\star s' \star^{\Sigma'} \in P_{(1 \Rightarrow A) \rightarrow 1}^{\text{prop}}$ and, by lifting equality under $\gamma_\$$ from threads to thread-interleavings, we have $\gamma_\$(\star s \star^{\Sigma}) = \gamma_\$(\star s' \star^{\Sigma'})$. But, since $\llbracket N \rrbracket$ is $\$$ -propagating, $\star s \star^{\Sigma} \in \llbracket N \rrbracket$ implies $\star s' \star^{\Sigma'} \in \llbracket N \rrbracket$ and therefore $\llbracket M'_2 \rrbracket; \llbracket N \rrbracket = \{\star\star\} = \llbracket C[M_2] \rrbracket$. By Lemma 22, then, $C[M_2] \Downarrow$. \square

For completeness, we again work our way through a finitary definability result, established via factorisations. The first factorisation brings us to handle-free strategies, from which the factorisations of the previous section can be applied. We say that an $\$$ -propagating strategy σ is $\$$ -finitary if the set $\{\llbracket \gamma(\$ (s)) \rrbracket \mid s \in \sigma\}$ is finite.

Lemma 25. *Let $\sigma : A \rightarrow B$ be a $\$$ -finitary strategy. There is some $n \in \omega$ and a ϕ -finitary handle-free strategy $\hat{\sigma} : \mathbb{A}_e \otimes A \otimes \mathbb{A}_{\text{exn}}^n \otimes ((1 \Rightarrow 1) \Rightarrow \mathbb{A}_e) \rightarrow B$ such that*

$$\sigma = \langle !; \llbracket \text{exn}() \rrbracket, \text{id} \rangle; \langle \text{id}, \pi_1; \langle \overline{\llbracket x : \text{exn} \vdash \text{ref}(x) \rrbracket}, \text{id} \rangle \rangle; \langle \pi_1, \overline{\llbracket z : \text{ref exn}, h : \text{exn} \vdash M \rrbracket} \rangle; \hat{\sigma}$$

with $M \equiv \lambda f. (\dots (f()); h \text{ handle} !z_1 \rightarrow !z_1) \text{ handle} !z_2 \rightarrow !z_2 \dots) \text{ handle} !z_n \rightarrow !z_n : \text{exn}$.

The above factorisation is identical to the corresponding one in the previous section, only that instead of simply delegating exception handling to the environment, the strategy $\hat{\sigma}$ also stores all exception names encountered, apart from those in $\$$ -removable moves, in the variables z_i .

Proposition 26. *Let $A \rightarrow B$ be a denotable arena. For each $\$$ -finitary strategy $\sigma : A \rightarrow B$ there is an $\text{ExnML}_{\$}$ term $u, \Gamma \vdash M : \theta$ such that $\llbracket M \rrbracket = \sigma$.*

We can now prove completeness, and thus full abstraction.

Theorem 27. *For all $\text{ExnML}_{\$}$ -terms $\vdash M_1, M_2 : \theta$, we have $\text{comp}_{\$}(M_1) \subseteq \text{comp}_{\$}(M_2)$ if, and only if, $\vdash M_1 \sqsubseteq M_2$.*

Proof. We show completeness (right-to-left). Let us write M'_i for $\lambda z. M_i$ ($i = 1, 2$). Suppose $s \in \text{comp}_{\$}(\llbracket M_1 \rrbracket) \setminus \text{comp}_{\$}(\llbracket M_2 \rrbracket)$. Let $A = \llbracket \theta \rrbracket$ and define the strategy $\rho : (1 \Rightarrow A) \rightarrow 1$ by: $\rho = \{t \in P_{(1 \Rightarrow A) \rightarrow 1}^{\text{prop}} \mid \gamma(\$ (t)) \sqsubseteq_{\text{even}} s\}$. By construction, ρ is $\$$ -propagating and $\$$ -finitary. Hence, there is a term $f : \text{unit} \rightarrow \theta \vdash N : \text{unit}$ such that $\llbracket N \rrbracket = \rho$. Moreover, $s \in \llbracket N \rrbracket$ and thus, by Lemma 22, $(\lambda f. N)M'_1 \Downarrow$. We claim that $\star \star \notin \llbracket M'_2 \rrbracket; \rho$ and therefore $(\lambda f. N)M'_2 \not\Downarrow$. For suppose $\star \star \in \llbracket M'_2 \rrbracket; \rho$, because of composing $\llbracket M'_2 \rrbracket$ with some play $t = \star s' \star^{\Sigma} \in \rho$. Then, $s' \in \text{comp}(\llbracket M_2 \rrbracket)$ and $\gamma(\$ (\star s' \star^{\Sigma})) \sqsubseteq s$. Since \star^{Σ} is not deleted by $\$$, we have in fact $\gamma(\$ (\star s' \star^{\Sigma})) = s$, hence $s \in \text{comp}_{\$}(\llbracket M_2 \rrbracket)$, contradicting the hypothesis. \square

Example 28. Let us revisit the terms from Examples 2 and 19. We have $\text{comp}_{\$}(\llbracket M_i \rrbracket) = \{\star^{\Sigma_0} q_0^{\Sigma_0} \star^{\Sigma_0} \star^{\Sigma_0}\}$ for $i = 1, 2$, because other plays from $\llbracket M_i \rrbracket$ would give rise to non-propagating interactions \hat{s} . Thus, in $\text{ExnML}_{\$}$ we do have $\vdash M_1 \cong M_2$.

References

1. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.-H.L., Stark, I.D.B.: Nominal games and full abstraction for the nu-calculus. In: LICS 2004 (2004)
2. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2002)
3. Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. Theoretical Computer Science 221(1-2), 393–456 (1999)
4. Laird, J.: A semantic analysis of control. PhD thesis, University of Edinburgh (1998)

5. Laird, J.: A fully abstract games semantics of local exceptions. In: LICS 2001 (2001)
6. Laird, J.: A game semantics of local names and good variables. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 289–303. Springer, Heidelberg (2004)
7. Laird, J.: A game semantics of names and pointers. *Ann. Pure Appl. Logic* 151, 151–169 (2008)
8. Laird, J.: Combining and relating control effects and their semantics. In: COS (2013)
9. Moggi, E.: Notions of computation and monads. *Inf. and Comput.* 93, 55–92 (1991)
10. Murawski, A.S., Tzevelekos, N.: Game semantics for good general references. In: LICS 2011 (2011)
11. Murawski, A.S., Tzevelekos, N.: Algorithmic games for full ground references. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 312–324. Springer, Heidelberg (2012)
12. Murawski, A.S., Tzevelekos, N.: Full abstraction for Reduced ML. *Ann. Pure Appl. Logic* 164(11), 1118–1143 (2013)
13. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) *Algorithmic Languages*, pp. 345–372. North Holland (1981)
14. Stark, I.D.B.: Names and Higher-Order Functions. PhD thesis, University of Cambridge, Technical Report No. 363 (1995)
15. Tzevelekos, N.: Nominal game semantics. D.Phil. thesis, Oxford University (2008)
16. Tzevelekos, N.: Full abstraction for nominal general references. *LMCS* 5(3) (2009)

Complexity of Model-Checking Call-by-Value Programs

Takeshi Tsukada^{1,2} and Naoki Kobayashi³

¹ University of Oxford

² JSPS Postdoctoral Fellow for Research Abroad

³ The University of Tokyo

Abstract. This paper studies the complexity of the reachability problem (a typical and practically important instance of the model-checking problem) for simply-typed call-by-value programs with recursion, Boolean values, and non-deterministic branch, and proves the following results. (1) The reachability problem for order-3 programs is nonelementary. Thus, unlike in the call-by-name case, the order of the input program does not serve as a good measure of the complexity. (2) Instead, the *depth* of types is an appropriate measure: the reachability problem for depth- n programs is n -EXPTIME complete. In particular, the previous upper bound given by the CPS translation is not tight. The algorithm used to prove the upper bound result is based on a novel intersection type system, which we believe is of independent interest.

1 Introduction

A promising approach to verifying higher-order functional programs is to use higher-order model checking [7,8,15], which is a decision problem about the trees generated by higher-order recursion schemes. Various verification problems such as the reachability problem and the resource usage verification [5] are reducible to the higher-order model checking [8].

This paper addresses a variant of the higher-order model checking, namely, the reachability problem for simply-typed *call-by-value* Boolean programs. It is the problem to decide, given a program with Boolean primitives and a special constant meaning the failure, whether the evaluation of the program fails. This is a practically important problem that can be a basis for verification of programs written in call-by-value languages such as ML and OCaml. In fact, MoCHi [11], a software model-checker for a subset of OCaml, reduces a verification problem to a reachability problem for a call-by-value Boolean program.

In the previous approach [11], the reachability problem for call-by-value programs was reduced to that for call-by-name programs via the CPS transformation. From a complexity theoretic point of view, however, this reduction via the CPS transformation has a bad effect: the order of a function is raised by 2 for each increase of the arity of the function. Since the reachability of order- n call-by-name programs is $(n - 1)$ -EXPTIME complete in general, the approach may suffer from double exponential blow-up of the time complexity for each increase

of the largest arity in a program. Thus, important questions are: Is the double exponential blow-up of the time complexity (with respect to the arity increase) inevitable? If not, what is the exact complexity of the reachability problem for call-by-value programs, and how can we achieve the exact complexity?

The above questions are answered in this paper. We first show that the *single* exponential blow-up with respect to the arity increase is inevitable for programs of order-3 or higher. This implies that when the arity is not fixed, the reachability problem for order-3 call-by-value programs is nonelementary. The key observation used in the proof is that the subset of natural numbers $\{0, 1, \dots, \mathbf{exp}_n(2) - 1\}$ (here $\mathbf{exp}_n(k)$ is the n th iterated exponential function, defined by $\mathbf{exp}_0(k) = k$ and $\mathbf{exp}_{n+1}(k) = 2^{\mathbf{exp}_n(k)}$) can be embedded into the

set of values of the type $\overbrace{\mathbb{B} \rightarrow \mathbb{B} \rightarrow \dots \rightarrow \mathbb{B}}^n \rightarrow \mathbb{B}$ by using non-determinism.

Second, we show the *depth* of types is an appropriate measure, i.e. the reachability problem for depth- n programs is n -EXPTIME complete. The depth of function type is defined by $\mathit{depth}(\kappa \rightarrow \kappa') = \max\{\mathit{depth}(\kappa) + 1, \mathit{depth}(\kappa') + 1\}$. In particular, the previous bound given by the CPS translation is not tight. To prove the upper-bound, we develop a novel intersection type system that completely characterises programs that reach the failure. Since the target is a call-by-value language with effects (i.e. divergence, non-determinism and failure), the proposed type system is much different from that for call-by-name calculi [18,7,9], which we believe is of the independent interest.

Organisation of the paper Section 2 defines the problem addressed in the paper. Section 3 proves that the reachability problem for order-3 programs is nonelementary. Section 4 provides a sketch of the proof of n -EXPTIME hardness of the reachability problem for depth- n programs. In Section 5, we develop an intersection type system that characterises the reachability problem, and a type-checking algorithm. We discuss related work in Section 6 and conclude in Section 7. For the space limitation, we omit some details and proofs, which are found in a long version available from the first author's web page.

2 Call-by-Value Reachability Problem

The target language of the paper is a simply-typed call-by-value calculus with recursion, product types (restricted to argument positions), Boolean and non-deterministic branch. Simple types are called *sorts* in order to avoid confusion with intersection types introduced later. The sets of *sorts*, *terms* and *function definitions* (*definitions* for short) are defined by the following grammar:

$$\begin{array}{ll}
 (\text{Sorts}) & \kappa, \iota \quad ::= \mathbb{B} \mid \kappa_1 \times \dots \times \kappa_n \rightarrow \iota \\
 (\text{Terms}) & s, t, u \quad ::= x \mid f \mid \lambda \langle x_1, \dots, x_n \rangle . t \mid t \langle u_1, \dots, u_n \rangle \\
 & \quad \mid t \oplus u \mid \mathbf{t} \mid \mathbf{f} \mid \mathbf{if}(t, u_1, u_2) \mid \mathfrak{F}_\kappa \mid \Omega_\kappa \\
 (\text{Definitions}) & D \quad ::= \{f_i = \lambda \langle x_{i,1}, \dots, x_{i,n_i} \rangle . t_i\}_{i \leq m},
 \end{array}$$

where $\langle x_1, \dots, x_n \rangle$ (resp. $\langle u_1, \dots, u_n \rangle$) is a non-empty sequence of variables (resp. terms). The sort \mathbb{B} is for Boolean values and the sort $\kappa_1 \times \dots \times \kappa_n \rightarrow \iota$ is for

$$\begin{array}{c}
\frac{b \in \{\mathbf{t}, \mathbf{f}\}}{\Delta \mid \mathcal{K} \vdash b :: \mathbb{B}} \quad \frac{x :: \kappa \in \mathcal{K}}{\Delta \mid \mathcal{K} \vdash x :: \kappa} \quad \frac{f :: \kappa \in \Delta}{\Delta \mid \mathcal{K} \vdash f :: \kappa} \quad \frac{\Delta \mid \mathcal{K}, \vec{x} :: \vec{\kappa} \vdash t :: \iota}{\Delta \mid \mathcal{K} \vdash \lambda(\vec{x}).t :: \vec{\kappa} \rightarrow \iota} \\
\frac{\Delta \mid \mathcal{K} \vdash t :: \vec{\kappa} \rightarrow \iota' \quad \Delta \mid \mathcal{K} \vdash \vec{u} :: \vec{\kappa}}{\Delta \mid \mathcal{K} \vdash t(\vec{u}) :: \iota} \quad \frac{\Delta \mid \mathcal{K} \vdash t :: \mathbb{B} \quad \Delta \mid \mathcal{K} \vdash u_i :: \kappa \ (i \in \{1, 2\})}{\Delta \mid \mathcal{K} \vdash \mathbf{if}(t, u_1, u_2) :: \kappa} \\
\frac{\Delta \mid \mathcal{K} \vdash t :: \kappa \quad \Delta \mid \mathcal{K} \vdash u :: \kappa}{\Delta \mid \mathcal{K} \vdash t \oplus u :: \kappa} \quad \frac{}{\Delta \mid \mathcal{K} \vdash \mathfrak{F}_\kappa :: \kappa} \quad \frac{}{\Delta \mid \mathcal{K} \vdash \Omega_\kappa :: \kappa}
\end{array}$$

Fig. 1. Sorting rules for terms

functions that take an n -tuple as the argument and returns a value of ι . A term is a variable x , a function symbol f (that is a variable expected to be defined in D), an abstraction $\lambda\langle x_1, \dots, x_n \rangle.t$ that takes an n -tuple as its argument, an application $t \langle u_1, \dots, u_n \rangle$ of t to n -tuple $\langle u_1, \dots, u_n \rangle$, a non-deterministic branch $t_1 \oplus t_2$, a truth value (\mathbf{t} or \mathbf{f}), a conditional branch $\mathbf{if}(t, u_1, u_2)$, a special constant \mathfrak{F}_κ (standing for ‘Fail’) to which the reachability is considered, or divergence Ω_κ . A function definition is a finite set of elements of the form $f = \lambda\langle x_1, \dots, x_n \rangle.t$, which defines functions by mutual recursion. If $(f = \lambda\langle \vec{x} \rangle.t) \in D$, we write $D(f) = \lambda\langle \vec{x} \rangle.t$. The domain $\text{dom}(D)$ of D is $\{f \mid (f = \lambda\langle \vec{x} \rangle.t) \in D\}$.

For notational convenience, we use the following abbreviations. We write \vec{x} for a *non-empty* sequence of variables x_1, \dots, x_n , and simply write $\lambda\langle x_1, \dots, x_n \rangle.t$ as $\lambda\langle \vec{x} \rangle.t$. Similarly, $t \langle u_1, \dots, u_n \rangle$ is written as $t \langle \vec{u} \rangle$, where \vec{u} indicates the sequence u_1, \dots, u_n , and $\kappa_1 \times \dots \times \kappa_n \rightarrow \iota$ as $\vec{\kappa} \rightarrow \iota$, where $\vec{\kappa} = \kappa_1, \dots, \kappa_n$. Note that $\vec{\kappa} \rightarrow \iota$ is *not* $\kappa_1 \rightarrow \dots \rightarrow \kappa_n \rightarrow \iota$. Sort annotation of \mathfrak{F}_κ and Ω_κ are often omitted. For a 1-tuple $\langle t \rangle$, we often write just t .

The sort system is defined straightforwardly. A *sort environment* is a finite set of sort bindings of the form $x :: \kappa$ (here a double-colon is used for sort bindings and judgements, in order to distinguish them from type bindings and judgements). We write $\mathcal{K}(x) = \kappa$ if $x :: \kappa \in \mathcal{K}$. A *sort judgement* is of the form $\Delta \mid \mathcal{K} \vdash t :: \kappa$, where Δ is the sort environment for function symbols and \mathcal{K} is the sort environment for free variables of t . Given sequences \vec{x} and $\vec{\kappa}$ of the same length, we write $\vec{x} :: \vec{\kappa}$ for $x_1 :: \kappa_1, \dots, x_n :: \kappa_n$. Given sequences \vec{t} and $\vec{\kappa}$ of the same length, we write $\Delta \mid \mathcal{K} \vdash \vec{t} :: \vec{\kappa}$ just if we have $\Delta \mid \mathcal{K} \vdash t_i :: \kappa_i$ for all $i \leq n$, where n is the length of \vec{t} . The sorting rules are listed in Fig. 1.

When term t does not contain function symbols, we simply write $\emptyset \mid \mathcal{K} \vdash t :: \kappa$ as $\mathcal{K} \vdash t :: \kappa$. We assume that terms in the sequel are explicitly typed, i.e. every term is equipped with a sort derivation for it and we can freely refer to sorts of subterms and variables in the term. For function definitions, a judgement is of the form $\vdash D :: \Delta$, which is derived by the following rule:

$$\frac{\Delta \mid \emptyset \vdash D(f) :: \kappa \quad (\text{for every } f :: \kappa \in \Delta)}{\vdash D :: \Delta}$$

A *program* is a pair of a definition D and a term t of the ground sort \mathbb{B} with $\vdash D :: \Delta$ and $\Delta \mid \emptyset \vdash t :: \mathbb{B}$ for some Δ . A program is written as **let rec** D **in** t . A program **let rec** \emptyset **in** t with no function symbols is simply written as t .

The set of *values* is defined by: $v, w ::= \lambda(\vec{x}).t \mid \mathbf{t} \mid \mathbf{f}$. Recall that \vec{x} is a non-empty sequence. *Evaluation contexts* are defined by: $E ::= \square \mid E \langle \vec{t} \rangle \mid v \langle w_1, \dots, w_{k-1}, E, t_{k+1}, \dots, t_n \rangle \mid \mathbf{if}(E, t_1, t_2)$. Therefore arguments are evaluated left-to-right. The reduction relation on terms is defined by the rules below:

$$\begin{array}{ll} E[(\lambda(\vec{x}).t) \langle \vec{v} \rangle] \longrightarrow E[(\vec{v}/\vec{x})t] & E[t_1 \oplus t_2] \longrightarrow E[t_i] \quad (\text{for } i = 1, 2) \\ E[\mathbf{if}(\mathbf{t}, t_1, t_2)] \longrightarrow E[t_1] & E[\mathbf{if}(\mathbf{f}, t_1, t_2)] \longrightarrow E[t_2]. \end{array}$$

We write \longrightarrow^* for the reflexive and transitive closure of \longrightarrow . The reduction relation is not deterministic because of the non-deterministic branch. A closed well-typed term t cannot be reduced just if (1) t is a value, (2) $t = E[\mathfrak{F}]$ or (3) $t = E[\Omega]$. In the second case, t immediately fails and in the third case, t never fails since Ω diverges. So we do not need to consider further reduction steps for $E[\mathfrak{F}]$ and $E[\Omega]$. By this design choice, \longrightarrow is terminating.

Lemma 1. *If $\emptyset \vdash t :: \kappa$, then t has no infinite reduction sequence.*

Given a function definition D , the reduction relation \longrightarrow_D is defined by the same rules as \longrightarrow and the following additional rule:

$$E[f] \longrightarrow_D E[D(f)].$$

We write \longrightarrow_D^* for the reflexive and transitive closure of \longrightarrow_D . Note that reduction by \longrightarrow_D does not terminate in general.

Definition 1 (Reachability Problem). We say a program $\mathbf{let\ rec\ } D \mathbf{\ in\ } t$ fails if $t \longrightarrow_D^* E[\mathfrak{F}]$ for some E . The *reachability problem* is the problem to decide whether a given program fails.

Example 1. Let $t_0 = \lambda f.\mathbf{if}(f \mathbf{t}, \mathbf{if}(f \mathbf{t}, \Omega, \mathfrak{F}), \Omega)$, which calls the argument f (at most) twice with the same argument \mathbf{t} and fails just if the first call returns \mathbf{t} and the second call \mathbf{f} . Let $u_0 = (\lambda x.\mathbf{t}) \oplus (\lambda x.\mathbf{f})$ and $e_1 = t_0 u_0$. Then e_1 has just two reduction sequences starting from $e_1 \longrightarrow t_0 (\lambda x.\mathbf{t})$ and $e_1 \longrightarrow t_0 (\lambda x.\mathbf{f})$, both of which do not fail. In the call-by-name setting, however, e_1 would fail since

$$\begin{aligned} e_1 &\longrightarrow \mathbf{if}(u_0 \mathbf{t}, \mathbf{if}(u_0 \mathbf{t}, \Omega, \mathfrak{F}), \Omega) \longrightarrow \mathbf{if}((\lambda x.\mathbf{t}) \mathbf{t}, \mathbf{if}(u_0 \mathbf{t}, \Omega, \mathfrak{F}), \Omega) \\ &\longrightarrow^* \mathbf{if}(u_0 \mathbf{t}, \Omega, \mathfrak{F}) \longrightarrow \mathbf{if}((\lambda x.\mathbf{f}) \mathbf{t}, \Omega, \mathfrak{F}) \longrightarrow^* \mathfrak{F}. \end{aligned}$$

Consider the program $e'_1 = t_0 u'_0$ where $u'_0 = \lambda x.(\mathbf{t} \oplus \mathbf{f})$, in which the non-deterministic branch is delayed by the abstraction. Then e'_1 would fail both in call-by-name and in call-by-value.

Example 2. Consider the program $P_2 = \mathbf{let\ rec\ } D_2 \mathbf{\ in\ } e_2$, where $D_2 = \{f = \lambda x.f x\}$ and $e_2 = (\lambda y.\mathfrak{F})(f \mathbf{t})$. Then P_2 never fails because

$$e_2 = (\lambda y.\mathfrak{F})(f \mathbf{t}) \longrightarrow_{D_2} (\lambda y.\mathfrak{F})((\lambda x.f x) \mathbf{t}) \longrightarrow_{D_2} (\lambda y.\mathfrak{F})(f \mathbf{t}) = e_2 \longrightarrow_{D_2} \dots$$

In the call-by-name case, however, P_2 would fail since $(\lambda x.\mathfrak{F})(f \mathbf{t}) \longrightarrow \mathfrak{F}$.

Example 3. Consider the program $e_3 = (\lambda x.t)\mathfrak{F}$. Then e_3 (immediately) fails because $e_3 = E[\mathfrak{F}]$, where $E = (\lambda x.t)\square$. In contrast, e_3 would not fail in the call-by-name setting, in which E is not an evaluation context and $e_3 \rightarrow t$.

We give a technically convenient characterisation of the reachability problem. Let $\{f_1, \dots, f_n\}$ be the set of function symbols in D . The m th approximation of f_i , written F_i^m , is the term obtained by expanding the definition m times, as is formally defined below:

$$\begin{aligned} F_i^0 &= \lambda \langle x_1, \dots, x_k \rangle. \Omega_\iota && \text{(where } f_i :: \kappa_1 \times \dots \times \kappa_k \rightarrow \iota \in \Delta) \\ F_i^{m+1} &= [F_1^m / f_1, \dots, F_n^m / f_n](D(f_i)). \end{aligned}$$

The m th approximation of t is defined by: $[t]_D^m = [F_1^m / f_1, \dots, F_n^m / f_n]t$.

Lemma 2. *Let $P = \text{let rec } D \text{ in } t$ be a program. Then $t \rightarrow_D^* E[\mathfrak{F}]$ for some E if and only if $[t]_D^n \rightarrow^* E'[\mathfrak{F}]$ for some n and E' .*

Size of terms and programs The size of sorts is inductively defined by $|\mathbb{B}| = 1$ and $|\kappa_1 \times \dots \times \kappa_n \rightarrow \iota| = 1 + |\iota| + \sum_{i=1}^n |\kappa_i|$. The size of sort environments is given by $|\mathcal{K}| = \sum_{x::\kappa \in \mathcal{K}} |\kappa|$. The size of a term is defined straightforwardly (e.g. $|x| = 1$ and $|t \langle u_1, \dots, u_n \rangle| = 1 + |t| + \sum_{i=1}^n |u_i|$) except for the abstraction $|\lambda \langle x_1, \dots, x_n \rangle. t| = 1 + |t| + \sum_{i=1}^n (1 + |\kappa_i|)$, where κ_i is the sort of x_i . Here a term t is considered to be explicitly sorted, and thus the size of annotated sorts should be added. For programs, $|\text{let rec } D \text{ in } t| = |t| + \sum_{f \in \text{dom}(D)} |D(f)|$.

Order and depth of programs. Order is a well-known measure that characterises complexity of the *call-by-name* reachability problem [10,15] (it is $(n-1)$ -EXPTIME complete for order- n programs) and, as we shall see, depth characterises complexity in the call-by-value case. *Order* and *depth* of sorts are defined by:

$$\begin{aligned} \text{order}(\mathbb{B}) &= \text{depth}(\mathbb{B}) = 0 \\ \text{order}(\vec{\kappa} \rightarrow \iota) &= \max\{\text{order}(\iota), \text{order}(\kappa_1)+1, \dots, \text{order}(\kappa_n)+1\} \\ \text{depth}(\vec{\kappa} \rightarrow \iota) &= \max\{\text{depth}(\iota)+1, \text{depth}(\kappa_1)+1, \dots, \text{depth}(\kappa_n)+1\} \end{aligned}$$

For a sort environment, $\text{depth}(\mathcal{K}) = \max\{\text{depth}(\kappa) \mid x :: \kappa \in \mathcal{K}\}$. Order and depth of judgements are defined by $\varphi(\Delta \mid \mathcal{K} \vdash t :: \kappa) = \varphi(\kappa)$, where $\varphi \in \{\text{order}, \text{depth}\}$. The order of a sort derivation is the maximal order of judgements in the derivation. The order of a sorted term t is the order of its sort derivation $\Delta \mid \mathcal{K} \vdash t :: \kappa$. The order of a program $\text{let rec } D \text{ in } t$ is the maximal order of terms t and $D(f)$ ($f \in \text{dom}(D)$). The depth of derivations, sorted terms and programs are defined similarly.

3 Order-3 Reachability is Nonelementary

This section proves the following theorem.

Theorem 1. *The reachability problem for order-3 programs is nonelementary.*

The key observation is that, for every n , the subset of natural numbers $\{0, 1, \dots, \mathbf{exp}_n(2) - 1\}$ can be implemented by $\overbrace{\mathbb{B} \rightarrow \dots \rightarrow \mathbb{B}}^n \rightarrow \mathbb{B}$ in a certain sense (see Definition 2). The non-determinism of the calculus is essential to the construction. Note that in the call-by-name case, the set of closed terms (modulo observational equivalence) of this sort can be bounded by 4^{4^n} , since

$$\overbrace{\mathbb{B} \rightarrow \dots \rightarrow \mathbb{B}}^n \rightarrow \mathbb{B} \cong \overbrace{\mathbb{B} \times \dots \times \mathbb{B}}^n \rightarrow \mathbb{B}.$$

The proof in this section can be sketched as follows. Let $L \subseteq \{0, 1\}^*$ be a language in n -EXPSpace. We can assume without loss of generality that there exists a Turing machine M that accepts L and runs in space $\mathbf{exp}_n(x)$ (here x is the size of the input). Given a word w , we reduce its acceptance by M to the reachability problem of a program (say $P_{M,w}$) of the call-by-value calculus in Section 2 extended to have natural numbers up to $N \geq \mathbf{exp}_n(x)$ (Lemma 3). The order of $P_{M,w}$ is independent from M and w : it is 3 when the order of the natural number type is defined to be 1. Recall that the natural numbers up to $\mathbf{exp}_{n+x}(2) \geq \mathbf{exp}_n(x)$ can be implemented by the order-1 sort $\overbrace{\mathbb{B} \rightarrow \dots \rightarrow \mathbb{B}}^{n+x} \rightarrow \mathbb{B}$. By replacing natural numbers in $P_{M,w}$ with the implementation, the acceptance of w by M can be reduced to the reachability problem of an order-3 program without natural numbers.

3.1 Simulating Turing Machine by Program with Natural Numbers

First of all, we define programs with natural numbers up to N , which is an extension of the typed calculus presented in Section 2. The syntax of sorts and terms is given by:

$$\begin{aligned} (\text{Sorts}) \quad \kappa, \iota &::= \dots \mid \mathbb{N} \\ (\text{Terms}) \quad s, t, u &::= \dots \mid \mathbf{S} \mid \mathbf{P} \mid \mathbf{EQ} \mid \underline{0} \mid \underline{1} \mid \dots \mid \underline{N-1} \end{aligned}$$

The extended calculus has an additional ground sort \mathbb{N} for (bounded) natural numbers. Constants \mathbf{S} and \mathbf{P} are functions of sort $\mathbb{N} \rightarrow \mathbb{N}$ meaning the successor and the predecessor functions, respectively, and \mathbf{EQ} is a constant of sort $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$ which checks if two arguments are equivalent. A constant \underline{n} indicates the natural number n . The set of values is defined by: $v ::= \dots \mid \mathbf{S} \mid \mathbf{P} \mid \mathbf{EQ} \mid \underline{n}$. Function definitions and evaluation contexts are given by the same syntax as in Section 2, but terms and values may contain natural numbers. The additional reduction rules are given by

$$\begin{aligned} E[\mathbf{S} \underline{n}] &\longrightarrow_D E[\underline{n+1}] && (\text{if } n+1 < N) \\ E[\mathbf{P} \underline{n}] &\longrightarrow_D E[\underline{n-1}] && (\text{if } n-1 \geq 0) \\ E[\mathbf{EQ} \langle \underline{n}, \underline{n} \rangle] &\longrightarrow_D E[\mathbf{t}] \\ E[\mathbf{EQ} \langle \underline{n}, \underline{m} \rangle] &\longrightarrow_D E[\mathbf{f}] && (\text{if } n \neq m). \end{aligned}$$

Note that $E[\underline{S} \ N - 1]$ and $E[\underline{P} \ 0]$ get stuck. A *program with natural numbers up to N* is a pair of a function definition D and a term t of sort \mathbb{B} , written as **let rec D in t** . We assume that programs in the sequel do not contain constant numbers except for $\underline{0}$. The order of \mathbb{N} is defined as 1.

Lemma 3. *Let $L \subseteq \{0, 1\}^*$ be a language and M be a deterministic Turing machine accepting L that runs in space $\mathbf{exp}_n(x)$ for some n . Then, for every word $w \in \{0, 1\}^*$ of length k and natural number $N \geq \mathbf{exp}_n(k)$, one can construct a program $P_{M,w}$ with natural numbers up to N such that $P_{M,w}$ fails if and only if $w \in L$. Furthermore $P_{M,w}$ is of order-3 and can be constructed in polynomial time with respect to k .*

Proof. Let M be a Turing machine with states Q and tape symbols Σ and w be a word of length k . We can assume without loss of generality that $Q = \{\mathbf{t}, \mathbf{f}\}^q$ (that is, the set of all sequences of length q consisting of \mathbf{t} and \mathbf{f}) and $\Sigma = \{\mathbf{t}, \mathbf{f}\}^l$.

A configuration is expressed as a value of sort¹

$$Config = \overbrace{\mathbb{B} \times \cdots \times \mathbb{B}}^q \times \overbrace{(\mathbb{N} \rightarrow \mathbb{B}) \times \cdots \times (\mathbb{N} \rightarrow \mathbb{B})}^l \times \mathbb{N},$$

where the first part represents the current state, the second part the tape and the third part the position of the tape head. The program $P_{M,w}$ has one recursive function *isAccepted* of sort $Config \rightarrow \mathbb{B}$. It checks if the current state is a final state and it fails if so. Otherwise it computes the next configuration and passes it to *isAccepted* itself. The body of the program generates the initial configuration determined by w and passes it to the function *isAccepted*.

Clearly we can construct $P_{M,w}$ in polynomial time with respect to k (the length of w) and the order of $P_{M,w}$ is 3. □

3.2 Implementing Natural Numbers

Let ν_n be the order-1 sort defined by $\nu_0 = \mathbb{B}$ and $\nu_{n+1} = \mathbb{B} \rightarrow \nu_n$. We shall show that natural numbers up to $\mathbf{exp}_n(2)$ can be implemented as values of ν_n .

Intuitive Explanation. We explain the intuition behind the construction by using the set-theoretic model. Let $\mathbf{N} = \{0, 1, \dots, N - 1\}$. We explain the way to express the set $2^{\mathbf{N}} \cong \{0, 1, \dots, 2^N - 1\}$ as (a subset of) non-deterministic functions of $\mathbb{B} \rightarrow \mathbf{N}$, i.e. functions of $\mathbb{B} \rightarrow \mathcal{P}(\mathbf{N})$, where $\mathcal{P}(\mathbf{N})$ is the powerset of \mathbf{N} . The set $(\mathbb{B} \rightrightarrows \mathbf{N}) \subseteq (\mathbb{B} \rightarrow \mathcal{P}(\mathbf{N}))$ is defined by:

$$(\mathbb{B} \rightrightarrows \mathbf{N}) = \{f : \mathbb{B} \rightarrow \mathcal{P}(\mathbf{N}) \mid f(\mathbf{t}) \cup f(\mathbf{f}) = \mathbf{N} \text{ and } f(\mathbf{t}) \cap f(\mathbf{f}) = \emptyset\}.$$

In other words, $f \in \mathbb{B} \rightarrow \mathcal{P}(\mathbf{N})$ is in $\mathbb{B} \rightrightarrows \mathbf{N}$ if and only if, for every $i \in \mathbf{N}$, exactly one of $i \in f(\mathbf{t})$ and $i \in f(\mathbf{f})$ holds. Hence a function $f : \mathbb{B} \rightrightarrows \mathbf{N}$ determines a function of $\mathbf{N} \rightarrow \mathbb{B}$, say \hat{f} , defined by $\hat{f}(i) = b$ iff $i \in f(b)$ ($b \in \{\mathbf{t}, \mathbf{f}\}$).

¹ Strictly speaking, it is not a sort in our syntax because products are restricted to argument positions. But there is no problem since occurrences of *Config* in the following construction are also restricted to argument positions.

There is a bijection between the set of functions $\mathbf{N} \rightarrow \mathbb{B}$ and the subset of natural numbers $\{0, 1, \dots, 2^N - 1\}$, given by binary encoding, i.e. $(\hat{f} : \mathbf{N} \rightarrow \mathbb{B}) \mapsto \sum_{i < N, \hat{f}(i) = \mathbf{t}} 2^i$. For example, consider the case that $N = 4$ and $\mathbf{N} = \{0, 1, 2, 3\}$. Then 6 ($= 0110$ in binary) is represented by \hat{f}_6 such that $\hat{f}_6(0) = \hat{f}_6(3) = \mathbf{f}$ and $\hat{f}_6(1) = \hat{f}_6(2) = \mathbf{t}$. Therefore f_6 is given by $f_6(\mathbf{t}) = \{1, 2\}$ and $f_6(\mathbf{f}) = \{0, 3\}$.

Now let us consider the way to define operations such as the successor, predecessor and equality test. The key fact is that there is a term (say **get**) that computes $\hat{f}(i)$ for $f \in \mathbb{B} \Rightarrow \mathbf{N}$ and $i \in \mathbf{N}$, and there exists a term (say **put**) that computes $g \in \mathbb{B} \Rightarrow \mathbf{N}$ such that $\hat{g} = \hat{f}[i \mapsto b]$ for $f \in \mathbb{B} \Rightarrow \mathbf{N}$, $i \in \mathbf{N}$ and $b \in \{\mathbf{t}, \mathbf{f}\}$. They are given by the following informal equations:

$$\begin{aligned} \text{get } \langle f, i \rangle &= \mathbf{if}(f \mathbf{t} = i, \mathbf{t}, \Omega) \oplus \mathbf{if}(f \mathbf{f} = i, \mathbf{f}, \Omega) \\ \text{put } \langle f, i, b \rangle &= \lambda c^{\mathbb{B}}. (\mathbf{if}(b = c, i, \Omega) \oplus ((\lambda j. \mathbf{if}(i \neq j, j, \Omega))(f c))) \end{aligned}$$

where $f :: \mathbb{B} \rightarrow \mathbf{N}$ and $i, j :: \mathbf{N}$ and $b, c :: \mathbb{B}$. Note that **put** would be incorrect in the call-by-name setting. By using these functions, we can write operations like successor, predecessor and equality test for $\mathbb{B} \Rightarrow \mathbf{N}$. For example, the equality test eq can be defined by $eq = \lambda \langle f, g \rangle. e \langle f, g, N - 1 \rangle$, where e is given by the following recursive definition:

$$e \langle f, g, i \rangle = \mathbf{if}((\text{get } \langle f, i \rangle) = (\text{get } \langle g, i \rangle), \quad \mathbf{if}(i = 0, \mathbf{t}, e \langle f, g, (i - 1) \rangle), \quad \mathbf{f}).$$

Formal Development. We formally define the notion of implementations and show that replacement of natural numbers with its implementations preserves reachability.

Definition 2 (Implementation of Natural Numbers). Let \mathbf{N} be the tuple $(N, D, \kappa, \{V_i\}_{i \in \{0, 1, \dots, N-1\}}, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$, where N is a natural number, D is a function definition, κ is a sort, $\{V_i\}_i$ is an indexed set of pairwise disjoint sets of closed values of sort κ , \mathbf{eq} is a closed value of sort $\kappa \times \kappa \rightarrow \mathbb{B}$, \mathbf{s} and \mathbf{p} are closed values of sort $\kappa \rightarrow \kappa$, and \mathbf{z} and \mathbf{max} are closed values of sort κ . Here we consider terms *without* natural numbers. We say \mathbf{N} is an *implementation of natural numbers up to N* just if the following conditions hold (here $V = \bigcup_i V_i$).

- For every $v, v' \in V$, evaluation of $\mathbf{eq} \langle v, v' \rangle$, $\mathbf{s} v$ and $\mathbf{p} v$ under D never fails.
- $\mathbf{z} \in V_0$ and $\mathbf{max} \in V_{N-1}$.
- For every $v \in V_n$ and $v' \in V_{n'}$, $\mathbf{eq} \langle v, v' \rangle \rightarrow_D^* \mathbf{t}$ if and only if $n = n'$, and $\mathbf{eq} \langle v, v' \rangle \rightarrow_D^* \mathbf{f}$ if and only if $n \neq n'$.
- For every $v \in V_n$, $\mathbf{s} v \rightarrow_D^* v'$ implies $v' \in V_{n+1}$ and if $n + 1 < N$ then $\mathbf{s} v \rightarrow_D^* v'$ for some $v' \in V_{n+1}$. Similarly, $\mathbf{p} v \rightarrow_D^* v'$ implies $v' \in V_{n-1}$ and if $n \geq 1$ then $\mathbf{p} v \rightarrow_D^* v'$ for some $v' \in V_{n-1}$.

The sort of \mathbf{N} is κ and the order of \mathbf{N} is that of κ .

Given an implementation \mathbf{N} of natural numbers up to N and a term t with natural numbers up to N , we write $t^{\mathbf{N}}$ for the term without natural numbers obtained by replacing constants with values given by \mathbf{N} , e.g.,

$$\underline{0}^{\mathbf{N}} = \mathbf{z} \quad \mathbf{S}^{\mathbf{N}} = \mathbf{s} \quad (t u)^{\mathbf{N}} = t^{\mathbf{N}} u^{\mathbf{N}} \quad (\lambda x. t)^{\mathbf{N}} = \lambda x. (t^{\mathbf{N}}).$$

Note that programs do not contain constant numbers except for $\underline{0}$ by definition. Given a function definition D , $D^{\mathbf{N}}$ can be defined straightforwardly. See the long version for the concrete definition.

Lemma 4. *Let $\text{let rec } D \text{ in } t$ be a program with natural numbers up to N , and \mathbf{N} be an implementation of natural numbers up to N . Then $\text{let rec } D \text{ in } t$ fails if and only if $\text{let rec } D^{\mathbf{N}} \text{ in } t^{\mathbf{N}}$ fails.*

Given a natural number $n \geq 1$, we present an implementation of natural numbers up to $\text{exp}_n(2)$ whose order is 1. By using the implementation to the program constructed in Lemma 3, the nonelementary result for the reachability problem for order-3 programs is established.

For every n , we shall define an implementation $\mathbf{N}(n)$ of natural numbers up to $\text{exp}_n(2)$ by induction on n . As for the base case, the natural numbers up to $\text{exp}_0(2) = 2$ (i.e. $\{0, 1\}$) can be naturally implemented by using \mathbb{B} . We call this implementation $\mathbf{N}(0)$. As for the induction step, assuming an implementation $\mathbf{N} = (N, D, \kappa, \{V_i\}_i, \mathbf{eq}, \mathbf{s}, \mathbf{p}, \mathbf{z}, \mathbf{max})$ of natural numbers up to N , it suffices to construct an implementation of natural numbers up to 2^N , say ${}^{\mathbb{B}}\mathbf{N} = (2^N, D \cup D', \mathbb{B} \rightarrow \kappa, \{V'_i\}_{i \in \{0, 1, \dots, 2^N - 1\}}, \mathbf{eq}', \mathbf{s}', \mathbf{p}', \mathbf{z}', \mathbf{max}')$.

- The additional function definition D' defines get , put and other auxiliary functions used to define \mathbf{s}' and others. The definitions of get and put are:

$$\begin{aligned} \text{get} &= \lambda \langle x^{\mathbb{B} \rightarrow \kappa}, i^{\kappa} \rangle. \mathbf{if}(\mathbf{eq} \langle x \mathbf{t}, i \rangle, \mathbf{t}, \Omega) \oplus \mathbf{if}(\mathbf{eq} \langle x \mathbf{f}, i \rangle, \mathbf{f}, \Omega) \\ \text{put} &= \lambda \langle x^{\mathbb{B} \rightarrow \kappa}, i^{\kappa}, b^{\mathbb{B}} \rangle. \lambda c^{\mathbb{B}}. (\mathbf{if}(b = c, i, \Omega) \oplus ((\lambda j. \mathbf{if}(\mathbf{eq} \langle i, j \rangle, \Omega, j)) (x c))) \end{aligned}$$

- Let $m < 2^N$ and $b_{N-1} \dots b_0$ be its binary representation. Then V'_m is the set of values v of sort $\mathbb{B} \rightarrow \kappa$ such that
 1. $b_i = 1$ iff $v \mathbf{t} \longrightarrow^* v'$ for some $v' \in V_i$,
 2. $b_i = 0$ iff $v \mathbf{f} \longrightarrow^* v'$ for some $v' \in V_i$, and
 3. $v \mathbf{t} \longrightarrow^* v'$ or $v \mathbf{f} \longrightarrow^* v'$ implies $v' \in \bigcup_{i \in \{0, \dots, N-1\}} V_i$.

Here $x = y$ is the shorthand for $\mathbf{if}(x, \mathbf{if}(y, \mathbf{t}, \mathbf{f}), \mathbf{if}(y, \mathbf{f}, \mathbf{t}))$. For $n \geq 1$, we define $\mathbf{N}(n+1) = {}^{\mathbb{B}}(\mathbf{N}(n))$. See the long version for the concrete definition of ${}^{\mathbb{B}}\mathbf{N}$.

Lemma 5. *$\mathbf{N}(n)$ is an implementation of natural numbers up to $\text{exp}_n(2)$. Furthermore, the sort, the function definition and the operations of $\mathbf{N}(n)$ can be constructed in time polynomial with respect to n .*

Proof (Theorem 1). The claim follows from Lemmas 3, 4 and 5. Note that (i) $\text{exp}_n(x) \leq \text{exp}_{n+x}(2)$, and (ii) given an order- n program with natural numbers up to $\text{exp}_m(2)$, the replacement of natural number constants with $\mathbf{N}(m)$ can be done in time polynomial with respect to m and the size of the program, and the resulting program is of order n (provided that $n \geq 2$). \square

4 Depth- n Reachability is n -EXPTIME Hard

In this section, we show a sketch of the proof of Theorem 2 below.

Theorem 2. *For every $n > 0$, the reachability problem for depth- n programs is n -EXPTIME hard.*

We reduce the emptiness problem of order- n alternating pushdown systems, which is known to be n -EXPTIME complete [4], to the reachability problem for depth- n programs. The basic idea originates from the work of Knapik et al. [6], which simulates a deterministic higher-order pushdown automaton by a safe higher-order grammar.

Since Knapik et al. [6] considered call-by-name grammars, we need to fill the gap between call-by-name and call-by-value. A problem arises when a divergent term that would not be evaluated in the call-by-name strategy appears in an argument position. We use the non-deterministic branch and the Boolean values to overcome the problem. Basically, by our reduction, every term of the ground sort is of the form $\mathbf{f} \oplus s$, and thus one can choose whether s is evaluated or not, by selecting one of the two possible reduction $\mathbf{f} \oplus s \rightarrow \mathbf{f}$ and $\mathbf{f} \oplus s \rightarrow s$. A detailed proof can be found in the long version.

5 Intersection-Type-Based Model-Checking Algorithm

We develop an intersection type system that completely characterises the reachability problem and give an upper bound of complexity of the reachability problem by solving the typability problem.

5.1 Types

The pre-types are given by the following grammar:

$$\begin{aligned} (\text{Value Pre-types}) \quad \theta &::= \mathbf{t} \mid \mathbf{f} \mid \bigwedge_{i \in I} (\theta_{1,i} \times \dots \times \theta_{n,i} \rightarrow \tau_i) \\ (\text{Term Pre-types}) \quad \tau, \sigma &::= \theta \mid \mathfrak{F}_\kappa \end{aligned}$$

The index I of the intersection is a finite set. We allow I to be the empty set, and we also write $\bigwedge \emptyset$ for the type. The subscript κ of \mathfrak{F}_κ is often omitted. We use infix notation for intersection, e.g. $(\theta_1 \rightarrow \tau_1) \wedge (\theta_2 \rightarrow \tau_2)$. The intersection connective is assumed to be associative, commutative and idempotent. Thus types $\bigwedge_{i \in I} (\theta_{1,i} \times \dots \times \theta_{n,i} \rightarrow \tau_i)$ and $\bigwedge_{j \in J} (\theta'_{1,j} \times \dots \times \theta'_{n,j} \rightarrow \tau'_{n,j})$ are equivalent if $\{(\theta_{1,i}, \dots, \theta_{n,i}, \tau_i) \mid i \in I\}$ and $\{(\theta'_{1,j}, \dots, \theta'_{n,j}, \tau'_j) \mid j \in J\}$ are equivalent sets.

Value pre-types are types for values and term pre-types are those for terms.

The value pre-type \mathbf{t} is for the Boolean value \mathbf{t} and \mathbf{f} for the Boolean value \mathbf{f} . The last one is for abstractions. It can be understood as the intersection of function types of the form $\theta_1 \times \dots \times \theta_n \rightarrow \tau$. The judgement $\lambda \langle \vec{x} \rangle . t : \theta_1 \times \dots \times \theta_n \rightarrow \tau$ means that, for all values $v_i : \theta_i$ (for every $i \leq n$), one has $[\vec{v}/\vec{x}]t : \tau$. For example, $\lambda x.x : \mathbf{t} \rightarrow \mathbf{t}$ and $\lambda x.x : \mathbf{f} \rightarrow \mathbf{f}$. The judgement $\lambda \langle \vec{x} \rangle . t : \bigwedge_{i \in I} (\theta_{1,i} \times \dots \times \theta_{n,i} \rightarrow \tau_i)$ means that, for every $i \in I$, one has $\lambda \langle \vec{x} \rangle . t : \theta_{1,i} \times \dots \times \theta_{n,i} \rightarrow \tau_i$. Therefore, $\lambda x.x : (\mathbf{t} \rightarrow \mathbf{t}) \wedge (\mathbf{f} \rightarrow \mathbf{f})$.

The term pre-type \mathfrak{F} means failure, i.e. $t : \mathfrak{F}$ just if $t \longrightarrow^* E[\mathfrak{F}]$. The term pre-type θ is for terms that is reducible to a value of type θ , i.e. $t : \theta$ just if $t \longrightarrow^* v$ and $v : \theta$ for some v . For example, consider $u_0 = (\lambda x.t) \oplus (\lambda x.f)$ and $u'_0 = \lambda x.(t \oplus f)$ in Example 1. Then $u_0 : \mathfrak{t} \rightarrow \mathfrak{t}$ since $u_0 \longrightarrow \lambda x.t$, and $u_0 : \mathfrak{t} \rightarrow \mathfrak{f}$ since $u_0 \longrightarrow \lambda x.f$. It is worth noting that $t : \theta_1$ and $t : \theta_2$ does not imply $t : \theta_1 \wedge \theta_2$, e.g. u_0 does not have type $(\mathfrak{t} \rightarrow \mathfrak{t}) \wedge (\mathfrak{t} \rightarrow \mathfrak{f})$. In contrast, $u'_0 : (\mathfrak{t} \rightarrow \mathfrak{t}) \wedge (\mathfrak{t} \rightarrow \mathfrak{f})$. So the difference between u_0 and u'_0 is captured by types.

Given a sort κ , the relation $\tau :: \kappa$, read “ τ is a refinement of κ ,” is inductively defined by the following rules:

$$\frac{}{\mathfrak{t} :: \mathbb{B}} \quad \frac{}{\mathfrak{f} :: \mathbb{B}} \quad \frac{}{\mathfrak{F} :: \kappa} \quad \frac{\theta_{k,i} :: \kappa_k \quad \tau_i :: \iota \quad (\text{for all } i \in I, k \in \{1, \dots, n\})}{\bigwedge_{i \in I} (\theta_{1,i} \times \dots \times \theta_{n,i} \rightarrow \tau_i) :: \kappa_1 \times \dots \times \kappa_n \rightarrow \iota}$$

Note that intersection is allowed only for pre-types of the same sort. So a pre-type like $((\mathfrak{t} \rightarrow \mathfrak{t}) \rightarrow \mathfrak{t}) \wedge (\mathfrak{t} \rightarrow \mathfrak{t})$ is not a refinement of any sort. A *type* is a value pre-type with its sort $\theta :: \kappa$ or a term pre-type with its sort $\tau :: \kappa$. A type is often simply written as θ or τ .

Let $\theta, \theta' :: \kappa$ be value types of the same sort. We define $\theta \wedge \theta'$ by:

$$\mathfrak{t} \wedge \mathfrak{t} = \mathfrak{t} \quad \mathfrak{f} \wedge \mathfrak{f} = \mathfrak{f} \quad \left(\bigwedge_{i \in I} (\vec{\theta}_i \rightarrow \tau_i) \right) \wedge \left(\bigwedge_{j \in J} (\vec{\theta}_j \rightarrow \tau_j) \right) = \bigwedge_{i \in I \cup J} (\vec{\theta}_i \rightarrow \tau_i)$$

and $\mathfrak{t} \wedge \mathfrak{f}$ and $\mathfrak{f} \wedge \mathfrak{t}$ are undefined.

5.2 Typing Rules

A *type environment* Γ is a finite set of type bindings of the form $x : \theta$ (here x is a variable or a function symbol). We write $\Gamma(x) = \theta$ if $x : \theta \in \Gamma$. We assume type bindings respect sorts, i.e. $x :: \kappa$ implies $\Gamma(x) :: \kappa$. A *type judgement* is of the form $\Gamma \vdash t : \tau$. The judgement intuitively means that, if each free variable x in t is bound to a value of type $\Gamma(x)$, then at least one possible evaluation of t results in a value of type τ . We abbreviate a sequence of judgements $\Gamma \vdash t_1 : \tau_1, \dots, \Gamma \vdash t_n : \tau_n$ as $\Gamma \vdash \vec{t} : \vec{\tau}$. The typing rules are listed in Fig. 2.

Here are some notes on typing rules. Rule (ABS) can be understood as the (standard) abstraction rule followed by the intersection introduction rule. Rule (APP) can be understood as the intersection elimination rule followed by the (standard) application rule. Note that intersection is introduced by (ABS) rule and eliminated by (APP) rule, which is the converse of the call-by-name case [7]. Rule (VAR) is designed for ensuring weakening. Rule (APP- $\mathfrak{F}1$) reflects the fact that, if $t \longrightarrow^* E[\mathfrak{F}]$, then $t \langle \vec{u} \rangle \longrightarrow^* E'[\mathfrak{F}]$ where $E' = E \langle \vec{u} \rangle$. Rule (APP- $\mathfrak{F}2$) reflects the fact that, if $t \longrightarrow v_0$ and $u_i \longrightarrow^* v_i$ for $i < l$, then $t \langle u_1, \dots, u_{l-1}, u_l, u_{l+1}, \dots, u_n \rangle \longrightarrow^* v_0 \langle v_1, \dots, v_{l-1}, E[\mathfrak{F}], u_{l+1}, \dots, u_n \rangle$. The premises $t : \theta_0$ and $u_i : \theta_i$ ($i < l$) ensure may-convergence of their evaluation.

Typability of a program is defined by using the notion of the n th approximation (see Section 2 for the definition). Let $P = \mathbf{let\ rec\ } D \mathbf{\ in\ } t$ be a program. Thus t is a term of sort \mathbb{B} with free occurrences of function symbols. We say the program P has type τ (written as $\vdash P : \tau$) just if $\vdash [t]_D^n : \tau$ for some n .

$$\begin{array}{c}
\frac{x : \theta \wedge \theta' \in \Gamma \text{ for some } \theta'}{\Gamma \vdash x : \theta} \quad (\text{VAR}) \\
\frac{b \in \{\mathbf{t}, \mathbf{f}\}}{\Gamma \vdash b : b} \quad (\text{BOOL}) \\
\frac{}{\Gamma \vdash \mathfrak{F} : \mathfrak{F}} \quad (\mathfrak{F}) \\
\frac{\Gamma, \vec{x} : \vec{\theta}_i \vdash t : \tau_i \text{ for all } i \in I}{\Gamma \vdash \lambda(\vec{x}).t : \bigwedge_{i \in I} (\vec{\theta}_i \rightarrow \tau_i)} \quad (\text{ABS}) \\
\frac{\Gamma \vdash t : \bigwedge_{i \in I} (\vec{\theta}_i \rightarrow \tau_i)}{\Gamma \vdash \vec{u} : \vec{\theta}_i \quad l \in I} \quad (\text{APP}) \\
\frac{\Gamma \vdash t : \mathfrak{F}}{\Gamma \vdash t \langle \vec{u} \rangle : \mathfrak{F}} \quad (\text{APP-}\mathfrak{F}1) \\
\frac{\Gamma \vdash t : \theta_0}{\Gamma \vdash u_1 : \theta_1} \\
\vdots \\
\frac{\Gamma \vdash u_{l-1} : \theta_{l-1}}{\Gamma \vdash u_l : \mathfrak{F}} \quad (\text{APP-}\mathfrak{F}2) \\
\frac{\Gamma \vdash t : \mathbf{t} \quad \Gamma \vdash s_1 : \tau}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \tau} \quad (\text{C-T}) \\
\frac{\Gamma \vdash t : \mathbf{f} \quad \Gamma \vdash s_2 : \tau}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \tau} \quad (\text{C-F}) \\
\frac{\Gamma \vdash t : \mathfrak{F}}{\Gamma \vdash \mathbf{if}(t, s_1, s_2) : \mathfrak{F}} \quad (\text{C-}\mathfrak{F}) \\
\frac{\exists i \in \{1, 2\} \quad \Gamma \vdash t_i : \tau}{\Gamma \vdash t_1 \oplus t_2 : \tau} \quad (\text{BR})
\end{array}$$

Fig. 2. Typing Rules

Soundness and completeness of the type system can be proved by using a standard technique for intersection type systems, except that Substitution and De-Substitution Lemmas are restricted to substitution of values and Subject Reduction and Expansion properties are restricted to call-by-value reductions. For more details, see the long version of the paper.

Theorem 3. $\vdash P : \mathfrak{F}$ if and only if P fails.

5.3 Type-Checking Algorithm and Upper Bound of Complexity

We provide an algorithm that decides the typability of a given depth- n program P in time $O(\mathbf{exp}_n(\mathit{poly}(|P|)))$ for some polynomial poly . Let $P = \mathbf{let\ rec\ } D \mathbf{\ in\ } t$ and suppose that $\vdash D :: \Delta$, $\Delta \vdash t :: \mathbb{B}$ and $\Delta = \{f_i :: \delta_i \mid i \in I\}$.

We define $\mathcal{T}(\kappa) = \{\tau \mid \tau :: \kappa\}$ and $\mathcal{T}(\Delta) = \{I \mid I :: \Delta\}$. For $\tau, \sigma \in \mathcal{T}(\kappa)$, we write $\tau \preceq \sigma$ just if $\tau = \sigma \wedge \sigma'$ for some σ' . The ordering for type environments is defined similarly. Let \mathcal{F}_D be a function on $\mathcal{T}(\Delta)$, defined by:

$$\mathcal{F}_D(\Theta) = \left\{ f : \bigwedge \{ \vec{\theta} \rightarrow \tau \mid \Theta, \vec{x} : \vec{\theta} \vdash t : \tau \} \mid (f = \lambda(\vec{x}).t) \in D \right\}.$$

The algorithm to decide whether $\vdash \mathbf{let\ rec\ } D \mathbf{\ in\ } t : \mathfrak{F}$ is shown in Fig. 3.

Termination of the algorithm comes from monotonicity of \mathcal{F}_D and finiteness of $\mathcal{T}(\Delta)$. Correctness is a consequence of the following lemma and the monotonicity of the approximation (i.e. if $[t]_D^m$ fails and $m \leq m'$, then $[t]_D^{m'}$ fails).

```

1  :   $\Theta_0 := \{f : \bigwedge \emptyset \mid f \in \text{dom}(\Delta)\}, \Theta_1 = \mathcal{F}_D(\Theta_0), i := 1$ 
2  :  while  $\Theta_i \neq \Theta_{i-1}$  do
2-1:     $\Theta_{i+1} := \mathcal{F}_D(\Theta_i)$ 
2-2:     $i := i + 1$ 
3  :  if  $\Theta_i \vdash t : \mathfrak{F}$  then yes else no

```

Fig. 3. Algorithm checking if $\vdash \text{let rec } D \text{ in } t : \mathfrak{F}$

Lemma 6. *Suppose $\Delta \mid \mathcal{K} \vdash t :: \mathbb{B}$. Then $\emptyset \vdash [t]_D^n : \tau$ if and only if $\Theta_n \vdash t : \tau$.*

We shall analyse the cost of the algorithm. For a set A , we write $\#A$ for the number of elements. The *height* of a poset A is the maximum length of strictly increasing chains in A .

Lemma 7. *Let κ be a sort of depth n . Then $\#\mathcal{T}(\kappa) \leq \mathbf{exp}_{n+1}(2|\kappa|)$ and the height of $\mathcal{T}(\kappa)$ is bounded by $\mathbf{exp}_n(2|\kappa|)$.*

Lemma 8. *Let $\Delta \mid \mathcal{K} \vdash t :: \kappa$ be a sorted term of depth n , and $\Theta :: \Delta$. Assume that $\text{depth}(\mathcal{K}) \leq n - 1$. Then $A_{\Theta,t} = \{(\Gamma, \tau) \in \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa) \mid \Theta, \Gamma \vdash t : \tau\}$ can be computed in time $O(\mathbf{exp}_n(\text{poly}(|t|)))$ for some polynomial *poly*.*

Proof. We can compute $A_{\Theta,t}$ by induction on t . An important case is that the sort κ is of depth n . In this case, there exists $B_{\Theta,t} \subseteq \mathcal{T}(\mathcal{K}) \times \mathcal{T}(\kappa)$ such that (1) $(\Gamma, \tau) \in A_{\Theta,t}$ if and only if $(\Gamma, \tau') \in B_{\Theta,t}$ for some $\tau' \preceq \tau$ and (2) for each Γ , the number of elements in $B_{\Theta,t} \upharpoonright \Gamma = \{\tau \mid (\Gamma, \tau) \in B_{\Theta,t}\}$ is bounded by $|t|$. See the long version for the proof of this claim. By using $B_{\Theta,t}$ as the representation of $A_{\Theta,t}$, $A_{\Theta,t}$ can be computed in the desired bound. For other cases, one can enumerate all the elements in $A_{\Theta,t}$, since $\#A_{\Theta,t} \leq \mathbf{exp}_n(2(|\mathcal{K}| + |\kappa|)) \leq \mathbf{exp}_n(2|t|)$ (here we assume w.l.o.g. that each variable in $\text{dom}(\mathcal{K})$ appears in t). \square

Theorem 4. *The reachability problem for depth- n programs is in n -EXPTIME.*

Proof. By Lemma 8, each iteration of loop 2 in Fig. 3 runs in n -EXPTIME. Since the height of $\mathcal{T}(\Delta)$ is bounded by $\mathbf{exp}_n(2|\Delta|)$, one needs at most $\mathbf{exp}_n(2|\Delta|)$ iterations for loop 2, and thus loop 2 runs in n -EXPTIME. Again by Lemma 8, step 3 can be computed in n -EXPTIME. Thus the algorithm in Fig. 3 runs in n -EXPTIME for depth- n programs. \square

6 Related Work

Higher-order model checking. Model-checking recursion schemes against modal μ -calculus (known as higher-order model checking) has been proved to be decidable by Ong [15], and applied to various verification problems of higher-order programs [7,11,12,17]. The higher-order model-checking problem is n -EXPTIME complete for order- n recursion schemes [15]. The reachability problem for *call-by-name* programs is an instance of the higher-order model checking, and $(n-1)$ -EXPTIME complete for order- n programs [10].

Model-checking call-by-value programs via the CPS translation. The previous approach for model-checking call-by-value programs is based on the CPS translation. Our result implies that the upper bound given by the CPS translation is not tight. However this does not imply that the CPS translation followed by call-by-name model-checking is inefficient. It depends on the model-checking algorithm. For example, the naïve algorithm in [7] following the CPS translation takes more time than our algorithm, but we conjecture that HORSAT [2] following the CPS translation meets the tight bound.

Sato et al. [16] employed the selective CPS translation [14] to avoid unnecessary growth of the order, using a type and effect system to capture effect-free fragments and then added continuation parameters to only effectful parts.

Intersection types for call-by-value calculi. Davies and Pfenning [3] studied an intersection type system for a call-by-value effectful calculus and pointed out that the value restriction on the intersection introduction rule is needed. In our type system, the intersection introduction rule is restricted immediately after the abstraction rule, which can be considered as a variant of the value restriction.

Similarly to the previous work on type-based approaches for higher-order model checking [7,8,9], our intersection type system is a variant of the Essential Type Assignment System in the sense of van Bakel [18], in which the typing rules are syntax directed. Our syntax of intersection types differs from the standard one for call-by-name calculi. Our syntax is inspired by the embedding of the call-by-value calculus into the linear lambda calculus [13], in which the call-by-value function type $A \rightarrow B$ is translated into $!(A \multimap B)$ (recall that function types in our intersection type system is $\bigwedge_i(\tau_i \rightarrow \sigma_i)$).

Zeilberger [19] proposed a principled design of the intersection type system based on the idea from focusing proofs [1]. Its connection to ours is currently unclear, mainly because of the difference of the target calculi.

Our type system is designed to be complete. This is a characteristic feature that the previous work for call-by-value calculi [3,19] does not have.

7 Conclusion

We have studied the complexity of the reachability problem for call-by-value programs, and proved the following results. First, the reachability problem for order-3 programs is non-elementary, and thus the order of the program does not serve as a good measure of the complexity, in contrast to the call-by-name case. Second, the reachability problem for depth- n programs is n -EXPTIME complete, which improves the previous upper bound given by the CPS translation.

For future work, we aim to (1) develop an efficient model-checker for call-by-value programs, using the type system proposed in the paper, and (2) study the relationship between intersection types and focused proofs [1,19].

Acknowledgement. This work is partially supported by JSPS KAKENHI Grant Number 23220001.

References

1. Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. *J. Log. Comput.* 2(3), 297–347 (1992)
2. Broadbent, C.H., Kobayashi, N.: Saturation-based model checking of higher-order recursion schemes. In: *CSL 2013. LIPIcs*, vol. 23, pp. 129–148, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
3. Davies, R., Pfenning, F.: Intersection types and computational effects. In: *ICFP 2000*, pp. 198–208. ACM (2000)
4. Engelfriet, J.: Iterated stack automata and complexity classes. *Inf. Comput.* 95(1), 21–75 (1991)
5. Igarashi, A., Kobayashi, N.: Resource usage analysis. *ACM Trans. Program. Lang. Syst.* 27(2), 264–313 (2005)
6. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) *FOSSACS 2002. LNCS*, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
7. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: *POPL 2009*, pp. 416–428. ACM (2009)
8. Kobayashi, N.: Model checking higher-order programs. *J. ACM* 60(3), 20 (2013)
9. Kobayashi, N., Ong, C.-H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: *LICS 2009*, pp. 179–188. IEEE Computer Society (2009)
10. Kobayashi, N., Ong, C.-H.L.: Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science* 7(4) (2011)
11. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: *PLDI 2011*, pp. 222–233. ACM (2011)
12. Kobayashi, N., Tabuchi, N., Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification. In: *POPL 2010*, pp. 495–508. ACM (2010)
13. 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)
14. Nielsen, L.R.: A selective cps transformation. *Electr. Notes Theor. Comput. Sci.* 45, 311–331 (2001)
15. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: *LICS*, pp. 81–90. IEEE Computer Society (2006)
16. Sato, R., Unno, H., Kobayashi, N.: Towards a scalable software model checker for higher-order programs. In: *PEPM 2013*, pp. 53–62. ACM (2013)
17. Tobita, Y., Tsukada, T., Kobayashi, N.: Exact flow analysis by higher-order model checking. In: Schrijvers, T., Thiemann, P. (eds.) *FLOPS 2012. LNCS*, vol. 7294, pp. 275–289. Springer, Heidelberg (2012)
18. van Bakel, S.: Intersection type assignment systems. *Theor. Comput. Sci.* 151(2), 385–435 (1995)
19. Zeilberger, N.: Refinement types and computational duality. In: *PLPV 2009*, pp. 15–26. ACM (2009)

Resource Reachability Games on Pushdown Graphs

Martin Lang*

RWTH Aachen University, Lehrstuhl für Informatik 7, D-52056 Aachen, Germany
lang@automata.rwth-aachen.de

Abstract. We consider two-player reachability games with additional resource counters on arenas that are induced by the configuration graphs of pushdown systems. For a play, we define the resource cost to be the highest occurring counter value. In this way, we quantify resources and memory that player 0 needs to win. We introduce the bounded winning problem: Is there a uniform bound k such that player 0 can win the game from a set of initial configurations with this bound k ? We provide an effective, saturation-based method to solve this problem for regular sets of initial and goal configurations.

1 Introduction

Pushdown automata have become an important tool in the formal analysis and verification of recursive programs. Since their introduction by A.G. Oettinger in 1961 and M.-P. Schützenberger in 1963, they have been intensively studied and are relatively well-understood today. Pushdown automata without input alphabet, which only operate with their control states on the stack, are usually called *pushdown systems*. The configuration graphs of such systems are called *pushdown graphs*. They can be used as a formal model for recursive programs because they combine good expressive power with an (efficiently) decidable point-to-point reachability problem. An example of their application in the area of formal verification is the model checker jMoped [14], which uses symbolic pushdown systems to verify Java bytecode.

However, mere reachability on transition systems lacks the possibility to model an environment system or possible user input. This can be achieved by two-player games on graphs. Such games were studied in the course of the controller synthesis problem proposed by A. Church in [8], and many positive algorithmic results are known today for games on finite graphs. Moreover, two player games on pushdown graphs with ω -regular winning conditions were solved in [15] by I. Walukiewicz. Later, T. Cachat showed in [4] that the well-known saturation approach for pushdown system reachability can be extended to reachability and Büchi games on pushdown graphs.

* Supported by DFG research grant *Automatentheoretische Verifikationsprobleme mit Ressourcenschranken*

Recently, several models of games with additional resource constraints were introduced to provide a model for systems with resource consumption. In this context, the resources are usually modeled by integer counters that can be modified by the players but not read during the game. In addition to usual winning conditions such as Büchi or Parity, the winning conditions of these games restrict the values of the resource counters throughout the game. Typically, it is required that these values are bounded by a global limit. Examples for such games are energy games (cf. [5]), energy parity games (cf. [6]), or consumption games (cf. [3]). However, all these previous games are defined over finite graphs. Only very recently, and independently from the author, games with such a structure were considered on graphs induced by pushdown systems (cf. [7]). Although the game model considered there is essentially the same, the questions we solve and the methods we use are quite different from those in [7].

In this work, we consider *resource pushdown systems*. These are pushdown systems that are extended with a finite set of non-negative integer counters. They can be used to model recursive programs with resource consumption. We examine two-player games on the configuration graphs of these systems. Every resource counter can be modified by the pushdown rules either by incrementing it (for short *i*), or resetting it to zero (for short *r*). Moreover, it is possible to leave a counter unchanged (no operation or *n*). The counters cannot be read during the game. This reflects a step-by-step consumption and all-at-once replenishment model of resources. The form of the counters is the same as in the ωB -games considered in [7] and very similar to the model used in consumption games (cf. [3]) on finite graphs. It is also used in the model of B-automata (cf. [9]).

We introduce reachability games with an additional bound on the resource counters – called *resource reachability games*. We fix some *resource bound* $k \in \mathbb{N}$. In a play on the configuration graph of a resource pushdown system, the counters are updated according to the operations associated with the used pushdown rules. In order to win the game w.r.t. the bound k , player 0 does not only have to reach a certain set of goal configurations F but also needs to ensure that all counter values throughout the play stay below k . We examine this kind of game and present a method to compute the winning region and winning strategies for player 0 in the case of regular sets of goal configurations. Furthermore, we investigate the *bounded winning problem*. Given a set A of initial configurations. Is there a *uniform* resource bound $k \in \mathbb{N}$ such that player 0 wins the resource reachability game from all configurations in A w.r.t. this bound k ? In order to solve this problem, we thoroughly analyze the propagation of counter operations in the saturation approach. Thereby, we can extend the saturation idea introduced by T. Cachat to effectively translate the question of winning cost into a membership query for alternating B-automata. In total, we reduce the bounded winning problem to a boundedness problem for B-automata. In contrast to [7], we consider only finite plays and are interested in finding *uniform* bounds that can be respected by all plays starting from sets of initial configurations. In [7], infinite games are considered with an interest in checking for a given initial configuration whether for each play there is an individual bound.

This work is grouped into three parts. First, we fix the notation and present the preliminaries. Second, we formally introduce resource pushdown systems and resource reachability games. Furthermore, we investigate some general properties of the games and calculate the winning region of player 0 for a given resource bound. In the third part, we consider the bounded winning problem and present our solution approach.

2 Preliminaries

For a set Σ , we denote the set of finite sequences (or words) over Σ by Σ^* . For a word $w \in \Sigma^*$, we write $w(i)$ for the i -th letter in the word (zero indexed). For sets A, B , we write B^A for the set of all functions from A to B and B_p^A for the set of all partial functions from A to B . We write \perp to indicate that a partial function is undefined on some value.

The fundamental model that we consider are two-player games on graphs. A game graph $\mathcal{A} = (V, E)$ is directed and its vertices V are partitioned into two sets V_0, V_1 indicating to which player the vertex belongs. Such a game graph is often called *arena*. The two players are called 0 (or Eve) and 1 (or Adam). A *play* of a game on \mathcal{A} is a (possibly infinite) sequence of moves in which the two players move a game pebble across the graph. A play starts in some vertex $v \in V$. In each step of the game, the player to which the current vertex belongs can move the pebble to one of the successor vertices. Formally, we say a play is the sequence τ of edges along which the pebble is moved.

The winner of a fixed play in a game is determined by the so-called *winning-condition*. In a reachability game, we fix a *goal set* $F \subseteq V$ of vertices in the game graph. Player 0 wins the reachability game if the play visits a vertex from F after a finite number of moves. Otherwise, player 1 wins the game. In general, we are interested in knowing whether one of the players can force to win (independent of how the other player moves) by following a certain *strategy*. A strategy for player i is a mapping σ that maps all past moves ($\in E^*$) of the play to the next edge to take whenever the current position is a vertex of V_i . A strategy is called *winning* for player i if all plays in which player i moves according to the strategy are winning for player i . A strategy is called *finite memory* if it can be implemented by a finite state Mealy machine that reads all the moves of the opponent and outputs the next move of the respective player. It is called *memoryless* or *positional* if the next move only depends on the current vertex. We call the vertices from which player i has a winning strategy the *winning region* of player i . A game is called *determined* if for every starting vertex either player 0 or player 1 has a winning strategy. Reachability games are known to be determined and admit positional winning strategies for both players on their respective winning regions. A comprehensive introduction to two-player games and proofs for the claims above can be found, e.g., in [11].

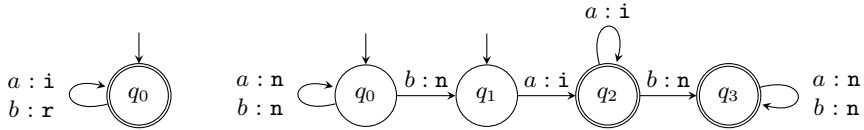


Fig. 1. Example B-automaton: left: count maximal length of uninterrupted a -block / right: count minimal length of uninterrupted a -block

2.1 Counters as Resource Model

We model resources by a finite set of non-negative integer counters. Each counter supports two kinds of operations. First, the counter can be incremented (for short i). This represents the usage of a single resource. Second, a counter can be reset to zero (for short r). This models the full replenishment of the resource. Additionally, we use n as a shorthand notation for no operation (the counter is left unchanged). The counters operate independently from each other. Thus, we can use multiple counters in order to model different types of resources. We associate the *resource usage* or *consumption* with the highest occurring counter value. This scheme of step-by-step consumption and all at once replenishment is motivated by scenarios such as battery driven systems or the usage of paper in a printer.

Finite state automata with similar counters have been studied in [1] (R-automata) and [9] (B-automata). In the context of this work, we use the model of B-automata and known results for this formalism as a tool. B-automata were introduced by T. Colcombet in [9] and extend finite state automata with a finite set of counters (denoted by Γ) as described above¹. The counters can be manipulated by the transitions but not read by the automaton. Throughout a run, the counters are updated according to the used transitions. The value of a run is the maximal counter value (over all counters) that occurs in the run. B-automata naturally define a function from words to $\mathbb{N} \cup \{\infty\}$. For a B-automaton \mathfrak{A} and a word w , we assign w to the infimum of the values of all accepting runs of \mathfrak{A} on w and denote this value by $\llbracket \mathfrak{A} \rrbracket(w)$. We also call it the (resource) cost of w . Note that $\llbracket \mathfrak{A} \rrbracket(w) = \infty$ if there is no accepting run for w .

Figure 1 shows two examples of B-automata. Their semantics are to count the maximal (left) / minimal (right) number of subsequent letters a without interruption. The left automaton just increments for each letter a and resets the counter to zero when it reads a b . Consequently, the unique run of a word has the value of the longest uninterrupted block of as . The right automaton calculates the minimal length of a blocks by nondeterministically guessing the position of the minimal block in the word. It changes to q_1 when the block starts (or starts in q_1 if this block is at the beginning) and counts its length.

In the context of systems with resources modeled by counters as described above, we are especially interested in the question of *boundedness*: Is there a bound on the resource consumption for a given set of runs? A solution to this

¹ In difference to the original publication we do not use the counter operation *check* because it is not necessary for our work and simplifies the overall presentation.

question is part of the realizability problem since real world systems can only have limited resources. With formal verification in mind, we are especially interested in decidable variants of this question. The *boundedness problem* for B-automata is decidable, i.e., one can algorithmically check if there is a global bound $k \in \mathbb{N}$ for a given B-automaton \mathfrak{A} such that for all words w , we have $\llbracket \mathfrak{A} \rrbracket(w) \leq k$. In the case of multiple counters, this was first shown by D. Kirsten in [13] for a slightly more restrictive counter model (hierarchical counters). He also proved that this problem is PSPACE-hard. In the case of B-automata, the boundedness problem was solved by T. Colcombet in [9].

2.2 Counter Profiles

In order to provide a well understandable way to reason about sequences of counter operations, we introduce a structured representation in the form of a well-partially ordered monoid. This enables us to present our results more generally for systems that are annotated with such a structure and to emphasize which properties are needed to obtain the results. For sequences of counter operations, we introduce the notion of *counter profiles* and use this model instead of sequences of counter operations in the context of the bounded winning problem. A counter profile is a 3-tuple $(i_{\leftarrow}^+, c_{max}, i_{\rightarrow}^+) \in (\mathbb{N} \cup \{\diagup\})^3$. It represents a sequence u of counter operations (from $\{\mathbf{i}, \mathbf{n}, \mathbf{r}\}^*$) with the following intuition. For the sake of simplicity, we assume that u does not contain any \mathbf{n} s since they have no influence on the counter. The component i_{\leftarrow}^+ represents the number of increments before the first reset, i.e., the largest $j \in \mathbb{N}$ such that \mathbf{i}^j is a prefix of u . The component c_{max} represents the maximal counter value between two subsequent resets, i.e., the largest $j \in \mathbb{N}$ such that $\mathbf{ri}^j\mathbf{r}$ is an infix of u . Lastly, i_{\rightarrow}^+ represents the number of increments after the last reset, i.e., the largest $j \in \mathbb{N}$ such that \mathbf{ri}^j is a suffix of u . If the sequence u contains only one (or even no) reset, the components c_{max} (and i_{\rightarrow}^+) are set to \diagup (read n/a). On these profiles, we define the concatenation \circ such that it reflects the concatenation of counter sequences. One can see by checking all cases that all counter profiles together with the concatenation and $(0, \diagup, \diagup)$ as neutral element form a monoid. Each of the three base operations directly corresponds to a profile – \mathbf{n} to $(0, \diagup, \diagup)$, \mathbf{i} to $(1, \diagup, \diagup)$ and \mathbf{r} to $(0, \diagup, 0)$. By translating each operation into its profile and concatenating all the profiles along a run, one obtains a profile that provides the value of this run by its maximal entry as well as all information necessary to interpret the sequence as part of a longer sequence. As a result, we can use counter profiles as an equivalent representation for counter sequences.

In contrast to counter sequences, counter profiles offer a natural way to define a partial order. For two profiles p_1 and p_2 we say p_1 is less than or equal to p_2 (and write $p_1 \leq p_2$) if all components of p_1 are less than or equal to p_2 (component-wise order). In each component we use the canonical order on \mathbb{N} and let the newly introduced \diagup be incomparable to all natural numbers. This order is a well-partial order in every component since it has neither infinitely decreasing chains nor infinite anti-chains. By a result of Higman (cf. [12]), we obtain that the component-wise order we defined on the counter profiles is a

well-partial order, too. We remark that the order is compatible with the monoid operation.

Systems with several counters can be represented by a vector of counter profiles. We extend the concatenation and the order to these vectors by applying the concatenation in each component and taking the component-wise order. Again by the result of Higman and simple checking, we obtain that these vectors of counter profiles still form a monoid and the order is a well-partial order. For a set of such vectors of profiles or a set of profiles A , we denote the set of maximal elements of A by $\max A$. We remark that $\max A$ may contain several elements because the order is not total. However, by definition of maximal, $\max A$ is an anti-chain and thus finite.

3 Resource Reachability Games

We introduce pushdown systems with a finite set of counters as model for recursive programs with resource consumption. These counters follow the previously described ideas and provide a way to model step-by-step usage and all at once replenishment of several resource types during the execution of recursive programs.

Definition 1. A resource pushdown system is a 4-tuple $\mathcal{P} = (P, \Sigma, \Delta, \Gamma)$ where P is a finite set of control states, Σ is a finite stack alphabet, $\Delta \subseteq P \times \Sigma \times \Sigma^* \times P \times \{\mathbf{i}, \mathbf{r}, \mathbf{n}\}^\Gamma$ is a finite transition relation and Γ is a finite set of counters.

Similar to normal pushdown systems, a configuration of a resource pushdown system is a pair of a state from P and a finite word from Σ^* . The successor relation on configurations is defined similar to normal pushdown systems. We additionally associate this step of the system with the counter operation f of the transition used. Formally, for two configurations $pu, qv \in P \times \Sigma^*$, we say qv is an f -successor of pu and write $pu \vdash_f qv$ if there is a common suffix w and a transition $(p, a', v', q, f) \in \Delta$ such that $u = a'w$ and $v = v'w$. We denote the configuration graph of \mathcal{P} by $\mathcal{C}_{\mathcal{P}} = (P \times \Sigma^*, \vdash)$. In our examples, we use systems with only one counter and write, e.g., $pa \xrightarrow{\mathbf{i}} qv$ as a shorthand notation for a transition (p, a, v, q, f) where f maps the unique counter to \mathbf{i} . Analogously, we write $pu \vdash_{\mathbf{i}} qv$ if $pu \vdash_f qv$ with $f(c) = \mathbf{i}$ for the unique counter c .

We obtain a game arena from the configuration graph of a resource pushdown system by providing an additional partition of the state space $P = P_0 \uplus P_1$. Configurations with a state in P_i belong to player i . A game on this arena is played as in the classical case but each move additionally provides counter operations according to the corresponding pushdown rule. As for B-automata, we simulate the counters along the play and associate the *resource consumption* at every point in the play with the highest counter value that occurred so far. On such arenas, we consider a combined reachability and resource limit objective. Let F be a set of goal configurations and $k \in \mathbb{N}$ be a resource limit. Player 0 wins the *resource reachability game* with respect to F and k if the play reaches a configuration in F and the resource consumption at this point is at most k .

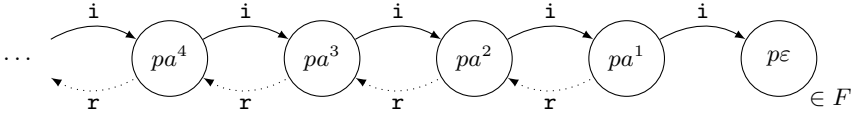


Fig. 2. Example for the configuration graph of a simple resource pushdown system

For these games, we consider different kinds of winning regions. First, we consider the resource independent winning region of player 0 denoted by $W_0(F)$. A configuration pw is in $W_0(F)$ if player 0 can reach F from pw with arbitrarily high resource consumption. Second, we consider the winning region with resource limit k denoted by $W_0^{(k)}(F)$. A configuration pw is in $W_0^{(k)}(F)$ if player 0 wins the resource reachability game with the respective limit k on the resource consumption. This second, new type of winning region immediately yields two algorithmic questions:

1. We fix F and $k \in \mathbb{N}$ and ask what is $W_0^{(k)}(F)$?
2. We fix a set A and ask whether there is a uniform resource bound k such that player 0 wins the resource reachability game with bound k from A , i.e., whether $A \subseteq W_0^{(k)}(F)$. We call this problem the *bounded winning problem*.

We illustrate the newly introduced concepts with the following example. Consider a resource pushdown system $\mathcal{P} = (P, \Sigma, \Delta, \Gamma)$ with only one state $p \in P$, the stack alphabet $\Sigma = \{a\}$ and only one pushdown rule $pa \xrightarrow{i} p\epsilon \in \Delta$. Figure 2 (without the dotted transitions) shows a part of the configuration graph of \mathcal{P} . On this configuration graph, we compare the different winning regions for the resource reachability game in which all configurations belong to player 0 and the goal set is $F = \{p\epsilon\}$. First, the resource independent winning region is $W_0(F) = \{pa^n \mid n \in \mathbb{N}\}$ because one can remove all letters a from the stack by successively applying the rule $pa \xrightarrow{i} p\epsilon$. However, each such step costs one increment of the resource counter. Hence, the winning region with resource bound k is $W_0^{(k)}(F) = \{pa^n \mid n \leq k\}$. Consequently, there is no uniform bound k such that player 0 wins on complete $W_0(F)$ with this bound.

Now, we add the pushdown rule $pa \xrightarrow{r} paa$ to Δ of \mathcal{P} . Then, the configuration graph includes the dotted transitions in Figure 2. This does not change $W_0(F)$ in the considered resource reachability game but reduces the resources needed to reach F from an arbitrary configuration to 2. For instance, let us start at configuration pa^3 . The sequence $pa^3 \vdash_i pa^2 \vdash_i pa \vdash_r pa^2 \vdash_i pa \vdash_i p\epsilon$ shows that F is reachable with a resource bound of 2. This idea of incrementing two times and then resetting one time can easily be extended to all configurations. Thus, we obtain that 2 is a uniform bound such that player 0 wins the resource reachability game, i.e., $W_0^{(2)}(F) = W_0(F)$. This example already shows that we generally cannot expect to obtain memoryless winning strategies for player 0 in resource reachability games. Moreover, it illustrates that memoryless strategies cannot obtain minimal resource bounds even on finite graphs.

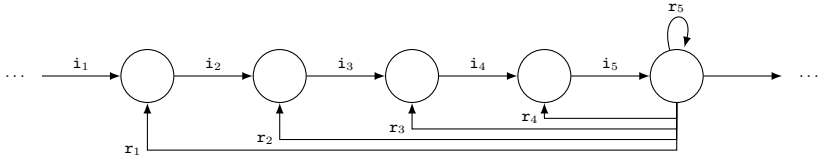


Fig. 3. Exponential memory in the number of counters is unavoidable to achieve the best resource-limit possible

In the following, we solve the two algorithmic questions for the case that F and A are regular. First, we consider the problem of calculating $W_0^{(k)}(F)$ for a fixed F and k . This problem can be reduced to solving (normal) reachability games on pushdown graphs. The reduction is based on the idea of simulating the counters up to the (finite) value k in the state space of the pushdown system. With standard techniques (see e.g. [11]), we can obtain the winning region and a winning strategy for the original game. We formalize this idea by

Proposition 2. *Let $\mathcal{P} = (P_0 \uplus P_1, \Sigma, \Delta, \Gamma)$ be a resource pushdown system. Let F be a regular goal set and $k \in \mathbb{N}$ a resource bound for the bounded reachability game on the configuration graph of \mathcal{P} . One can effectively compute the winning region $W_0^{(k)}(F)$ and a corresponding finite memory winning strategy.*

In a similar way as previously described, we obtain winning strategies for player 1 for all configurations in $P \times \Sigma^* \setminus W_0^{(k)}(F)$. As a direct consequence, we obtain that resource reachability games are determined. We remark that this idea can be easily extended to all ω -regular winning conditions.

The strategy obtained from the above reduction uses a memory structure that is exponential in the number of counters. The example in Figure 3 shows that this is generally unavoidable if the strategy should achieve the lowest possible resource bound. We use 5 counters in the example and denote the increment/reset of counter j by i_j/r_j . While it is possible to get through the shown gadget with resource limit 1 and all counters reset to zero before leaving, the strategy of player 0 has to store the state of all counters in order to achieve this ($2^5 = 32$). Nevertheless, if we allow a resource limit of 5, all counters can be reset to zero before leaving the gadget with a memory structure of size 6.

4 The Bounded Winning Problem

In this section, we first show that the bounded winning problem is at least as complex as solving the boundedness problem for B-automata. As already mentioned, this is known to be PSPACE-hard (cf. [13]) even for a slightly simpler version of automata (with hierarchical counters). Furthermore, there is a 2-EXPSpace algorithm (cf. [1]) to solve it. Subsequently, we introduce an alternating variant of automata with monoid annotations, such as counter profiles. This model enables us to extend a saturation-based solution approach for normal reachability games on pushdown graphs to resource reachability games.

Proposition 3. *The bounded winning problem is at least as complex as the boundedness problem for B-automata.*

Proof. Let $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, \text{Fin}, \Gamma)$ be a B-automaton. We define a resource pushdown system with the following idea in mind: The pushdown system simulates the automaton by letter-wise consuming the stack contents and simulating the operation of \mathfrak{A} in its state space and with its counters. Formally, we define the resource pushdown system by $\mathcal{P} = (Q, \Sigma, \Delta', \Gamma)$ where

$$\Delta' := \{(p, a, \varepsilon, q, f) \mid (p, a, q, f) \in \Delta\}$$

With this definition, we obtain for all words $w \in \Sigma^*$ the equivalence that $\llbracket \mathfrak{A} \rrbracket(w) \leq k$ iff there is a $q_f \in \text{Fin}$ and a sequence of pushdown operations to reach $q_f \varepsilon$ from $q_0 w$ with resource limit k . Consequently, there is a global bound k such that for all words $w \in \Sigma^* : \llbracket \mathfrak{A} \rrbracket(w) \leq k$ iff there is a bound k such that one can reach $\text{Fin} \times \{\varepsilon\}$ from all configurations in $q_0 \Sigma^*$ with bound k . That is exactly the bounded winning problem on \mathcal{P} for $P_0 = Q, P_1 = \emptyset, F = \text{Fin} \times \{\varepsilon\}$ and $A = \{q_0 w \mid w \in \Sigma^*\}$. \square

The main tool in constructing our saturation method for resource reachability games is the model of alternating automata with B-automaton like counters. We introduce and argue on the base of a slightly more general model with annotations from well-partially ordered monoids. This shows the properties we use more clearly, and simplifies the presentation. In the analysis of resource reachability games, we instantiate the model with (vectors of) counter profiles as a formalism equivalent to B-automaton counters. We model the alternation by nondeterministic choice among transitions with possibly several target states. A run of the alternating automaton has the form of a tree. For the transition chosen, the run has to be continued from all target states of the respective transition. This is an explicit presentation of the otherwise often used positive boolean formula notation for transitions of alternating automata. It has the advantage, for our purpose, that we can associate the different paths in the automaton with different annotations more easily. This is needed to reflect the multiple choices in the game. Formally, we have

Definition 4. *An annotated alternating automaton is a tuple $\mathfrak{A} = (Q, \Sigma, \text{In}, \Delta, F, \mathcal{M})$. The components Q, Σ, In and F are defined as usually for automata. $\mathcal{M} = (M, \circ, e_{\mathcal{M}}, \leq)$ is a well-partially ordered monoid. The transition relation Δ is a finite set $\Delta \subseteq Q \times \Sigma \times (\text{AntiChain}(M))_{\mathbb{P}}^Q$ where $\text{AntiChain}(M)$ is the set of all anti-chains with elements from M . For a transition $t = (p, a, f)$, we define the successor states of the transition by $\text{Succ}(t) := \text{dom}(f)$. The automaton is called normalized, if states in In have no ingoing transitions.*

In order to define a run and the (annotation) values associated with the run, we need the notion of a tree.

Definition 5. *A tree \mathfrak{T} consists of a set of nodes T , a root node $t_0 \in T$ and a child (or successor) function $s_{\mathfrak{T}} : T \rightarrow \text{Pow}(T)$ such that:*

1. for every node $v \in T \setminus \{t_0\}$ there is a unique parent node $p \in T$ such that $v \in s_{\mathfrak{T}}(p)$.
2. the child function has no loops, i.e., there is no sequence v_0, v_1, \dots, v_n with $v_{i+1} \in s_{\mathfrak{T}}(v_i)$ for all $i = 1, \dots, n-1$ such that $v_0 = v_n$.
3. every node is reachable from the root, i.e., for all nodes $v \in T$ there is a sequence $t_0 = v_0, \dots, v_n = v$ such that $v_{i+1} \in s_{\mathfrak{T}}(v_i)$.

We use the following common operations on trees. The parent function $\pi_{\mathfrak{T}} : T \setminus \{t_0\} \rightarrow T$ maps all nodes but the root to their unique parent nodes. The distance function $d_{\mathfrak{T}} : T \rightarrow \mathbb{N}$ maps every node v to its distance from the root node. The leaves of a tree are denoted by $\text{Leafs}_{\mathfrak{T}} = \{v \in T \mid s_{\mathfrak{T}}(v) = \emptyset\}$. A level of a tree is a maximal set of nodes $T' \subseteq T$ that all have the same distance from the root, i.e., for $v, v' \in T'$ we always have $d_{\mathfrak{T}}(v) = d_{\mathfrak{T}}(v')$.

A run of an alternating automaton on a word $w = a_1 \dots a_n$ follows the idea of an inductive tree construction. It starts with the root node and associates this node with the initial state of the automaton. Then, a transition $(p, a_1, f) \in \Delta$ is selected and child nodes are created for all states in $q \in \text{dom}(f)$ with their different annotations $f(q)$. For all child nodes, this construction continues on the rest of the word a_2, \dots, a_n . A run is called accepting if all the leaf nodes of the tree are associated with final states of the automaton. Moreover, such a run yields values from the annotation monoid by multiplying the annotations along each path. We formalize this idea in the following two definitions.

Definition 6. *A run of an annotated alternating automaton \mathfrak{A} on a word w is a 4-tuple $\rho = (\rho_Q, \rho_{\Delta}, \rho_M, \mathfrak{T})$ of three labeling functions and a tree \mathfrak{T} . The function $\rho_Q : T \rightarrow Q$ is called state labeling function. The function $\rho_{\Delta} : T \setminus \text{Leafs}_{\mathfrak{T}} \rightarrow \Delta$ is called transition labeling function. The function $\rho_M : T \setminus \{t_0\} \rightarrow M$ is the annotation labeling function. They satisfy the following consistency properties:*

1. $\rho_Q(t_0) \in \text{In}$
2. The state labeling and the transition labeling are consistent with each other and with the word w : For all $v \in T \setminus \text{Leafs}_{\mathfrak{T}}$ with labeled state $\rho_Q(v) = q$ and selected transition $\rho_{\Delta}(v) = t = (p, a, f)$ we have $w(d_{\mathfrak{T}}(v)) = a$, $q = p$, $\rho_Q(s_{\mathfrak{T}}(v)) = \text{Succ}(t)$
3. For each node $v \in T \setminus \text{Leafs}_{\mathfrak{T}}$ with $\rho_{\Delta}(v) = t$ and every state $q \in \text{Succ}(t)$, there is exactly one child per annotation. Let $V_q = \{v' \in s_{\mathfrak{T}}(v) \mid \rho_Q(v') = q\}$. We have $|V_q| = |f(q)|$ and $\rho_M(V_q) = f(q)$.

We call a run partial if it does not start in In , and call it accepting if $\rho_Q(\text{Leafs}_{\mathfrak{T}}) \subseteq F$. If there is an accepting run of \mathfrak{A} on w , we write $w \in L(\mathfrak{A})$. Moreover, if there is a run of \mathfrak{A} on w with $\rho_Q(\text{Leafs}_{\mathfrak{T}}) \subseteq S$, we write $w \in L_S(\mathfrak{A})$.

Definition 7. *Let $\rho = (\rho_Q, \rho_{\Delta}, \rho_M, \mathfrak{T})$ be a runtree of an automaton $\mathfrak{A} = (Q, \Sigma, \text{In}, \Delta, F, \mathcal{M})$. For each $q \in \rho_Q(\text{Leafs}_{\mathfrak{T}})$, we define the value of ρ inductively over the runtree.*

$$\text{val}_\rho^q(v) = \begin{cases} \{e_{\mathcal{M}}\} & \text{if } \rho_Q(v) = q \\ \emptyset & \text{otherwise} \end{cases} \quad \text{if } v \in \text{Leaf}_{s_{\bar{\tau}}} \\ \text{val}_\rho^q(v) = \max\{m_1 \circ m_2 \mid v' \in s_{\bar{\tau}}(v), m_1 = \rho_M(v'), \\ m_2 \in \text{val}_\rho^q(v')\} \quad \text{otherwise}$$

Additionally, we define the total value $\text{val}_\rho(v) = \max_{q \in \rho_Q(\text{Leaf}_{s_{\bar{\tau}}})} \text{val}_\rho^q(v)$. We write $\text{val}^q(\rho)$ as a shorthand for $\text{val}_\rho^q(t_0)$ and $\text{val}(\rho)$ as a shorthand for $\text{val}_\rho(t_0)$.

In the context of resource reachability games on pushdown graphs, we take the annotation monoid \mathcal{M} to be the vector of counter profiles with its dimension matching the number of counters. Furthermore, we use the idea of P -automata to read pushdown configurations. For a pushdown system \mathcal{P} with control states P and an automaton \mathfrak{A} reading configurations from \mathcal{P} , we assume that P is part of the state space of \mathfrak{A} . A run of \mathfrak{A} on some pushdown configuration pw then starts in state p of \mathfrak{A} . This way \mathfrak{A} operates only on Σ and we do not need to distinguish the different kinds of input symbols (from P and Σ).

The traditional saturation approach for reachability in pushdown systems gradually extends a finite automaton with transitions that enable the automaton to simulate replacement steps of the pushdown system. It starts with a P -automaton \mathfrak{A} that recognizes some set F of pushdown configurations to be reached. For a pushdown rule $pa \rightarrow qu$ it searches states q' that can be reached in \mathfrak{A} when reading the configuration qu . Then, an a -transition from p is added to q' . This new transition enables the automaton to behave as if it had read qu although it actually read pa . Hence, the automaton can now simulate the pushdown rule $pa \rightarrow qu$. This is repeated until no more transitions can be added. The resulting automaton recognizes the set of all predecessor configurations of F . A complete presentation of this basic idea can be found in [2].

In [4], T. Cachat used alternating automata to lift this basic idea to reachability games. His approach uses the similarities between games and the semantics of alternating automata in the following way. Let \mathfrak{A} be an alternating automaton recognizing a regular goal set F of the reachability game. Consider a player 1 state p of the pushdown system and let $pa \rightarrow q_1u_1, \dots, pa \rightarrow q_nu_n$ be all pushdown rules originating in p with letter a on top of the stack. The saturation method now looks for states q'_i in \mathfrak{A} that can be reached when reading q_iu_i ($i \in \{1, \dots, n\}$) on \mathfrak{A} . It then adds an a -transition from p to $\{q'_1, \dots, q'_n\}$ in \mathfrak{A} . By the semantics of alternating automata, a run of the automaton is continued in all target states of the transition. This reflects the fact that player 1 can choose among $pa \rightarrow q_1u_1, \dots, pa \rightarrow q_nu_n$ and player 0 has to be able to reach the goal set F for all possible choices in order to win the reachability game. The case of player 0 states can be handled similar to the case of mere reachability. Again, this procedure is repeated until no more transitions can be added to the automaton. The resulting alternating automaton recognizes the winning region of player 0.

We extend this idea with the overall goal of constructing an automaton with counters that recognizes a pushdown configuration with cost k iff player 0 has

a strategy to win the resource reachability game with a resource limit of k . To realize this, the designed saturation method has to keep track of the resource counter operations executed by the simulated pushdown transitions. We use the monoid annotation of the automaton to store the counter profile associated with the resource counter operation of the simulated pushdown rule. The major difficulty arises from the fact that there are incomparable choices in the game. While it is easy to see that two increments are better than three, there are (even for finite game graphs) situations that do not have a unique best choice. For example, consider a resource pushdown system with two counters and two nearly identical pushdown rules. The first rule resets the first counter and increments the second whereas the other increments the first and resets the second. The decision which one is better depends on the context in this case. Thus, the designed method has to represent all such situations.

We illustrate the intuition behind newly added transitions with an example. Again, let \mathfrak{A} be the automaton to be saturated. Consider a player 1 state p of the pushdown system and let $pa \xrightarrow{m_1} q_1u_1, \dots, pa \xrightarrow{m_n} q_nu_n$ be all pushdown rules originating in p that can be applied with an a on top of the stack. For the sake of clarity, assume that there are linear (non-branching) runs of \mathfrak{A} on q_iu_i . These runs end in a state q'_i and have an accumulated annotation m'_i . Then, the saturation procedure adds a transition $t = (p, a, f)$ with target states $\text{dom}(f) = \{q'_1, \dots, q'_n\}$. This part is identical to the situation without annotations. Similar to the target states of t , which represent the states into which player 1 can force player 0, the annotation in \mathfrak{A} has to represent the possible annotations in the game that player 1 can enforce while reaching this state. Consequently, the annotation $f(q)$ for a target state $q \in \text{dom}(f)$ is the maximum over all combined annotations $m_i \circ m'_i$ (first apply the pushdown rule, then the skipped part of the original run) of runs that end in a state q'_i equal to q .

In order to simplify the presentation of the saturation algorithm and the subsequent termination argument, we introduce an order on the transitions of the automaton and show that this order is a well-partial order. First, we order sets of elements from ordered monoids. Let \mathcal{M} be a well-partially ordered monoid and $A, B \subseteq M$ two sets. We say A is dominated by B and write $A \leq B$ if for all elements $a \in A$ there is an element $b \in B$ such that $a \leq b$. Now, let $t = (p, a, f), t' = (p', a', f')$ be two transitions of an annotated alternating automaton. We order t and t' by $t \leq t'$ if $p = p', a = a', \text{dom}(f) = \text{dom}(f')$ and for all $q \in \text{dom}(f)$ $f(q) \leq f'(q)$. With the finiteness of the states, the alphabet, and arguments about well-partial orders from [12], we deduce:

Lemma 8. *Let $\mathfrak{A} = (Q, \Sigma, \text{In}, \Delta, F, \mathcal{M})$ be an annotated alternating automaton. The order on Δ is a well-partial order.*

With all previous preparations, we are now able to present the complete saturation procedure in Algorithm 1. It operates on arbitrary monoid annotations and assumes the resource pushdown system to be labeled with counter profiles. The resulting automaton and the arguments are stated in terms of counter profiles. To this end, we say that player 0 wins the resource reachability game with profile bound B if the resulting play has a combined counter profile p such that

Algorithm 1. Saturation procedure

input : resource pushdown system $\mathcal{P} = (P, \Sigma, \Delta_{\mathcal{P}})$, state partition $P = P_1 \uplus P_2$, normalized annotated alternating P -automaton $\mathfrak{A} = (Q, \Sigma, P, \Delta, F, \mathcal{M})$ (all annotations are $\{e_{\mathcal{M}}\}$, and for all $(q, a, f) \in \Delta \mid \text{dom}(f) = 1$)

output: annotated alternating P -automaton $\mathfrak{A}^* = (Q, \Sigma, P, \Delta^*, F, \mathcal{M})$

```

1   $\mathfrak{A}^0 := \mathfrak{A} ; i := 0$ 
2  while automaton can be updated do
3      For  $p \in P_0, pa \xrightarrow{m} qw \in \Delta_{\mathcal{P}}$  do
4          Find Runtree  $\rho = (\rho_Q, \rho_{\Delta}, \mathfrak{T})$  of  $\mathfrak{A}^i$  on  $qw$ 
5           $t := (p, a, f)$  with
6           $f := q \mapsto \begin{cases} \max\{m \circ m_{\rho} \mid m_{\rho} \in \text{val}^q(\rho)\} & \text{if } q \in \rho_Q(\text{Leaf}_{\mathfrak{T}}) \\ \perp & \text{otherwise} \end{cases}$ 
7          if  $\exists t' \in \Delta^i$  with  $t' > t$  then
8               $\Delta^{i+1} := \Delta^i \setminus \{t' \in \Delta^i \mid t' > t\} \cup \{t\} ; i := i + 1$ 
9          else if  $\neg(\exists t' \in \Delta^i \text{ with } t' \leq t)$  then
10              $\Delta^{i+1} := \Delta^i \cup \{t\} ; i := i + 1$ 
11     For  $p \in P_1$  and  $pa \xrightarrow{m_1} q_1 w_1 \in \Delta_{\mathcal{P}}, \dots, pa \xrightarrow{m_n} q_n w_n \in \Delta_{\mathcal{P}}$  all  $a$ -pushdown
12     rules starting in  $p$  do
13         Find Runtrees  $\rho^j = (\rho_Q^j, \rho_{\Delta}^j, \mathfrak{T}^j)$  of  $\mathfrak{A}^i$  on  $q_j w_j$  for  $j \in \{1, \dots, n\}$ 
14          $t := (p, a, f)$  with  $f$  defined by
15          $q \mapsto \begin{cases} \max\{m_j \circ m_{\rho} \mid m_{\rho} \in \text{val}^q(\rho^j), \\ j \in \{1, \dots, n\}\} & \text{if } q \in \bigcup_{j=1}^n \rho_Q^j(\text{Leaf}_{\mathfrak{T}^j}) \\ \perp & \text{otherwise} \end{cases}$ 
16         if  $\exists t' \in \Delta^i$  with  $t' > t$  then
17              $\Delta^{i+1} := \Delta^i \setminus \{t' \in \Delta^i \mid t' > t\} \cup \{t\} ; i := i + 1$ 
18         else if  $\neg(\exists t' \in \Delta^i \text{ with } t' \leq t)$  then
19              $\Delta^{i+1} := \Delta^i \cup \{t\} ; i := i + 1$ 

```

Result: $\mathfrak{A}^* := \mathfrak{A}^i$

$\{p\} \leq B$. Accordingly, we write $W_0^{(B)}(F)$ to denote the region in which player 0 wins with profile bound B .

Lemma 9. *Algorithm 1 terminates for all inputs.*

Proof. We remark that the set of all transitions always forms an anti-chain by construction. If the algorithm does not terminate either ll. 7, 13 or ll. 9,15 are executed infinitely many times. Assume the instructions in ll. 7 or 13 are executed infinitely many times. Since there are only finitely many transitions in Δ^i at each point in time, there has to be a descending chain of transitions. This is a contradiction to the fact that the order is a well-partial order. Otherwise,

ll. 9 or 15 are executed infinitely often. Since both update rules only add a new transition relation, we obtain $\Delta^i \subsetneq \Delta^{i+1}$ for all $i > j$ for some threshold j . Thus, the set $\bigcup_{i=j}^{\infty} \Delta^i$ is an infinite anti-chain. Also a contradiction. \square

In order to prove the correctness of the saturation procedure, we show a direct correspondence between the games winning with certain profiles and runs on the saturated automaton.

Lemma 10. *Let F be a regular set of configurations represented by a normalized annotated alternating P -automaton \mathfrak{A} as stated in the precondition of Algorithm 1. Furthermore, let \mathfrak{A}^* be the result of the algorithm and $B \in \text{AntiChain}(M)$ such that $\{e_{\mathcal{M}}\} \leq B$.*

- (i) *Let $qw \in W_0^{(B)}(F)$. Then, there is an accepting run ρ of \mathfrak{A}^* on qw such that $\text{val}(\rho) \leq B$.*
- (ii) *Let $\rho = (\rho_Q, \rho_{\Delta}, \rho_M, \mathfrak{T})$ be a run of \mathfrak{A}^* on qw with $S = \rho_Q(\text{Leaf}_{\mathfrak{T}})$. Then player 0 has a strategy σ_{ρ} to reach a configuration in $L(\mathfrak{A}_S)$. Moreover, for a play τ that is played according to σ_{ρ} and that ends in a configuration $q'w'$, let $\rho^{\mathfrak{A}}$ be an accepting run of \mathfrak{A} on $q'w'$ with (single) final state r . The value of τ is bounded by $\text{val}^r(\rho)$.*

The above presented procedure effectively reduces the bounded winning problem to a boundedness problem for alternating B-automata. After exchanging the counter profiles back to direct counter operations, we obtain an automaton that recognizes a configuration pw with some cost k iff player 0 wins the resource reachability game from this configuration with bound k . Since the set of initial configurations A is regular, we can easily construct an automaton that recognizes the complement \bar{A} with cost 0 and A with cost ∞ . By the closure of B-automata under taking the minimum (cf. [10]) we obtain an automaton that yields value 0 for all elements outside of A . As a consequence, the boundedness of this automaton only depends on the boundedness on A . By [10], we know that boundedness for these automata is decidable. Altogether, we obtain our main result:

Theorem 11. *Let $\mathcal{P} = (P_0 \uplus P_1, \Sigma, \Delta, \Gamma)$ be a resource pushdown system. Let F be a regular goal set for the bounded reachability game on the configuration graph of \mathcal{P} and A a regular set of start configurations. It is decidable whether there is a $k \in \mathbb{N}$ such that $A \subseteq W_0^{(k)}(F)$.*

We remark that the winning strategies constructed in the inductive proof cannot be directly transformed into pushdown strategies as in the traditional case. This difference arises from the fact that we do not have memoryless runs on our alternating automaton model because of incomparable annotations. However, once the bound k is determined, one can use the methods of the previous section to obtain a finite memory winning strategy.

5 Conclusion

We considered two-player games on pushdown graphs with additional non-negative integer counters as model for reactive, recursive systems with resource consumption. On these games, we examined a combined reachability and resource

limit winning condition. We showed that these games are determined and that for regular goal sets one can compute the winning region of player 0 with a certain resource limit as well as decide whether there is a resource limit such that player 0 wins from a given regular set of initial configurations with this limit.

In the main theorem, we solved the bounded winning problem by an extension of the traditional saturation idea. Our approach can be used for a variety of annotated pushdown games. We only require that the annotations form a well-partially ordered monoid. In the semantics, the value of a play has to be associated with the concatenation of all the values on the transitions along the play and the order has to reflect that smaller annotations are preferred. For the specific case of counter profiles as annotation, we obtain a reduction of the bounded winning problem to the boundedness problem of alternating B-automata.

References

1. Abdulla, P.A., Krcal, P., Yi, W.: R-automata. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 67–81. Springer, Heidelberg (2008)
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
3. Brázdil, T., Chatterjee, K., Kučera, A., Novotný, P.: Efficient controller synthesis for consumption games with multiple resource types. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 23–38. Springer, Heidelberg (2012)
4. Cachat, T.: Symbolic strategy synthesis for games on pushdown graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 704–715. Springer, Heidelberg (2002)
5. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
6. Chatterjee, K., Doyen, L.: Energy parity games. *Theoretical Computer Science* 458, 49–60 (2012)
7. Chatterjee, K., Fijalkow, N.: Infinite-state games with finitary conditions. In: CSL, pp. 181–196 (2013)
8. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic* 1, 3–50 (1957)
9. Colcombet, T.: Regular cost functions over words (2009)
10. Colcombet, T., Löding, C.: Regular cost functions over finite trees. In: LICS, pp. 70–79 (2010)
11. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
12. Higman, G.: Ordering by Divisibility in Abstract Algebras. *Proceedings London Mathematical Society* s3-2(1), 326–336 (1952)
13. Kirsten, D.: Distance desert automata and the star height problem. *RAIRO - Theoretical Informatics and Applications* 39, 455–509 (2005)
14. Suwimonteerabuth, D., Schwoon, S., Esparza, J.: jMoped: A java bytecode checker based on moped. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 541–545. Springer, Heidelberg (2005)
15. Walukiewicz, I.: Pushdown processes: Games and model-checking. *Inf. Comput.* 164(2), 234–263 (2001)

Perfect-Information Stochastic Mean-Payoff Parity Games^{*,**}

Krishnendu Chatterjee¹, Laurent Doyen², Hugo Gimbert³, and Youssef Oualhadj⁴

¹ IST Austria

² LSV, ENS Cachan & CNRS, France

³ LaBRI, CNRS, France

⁴ LIF, Université Aix Marseille, France

Abstract. The theory of graph games is the foundation for modeling and synthesizing reactive processes. In the synthesis of stochastic processes, we use $2\frac{1}{2}$ -player games where some transitions of the game graph are controlled by two adversarial players, the System and the Environment, and the other transitions are determined probabilistically. We consider $2\frac{1}{2}$ -player games where the objective of the System is the conjunction of a qualitative objective (specified as a parity condition) and a quantitative objective (specified as a mean-payoff condition). We establish that the problem of deciding whether the System can ensure that the probability to satisfy the mean-payoff parity objective is at least a given threshold is in $\text{NP} \cap \text{coNP}$, matching the best known bound in the special case of 2-player games (where all transitions are deterministic). We present an algorithm running in time $O(d \cdot n^{2d} \cdot \text{MeanGame})$ to compute the set of *almost-sure* winning states from which the objective can be ensured with probability 1, where n is the number of states of the game, d the number of priorities of the parity objective, and MeanGame is the complexity to compute the set of almost-sure winning states in $2\frac{1}{2}$ -player mean-payoff games. Our results are useful in the synthesis of stochastic reactive systems with both functional requirement (given as a qualitative objective) and performance requirement (given as a quantitative objective).

1 Introduction

Perfect-information stochastic games. A perfect-information stochastic graph game [16] is played on a finite directed graph with three kinds of states (or vertices): player-Max, player-Min, and probabilistic states. At player-Max states, player Max chooses a successor state; at player-Min states, player Min (the adversary of player Max) chooses a successor state; and at probabilistic states, a successor state is chosen according to a fixed probability distribution. The result of playing the game forever is an infinite path through the graph. If there are no probabilistic states, we refer to the game as a *2-player graph game*; otherwise, as a *$2\frac{1}{2}$ -player graph game*. There has been

* This research was supported by Austrian Science Fund (FWF) Grant No P23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), Microsoft Faculty Fellowship Award, and European project Cassting (FP7-601148).

** Fuller version: IST Technical Report No IST-2013-128.

a long history of using 2-player graph games for modeling and synthesizing reactive processes [7,23,26]: a reactive system and its environment represent the two players, whose states and transitions are specified by the states and edges of a game graph. Consequently, $2\frac{1}{2}$ -player graph games provide the theoretical foundation for modeling and synthesizing processes that are both reactive and stochastic [17,25]. They subsume both 2-player games which have no probabilistic states, and Markov decision processes (MDPs) which have no player-Min states.

Qualitative and quantitative objectives. In the analysis of reactive systems, the goal is specified as a set of desired paths (such as ω -regular specifications), or as a quantitative optimization objective for a payoff function on the paths. In verification and synthesis of reactive systems all commonly used properties are expressed as ω -regular objectives, and parity objectives are a canonical way to express ω -regular objectives [27]. In a parity objective, an integer priority is assigned to every state, and a path satisfies the objective for player Max if the maximum priority visited infinitely often is even. The most classical example of quantitative objective is the mean-payoff objective [17,24], where a reward is associated with every state and the payoff of a path is the long-run average of the rewards of the path. While traditionally the verification and the synthesis problems were considered with qualitative objectives, recently combinations of qualitative and quantitative objectives have received a lot of attention. Qualitative objectives such as ω -regular objectives specify the functional requirements of reactive systems, whereas the quantitative objectives specify resource consumption requirements (such as for embedded systems or power-limited systems). Combining quantitative and qualitative objectives is crucial in the design of reactive systems with both resource constraints and functional requirements [9,14,5,3]. For example, mean-payoff parity objectives are relevant in synthesis of optimal performance lock-synchronization for concurrent programs [8], where one player is the synchronizer, the opponent is the environment, and the randomization arises due to the randomized scheduler; the performance objective is specified as mean-payoff condition and the functional requirement (e.g., data-race freedom or liveness) as an ω -regular objective. Mean-payoff parity objectives have also been used in other applications such as to define permissivity for parity games [6]. Thus $2\frac{1}{2}$ -player mean-payoff parity games provide the theoretical foundation for analysis of stochastic reactive systems with functional as well as performance requirements.

Algorithmic questions in $2\frac{1}{2}$ -player games. The study of $2\frac{1}{2}$ -player games has a wealth of algorithmic problems. For example, given a $2\frac{1}{2}$ -player game with reachability objective (where the goal is to reach a target set of states), whether the player Max can ensure the objective with probability at least $\frac{1}{2}$ (called the value-strategy problem) is in $\text{NP} \cap \text{coNP}$ [16]. This is one of the rare combinatorial problems that belong to $\text{NP} \cap \text{coNP}$, but are not known to be solvable in polynomial time. It is a major and long-standing open question whether the problem can be solved in polynomial time. Moreover, 2-player games with mean-payoff (resp. parity) objectives lies in $\text{NP} \cap \text{coNP}$ (even in $\text{UP} \cap \text{coUP}$) [21,29,20], and again no polynomial time algorithm is known. Both 2-player parity games and 2-player mean-payoff games admit a polynomial reduction to the value-strategy problem of $2\frac{1}{2}$ -player reachability games. The value-strategy problem for $2\frac{1}{2}$ -player mean-payoff (resp. parity) games also lie in $\text{NP} \cap \text{coNP}$: the key property to show that the problem is in $\text{NP} \cap \text{coNP}$ for mean-payoff (resp. parity) games

is to show that it is sufficient to consider positional strategies (that are independent of the past history and depend only on the current state), see [22] for mean-payoff and [15] for parity objectives. In this work we consider $2\frac{1}{2}$ -player games with conjunction of mean-payoff and parity objectives for player Max. The study of $2\frac{1}{2}$ -player games with conjunction of mean-payoff and parity objectives poses new algorithmic challenges as *infinite-memory* strategies are required. The key challenge is to obtain *succinct* (polynomial) witnesses for the infinite-memory strategies and their characterization to obtain complexity results matching the simpler classes of games where positional strategies suffice. Besides the complexity result, our characterization of strategies will also allow us to obtain algorithms to solve $2\frac{1}{2}$ -player mean-payoff parity games.

Contributions. The details of our contributions are as follows:

1. We first present polynomial witnesses for infinite-memory strategies required by player Max, and a polynomial-time verification procedure for the witnesses, thereby establishing that the value-strategy problem (of whether player Max can ensure that the probability to satisfy the objective is at least a given threshold) is in NP. The fact that player Max requires infinite-memory strategies follows from the special case of 2-player mean-payoff parity games [14].
2. We show that positional strategies are sufficient for player Min (note that player Max and Min are asymmetric since player Max has a conjunction of parity and mean-payoff objectives to satisfy, whereas player Min has disjunction of parity or mean-payoff objectives to falsify). From the existence of positional strategies for player Min it follows that the value-strategy problem is also in coNP. Our $\text{NP} \cap \text{coNP}$ bound for the problem matches the special cases of 2-player mean-payoff parity games.
3. We present an algorithm for the computation of the almost-sure winning set (the set of states where the objective can be ensured with probability 1 by player Max) for $2\frac{1}{2}$ -player mean-payoff parity games in time $O(d \cdot n^{2d} \cdot \text{MeanGame})$, where n is the number of states of the game graph, d the number of priorities of the parity objective, and MeanGame denotes the complexity to compute the almost-sure winning set in $2\frac{1}{2}$ -player mean-payoff games.

In summary, we present results that establish computational, strategy, and algorithmic complexity of solving $2\frac{1}{2}$ -player mean-payoff parity games.

Technical difficulty. For 2-player games the $\text{NP} \cap \text{coNP}$ result for mean-payoff parity objectives was established in [10]: the technique relied on reduction of 2-player mean-payoff parity games to 2-player energy-parity games, and in 2-player energy-parity games finite-memory strategies suffice (for details related to energy objectives see [10,12]). However the technique of reduction of mean-payoff games to energy games (even without the parity condition) for almost-sure winning does not hold in the presence of stochastic transitions because for energy conditions (which are like safety conditions) the precise probabilities do not matter, whereas they matter for mean-payoff conditions. Hence the techniques for 2-player mean-payoff parity games do not extend to $2\frac{1}{2}$ -player games, and we need to explicitly construct succinct witness to show the $\text{NP} \cap \text{coNP}$ result. The succinct witness construction of infinite-memory strategies in

the presence of adversary and stochastic transitions is the main technical challenge in $2\frac{1}{2}$ -player mean-payoff parity games.

Related works. The problem of 2-player mean-payoff parity games was first studied in [14]. The $\text{NP} \cap \text{coNP}$ complexity bound was established in [10], and an improved algorithm for the problem was given in [6]. The algorithmic analysis of $2\frac{1}{2}$ -player mean-payoff games has been studied in [1,4]: a reduction to $2\frac{1}{2}$ -player reachability games was presented in [1], and approximation schemes were considered in [4]. The polynomial time complexity for MDPs with mean-payoff parity objectives was established in [11] and the polynomial time complexity for MDPs with positive average parity objectives was shown in [19]. The generalization to $2\frac{1}{2}$ -player games with mean-payoff parity objectives gives rise to many delicate issues, such as dealing at the same time with infinite-memory strategies, stochastic transitions, as well as the opponent.

2 Definitions

In this section we present definitions of game graphs, objectives, and the basic decision problems.

Probability Distributions. For a finite set S , we denote by $\Delta(S)$ the set of all probability distributions over S , i.e., the set of functions $p : S \rightarrow [0, 1]$ such that $\sum_{s \in S} p(s) = 1$. For a set $U \subseteq S$ we use the following notation: $p(U) = \sum_{s \in U} p(s)$.

Stochastic Games. A *perfect-information stochastic game graph* (for brevity, stochastic game) is a tuple $\mathcal{G} = (S, (S_{\text{Max}}, S_{\text{Min}}), A, \delta)$, where S is a finite set of states, $(S_{\text{Max}}, S_{\text{Min}})$ is a partition of S such that S_{Max} is the set of states controlled by player Max and S_{Min} is the set of states controlled by player Min, A is a finite set of actions, and $\delta : S \times A \rightarrow \Delta(S)$ is a probabilistic transition function. Stochastic games are also known as $2\frac{1}{2}$ -player games where probabilistic states are explicitly present. In our model, the probabilistic states can be embedded in the probabilistic transition function. A *Markov decision process* (MDP) is the special case of a stochastic game where either $S_{\text{Max}} = \emptyset$, or $S_{\text{Min}} = \emptyset$. Typically in this paper, we obtain MDPs from stochastic games after fixing the action choices of one of the players.

For complexity issues, we assume that the probabilities in stochastic games are rational numbers whose numerator and denominator are encoded in binary. We denote by $|\delta|$ the size of the encoding of the probabilistic transition function δ .

Subgames and Traps. Given a stochastic game \mathcal{G} , a set $U \subseteq S$ of states induces a subgame if for all $s \in U$, there exists an action $a_s \in A$ such that $\delta(s, a_s)(U) = 1$; the induced subgame is $\mathcal{G}[U] = (U, (U \cap S_{\text{Max}}, U \cap S_{\text{Min}}), A, \delta')$ where, for all states $s \in U$ and action $a \in A$, we have $\delta'(s, a) = \delta(s, a)$ if $\delta(s, a)(U) = 1$, and $\delta'(s, a) = \delta(s, a_s)$ otherwise. We take this definition of subgame to keep the same alphabet of actions in every state. The subgame $\mathcal{G}[U]$ is a *trap* for player Min in the original game \mathcal{G} if for all $s \in U \cap S_{\text{Min}}$ and for all $a \in A$ we have $\delta(s, a)(U) = 1$. A trap for player Max is defined similarly.

Plays and Strategies. A *play* $\rho = s_0 s_1 \cdots \in S^\omega$ is an infinite sequence of states such that for all $i \geq 0$ there exists $a \in A$ such that $\delta(s_i, a)(s_{i+1}) > 0$. A *strategy* for Max is a recipe to describe what is the next action to play; formally, it is a function

$\sigma : S^* S_{\text{Max}} \rightarrow A$. A *positional* strategy is independent of the past and depends only on the current state. We view it as a function $\sigma : S_{\text{Max}} \rightarrow A$.

A strategy σ uses *finite memory* if there exists an equivalence relation \sim on S^ω of finite index, such that $\sigma(\rho_1) = \sigma(\rho_2)$ for all plays ρ_1, ρ_2 such that $\rho_1 \sim \rho_2$. We define strategies, positional strategies, and finite-memory strategies analogously for Min. A strategy that is not finite-memory is referred to as an infinite-memory strategies.

Probability Measures. Given a finite prefix $\rho \in S^*$ of a play, denote by $|\rho|$ the length of ρ and by $\text{Cone}(\rho)$ the set of plays with prefix ρ . If $\rho \in S^+$ is nonempty, we denote by $\text{Last}(\rho)$ the last state of ρ . Given a pair of strategies (σ, τ) for Max and Min, and an initial state s , we first define the probability measure on cones inductively as follows: for all $s' \in S$, let

$$\mathbb{P}_s^{\sigma, \tau}(\text{Cone}(s')) = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{if } s' \neq s \end{cases}$$

and for all $\rho \in S^+$ (where $S^+ = S^* \setminus \{\epsilon\}$ and ϵ is the empty string), let

$$\mathbb{P}_s^{\sigma, \tau}(\text{Cone}(\rho \cdot s')) = \begin{cases} \mathbb{P}_s^{\sigma, \tau}(\text{Cone}(\rho)) \cdot \delta(\text{Last}(\rho), \sigma(\rho))(s') & \text{if } \text{Last}(\rho) \in S_{\text{Max}} \\ \mathbb{P}_s^{\sigma, \tau}(\text{Cone}(\rho)) \cdot \delta(\text{Last}(\rho), \tau(\rho))(s') & \text{if } \text{Last}(\rho) \in S_{\text{Min}} \end{cases}$$

By Caratheodary's extension theorem, there is a unique extension of this probability measure to S^ω which is also denoted as $\mathbb{P}_s^{\sigma, \tau}(\cdot)$ [2].

Mean-Payoff Parity Objectives. An *objective* is a measurable set $\varphi \subseteq S^\omega$ of plays. Let $\text{rwd} : S \times S \rightarrow \mathbb{Q}$ be a *reward function* defined on edges and $\chi : S \rightarrow \mathbb{N}$ be a *priority function* defined on states. Given a set of states $U \subseteq S$ and a priority $d \in \mathbb{N}$, we denote by $U(d)$ the set $\{s \in U \mid \chi(s) = d\}$ of states with priority d . The *mean-payoff* objective $\text{Mean} = \{s_0 s_1 \cdots \in S^\omega \mid \limsup_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \text{rwd}(s_i, s_{i+1}) \geq 0\}$ requires that the long-run average of rewards be non-negative. The *parity objective* $\text{Par} = \{s_0 s_1 \cdots \in S^\omega \mid \limsup_{n \rightarrow \infty} \chi(s_n) \text{ is even}\}$ requires that the maximal priority visited infinitely often be even. The *mean-payoff parity objective* $\text{Mean} \cap \text{Par}$ is the conjunction of a mean-payoff objective Mean and a parity objective Par.

Almost-Sure and Positive Winning. We say that player Max wins almost-surely (resp., positively) from an initial state s for an objective φ if there exists a strategy σ for Max such that for every strategy τ of player Min we have $\mathbb{P}_s^{\sigma, \tau}(\varphi) = 1$ (resp., $\mathbb{P}_s^{\sigma, \tau}(\varphi) > 0$). The state s is called *almost-sure* (resp., *positive*) winning for Max. In the sequel, we say that a game \mathcal{G} is almost-sure (resp., positive) winning, if every state in \mathcal{G} is almost-sure (resp., positive) winning for Max. We use analogous definitions for player Min. Note that almost-sure winning for Max is the dual of positive winning for Min.

Value-Strategy Problem and Reduction to Almost-Sure Winning. Given a threshold λ , the *value-strategy* problem for an objective asks whether there exists a strategy for player Max to ensure against all strategies of player Min that the objective is satisfied with probability at least λ . A strategy for player Max is *optimal* if it ensures the maximal value λ (for stochastic mean-payoff parity games, optimal strategies are guaranteed to exist [18]). In this paper we focus on the *almost-sure winning problem*, which is to decide whether there exists an almost-sure winning strategy for player Max for a mean-payoff parity objective, that is the value-strategy problem for $\lambda = 1$. While for player Max infinite-memory strategies are necessary [14], we will show that for

player Min positional strategies are sufficient, and that the almost-sure winning problem is in $\text{NP} \cap \text{coNP}$.

Remark 1. It follows from the results of [13, Lemma 7] and [18, Theorem 4.1] that since mean-payoff parity objectives are *tail* objectives (independent of finite prefixes), the memory requirement for optimal strategies of both players is the same as for almost-sure winning strategies, and if the almost-sure winning problem is in $\text{NP} \cap \text{coNP}$, then the value-strategy problem is also in $\text{NP} \cap \text{coNP}$. The details are as follows: The results of [13, Lemma 7] and [18, Theorem 4.1] show that for the quantitative analysis of tail objectives it suffices to guess the *value classes* (where a value class for r , with $0 \leq r \leq 1$, is the set of states with value r), almost-sure winning witness in a modified game for each value class, and then the verification problem requires the almost-sure witness verification in each value class, and verification of MDPs which is polynomial time. Since $\text{NP} \cap \text{coNP}$ bound for the almost-sure problem imply polynomial witness and polynomial-time verification for the witness, it follows (using the results of [13,18]) that the $\text{NP} \cap \text{coNP}$ bound for almost-sure winning imply that there exists polynomial witness and polynomial-time verification for quantitative analysis, and thereby establish the $\text{NP} \cap \text{coNP}$ bound. Thus from our results it will follow that the value-strategy problem is in $\text{NP} \cap \text{coNP}$ for $2^{\frac{1}{2}}$ -player mean-payoff parity games.

Positive Attractors. Given a stochastic game \mathcal{G} , let $U \subseteq S$ induce a subgame $\mathcal{G}[U]$ with probabilistic transition function $\delta : U \times A \rightarrow \Delta(U)$. For $T \subseteq U$, let $f_T : 2^U \rightarrow 2^U$ be the operator such that for all $Z \subseteq U$,

$$f_T(Z) = T \cup \{s \in S_{\text{Max}} \cap U \mid \exists a \in A : \delta(s, a)(Z) > 0\} \\ \cup \{s \in S_{\text{Min}} \cap U \mid \forall a \in A : \delta(s, a)(Z) > 0\}.$$

Then $\text{Attr}_{\text{Max}}(T, \mathcal{G}[U])$ is the least fixed point of f_T , called the *positive attractor* for Max to T in $\mathcal{G}[U]$. It can be computed as the limit of the iteration $(f_T^i(\emptyset))_{i \in \mathbb{N}}$. There exists a positional strategy for Max (referred to as *positive-attractor strategy*) to ensure that from all states in $\text{Attr}_{\text{Max}}(T, \mathcal{G}[U])$, the set T is reached within $|U|$ steps with positive probability. We define $\text{Attr}_{\text{Min}}(T, \mathcal{G}[U])$ as the positive attractor for Min in an analogous way. An important property of positive attractors is that if X is a positive attractor for Max in $\mathcal{G}[U]$, then $\mathcal{G}[U \setminus X]$ is a subgame and it is a trap for Max. Analogous statement holds for Min.

3 Characterization of the Almost-Sure Winning Set

In this section we present the key lemmas that enable an inductive characterization of certificates and a polynomial-time verification procedure for the existence of almost-sure winning strategies, showing that the almost-sure winning problem is in NP for stochastic games with mean-payoff parity objectives.

It follows from the results of [14] that finite-memory strategies are not sufficient for Max and infinite-memory strategies are required for almost-sure winning. We present polynomial witnesses and polynomial-time verification procedure for the infinite-memory almost-sure winning strategies. The polynomial witnesses consists of

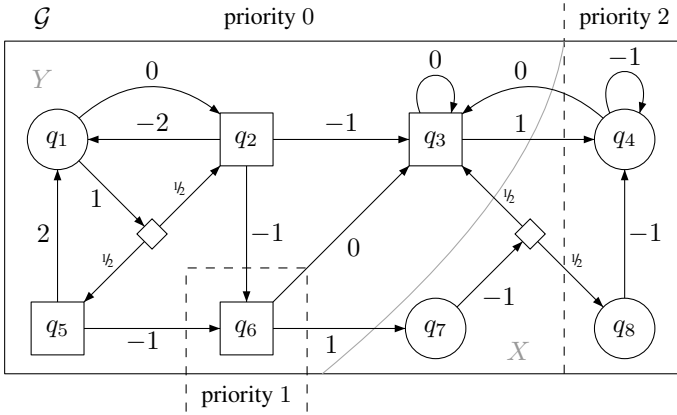


Fig. 1. Stochastic game \mathcal{G} with largest priority even

a trap U for player Min that defines a subgame where all states are almost-sure winning for player Max, together with a certificate defined as an inductive decomposition of the subgame induced by U constructed according to the parity of the largest priority d in U . If d is even we refer to the certificate as an *even certificate*, if d is odd as an *odd certificate*.

Intuitive description. To present the intuition of the (inductive) certificates, we informally explain some key properties in establishing that all states in a (sub)game are almost-sure winning for Max. In figures, we denote states of player Max by circles, and states of player Min by square boxes. Probability distributions over states are shown by a diamond. We omit actions and assume that every outgoing edge from player-Max and player-Min states corresponds to a different action. Let \mathcal{G} be a (sub)game with state space S where all states are almost-sure winning. Then, we describe a certificate according to the parity of the largest priority d in \mathcal{G} as follows.

1. If d is even (see Example 1 and Fig. 1), let $X = \text{Attr}_{\text{Max}}(S(d), \mathcal{G})$ and $Y = S \setminus X$. An even certificate for \mathcal{G} ensures that (1) in \mathcal{G} all states are almost-sure winning for the objective Mean; and (2) in $\mathcal{G}[Y]$ all states are almost-sure winning for Max for the objective $\text{Mean} \cap \text{Par}$ (using a certificate defined recursively in the subgame $\mathcal{G}[Y]$, which has at least one less priority as there is no priority- d state in Y). In other words, the even certificate consists of (i) a positional positive attractor strategy in X for the target $S(d)$; (ii) a positional almost-sure winning strategy in \mathcal{G} for the mean-payoff objective; and (iii) a certificate for $\mathcal{G}[Y]$. We establish that the above two conditions ensure that in \mathcal{G} all states are almost-sure winning for Max for the objective $\text{Mean} \cap \text{Par}$. An almost-sure winning strategy for Max is as follows: if the current state is in the subgame $\mathcal{G}[Y]$, then player Max ignores the history of the play up to the last state that was not in Y , and uses an almost-sure winning strategy in $\mathcal{G}[Y]$ (such a strategy exists in $\mathcal{G}[Y]$ by the certificate). If the opponent decides to visit the positive attractor X , then player Max switches to a (positional) positive-attractor strategy for at most $|S|$ steps. Then, either after $|S|$ steps or before (e.g., if

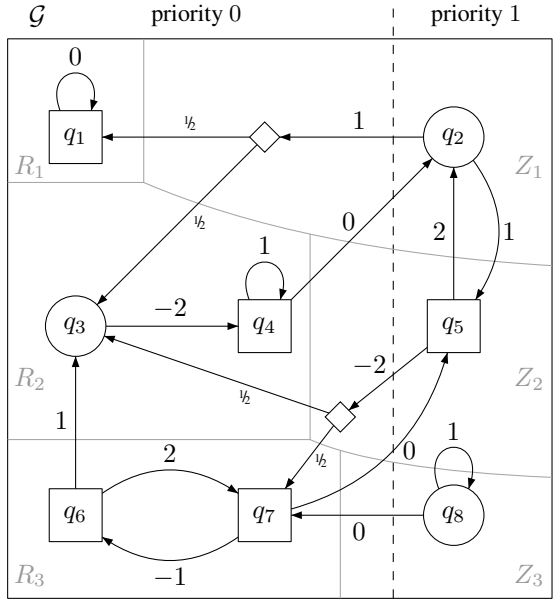


Fig. 2. Stochastic game \mathcal{G} with largest priority odd

a state with priority d is reached), player Max switches to an almost-sure winning strategy for Mean and plays it for a long finite time (that increases over the play). After that, the play might be in Y or in X , and player Max restarts from scratch the same process of playing. Intuitively, if the play keeps visiting X , then with probability 1 the positive-attractor strategy ensures infinitely many visits to a state with priority d (thus the parity condition is satisfied), and the almost-sure winning strategy for Mean played for increasing number of steps ensures that the mean-payoff objective is satisfied. On the other hand, if the play eventually stays in $\mathcal{G}[Y]$ forever, then the almost-sure winning strategy in $\mathcal{G}[Y]$ ensures the mean-payoff parity objective is satisfied with probability 1 (since the objective is independent of finite prefixes).

Example 1. Consider the stochastic game \mathcal{G} in Fig. 1 where the largest priority is 2. All states are almost-sure winning for the Mean objective, and a positional strategy for player Max is as follows: for state q_1 choose the edge labeled reward 1; and for state q_4 choose the edge to q_3 . The positive attractor for Max to the largest priority is $X = \{q_4, q_7, q_8\}$. In the subgame induced by $Y = \{q_1, q_2, q_3, q_5, q_6\}$ there is one less priority, and player Min can decide to leave the subgame in states q_3 and q_6 . An (odd) certificate defined in the subgame $\mathcal{G}[Y]$ witnesses that all states in $\mathcal{G}[Y]$ are almost-sure winning for the mean-payoff parity objective. Thus the even certificate consists of the positional strategy for Mean, the positive-attractor strategy, and a certificate for $\mathcal{G}[Y]$.

2. If d is odd (see Example 2 and Fig. 2), an odd certificate is a *layer-decomposition* of the state space of \mathcal{G} into non-empty sets R_1, \dots, R_k and Z_1, \dots, Z_k defined recursively as follows: (1) $R_1 \subseteq S \setminus S(d)$ is a trap for player Min in \mathcal{G} that contains no priority- d state, and such that all states in R_1 are almost-sure winning for Max for the objective $\text{Mean} \cap \text{Par}$ (using a certificate defined recursively in the subgame $\mathcal{G}[R_1]$, which has at least one less priority since priority d does not occur in R_1), (2) $Z_1 = \text{Attr}_{\text{Max}}(R_1, \mathcal{G})$ is the positive attractor for player Max to R_1 in \mathcal{G} , and (3) the sets R_2 and Z_2 are defined analogously in the subgame $\mathcal{G}[S \setminus Z_1]$, and the sets R_3 and Z_3 in the subgame $\mathcal{G}[S \setminus Z_2]$ where $Z_2 = \text{Attr}_{\text{Max}}(R_2, \mathcal{G}[S \setminus Z_1])$, and so on to obtain the layer-decomposition of \mathcal{G} . Such a decomposition must cover the state space, and thus the sets Z_1, \dots, Z_k form a partition of S (and $k \leq |S|$). An almost-sure winning strategy for player Max is as follows: if the current state is in a subgame R_i , then player Max ignores the history of the play up to the last state that was not in R_i , and uses an almost-sure winning strategy (that exists in R_i by the certificate). If the current state is in $Z_i \setminus R_i$, then player Max uses the positive-attractor strategy defined in Z_i . We show that almost-surely, one of the sets R_i is never left from some point on, and then the almost-sure winning strategy in $\mathcal{G}[R_i]$ ensures that the mean-payoff parity objective is satisfied with probability 1 (since the objective is independent of finite prefixes).

Example 2. Consider the stochastic game \mathcal{G} in Fig. 2 where the largest priority is 1. A layer-decomposition is shown where $R_1 = \{q_1\}$ is a trap of almost-sure winning states for Max, and $Z_1 = \{q_1, q_2\}$ is the positive attractor to R_1 . In the subgame $\mathcal{G}[S \setminus Z_1]$, there is no edge from q_4 to q_2 , and it follows that the states in $R_2 = \{q_3, q_4\}$ form a trap of almost-sure winning states in this subgame, and the positive attractor to R_2 is $Z_2 = R_2 \cup \{q_5\}$. The last layer consists of $R_3 = \{q_6, q_7\}$ and $Z_3 = R_3 \cup \{q_8\}$. As this layer-decomposition covers the state space of \mathcal{G} , it gives an odd certificate for player Max.

Given the basic intuitions, we now present the formal proofs. We start with a basic lemma, and then consider the two cases when the largest priority is even or odd.

Lemma 1. *Let \mathcal{G} be a stochastic mean-payoff game with state space S where all states are almost-sure winning for the mean-payoff objective Mean. Then there exists a positional strategy σ for player Max such that against all strategies τ for Min, for all $s \in S$ and for all $\epsilon > 0$, there exists k_ϵ such that for all $k \geq k_\epsilon$ we have $\mathbb{P}_s^{\sigma, \tau} \left(\left\{ s_0 s_1 \dots \in S^\omega \mid \sum_{i=0}^{k-1} \frac{1}{k} \cdot \text{rwd}(s_i, s_{i+1}) \geq -\epsilon \right\} \right) \geq 1 - \epsilon$.*

Lemma 2. *Let \mathcal{G} be a stochastic mean-payoff parity game with state space S and such that the largest priority d in \mathcal{G} is even. Let $X = \text{Attr}_{\text{Max}}(S(d), \mathcal{G})$ and $Y = S \setminus X$. All states in \mathcal{G} are almost-sure winning for player Max with the mean-payoff parity objective $\text{Mean} \cap \text{Par}$ if and only if:*

1. *all states in \mathcal{G} are almost-sure winning for the mean-payoff objective Mean for Max, and*
2. *all states in $\mathcal{G}[Y]$ are almost-sure winning for the mean-payoff parity objective $\text{Mean} \cap \text{Par}$ for Max.*

Proof. Let \mathcal{G} satisfy the conditions of the lemma. We first show that all states in \mathcal{G} are almost-sure winning for Max for the objective $\text{Mean} \cap \text{Par}$. Let σ_{Sub} be an almost-sure winning strategy for $\text{Mean} \cap \text{Par}$ in the subgame $\mathcal{G}[Y]$ induced by Y , let σ_{Attr} be a positional positive-attractor strategy to $S(d)$ in \mathcal{G} , and let σ_{Mean} be an almost-sure winning strategy for Mean in \mathcal{G} . Let $W = \max_{s, s' \in S} |\text{rwd}(s, s')|$ be the largest absolute reward and for every $j > 0$, let $\epsilon_j = \frac{1}{j}$ and let $K_j = \max \{k_{\epsilon_j}, j^2 \cdot W\}$ where k_{ϵ_j} is defined in Lemma 1.

The strategy σ that Max uses is played in rounds numbered $1, 2, \dots$, and at round i , the strategy σ is defined as follows:

Phase 1: (Mean-payoff phase). Let j be the length of the current play prefix until the end of phase 3 of round $i - 1$; then play according to the positional strategy σ_{Mean} for K_j steps. Switch to Phase 2.

Phase 2: (Subgame phase). While the current play ρ is in Y , let ρ' be the suffix of ρ obtained by ignoring the prefix of ρ up to the end of Phase 1 of the current round. Play $\sigma_{\text{Sub}}(\rho')$. If the play leaves Y (and thus reaches X), then switch to Phase 3.

Phase 3: (Attractor phase). Play σ_{Attr} for at most $|S|$ steps, or until a state with priority d is reached, or the positive attractor X is left. Switch to Phase 1 in round $i + 1$.

We show that σ is almost-sure winning for the $\text{Mean} \cap \text{Par}$ objective. Consider the following events:

$$\begin{aligned} A &= \{s_0 s_1 \dots \mid \exists J \geq 0 \cdot \forall j \geq J : s_j \in Y\}, \\ B &= \{s_0 s_1 \dots \mid \forall J \geq 0 \cdot \exists j \geq J : s_j \in X\}. \end{aligned}$$

Intuitively, A denotes that from some point on the play remains only in the subgame Y (and thus the strategy σ remains forever in the subgame phase), and B denotes that the set X (the positive attractor to priority d) is visited infinitely often. Let τ be a strategy for Min, then any play consistent with (σ, τ) belongs to $A \cup B$ and since $A \cap B = \emptyset$ we have $\mathbb{P}_s^{\sigma, \tau}(A \cup B) = \mathbb{P}_s^{\sigma, \tau}(A) + \mathbb{P}_s^{\sigma, \tau}(B) = 1$. We now consider two cases to establish that σ is almost-sure winning.

1. (*Under event A*). Observe that both parity and mean-payoff objectives are independent of finite prefixes, and if a play belongs to A , then the finite prefix of the play after which the play only visits states in Y does not change the mean-payoff nor the parity objective. Since σ_{Sub} is almost-sure winning in the subgame induced by Y , it follows that for all $s \in S$ and all strategies τ of player Min in \mathcal{G} we have $\mathbb{P}_s^{\sigma, \tau}(\text{Mean} \cap \text{Par} \mid A) = 1$ (if $\mathbb{P}_s^{\sigma, \tau}(A) \neq 0$).
2. (*Under event B*). We now reason under the event B and show that both the parity and the mean-payoff objectives are satisfied almost-surely. We first show that the parity objective is satisfied almost-surely. Consider an arbitrary strategy τ for player Min in \mathcal{G} and a state $s \in S$.

Parity objective almost-surely. Given the event B , the strategy is in attractor mode infinitely often. Given the strategy is in the attractor phase, the probability to reach a priority- d state within the next $|S|$ steps after the attractor mode starts is at least

$x = (p_{\min})^{|S|} > 0$, where p_{\min} is the minimum positive transition probability (i.e., $p_{\min} = \min \{\delta(s, a)(t) > 0 \mid s, t \in S, a \in A\}$). It follows that if the strategy is switching k times to the attractor phase, then the probability not to visit the priority- d set is at most $(1 - x)^k$. The event B ensures that the strategy is in the attractor phase infinitely often, and thus the probability that given the event B after some point a priority d state is not visited at all is $\lim_{k \rightarrow \infty} (1 - x)^k = 0$. Hence given event B , the best even priority d is visited infinitely often almost-surely, ensuring that the parity objective is satisfied, that is for all $s \in S$ and all strategies τ of player Min in \mathcal{G} we have $\mathbb{P}_s^{\sigma, \tau}(\text{Par} \mid B) = 1$ (if $\mathbb{P}_s^{\sigma, \tau}(B) \neq 0$).

In other words, given that the positive attractor to a set T is visited infinitely often, it follows that the set T is visited infinitely often with probability 1, and we refer to this property as the *almost-sure positive attractor property*.

Mean-payoff objective almost-surely. We now prove that the mean-payoff objective is almost-surely satisfied. Given the event B , the strategy σ is in the mean-payoff phase infinitely often. Consider the finite prefixes of play $\rho = s_0 \cdots s_{j+1}$ consistent with (σ, τ) that are in the mean-payoff phase for the first time in the current round. Then by the definition of the strategy σ , every play prefix $\rho' = \rho \cdot s_{j+1} \cdots s_{j+i}$ consistent with (σ, τ) that extends ρ , for all $0 < i \leq K_j$, is in the mean-payoff phase. The sum of the rewards for all prefixes of length j is at least $-j \cdot W$ and then applying Lemma 1 we have

$$\mathbb{P}_s^{\sigma, \tau} \left(\left\{ s_0 s_1 \cdots \mid \frac{1}{j + K_j} \cdot \sum_{i=0}^{j+K_j} \text{rwd}(s_i, s_{i+1}) \geq -\frac{\epsilon_j \cdot K_j + j \cdot W}{j + K_j} \mid \text{Cone}(\rho) \right\} \right) \geq 1 - \epsilon_j$$

By the choice of K_j (that $K_j \geq j^2 \cdot W$) and $\epsilon_j = \frac{1}{j}$, we have $-\frac{\epsilon_j \cdot K_j + j \cdot W}{j + K_j} \geq -\frac{\epsilon_j \cdot K_j}{K_j} - \frac{j \cdot W}{j^2 \cdot W} \geq -\frac{2}{j}$. Consider the function f that given a number ℓ returns the maximum number j such that $j + K_j \leq \ell$. Note that f is a non-decreasing function and as ℓ tends to ∞ , also $f(\ell)$ tends to ∞ . Given the event B , there are infinitely many prefixes ρ consistent with (σ, τ) that are in the mean-payoff phase for the first time in the current round. Hence we have

$$\limsup_{\ell \rightarrow \infty} \mathbb{P}_s^{\sigma, \tau} \left(\left\{ s_0 s_1 \cdots \mid \frac{1}{\ell} \cdot \sum_{i=0}^{\ell} \text{rwd}(s_i, s_{i+1}) \geq -\frac{2}{f(\ell)} \mid B \right\} \right) \geq \limsup_{\ell \rightarrow \infty} 1 - \frac{1}{f(\ell)} = 1.$$

By Fatou's lemma [2] we know that for an event sequence \mathcal{E}_ℓ we have that $\limsup_{\ell \rightarrow \infty} \mathbb{P}(\mathcal{E}_\ell) \leq \mathbb{P}(\limsup_{\ell \rightarrow \infty} \mathcal{E}_\ell)$. Hence an application of the Fatou's lemma gives us that

$$\mathbb{P}_s^{\sigma, \tau} \left(\limsup_{\ell \rightarrow \infty} \left\{ s_0 s_1 \cdots \mid \frac{1}{\ell} \cdot \sum_{i=0}^{\ell} \text{rwd}(s_i, s_{i+1}) \geq -\frac{2}{f(\ell)} \mid B \right\} \right) = 1.$$

Let $\varphi_\ell = \left\{ s_0 s_1 \cdots \mid \frac{1}{\ell} \cdot \sum_{i=0}^{\ell} \text{rwd}(s_i, s_{i+1}) \geq -\frac{2}{f(\ell)} \right\}$ and $\varphi = \limsup_{\ell \rightarrow \infty} \varphi_\ell$. Consider a play $\rho = s_0 s_1 \cdots \in \varphi$. Fix $\epsilon > 0$, and consider ℓ_0 such that $\frac{2}{f(\ell_0)} \leq \epsilon$. Since $\rho \in \varphi$, there exists infinitely many $\ell \geq \ell_0$ such that $\rho \in \varphi_\ell$, and hence for infinitely many ℓ we have $\frac{1}{\ell} \cdot \sum_{i=1}^{\ell-1} \text{rwd}(s_i, s_{i+1}) \geq -\epsilon$. Hence

$\limsup_{\ell \rightarrow \infty} \frac{1}{\ell} \cdot \sum_{i=1}^{\ell-1} \text{rwd}(s_i, s_{i+1}) \geq -\epsilon$. Since this holds for all $\epsilon > 0$, it follows that $\limsup_{\ell \rightarrow \infty} \frac{1}{\ell} \cdot \sum_{i=1}^{\ell-1} \text{rwd}(s_i, s_{i+1}) \geq 0$. In other words, we have $\varphi \subseteq \text{Mean}$ and hence for all $s \in S$ and all strategies τ of player Min in \mathcal{G} we have $\mathbb{P}_s^{\sigma, \tau}(\text{Mean} \mid B) = 1$ (if $\mathbb{P}_s^{\sigma, \tau}(B) \neq 0$).

Thus given either event A or B , the mean-payoff parity objective is satisfied almost-surely. Note that if one of the event has probability 0, then the other has probability 1. It follows that the mean-payoff parity objective is satisfied almost-surely. This concludes one direction of the proof that if the conditions of the lemma are satisfied, then almost-sure winning for $\text{Mean} \cap \text{Par}$ is ensured with probability 1.

We now prove the converse. Consider a game \mathcal{G} such that all states in its state space S are almost-sure winning for the objective $\text{Mean} \cap \text{Par}$ for player Max. First, observe that since $\text{Mean} \cap \text{Par} \subseteq \text{Mean}$, almost-sure winning for $\text{Mean} \cap \text{Par}$ implies almost-sure winning for Mean . This implies the first condition. Second, observe that Y is a trap for player Max. If player Max does not have an almost-sure winning strategy for a non-empty set $Z \subseteq Y$ in the subgame $\mathcal{G}[Y]$, then player Max does not have an almost-sure winning strategy from Z in \mathcal{G} , which contradicts that all states in \mathcal{G} are almost-sure winning. This proves the second condition of the lemma and completes the proof. \square

Lemma 3. *Let \mathcal{G} be a stochastic mean-payoff parity game with state space S , and such that the largest priority d in \mathcal{G} is odd. All states in \mathcal{G} are almost-sure winning for the objective $\text{Mean} \cap \text{Par}$ if and only if there exists a partition $\{Z_i\}_{1 \leq i \leq k}$ of S and non-empty sets R_i, U_i for $i = 1, \dots, k$, and U_{k+1} such that $U_1 = S$ and for all $1 \leq i \leq k$: (1) $R_i \subseteq U_i \setminus U_i(d)$ is a trap for Min in $\mathcal{G}[U_i]$, and all states in R_i are almost-sure winning for the objective $\text{Mean} \cap \text{Par}$ in $\mathcal{G}[U_i]$; (2) $Z_i = \text{Attr}_{\text{Max}}(R_i, \mathcal{G}[U_i])$; and (3) $U_{i+1} = U_i \setminus Z_i$.*

Lemma 3 presents a characterization of the certificate for almost-sure winning when the largest priority is odd. The key correctness argument uses the almost-sure positive attractor property to show that the event that from some point on only states in R_i are visited for some i has probability 1. From the above fact and the almost-sure winning strategies in R_i we obtain an almost-sure winning strategy in \mathcal{G} .

We remark that it follows from our proofs that the infinite-memory required by the strategies can be captured in terms of counter-based strategies that keep track of the number of steps that certain positional strategies need to be played.

4 Algorithm

In this section we present an algorithm for the almost-sure winning problem. Let \mathcal{G} be a stochastic mean-payoff parity game with largest priority d . Our algorithm computes the set R of almost-sure winning states for Max, by iterations that, from the state space S of \mathcal{G} remove positive winning states of player Min. When a fixpoint is obtained, we show that it satisfies the characterization of Lemma 2 and Lemma 3, hence it is the almost-sure winning set. Starting with $R = S$, the algorithm considers two cases:

- (a) If d is even: First, compute the almost-sure winning region U for the Mean objective in $\mathcal{G}[R]$. Compute the positive attractor X for player Max to the set of states with

- priority d in U , and let Y be the complement. Recursively compute the almost-sure winning region R' in $\mathcal{G}[Y]$ for the mean-payoff parity objective, and iterate (until $R' = Y$) in the subgame induced by the complement $U \setminus Z$ of the player-Min positive attractor $Z = \text{Attr}_{\text{Min}}(Y \setminus R', \mathcal{G}[U])$ (i.e., removing some positive winning states for player Min).
- (b) If d is odd: In each iteration of the main loop, the algorithm computes a set of positive winning states for player Min as the positive attractor (for Min) to the set U computed in the inner loop. The inner loop computes in R' the almost-sure winning states of player Max in the subgame induced by the complement Y of player-Min positive attractor to priority d , using a recursive call. The positive attractor for Max to R' is removed, and the next iteration starts (if $R' \neq \emptyset$) with a strictly smaller state space U . The main loop terminates when there is nothing to remove ($U = \emptyset$).

Correctness and termination. The correctness and termination of our algorithm (which we refer to as AlgStMPP, algorithm for stochastic mean-payoff parity games) is established using an argument by induction on the depth of the recursive calls, which are always invoked with games that have at least one less priority than the current game, and using Lemma 2 and Lemma 3.

The complexity of AlgStMPP is exponential in the number of priorities in the game, like the basic algorithm for parity games [28]. The key differences to the basic algorithm for parity games are as follows: (i) in our algorithm there is an extra nested loop when the maximum priority is odd; and (ii) in addition to the basic attractor computation for parity games we also need to compute the almost-sure winning set for stochastic mean-payoff games.

Theorem 1. *Given a stochastic mean-payoff parity game \mathcal{G} with n states, probabilistic transition function δ , priorities in $\{0, 1, \dots, d - 1\}$, and largest absolute reward W , AlgStMPP computes the almost-sure winning region of \mathcal{G} in time $O(d \cdot n^{2d} \cdot \text{MeanGame}(n, |\delta|, W))$ where $\text{MeanGame}(n, |\delta|, W)$ is the time complexity of solving the almost-sure winning problem for stochastic games with only a mean-payoff objective.*

Note that $\text{MeanGame}(n, |\delta|, W) \in |A|^n \cdot \text{Poly}(n, |\delta|, W)$ by simply enumerating over all positional strategies and then solving in polynomial time the MDP obtained by fixing the positional strategy.

5 Computational Complexity

In this section we establish the $\text{NP} \cap \text{coNP}$ complexity bound for the almost-sure winning problem.

The NP Membership. Although infinite-memory strategies are necessary for player Max to win mean-payoff parity games almost surely [14], we show that the almost-sure winning problem can be solved in NP by guessing a polynomial-size decomposition of the state space along with positional strategies that allow to construct an

almost-sure winning strategy, possibly with infinite memory. The polynomial certificate is obtained from the characterization of Lemma 2 and Lemma 3; and the verification procedure requires solving MDPs with mean-payoff parity objectives, which can be done in polynomial time [11].

Lemma 4. *The almost-sure winning problem for stochastic mean-payoff parity games is in NP.*

The coNP Membership. We show that positional strategies are sufficient for player Min to win positively in stochastic mean-payoff parity games. Using the fact that AlgStMPP maintains in variable R an over-approximation of the almost-sure winning set for player Max, we construct a positional strategy for player Min from all states that are removed from R by the algorithm.

Lemma 5. *To win positively in stochastic mean-payoff parity games, positional strategies are sufficient for player Min.*

We then show how to use the positional strategy for positive winning to obtain a positional strategy for almost-sure winning for player Min. By Remark 1 it follows that positional optimal strategies exist for player Min. Lemma 4, the existence of positional optimal strategies for player Min, and the fact that MDPs with mean-payoff parity objectives can be solved in polynomial time [11], gives us the following result.

Theorem 2. *The following assertions hold: (1) Positional optimal strategies exist for player Min in stochastic mean-payoff parity games (2) The almost-sure winning and the value-strategy problem for stochastic mean-payoff parity games can be decided in $\text{NP} \cap \text{coNP}$.*

Remark 2. The complexity result of Theorem 2 matches the best known complexity for stochastic mean-payoff games [22], stochastic parity games [15] (also see [1] for relationship of stochastic mean-payoff and stochastic parity games), and (non-stochastic) mean-payoff parity games [12].

Concluding Remarks. In this work we studied the computational and strategy complexity of the value-strategy problem for $2\frac{1}{2}$ -player mean-payoff parity games. In addition we presented an algorithm for computing the almost-sure winning states which requires the computation of the almost-sure winning states for $2\frac{1}{2}$ -player mean-payoff games. Improved algorithmic solutions for the computation of the almost-sure winning states in $2\frac{1}{2}$ -player mean-payoff games is an interesting question. Our algorithm for almost-sure winning and the general technique mentioned in Remark 1 for $2\frac{1}{2}$ -player games with tail objectives provide an exponential-time algorithm for the value-strategy problem. Whether more specialized algorithms (such as strategy-iteration algorithms) can be developed for the value-strategy problem in $2\frac{1}{2}$ -player mean-payoff parity games is another interesting algorithmic question.

References

1. Andersson, D., Miltersen, P.B.: The complexity of solving stochastic games on graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 112–121. Springer, Heidelberg (2009)
2. Billingsley, P. (ed.): Probability and Measure. Wiley-Interscience (1995)
3. Bloem, R., Chatterjee, K., Henzinger, T.A., 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)
4. Boros, E., Elbassioni, K., Fouz, M., Gurvich, V., Makino, K., Manthey, B.: Stochastic mean payoff games: Smoothed analysis and approximation schemes. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 147–158. Springer, Heidelberg (2011)
5. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)
6. Bouyer, P., Markey, N., Olschewski, J., Ummels, M.: Measuring permissiveness in parity games: Mean-payoff parity games revisited. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 135–149. Springer, Heidelberg (2011)
7. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. *Transactions of the AMS* 138, 295–311 (1969)
8. Černý, P., Chatterjee, K., Henzinger, T.A., Radhakrishna, A., Singh, R.: Quantitative synthesis for concurrent programs. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 243–259. Springer, Heidelberg (2011)
9. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
10. Chatterjee, K., Doyen, L.: Energy parity games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 599–610. Springer, Heidelberg (2010)
11. Chatterjee, K., Doyen, L.: Energy and mean-payoff parity Markov decision processes. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 206–218. Springer, Heidelberg (2011)
12. Chatterjee, K., Doyen, L.: Energy parity games. *TCS* 458(2), 49–60 (2012)
13. Chatterjee, K., Henzinger, T.A., Horn, F.: Stochastic games with finitary objectives. In: Královíč, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 34–54. Springer, Heidelberg (2009)
14. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Mean-payoff parity games. In: Proc. of LICS, pp. 178–187. IEEE Computer Society (2005)
15. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: Proc. of SODA, pp. 121–130. SIAM (2004)
16. Condon, A.: The complexity of stochastic games. *Inf. Comput.* 96(2), 203–224 (1992)
17. Filar, J.A., Vrieze, K.: *Competitive Markov Decision Processes*. Springer (1997)
18. Gimbert, H., Horn, F.: Solving simple stochastic tail games. In: Proc. of SODA, pp. 847–862. SIAM (2010)
19. Gimbert, H., Oualhadj, Y., Paul, S.: Computing optimal strategies for Markov decision processes with parity and positive-average conditions. Technical report, LaBRI, Université de Bordeaux II (2011)
20. Gurvich, V., Karzanov, A., Khachivan, L.: Cyclic games and an algorithm to find min-max cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics* 28(5), 85–91 (1988)

21. Jurdziński, M.: Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters* 68(3), 119–124 (1998)
22. Liggett, T.A., Lippman, S.A.: Stochastic games with perfect information and time average payoff. *Siam Review* 11, 604–607 (1969)
23. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: *Proc. of POPL*, pp. 179–190. ACM Press (1989)
24. Puterman, M.L.: *Markov Decision Processes*. John Wiley and Sons (1994)
25. Raghavan, T.E.S., Filar, J.A.: Algorithms for stochastic games — a survey. *ZOR — Methods and Models of Operations Research* 35, 437–472 (1991)
26. 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)
27. Thomas, W.: Languages, automata, and logic. In: *Handbook of Formal Languages, Beyond Words*, vol. 3, ch. 7, pp. 389–455. Springer (1997)
28. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science* 200(1-2), 135–183 (1998)
29. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. *TCS* 158(1&2), 343–359 (1996)

Latticed-LTL Synthesis in the Presence of Noisy Inputs

Shaull Almagor and Orna Kupferman

The Hebrew University, Jerusalem, Israel

Abstract. In the classical synthesis problem, we are given a linear temporal logic (LTL) formula ψ over sets of input and output signals, and we synthesize a finite-state transducer that realizes ψ : with every sequence of input signals, the transducer associates a sequence of output signals so that the generated computation satisfies ψ . In recent years, researchers consider extensions of the classical Boolean setting to a multi-valued one. We study a setting in which the truth values of the input and output signals are taken from a finite lattice, and the specification formalism is Latticed-LTL (LLTL), where conjunctions and disjunctions correspond to the meet and join operators of the lattice, respectively. The lattice setting arises in practice, for example in specifications involving priorities or in systems with inconsistent viewpoints.

We solve the LLTL synthesis problem, where the goal is to synthesize a transducer that realizes ψ in desired truth values.

For the classical synthesis problem, researchers have studied a setting with incomplete information, where the truth values of some of the input signals are hidden and the transducer should nevertheless realize ψ . For the multi-valued setting, we introduce and study a new type of incomplete information, where the truth values of some of the input signals may be noisy, and the transducer should still realize ψ in a desired value. We study the problem of noisy LLTL synthesis, as well as the theoretical aspects of the setting, like the amount of noise a transducer may tolerate, or the effect of perturbing input signals on the satisfaction value of a specification.

1 Introduction

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it adheres to its specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [10]. The modern approach to synthesis was initiated by Pnueli and Rosner, who introduced LTL (linear temporal logic) synthesis [24]: We are given an LTL formula ψ over sets I and O of input and output signals, and we synthesize a finite-state system that *realizes* ψ . At each moment in time, the system reads a truth assignment, generated by the environment, to the signals in I , and it generates a truth assignment to the signals in O . Thus, with every sequence of inputs, the transducer associates a sequence of outputs, and it *realizes* ψ if all the computations that are generated by the interaction satisfy ψ . Synthesis has attracted a lot of research and interest [28].

In recent years, researchers have considered extensions of the classical Boolean setting to a multi-valued one, where the atomic propositions are multi-valued, and so is

the satisfaction value of specifications. The multi-valued setting arises directly in systems in which the designer can give to the atomic propositions rich values, expressing, for example, energy consumption, waiting time, or different levels of confidence [5,1], and arises indirectly in probabilistic settings, systems with multiple and inconsistent view-points, specifications with priorities, and more [20,14,2]. Adjusting the synthesis problem to this setting, one works with multi-valued specification formalisms. In such formalisms, a specification ψ maps computations in which the atomic propositions take values from a domain D to a satisfaction value in D . For example, ψ may map a computation in $(\{0, 1, 2, 3\}^{\{p\}})^\omega$ to the maximal value assigned to the (multi-valued) atomic proposition p during the computation. Accordingly, the synthesis problem in the multi-valued setting gets as input a specification ψ and a predicate $P \subseteq D$ of desired values, and seeks a system that reads assignments in D^I , responds with assignments in D^O , and generates only computations whose satisfaction value is in P . The synthesis problem has been solved for several multi-valued settings [4,1].

A different extension of the classical setting of synthesis considers settings in which the system has *incomplete information* about its environment. Early work on incomplete information considers settings in which the system can read only a subset of the signals in I and should still generate only computations that satisfy the specification, which refers to all the signals in $I \cup O$ [18,6,7]. The setting is equivalent to a game with incomplete information, extensively studied in [25]. As shown there, the common practice in handling incomplete information is to move to an exponentially-larger game of complete information, where each state corresponds to a set of states that are indistinguishable by a player with incomplete information in the original game.

More recent work on synthesis with incomplete information studies richer types of incomplete information. In [8], the authors study a setting in which the transducer can read some of the input signals some of the time. In more detail, *sensing* the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget. In [30], the authors study games with *errors*. Such games correspond to a synthesis setting in which there are positions during the interaction in which input signals are read by the system with an error. The games are characterized by the number or rate of errors that the system has to cope with, and by the ability of the system to detect whether a current input is erred.

In this work we introduce and study a different model of incomplete information in the multi-valued setting. In our model, the system always reads all input signals, but their value may be perturbed according to a known noise function. This setting naturally models incomplete information in real-life multi-valued settings. For example, when the input is read by sensors that are not accurate (e.g., due to bounded precision, or to probabilistic measuring) or when the input is received over a noisy channel and may come with some distortion. The multi-valued setting we consider is that of *finite lattices*. A lattice is a partially-ordered set $\mathcal{L} = \langle A, \leq \rangle$ in which every two elements ℓ and ℓ' have a least upper bound (ℓ *join* ℓ' , denoted $\ell \vee \ell'$) and a greatest lower bound (ℓ *meet* ℓ' , denoted $\ell \wedge \ell'$). Of special interest are two classes of lattices: (1) Fully ordered, where $\mathcal{L} = \langle \{1, \dots, n\}, \leq \rangle$, for an integer $n \geq 1$ and the usual “less than or equal” order. In this lattice, the operators \vee and \wedge correspond to \max and \min , respectively.

(2) Power-set lattices, where $\mathcal{L} = \langle 2^X, \subseteq \rangle$, for a finite set X , and the containment (partial) order. In this lattice, the operators \vee and \wedge correspond to \cup and \cap , respectively.

The lattice setting is a good starting point to the multi-valued setting. While their finiteness circumvents the infinite-state space of dense multi-values, lattices are sufficiently rich to capture many quantitative settings. Fully-ordered lattices are sometimes useful as is (for example, when modeling priorities [2]), and sometimes thanks to the fact that real values can often be abstracted to finitely many linearly ordered classes. The power-set lattice models a wide range of partially-ordered values. For example, in a setting with inconsistent viewpoints, we have a set X of agents, each with a different viewpoint of the system, and the truth value of a signal or a formula indicates the set of agents according to whose viewpoint the signal or the formula are true. As another example, in a peer-to-peer network, one can refer to the different attributes of the communication channels by assigning with them subsets of attributes. From a technical point of view, the fact that lattices are partially ordered poses challenges that do not exist in (finite and infinite) full orders. For example, as we are going to see, the fact that a specification is realizable with value ℓ and with value ℓ' does not imply it is realizable with value $\ell \vee \ell'$, which trivially holds for full orders.

We start by defining lattices and the logic *Latticed LTL* (LLTL, for short). We then study theoretical properties of LLTL: We study cases where the set of attainable truth values of an LLTL formula are closed under \vee , thus a maximal attainable value exists, even when the lattice elements are partially ordered. We also study stability properties, namely the affect of perturbing the values of the atomic propositions on the satisfaction value of formulas. We continue to the synthesis and the noisy-synthesis problems for LLTL, which we solve via a translation of LLTL formulas to Boolean automata. We show that by working with universal automata, we can handle the exponential blow-up that incomplete information involves together with the exponential blow-up that determination (or alternation removal, if we take a Safraless approach) involves, thus the noisy-synthesis problem stays 2EXPTIME-complete, as it is for LTL.

Due to lack of space, some of the proofs are omitted and can be found in the full version, in the authors' home pages.

2 Preliminaries

2.1 Lattices

Consider a set A , a partial order \leq on A , and a subset P of A . An element $\ell \in A$ is an *upper bound* on P if $\ell \geq \ell'$ for all $\ell' \in P$. Dually, ℓ is a *lower bound* on P if $\ell \leq \ell'$ for all $\ell' \in P$. The pair $\langle A, \leq \rangle$ is a *lattice* if for every two elements $\ell, \ell' \in A$, both the least upper bound and the greatest lower bound of $\{\ell, \ell'\}$ exist, in which case they are denoted $\ell \vee \ell'$ (ℓ join ℓ') and $\ell \wedge \ell'$ (ℓ meet ℓ'), respectively. We use $\ell < \ell'$ to indicate that $\ell \leq \ell'$ and $\ell \neq \ell'$. We say that ℓ is a *child* of ℓ' , denoted $\ell \prec \ell'$, if $\ell < \ell'$ and there is no ℓ'' such that $\ell < \ell'' < \ell'$.

A lattice $\mathcal{L} = \langle A, \leq \rangle$ is *finite* if A is finite. Note that finite lattices are *complete*: every subset of A has a least-upper and a greatest-lower bound. We use \top (*top*) and \perp (*bottom*) to denote the least-upper and greatest-lower bounds of A , respectively. A lattice is *distributive* if for every $\ell_1, \ell_2, \ell_3 \in A$, we have $\ell_1 \wedge (\ell_2 \vee \ell_3) = (\ell_1 \wedge \ell_2) \vee (\ell_1 \wedge \ell_3)$.

ℓ_3) and $\ell_1 \vee (\ell_2 \wedge \ell_3) = (\ell_1 \vee \ell_2) \wedge (\ell_1 \vee \ell_3)$. The traditional disjunction and conjunction logic operators correspond to the join and meet lattice operators. In a general lattice, however, there is no natural counterpart to negation. A *De-Morgan* (or *quasi-Boolean*) lattice is a lattice in which every element a has a unique complement element $\neg \ell$ such that $\neg \neg \ell = \ell$, De-Morgan rules hold, and $\ell \leq \ell'$ implies $\neg \ell' \leq \neg \ell$. In the rest of this paper we consider only finite distributive De-Morgan lattices. We focus on two classes of such lattices: (1) Fully ordered, where $\mathcal{L} = \langle \{1, \dots, n\}, \leq \rangle$, for an integer $n \geq 1$ and the usual “less than or equal” order. Note that in this lattice, the operators \vee and \wedge correspond to \max and \min , respectively, and $\neg i = n - i + 1$. (2) Power-set lattices, where $\mathcal{L} = \langle 2^X, \subseteq \rangle$, for a finite set X , and the containment (partial) order. Note that in this lattice, the operators \vee and \wedge correspond to \cup and \cap , respectively, and negation corresponds to complementation.

Consider a lattice $\mathcal{L} = \langle A, \leq \rangle$. A *join irreducible* element in \mathcal{L} is $l \in A$ such that $l \neq \perp$ and for all elements $l_1, l_2 \in A$, if $l_1 \vee l_2 \geq l$, then $l_1 \geq l$ or $l_2 \geq l$. For example, the join irreducible elements in $\langle 2^X, \subseteq \rangle$ are all singletons $\{x\}$, for $x \in X$. By *Birkhoff’s representation theorem* for finite distributive lattices, in order to prove that $l_1 = l_2$, it is sufficient to prove that for every join irreducible element l it holds that $l_1 \geq l$ iff $l_2 \geq l$. We denote the set of join irreducible elements of \mathcal{L} by $\text{JI}(\mathcal{L})$. For convenience, we often talk about a lattice \mathcal{L} without specifying A and \leq . We then abuse notations and refer to \mathcal{L} as a set of elements and talk about $l \in \mathcal{L}$ or about assignments in \mathcal{L}^{AP} (rather than $l \in A$ or assignments in A^{AP}).

2.2 The Logic LLTL

The logic LLTL is a natural generalization of LTL to a multi-valued setting, where the atomic propositions take values from a lattice \mathcal{L} [9,16]. Given a (finite distributive De-Morgan) lattice \mathcal{L} , the syntax of LLTL is given by the following grammar, where p ranges over a set AP of atomic propositions, and ℓ ranges over \mathcal{L} .

$$\varphi := \ell \mid p \mid \neg \varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi \text{U} \psi.$$

The semantics of LLTL is defined with respect to a *computation* $\pi = \pi_0, \pi_1, \dots \in (\mathcal{L}^{AP})^\omega$. Thus, in each moment in time the atomic propositions get values from \mathcal{L} . Note that classical LTL coincides with LLTL defined with respect to the two-element fully-ordered lattice. For a position $i \geq 0$, we use π^i to denote the suffix π_i, π_{i+1}, \dots of π . Given a computation π and an LLTL formula φ , the *satisfaction value* of φ in π , denoted $\llbracket \pi, \varphi \rrbracket$, is defined by induction on the structure of φ as follows (the operators on the right-hand side are the join, meet, and complementation operators of \mathcal{L}).¹

- $\llbracket \pi, \ell \rrbracket = \ell$.
- $\llbracket \pi, p \rrbracket = \pi_0(p)$.
- $\llbracket \pi, \neg \varphi \rrbracket = \neg \llbracket \pi, \varphi \rrbracket$.
- $\llbracket \pi, \varphi \vee \psi \rrbracket = \llbracket \pi, \varphi \rrbracket \vee \llbracket \pi, \psi \rrbracket$.
- $\llbracket \pi, X\varphi \rrbracket = \llbracket \pi^1, \varphi \rrbracket$.
- $\llbracket \pi, \varphi \text{U} \psi \rrbracket = \bigvee_{i \geq 0} (\llbracket \pi^i, \psi \rrbracket \wedge \bigwedge_{0 \leq j < i} \llbracket \pi^j, \varphi \rrbracket)$.

Example 1. Consider a setting in which three agents a, b , and c have different viewpoints on a system S . A truth assignment for the atomic propositions is then a function

¹ Unlike LTL, where the constants **True** and **False** do not increase the expressive power, in LLTL the constants $\ell \in \mathcal{L}$ do increase the expressive power.

in $(2^{\{a,b,c\}})^{AP}$ assigning to each $p \in AP$ the set of agents according to whose view-point p is true. We reason about \mathcal{S} using the lattice $\mathcal{L} = \langle 2^{\{a,b,c\}}, \subseteq \rangle$. For example, the truth value of the formula $\psi = G(req \rightarrow Fgrant)$ in a computation is the set of agents according to whose view-point, whenever a request is sent, it is eventually granted.

2.3 LLTL Synthesis

Consider a lattice \mathcal{L} and finite disjoint sets I and O of input and output signals that take values in \mathcal{L} . An (I/O) -transducer over \mathcal{L} models an interaction between an environment that generates in each moment in time an input in \mathcal{L}^I and a system that responds with outputs in \mathcal{L}^O . Formally, an (I/O) -transducer over \mathcal{L} (transducer, when I , O , and \mathcal{L} are clear from the context) is a tuple $\mathcal{T} = \langle \mathcal{L}, I, O, S, s_0, \eta, \tau \rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\eta : S \times \mathcal{L}^I \rightarrow S$ is a deterministic transition function, and $\tau : S \rightarrow \mathcal{L}^O$ is a labeling function. We extend η to words in $(\mathcal{L}^I)^*$ in the straightforward way. Thus, $\eta : (\mathcal{L}^I)^* \rightarrow S$ is such that $\eta(\epsilon) = s_0$, and for $x \in (\mathcal{L}^I)^*$ and $i \in \mathcal{L}^I$, we have $\eta(x \cdot i) = \eta(\eta(x), i)$. Each transducer \mathcal{T} induces a strategy $f_{\mathcal{T}} : (\mathcal{L}^I)^* \rightarrow \mathcal{L}^O$ where for all $w \in (\mathcal{L}^I)^*$, we have $f_{\mathcal{T}}(w) = \tau(\eta(w))$. Thus, $f_{\mathcal{T}}(w)$ is the letter that \mathcal{T} outputs after reading the sequence w of input letters. Given a sequence $i_0, i_1, i_2, \dots \in (\mathcal{L}^I)^\omega$ of input assignments, the transducer generates the computation $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \dots \in (\mathcal{L}^{I \cup O})^\omega$, where for all $j \geq 1$, we have $o_j = f_{\mathcal{T}}(i_0 \cdots i_{j-1})$.

Consider a lattice \mathcal{L} , an LLTL formula φ over the atomic propositions $I \cup O$, and a predicate $P \subseteq \mathcal{L}$. We say that a transducer \mathcal{T} realizes $\langle \varphi, P \rangle$ if for every computation ρ of \mathcal{T} , it holds that $\llbracket \rho, \varphi \rrbracket \in P$. The realizability problem for LLTL is to determine, given φ and P , whether there exists a transducer that realizes $\langle \varphi, P \rangle$. We then say that φ is (I/O) -realizable with values in P . The synthesis problem is then to generate such a transducer. Of special interest are predicates P that are upward closed. Thus, P is such that for all $\ell \in \mathcal{L}$, if $\ell \in P$ then $\ell' \in P$ for all $\ell' \geq \ell$.

2.4 Noisy Synthesis

Consider an LLTL formula φ over atomic proposition $I \cup O$ and a predicate P . In noisy synthesis, we consider the synthesis problem in a setting in which the inputs are read with some perturbation and the goal is to synthesize a transducer that nevertheless realizes $\langle \varphi, P \rangle$.

In order to formalize the above intuition, we first formalize the notion of noise. Consider a lattice $\mathcal{L} = \langle A, \leq \rangle$ and two elements $\ell_1, \ell_2 \in \mathcal{L}$. We define the distance between ℓ_1 and ℓ_2 , denoted $d(\ell_1, \ell_2)$, as the shortest path from ℓ_1 to ℓ_2 in the undirected graph $\langle A, E_{\prec} \rangle$ in which $E_{\prec}(v, v')$ iff $v \prec v'$ or $v' \prec v$. For example, in the fully-ordered lattice \mathcal{L} , we have $d(i, j) = |i - j|$, and in the power-set lattice, the distance coincides with the Hamming distance, thus $d(X_1, X_2) = |(X_1 \setminus X_2) \cup (X_2 \setminus X_1)|$. For two assignments $f, f' \in \mathcal{L}^{AP}$, we define $d(f, f') = \max_{p \in AP} d(f(p), f'(p))$.

We assume we are given a noise function $\nu : \mathcal{L}^I \rightarrow 2^{\mathcal{L}^I}$, describing the possible perturbations of each input. That is, for every $i \in \mathcal{L}^I$ the set $\nu(i)$ consists of the inputs that may have been actually generated by the environment, when the system reads i . A natural noise function is $\nu(i) = \{j : d(i, j) \leq \gamma\}$, for some constant γ , which is the

γ -units ball around i . Given a noise function ν and two computations $\pi, \pi' \in (\mathcal{L}^{I \cup O})^\omega$, we say that π' is ν -*indistinguishable* from π if for every $i \geq 0$, we have that $\pi'_i|_I \in \nu(\pi_i|_I)$ and $\pi'_i|_O = \pi_i|_O$, where $\sigma|_I$ is the restriction of $\sigma \in \mathcal{L}^{I \cup O}$ to inputs in I , and similarly for $\sigma|_O$ and O . Thus, π' is obtained from π by changing only the assignment to input signals, within ν . Note that ν need not be a symmetric function, nor is the definition of ν -indistinguishability. We say that a transducer \mathcal{T} *realizes* $\langle \varphi, P \rangle$ with noise ν if for every computation π of \mathcal{T} , we have that $\llbracket \pi', \varphi \rrbracket \in P$ for all computations π' that are ν -indistinguishable from π . Thus, the reaction of \mathcal{T} on every input sequence satisfies φ in a desired satisfaction value even if the input sequence is read with noise ν .

2.5 Automata and Games

An *automaton over infinite words* is $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function, and α is an acceptance condition. When \mathcal{A} is a *generalized Büchi* or a *generalized co-Büchi* automaton, then $\alpha \subseteq 2^Q$ is a set of sets of accepting states. When \mathcal{A} is a *parity* automaton, then $\alpha = \langle F_1, \dots, F_d \rangle$, where the sets in α form a partition of Q . The number of sets in α is the *index* of \mathcal{A} . An automaton is *deterministic* if $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$. A run $r = r_0, r_1, \dots$ of \mathcal{A} on a word $w = w_1 \cdot w_2 \cdot \dots \in \Sigma^\omega$ is an infinite sequence of states such that $r_0 \in Q_0$, and for every $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, w_{i+1})$. We denote by $\text{inf}(r)$ the set of states that r visits infinitely often, that is $\text{inf}(r) = \{q : r_i = q \text{ for infinitely many } i \in \mathbb{N}\}$. The run r is *accepting* if it satisfies α . For generalized Büchi automata, a run is accepting if it visits all the sets in α infinitely often. Formally, for every set $F \in \alpha$, we have that $\text{inf}(r) \cap F \neq \emptyset$. Dually, in generalized co-Büchi automata, there should exist a set $F \in \alpha$ for which $\text{inf}(r) \cap F = \emptyset$. For parity automata, a run r is accepting if the minimal index i for which $\text{inf}(r) \cap F_i \neq \emptyset$ is even.

When \mathcal{A} is a *nondeterministic* automaton, it accepts a word w if it has an accepting run of on w . When \mathcal{A} is a *universal* automaton, it accepts a word w if all its runs on w are accepting. The language of \mathcal{A} , denoted $L(\mathcal{A})$, is the set of words that \mathcal{A} accepts.

A *parity game* is $\mathcal{G} = \langle \Sigma_1, \Sigma_2, S, s_0, \delta, \alpha \rangle$, where Σ_1 and Σ_2 are alphabets for Players 1 and 2, respectively, S is a finite set of states, $s_0 \in S$ is an initial state, $\delta : S \times \Sigma_1 \times \Sigma_2 \rightarrow S$ is a transition function, and $\alpha = \langle F_1, \dots, F_d \rangle$ is a parity acceptance condition, as described above. A play of the game starts in s_0 . In each turn Player 1 chooses a letter $\sigma \in \Sigma_1$ and Player 2 chooses a letter $\tau \in \Sigma_2$. The play then moves from the current state s to the state $\delta(s, \sigma, \tau)$. Formally, a *play* of \mathcal{G} is an infinite sequence $\rho = \langle s_0, \sigma_0, \tau_0 \rangle, \langle s_1, \sigma_1, \tau_1 \rangle, \dots$ such that for every $i \geq 0$, we have that $s_{i+1} = \delta(s_i, \sigma_i, \tau_i)$. We define $\text{inf}(\rho) = \{s \in S : s = s_i \text{ for infinitely many } i \in \mathbb{N}\}$. A play ρ is *winning for Player 1* if the minimal index i for which $\text{inf}(\rho) \cap F_i \neq \emptyset$ is even. A *strategy* for Player 1 is a function $f : (S \times \Sigma_1 \times \Sigma_2)^* \times S \rightarrow \Sigma_1$ that assigns, for every finite prefix of a play, the next move for Player 1. Similarly, a strategy for Player 2 is a function $g : (S \times \Sigma_1 \times \Sigma_2)^* \times S \times \Sigma_1 \rightarrow \Sigma_2$. A strategy is *memoryless* if it does not depend on the history of the play. Thus, a memoryless strategy for Player 1 is a function $f : S \rightarrow \Sigma_1$ and for Player 2 it is a function $g : S \times \Sigma_1 \rightarrow \Sigma_2$.

A pair of strategies f, g for Players 1 and 2, respectively, induces a single play that conforms with the strategies. We say that Player 1 wins \mathcal{G} if there exists a strategy f

for Player 1 such that for every strategy g for Player 2, the play induced by f and g is winning for Player 1. Otherwise, Player 2 wins. By determinacy of Parity games [21], Player 2 wins \mathcal{G} if there exists a strategy g for Player 2 such that for every strategy f of Player 1, the play induced by f and g is not winning for Player 1.

2.6 Solving the Boolean Synthesis Problem

The classical solution for the synthesis problem for LTL goes via games [24].² It involves a translation of the specification into a deterministic parity word automaton (DPW) over the alphabet $2^{I \cup O}$, which is then transformed into a game in which the players alphabets are 2^I and 2^O . More recent solutions avoid the determination and the solution of parity games and use instead alternating tree automata [19,13]. The complexity of both approaches coincide. Below we describe the classical solution for the synthesis problem, along with its complexity, when the starting point is a specification given by a DPW.³ In Remark 1, we describe an alternative, Safraless, approach, where the starting point is a universal co-Büchi automaton.

Theorem 1. *Consider a specification φ over I and O given by means of a DPW \mathcal{D}_φ of size t over the alphabet $2^{I \cup O}$, with index k . The synthesis problem for φ can be solved in time $O(t^k)$.*

Proof. Let $\mathcal{D}_\varphi = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We define a game \mathcal{G}_φ that models an interaction that simulates \mathcal{D}_φ between a system (Player 1) that generates assignments in 2^O and an environment (Player 2) that generates assignments in 2^I . Formally, $\mathcal{G}_\varphi = \langle 2^O, 2^I, Q, q_0, \eta, \alpha \rangle$, where $\eta : Q \times 2^O \times 2^I \rightarrow Q$ is such that for every $q \in Q, i \in 2^I$, and $o \in 2^O$, we have that $\eta(q, i, o) = \delta(q, i \cup o)$. By [11], the game is determined and one of the players has a memoryless winning strategy. Such a strategy for Player 1 in \mathcal{G}_φ is then a transducer that realizes φ . The game \mathcal{G}_φ is of size $O(t)$ and index k . Hence, by [15,27], we can find a memoryless strategy for the winner in time $O(t^k)$. \square

3 Properties of LLTL

In this section we study properties of the logic LLTL. We focus on the set of attainable satisfaction values of an LLTL formula and on stability properties, namely the affect of perturbing the values of the atomic propositions on the satisfaction value of formulas.

3.1 Attainable Values

Consider a lattice \mathcal{L} . We say that \mathcal{L} is *pointed* if for all LLTL formulas φ , partitions $I \cup O$ of AP , and values $\ell_1, \ell_2 \in \mathcal{L}$, if φ is (I/O) -realizable with value ℓ_1 and with value ℓ_2 , then φ is also (I/O) -realizable with value $\ell_1 \vee \ell_2$. Observe that if \mathcal{L} is pointed, then every LLTL formula over \mathcal{L} has a transducer that realizes it with a maximal value.

² In [24] and other early works the games are formulated by means of tree automata.

³ State-of-the-art algorithms for solving parity games achieve a better complexity [15,27]. The bound, however, remains polynomial in the size of the game and exponential in its index. Since the challenge of solving parity games is orthogonal to our contribution here, we keep this component of our contribution simple.

We start by showing that in general, not all lattices are pointed. In fact, our example has $O = \emptyset$, where (I/O) -realizability coincides with satisfiability. The lattices we focus on, are, however, pointed.

Theorem 2. *Not all distributive De-Morgan lattices are pointed. Fully-ordered lattices and power-set lattices are pointed.*

Proof. The proof of the positive result is in the full version. For the negative one, consider the lattice $\mathcal{L} = \langle 2^{\{a,b\}} \times \{0, 1\}, \leq \rangle$ where $\langle S_1, v_1 \rangle \leq \langle S_2, v_2 \rangle$ iff $v_1 \leq v_2$ or $(v_1 = v_2$ and $S_1 \subseteq S_2)$. We define $\neg \langle S, v \rangle = \langle \{a, b\} \setminus S, 1 - v \rangle$. It is easy to verify that \mathcal{L} is a distributive De-Morgan lattice.

Let $I = \{p\}$ and consider the formula $\varphi = (p \wedge \langle \{a\}, 1 \rangle) \vee (\neg p \wedge \langle \{b\}, 1 \rangle)$. Both $\langle \{a\}, 1 \rangle$ and $\langle \{b\}, 1 \rangle$ are attainable satisfaction values of φ . For example, by setting p to $\langle \{a\}, 1 \rangle$ or to $\langle \{a\}, 0 \rangle$. On the other hand, for every assignment ℓ to p , the second component of either ℓ or $\neg \ell$ is 0. Consequently, $\langle \{a, b\}, 1 \rangle$ is not attainable, thus \mathcal{L} is not pointed. \square

3.2 Stability

For two computations $\pi = \pi_0, \pi_1, \dots$ and $\pi' = \pi'_0, \pi'_1, \dots$, both in $(\mathcal{L}^{AP})^\omega$, we define the *global distance* between π and π' , denoted $gd(\pi, \pi')$, as $\sum_{i \geq 0} d(\pi_i, \pi'_i)$. Note that $gd(\pi, \pi')$ may be infinite. We define the *local distance* between π and π' , denoted $ld(\pi, \pi')$, as $\max_{i \geq 0} d(\pi_i, \pi'_i)$. Note that $ld(\pi, \pi') \leq |\mathcal{L}|$.

Consider an LLTL formula φ over AP and \mathcal{L} . We say that φ is *globally stable* if for every pair π and π' of computations, we have $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) \leq gd(\pi, \pi')$. Thus, the difference between the satisfaction value of φ in π and π' is bounded by the sum of differences between matching locations in π and π' . Also, φ is *locally stable* if for every pair π and π' of computations, we have $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) \leq ld(\pi, \pi')$. Thus, the difference between the satisfaction value of φ in π and π' is bounded by the maximal difference between matching locations in π and π' . Here, we study stability of all LLTL formulas. In Section 5.3, we study the problem of deciding whether a given LLTL formula is stable, and discuss the relevancy of stability to synthesis with noise.

Consider an LLTL formula φ over the atomic propositions AP , and consider computations $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$. Assume that $gd(\pi, \pi') \leq 1$. That is, π and π' differ only in one location, where they differ in the value of a single atomic proposition, whose value in π is a child of its value in π' or vice versa. It is tempting to think that then, $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) \leq 1$, which would imply that φ should be globally stable.

We start by breaking this intuition, showing that for non-distributive lattices, this is false. The proof makes use of an *N5 structure*. Formally, an N5 structure in a lattice \mathcal{L} is a tuple $\langle x, y, z, w, s \rangle$ such that the following relations hold: $s < x < y < w$, $s < z < w$, $y \not\leq z$, $z \not\leq y$, $x \not\leq z$, and $z \not\leq x$. Note that $x \vee (z \wedge y) = x \vee s = x$, whereas $(x \vee z) \wedge (x \vee y) = w \wedge y = y$. Hence, the structure of N5 is never a sub-lattice in a distributive lattice.

Theorem 3. *LLTL formulas may not be globally stable with respect to non-distributive lattices.*

Proof. Consider the lattice N5, the formula $\varphi = p \vee q$, and a computation π such that $\pi_0(p) = s$ and $\pi_0(q) = x$. Clearly $\llbracket \pi, \varphi \rrbracket = x$. Now, let π' be the computation obtained from π by setting $\pi'_0(p) = z$. It holds that $gd(\pi, \pi') = 1$. However, $\llbracket \pi', \varphi \rrbracket = z \vee x = w$, and $d(x, w) = 2$. Thus, φ is not globally stable over the lattice N5. \square

We now proceed to show that when defined with respect to a distributive lattice, all LLTL formulas are globally stable.

Theorem 4. *LLTL formulas over De-Morgan distributive lattices are globally stable.*

Proof. We prove that for every LLTL formula φ and computations $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$, if $gd(\pi, \pi') = 1$, then $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) \leq 1$. We then proceed by induction on $gd(\pi, \pi')$.

Consider an LLTL formula φ and computations π, π' such that $gd(\pi, \pi') = 1$. That is, there exists a single index $i \geq 0$ such that $d(\pi_i, \pi'_i) = 1$ and $\pi_j = \pi'_j$ for all $j \neq i$. W.l.o.g, there is $p \in AP$ such that $\pi_i(p) \preceq \pi'_i(p)$. By Birkhoff's representation theorem, there exists a unique element $u \in \text{JI}(\mathcal{L})$ such that $\pi'_i(p) = \pi_i(p) \vee u$. We prove, by induction over the structure of φ , that $\llbracket \pi', \varphi \rrbracket \in \{\llbracket \pi, \varphi \rrbracket \wedge \neg u, \llbracket \pi, \varphi \rrbracket, \llbracket \pi, \varphi \rrbracket \vee u\}$ and that $d(\llbracket \pi', \varphi \rrbracket, \llbracket \pi, \varphi \rrbracket) \leq 1$.

The proof appears in the full version. As detailed there, the interesting case is when $\varphi = \psi \vee \theta$, where we use the fact that a distributed lattice cannot have an N5 structure. \square

We now turn to study local stability. Since local stability refers to the maximal change along a computation, it is a very permissive notion. In particular, it is not hard to see that in a fully-ordered lattice, a local change of 1 entails a change of at most 1 in the satisfaction value. Thus, we have the following.

Theorem 5. *LLTL formulas are locally stable with respect to fully-ordered lattices.*

In partially-ordered lattices, however, things are more involved, as local changes may be in different “directions”. Formally, we have the following.

Theorem 6. *LLTL formulas may not be locally stable.*

Proof. Consider the power-set lattice $\langle 2^{a,b}, \subseteq \rangle$ and the LLTL formula $\varphi = p \vee Xp$. Consider computations π and π' with $\pi_0(p) = \pi_1(p) = \emptyset$, $\pi'_0(p) = \{a\}$, and $\pi'_1(p) = \{b\}$. It holds that $ld(\pi, \pi') = 1$, whereas $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) = d(\emptyset, \{a, b\}) = 2$. We conclude that φ is not locally stable. \square

4 Translating LLTL to Automata

In this section we describe an automata-theoretic approach for reasoning about LLTL specifications. One approach is to develop a framework that is based on *lattice automata* [16]. Like LLTL formulas, lattice automata map words to values in a lattice. Lattice automata have proven to be useful in solving the satisfiability and the model-checking problems for LLTL [16]. However, the solution of the synthesis problem involves automata-theoretic constructions for which the latticed counterpart is either not known or is very complicated. In particular, Safra's determinization construction has not yet been studied for lattice automata, and a latticed counterpart of it is not going

to be of much fun. Likewise, the solution of two-player games (even reachability, and moreover parity) in the latticed setting is much more complicated than in the Boolean setting. In particular, obtaining a value $\ell_1 \vee \ell_2$ in a latticed game may require one strategy for obtaining ℓ_1 and a different strategy for obtaining ℓ_2 [17]. When the game is induced by a realizability problem, it is not clear how to combine such strategies into a single transducer that realizes the underlying specification with value $\ell_1 \vee \ell_2$.

Accordingly, a second approach, which is the one we follow, is to use Boolean automata. The fact LLTL formulas have finitely many possible satisfaction values suggests that this is possible. For fully-ordered lattices, a similar approach has been taken in [12,1]. Beyond the challenge in these works of maintaining the simplicity of the automata-theoretic framework of LTL, an extra challenge in the latticed setting is caused by the fact values may be only partially ordered. We will elaborate on this point below.

In order to explain our framework, let us recall first the translation of LTL formulas to nondeterministic generalized Büchi automata (NGBW), as introduced in [29]. There, each state of the automaton is associated with a set of formulas, and the NGBW accepts a computation from a state q iff the computation satisfies exactly all the formulas associated with q . The state space of the NGBW contains only states associated with maximal and consistent sets of formulas, the transitions are defined so that requirements imposed by temporal formulas are satisfied, and the acceptance condition is used in order to guarantee that requirements that involve the satisfaction of eventualities are not delayed forever.

In the construction here, each state of the NGBW assigns a satisfaction value to every subformula. While it is not difficult to extend the local consistency rules to the latticed settings, handling of eventualities is more complicated. To see why, consider for example the formula Fp , for $p \in AP$, and the computation π in which the satisfaction value of p is $(\{a\}, \{b\}, \{c\})^\omega$. While $\llbracket \pi, Fp \rrbracket = \{a, b, c\}$, the computation never reaches a position in which the satisfaction value of the eventuality p is $\{a, b, c\}$. This poses a problem on translations of LTL formulas to automata, where eventualities are handled by making sure that each state in which the satisfaction of $\psi_1 U \psi_2$ is guaranteed, is followed by a state in which the satisfaction of ψ_2 is guaranteed. For a multi-valued setting with fully-ordered values, as is the case in [12,1], the latter can be replaced by a requirement to visit a state in which the guaranteed satisfaction value of ψ exceeds that of $\psi_1 U \psi_2$. As the example above demonstrates, such a position need not exist when the values are partially ordered. In order to address the above problem, every state in the NGBW associates with every subformula of the form $\psi_1 U \psi_2$ a value in \mathcal{L} that ψ_2 still needs “accumulate” in order for $\psi_1 U \psi_2$ to have its assigned satisfaction value. Thus, as in other break-point constructions [29,22], we decompose the requirement to obtain a value ℓ to requirements to obtain join-irreducible values whose join is ℓ , and we check these requirements together.

Theorem 7. *Let φ be an LLTL formula over \mathcal{L} and $P \subseteq \mathcal{L}$ be a predicate. There exists an NGBW $\mathcal{A}_{\varphi, P}$ such that for every computation $\pi \in (2^{AP})^\omega$, it holds that $\llbracket \pi, \varphi \rrbracket \in P$ iff $\mathcal{A}_{\varphi, P}$ accepts π . The state space and transitions of $\mathcal{A}_{\varphi, P}$ are independent of P , which only influences the set of initial states. The NGBW $\mathcal{A}_{\varphi, P}$ has at most $|\mathcal{L}|^{O(|\varphi|)}$ states and index at most $|\varphi|$.*

Proof. We define $\mathcal{A}_{\varphi,P} = \langle \mathcal{L}^{AP}, Q, \delta, Q_0, \alpha \rangle$ as follows. Let $cl(\varphi)$ be the set of φ 's subformulas, and let $ucl(\varphi)$ be the set of φ 's subformulas of the form $\psi_1 \cup \psi_2$. Let G_φ and F_φ be the collection of functions $g : cl(\varphi) \rightarrow \mathcal{L}$ and $f : ucl(\varphi) \rightarrow \mathcal{L}$, respectively. For an element $v \in \mathcal{L}$, let $\text{JI}(v)$ be the minimal set $S \subseteq \text{JI}(\mathcal{L})$ such that $v = \bigvee_{s \in S} s$. By Birkhoff's theorem, this set is well defined, and the JI mapping is a bijection.

For a pair of functions $\langle g, f \rangle \in G_\varphi \times F_\varphi$, we say that $\langle g, f \rangle$ is *consistent* if for every $\psi \in cl(\varphi)$, the following holds.

- If $\psi = v \in \mathcal{L}$, then $g(\psi) = v$.
- If $\psi = \neg\psi_1$, then $g(\psi) = \neg g(\psi_1)$.
- If $\psi = \psi_1 \vee \psi_2$, then $g(\psi) = g(\psi_1) \vee g(\psi_2)$.
- If $\psi = \psi_1 \cup \psi_2$, then $\text{JI}(f(\psi)) \cap \text{JI}(g(\psi_2)) = \emptyset$.

The state space Q of $\mathcal{A}_{\varphi,\ell}$ is the set of all consistent pairs of functions in $G_\varphi \times F_\varphi$. Intuitively, while the function g describes the satisfaction value of the formulas in the closure, the function f describes, for each subformula of the form $\psi_1 \cup \psi_2$, the values in which ψ_2 still has to be satisfied in order for the satisfaction value $g(\psi_1 \cup \psi_2)$ to be fulfilled. Accordingly, if a value is in $\text{JI}(g(\psi_2))$, it can be removed from $f(\psi_1 \cup \psi_2)$, explaining why $\text{JI}(f(\psi_1 \cup \psi_2)) \cap \text{JI}(g(\psi_2)) = \emptyset$.

Then, $Q_0 = \{g \in Q : g(\varphi) \in P\}$ contains all states in which the value assigned to φ is in P .

We now define the transition function δ . For two states $\langle g, f \rangle$ and $\langle g', f' \rangle$ in Q and a letter $\sigma \in \mathcal{L}^{AP}$, we have that $\langle g', f' \rangle \in \delta(\langle g, f \rangle, \sigma)$ iff the following hold.

- For all $p \in AP$, we have that $\sigma(p) = g(p)$.
- For all $\text{X}\psi_1 \in cl(\varphi)$, we have $g(\text{X}\psi_1) = g'(\psi_1)$.
- For all $\psi_1 \cup \psi_2 \in cl(\varphi)$, we have $g(\psi_1 \cup \psi_2) = g(\psi_2) \vee (g(\psi_1) \wedge g'(\psi_1 \cup \psi_2))$ and

$$f'(\psi_1 \cup \psi_2) = \begin{cases} \text{JI}(f(\psi_1 \cup \psi_2)) \setminus \text{JI}(g'(\psi_2)) & \text{If } \text{JI}(f(\psi_1 \cup \psi_2)) \neq \emptyset, \\ \text{JI}(g'(\psi_1 \cup \psi_2)) \setminus \text{JI}(g'(\psi_2)) & \text{Otherwise.} \end{cases}$$

Finally, every formula of the form $\psi_1 \cup \psi_2$ contributes to the acceptance condition α the set $F_{\psi_1 \cup \psi_2} = \{\langle g, f \rangle : \text{JI}(f(\psi_1 \cup \psi_2)) = \emptyset\}$.

Observe that while δ is nondeterministic, it is only nondeterministic in the first component. That is, once the function g' is chosen, there is a single function f' that can match the transition. The correctness proof can be found in the full version. \square

5 LLTL Synthesis

Recall that in the synthesis problem we are given an LLTL formula φ over sets I and O of input and output variables, taking truth values from a lattice \mathcal{L} , and we want to generate an (I/O) -transducer over \mathcal{L} all whose computations satisfy φ in a value from some desired set P of satisfaction values. In the noisy setting, the transducer may read a perturbed value of the input signals, and still all its computations need to satisfy φ as required. In this section we use the construction in Theorem 7 in order to solve both variants of the synthesis problem.

5.1 Solving the LLTL Synthesis Problem

Theorem 8. *The synthesis problem for LLTL is 2EXPTIME-complete. Given an LLTL formula φ over a lattice \mathcal{L} and a predicate $P \subseteq \mathcal{L}$, we can solve the synthesis problem for $\langle \varphi, P \rangle$ in time $2^{|\mathcal{L}|^{O(|\varphi|)}}$.*

Proof. Let m denote the size of \mathcal{L} , and let n denote the length of φ . The construction in Theorem 7 yields an NGBW with $m^{O(n)}$ states and index n . By determinizing the NGBW we obtain an equivalent DPW $\mathcal{D}_{\varphi, P}$ of size $2^{m^{O(n)} \log m^{O(n)}} = 2^{O(n)m^{O(n)}} = 2^{m^{O(n)}}$ and index $m^{O(n)}$ [26,23]. Following the same lines as the proof of Theorem 1, we see that in order to solve the LLTL synthesis problem, it suffices to solve the parity game that is obtained from \mathcal{D}_{φ} , except that here the alphabets of Players 1 and 2 are \mathcal{L}^O and \mathcal{L}^I , respectively. Accordingly, a winning memoryless strategy for Player 1 is an (I/O) -transducer over \mathcal{L} that realizes $\langle \varphi, P \rangle$.

As stated in Theorem 1, the parity game that is obtained from $\mathcal{D}_{\varphi, P}$ can be solved in time $(2^{m^{O(n)}})^{m^{O(n)}} = 2^{m^{O(n)}}$. We conclude that the LLTL-synthesis problem is in 2EXPTIME. Hardness in 2EXPTIME follow from the hardness of the synthesis problem in the Boolean setting, which corresponds to a fully-ordered lattice with two values. \square

5.2 Solving the Noisy LLTL Synthesis Problem

Consider an LLTL formula φ over the atomic propositions $I \cup O$, a predicate $P \subseteq \mathcal{L}$, and a noise function $\nu : \mathcal{L}^I \rightarrow 2^{\mathcal{L}^I}$. Recall that the goal in noisy synthesis is to find a transducer \mathcal{T} that realizes $\langle \varphi, P \rangle$ with noise ν . Our goal is to construct a DPW on which we can apply the algorithm described in Theorem 1. For this, we proceed in three steps. First, we translate φ to a universal generalized co-Büchi word automaton (UGCW). Then, we incorporate the noise in the constructed UGCW. Finally, we determinize the UGCW to obtain a DPW, from which we proceed as described in Theorem 1. We start by showing how to incorporate noise in universal automata.

Lemma 1. *Consider a UGCW \mathcal{D} and a noise function ν . There exists a UGCW \mathcal{D}' such that \mathcal{D}' accepts a computation ρ iff \mathcal{D} accepts every computation ρ' that is ν -indistinguishable from ρ . Moreover, \mathcal{D}' has the same state space and acceptance condition as \mathcal{D} .*

Proof. Let $\mathcal{D} = \langle I \cup O, Q, Q_0, \delta, \alpha \rangle$. We obtain $\mathcal{D}' = \langle I \cup O, Q, Q_0, \delta', \alpha \rangle$ from \mathcal{D} by modifying δ as follows. For every $\sigma \in I \cup O$, let $\Gamma_\sigma = \{\gamma : \gamma|_O = \sigma|_O \text{ and } \gamma|_I \in \nu(\sigma|_I)\}$. Thus, Γ_σ contains all letters that are ν -indistinguishable from σ . Then, for every state $q \in Q$, we have that $\delta'(q, \sigma) = \bigcup_{\gamma \in \Gamma_\sigma} \delta(q, \gamma)$. Thus, reading the letter σ , the UGCW \mathcal{D}' simulates all the runs of \mathcal{D} on all the letters that \mathcal{D} may read when the actual letter in the input is σ .

It is not hard to show that the set of runs of \mathcal{D}' on a computation ρ is exactly the set of all the runs of \mathcal{D} on all the computations that are ν -indistinguishable from ρ . From this, the correctness of the construction follows. \square

Theorem 9. *The noisy synthesis problem for LLTL is 2EXPTIME-complete. Given an LLTL formula φ over a lattice \mathcal{L} , a predicate $P \subseteq \mathcal{L}$, and a noise function ν , we can solve the synthesis problem for $\langle \varphi, P \rangle$ with noise ν in time $2^{m^{O(n)}}$.*

Proof. Let $\overline{P} = \mathcal{L} \setminus P$, and let $\mathcal{A}_{\varphi, \overline{P}}$ be the NGBW constructed for φ and \overline{P} in Theorem 7. Observe that $\mathcal{A}_{\varphi, \overline{P}}$ accepts a computation ρ iff $\llbracket \rho, \varphi \rrbracket \notin P$. Next, we dualize $\mathcal{A}_{\varphi, \overline{P}}$ and obtain a UGCW $\mathcal{D}_{\varphi, P}$ for the complement language, namely all computations ρ such that $\llbracket \rho, \varphi \rrbracket \in P$. We now apply the procedure in Lemma 1 to $\mathcal{D}_{\varphi, P}$ and obtain a UGCW $\mathcal{D}'_{\varphi, P}$ that accepts ρ iff $\mathcal{D}_{\varphi, P}$ accepts every computation ρ' that is ν -indistinguishable from ρ . Next, we determinize $\mathcal{D}'_{\varphi, P}$ to an equivalent DPW $\mathcal{D}''_{\varphi, P}$.

We claim that the algorithm described in the proof of Theorem 1 can be applied to $\mathcal{D}''_{\varphi, P}$. To see this, let $\mathcal{D}''_{\varphi, P} = \langle I \cup O, S, s_0, \eta, \beta \rangle$ and consider the game \mathcal{G} that is obtained from $\mathcal{D}''_{\varphi, P}$. That is, $\mathcal{G} = \langle \mathcal{L}^O, \mathcal{L}^I, S, s_0, \eta, \beta \rangle$, where for every $q \in S, i \in \mathcal{L}^I$, and $o \in \mathcal{L}^O$, we have that $\eta(q, i, o) = \mu(q, i \cup o)$.

A (memoryless) winning strategy f for Player 1 in \mathcal{G} is then an (I/O) -transducer over \mathcal{L} with the following property: for every strategy g of the environment, consider the play ρ that is induced by f and g . The play ρ induces a computation $w \in \mathcal{L}^{I \cup O}$ that is accepted by $\mathcal{D}''_{\varphi, P}$. By the construction of $\mathcal{D}''_{\varphi, P}$, this means that for every computation w' that is ν -indistinguishable from w , the run of $\mathcal{D}_{\varphi, P}$ on w' is accepting. Hence, $\llbracket w', \varphi \rrbracket \in P$, which in turn implies that f realizes $\langle \varphi, P \rangle$ with noise ν .

We now analyze the complexity of the algorithm. Let m denote the size of \mathcal{L} , and let n denote the length of φ . By Theorem 7, the size of $\mathcal{A}_{\varphi, \overline{P}}$ is $m^{O(n)}$ and it has index at most n . Dualizing results in a UGCW of the same size and acceptance condition, and so is the transition to $\mathcal{D}'_{\varphi, P}$. Determinization involves an exponential blowup, such that $\mathcal{D}''_{\varphi, P}$ is of size $2^{m^{O(n)} \log m^{O(n)}} = 2^{m^{O(n)}}$ and index $m^{O(n)}$. Finally, solving the parity game can be done in time $(2^{m^{O(n)}})^{m^{O(n)}} = 2^{m^{O(n)}}$. We conclude that the LLTL-noisy-synthesis problem is in 2EXPTIME. Hardness in 2EXPTIME again follows from the hardness of the synthesis problem in the Boolean setting. \square

Remark 1. The approach described in the proofs of Theorems 1, 8, and 9 is Safrafull, in the sense it involves a construction of a DPW. As has been the case with Boolean synthesis [19], it is possible to proceed Safralessly also in LLTL synthesis with noise. To see this, note that the starting point in Theorem 1 can also be a UGCW, and that Lemma 1 works with UGCWs. In more details, once we construct a UGCW \mathcal{U} for the specification, possibly with noise incorporated, the Safraless approach expands \mathcal{U} to a universal co-Büchi tree automaton that accepts winning strategies for the system in the corresponding synthesis game, and checks its emptiness. In terms of complexity, rather than paying an additional exponent in the translation of the specification to a deterministic automaton, we pay it in the non-emptiness check of the tree automaton.

5.3 Local Stability Revisited

In Section 3.2 we have seen that not all LLTL formulas are locally stable. This gives rise to the question of deciding whether a given LLTL formula is locally stable. In the context of synthesis, if φ is known to be locally stable and we have a transducer \mathcal{T} that realizes $\langle \varphi, P \rangle$ with no noise, we know that \mathcal{T} realizes $\langle \varphi, P \oplus \gamma \rangle$ with noise ν_γ , where $\nu_\gamma(\sigma) = \{\tau : d(\sigma, \tau) \leq \gamma\}$, and $P \oplus \gamma$ is the extension of P to noise ν_γ . Thus, $\ell \in P \oplus \gamma$ iff there is $\ell' \in P$ such that $d(\ell, \ell') \leq \gamma$.

Theorem 10. *Given an LLTL formula φ over a lattice \mathcal{L} , deciding whether φ is locally stable is PSPACE-complete.*

Proof. In order to show that the problem is in PSPACE, we consider the following, more general, problem: given an LLTL formula φ and a noise-threshold γ , we want to compute the maximal *distraction*, denoted $\Delta_{\varphi,\gamma}$, that noise γ may cause to φ . Formally,

$$\Delta_{\varphi,\gamma} = \max \{d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) : \pi, \pi' \in (\mathcal{L}^{AP})^\omega \text{ and } ld(\pi, \pi') \leq \gamma\}.$$

Observe that finding $\Delta_{\varphi,\gamma}$ allows us to decide local stability by iterating over all elements $\gamma \in \{1, \dots, |\mathcal{L}|\}$ and verifying that $\Delta_{\varphi,\gamma} \leq \gamma$. Furthermore, in order to compute $\Delta_{\varphi,\gamma}$, it is enough to decide whether $\Delta_{\varphi,\gamma} \leq \mu$ for a threshold $\mu \in \{1, \dots, |L|\}$, since we can then iterate over thresholds.

We solve the dual problem, namely deciding whether there exist $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$ such that $ld(\pi, \pi') \leq \gamma$ and $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) > \mu$. In order to solve this problem, we proceed as follows. In Theorem 7 we showed how to construct a NGBW $\mathcal{A}_{\varphi,\ell}$ such that $\mathcal{A}_{\varphi,\ell}$ accepts a computation π iff $\llbracket \pi, \varphi \rrbracket = \ell$. In Section 5.2, we showed how to construct a UGCW $\mathcal{D}'_{\varphi,\ell \oplus \mu}$ such that $\mathcal{D}'_{\varphi,\ell \oplus \mu}$ accepts π iff $\llbracket \pi', \varphi \rrbracket \in \ell \oplus \mu$ for every computation π' that is ν_γ -indistinguishable from π . Now, there exist $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$ such that $ld(\pi, \pi') \leq \gamma$ and $d(\llbracket \pi, \varphi \rrbracket, \llbracket \pi', \varphi \rrbracket) > \mu$ iff there exists $\ell \in \mathcal{L}$ such that $\llbracket \pi, \varphi \rrbracket = \ell$ and the latter conditions hold. Observe that these conditions hold iff there exists a computation π that is accepted by $\mathcal{A}_{\varphi,\ell}$ but not by $\mathcal{D}'_{\varphi,\ell \oplus \mu}$. Thus, it suffices to decide whether $L(\mathcal{A}_{\varphi,\ell}) \cap \overline{L(\mathcal{D}'_{\varphi,\ell \oplus \mu})} = \emptyset$ for every $\ell \in \mathcal{L}$.

Finally, we analyze the complexity of this procedure. Let $|\mathcal{L}| = m$ and $|\varphi| = n$. Complementation of $\mathcal{D}'_{\varphi,\ell \oplus \mu}$ can be done by constructing $\overline{\mathcal{D}'_{\varphi,\ell \oplus \mu}}$. Hence, both $\mathcal{A}_{\varphi,\ell}$ and $\overline{\mathcal{D}'_{\varphi,\ell \oplus \mu}}$ have $m^{O(n)}$ states. Checking the emptiness of their intersection can be done on-the-fly in PSPACE, implying the required upper bound.

We prove hardness in PSPACE by describing a polynomial time reduction from the satisfiability problem for LTL to the complement of the local-stability problem. Consider an LTL formula φ over AP . We assume that φ is not valid, thus there is a computation that does not satisfy it (clearly LTL satisfiability is PSPACE-hard also with this promise). We construct an LLTL formula ψ over the lattice $\mathcal{L} = \langle 2^{\{a,b\}}, \subseteq \rangle$ as follows. Let $AP' = \{p' : p \in AP\}$ be a tagged copy of AP . We define $\psi = \varphi \vee \varphi'$ over $AP \cup AP'$, where φ' is obtained from φ by replacing each atomic proposition by its tagged copy. Clearly this reduction is polynomial. In the full version, we show that φ is satisfiable iff ψ is not locally stable. \square

References

1. Almagor, S., Boker, U., Kupferman, O.: Formalizing and reasoning about quality. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 15–27. Springer, Heidelberg (2013)
2. 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)

3. Bloem, R., Chatterjee, K., Henzinger, T.A., 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)
4. Cerný, P., Henzinger, T.: From boolean to quantitative synthesis. In: EMSOFT, pp. 149–154 (2011)
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
6. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Algorithms for omega-regular games with imperfect information. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 287–302. Springer, Heidelberg (2006)
7. Chatterjee, K., Majumdar, R.: Minimum attention controller synthesis for omega-regular objectives. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 145–159. Springer, Heidelberg (2011)
8. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Controller synthesis with budget constraints. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 72–86. Springer, Heidelberg (2008)
9. Chechik, M., Devereux, B., Gurfinkel, A.: Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 16–36. Springer, Heidelberg (2001)
10. Church, A.: Logic, arithmetics, and automata. In: Proc. Int. Congress of Mathematicians, 1962, pp. 23–35. Institut Mittag-Leffle (1962)
11. Emerson, E., Jutla, C.: Tree automata, μ -calculus and determinacy. In: Proc. 32nd FOCS, pp. 368–377 (1991)
12. Faella, M., Legay, A., Stoelinga, M.: Model checking quantitative linear time logic. *Electr. Notes Theor. Comput. Sci.* 220(3), 61–77 (2008)
13. Filiot, E., Jin, N., Raskin, J.-F.: An antichain algorithm for LTL realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
14. Huth, M., Pradhan, S.: Consistent partial model checking. *Electr. Notes Theor. Comput. Sci.* 73, 45–85 (2004)
15. Jurdzinski, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing* 38(4), 1519–1532 (2008)
16. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
17. Kupferman, O., Lustig, Y.: Latticed simulation relations and games. *International Journal on the Foundations of Computer Science* 21(2), 167–189 (2010)
18. Kupferman, O., Vardi, M.: Church’s problem revisited. *The Bulletin of Symbolic Logic* 5(2), 245–263 (1999)
19. Kupferman, O., Vardi, M.: Safraless decision procedures. In: Proc. 46th FOCS, pp. 531–540 (2005)
20. Kwiatkowska, M.: Quantitative verification: models techniques and tools. In: ESEC/SIGSOFT FSE, pp. 449–458 (2007)
21. Martin, D.A.: Borel Determinacy. *Annals of Mathematics* 65, 363–371 (1975)
22. Miyano, S., Hayashi, T.: Alternating finite automata on ω -words. *TCS* 32, 321–330 (1984)
23. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. 21st LICS, pp. 255–264. IEEE (2006)
24. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th POPL, pp. 179–190 (1989)
25. Reif, J.: The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science* 29, 274–301 (1984)
26. Safra, S.: Exponential Determinization for ω -Automata with Strong-Fairness Acceptance Condition. In: Proc. 24th STOC, pp. 275–282 (1992)

27. Schewe, S.: Solving Parity Games in Big Steps. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 449–460. Springer, Heidelberg (2007)
28. Vardi, M.Y.: From verification to synthesis. In: Shankar, N., Woodcock, J. (eds.) VSTTE 2008. LNCS, vol. 5295, p. 2. Springer, Heidelberg (2008)
29. Vardi, M., Wolper, P.: Reasoning about infinite computations. *Information and Computation* 115(1), 1–37 (1994)
30. Velner, Y., Rabinovich, A.: Church synthesis problem for noisy input. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 275–289. Springer, Heidelberg (2011)

The Complexity of Partial-Observation Stochastic Parity Games with Finite-Memory Strategies^{*}

Krishnendu Chatterjee¹, Laurent Doyen², Sumit Nain³, and Moshe Y. Vardi³

¹ IST Austria

² CNRS, LSV, ENS Cachan

³ Rice University, USA

Abstract. We consider two-player partial-observation stochastic games on finite-state graphs where player 1 has partial observation and player 2 has perfect observation. The winning condition we study are ω -regular conditions specified as parity objectives. The qualitative-analysis problem given a partial-observation stochastic game and a parity objective asks whether there is a strategy to ensure that the objective is satisfied with probability 1 (resp. positive probability). These qualitative-analysis problems are known to be undecidable. However in many applications the relevant question is the existence of finite-memory strategies, and the qualitative-analysis problems under finite-memory strategies was recently shown to be decidable in 2EXPTIME. We improve the complexity and show that the qualitative-analysis problems for partial-observation stochastic parity games under finite-memory strategies are EXPTIME-complete; and also establish optimal (exponential) memory bounds for finite-memory strategies required for qualitative analysis.

1 Introduction

Games on graphs. Two-player stochastic games on finite graphs played for infinite rounds is central in many areas of computer science as they provide a natural setting to model nondeterminism and reactivity in the presence of randomness. In particular, infinite-duration games with omega-regular objectives are a fundamental tool in the analysis of many aspects of reactive systems such as modeling, verification, refinement, and synthesis [2,16]. For example, the standard approach to the synthesis problem for reactive systems reduces the problem to finding the winning strategy of a suitable game [22]. The most common approach to games assumes a setting with perfect information, where both players have complete knowledge of the state of the game. In many settings, however, the assumption of perfect information is not valid and it is natural to allow an information asymmetry between the players, such as, controllers with noisy sensors and software modules that expose partial interfaces [23].

^{*} This research was supported by Austrian Science Fund (FWF) Grant No P23499- N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), Microsoft Faculty Fellowship Award, European project Casting (FP7-601148), NSF grants CNS 1049862 and CCF-1139011, by NSF Expeditions in Computing project “ExCAPE: Expeditions in Computer Augmented Program Engineering”, by BSF grant 9800096, and by gift from Intel.

Partial-observation stochastic games. Partial-observation stochastic games are played between two players (player 1 and player 2) on a graph with finite state space. The game is played for infinitely many rounds where in each round either player 1 chooses a move or player 2 chooses a move, and the successor state is determined by a probabilistic transition function. Player 1 has partial observation where the state space is partitioned according to observations that she can observe i.e., given the current state, the player only views its observation (the partition the state belongs to), but not the precise state. Player 2 (adversary to player 1) has perfect observation and observes the precise state.

The class of ω -regular objectives. An objective specifies the desired set of behaviors (or paths) for player 1. In verification and control of stochastic systems an objective is typically an ω -regular set of paths. The class of ω -regular languages extends classical regular languages to infinite strings, and provides a robust specification language to express all commonly used specifications [24]. In a parity objective, every state of the game is mapped to a non-negative integer priority and the goal is to ensure that the minimum priority visited infinitely often is even. Parity objectives are a canonical way to define such ω -regular specifications. Thus partial-observation stochastic games with parity objective provide a general framework for analysis of stochastic reactive systems.

Qualitative and quantitative analysis. Given a partial-observation stochastic game with a parity objective and a start state, the *qualitative-analysis* problem asks whether the objective can be ensured with probability 1 (*almost-sure winning*) or positive probability (*positive winning*); whereas the *quantitative-analysis* problem asks whether the objective can be satisfied with probability at least λ for a given threshold $\lambda \in (0, 1)$.

Previous results. The quantitative analysis problem for partial-observation stochastic games with parity objectives is undecidable, even for the very special case of probabilistic automata with reachability objectives [21]. The qualitative-analysis problems for partial-observation stochastic games with parity objectives are also undecidable [3], even for probabilistic automata. In many practical applications, however, the more relevant question is the existence of finite-memory strategies. The quantitative analysis problem remains undecidable for finite-memory strategies, even for probabilistic automata [21]. The qualitative-analysis problems for partial-observation stochastic parity games were shown to be decidable with 2EXPTIME complexity for finite-memory strategies [20]; and the exact complexity was open which we settle in this work.

Our contributions. Our contributions are as follows: for the qualitative-analysis problems for partial-observation stochastic parity games under finite-memory strategies we show that (i) the problems are EXPTIME-complete; and (ii) if there is a finite-memory almost-sure (resp. positive) winning strategy, then there is a strategy that uses at most exponential memory (matching the exponential lower bound known for the simpler case of reachability and safety objectives). Thus we establish both optimal computational and strategy complexity results. Moreover, once a finite-memory strategy is fixed for player 1, we obtain a finite-state perfect-information Markov decision process (MDP) for player 2 where finite-memory is as powerful as infinite-memory [12]. Thus our results apply to both cases where player 2 has infinite-memory or restricted to finite-memory strategies.

Technical contribution. The 2EXPTIME upper bound of [20] is achieved via a reduction to the emptiness problem of alternating parity tree automata. The reduction of [20]

to alternating tree automata is exponential as it requires enumeration of the end components and recurrent classes that can arise after fixing strategies. We present a polynomial reduction, which is achieved in two steps. The first step is as follows: a *local gadget-based* reduction (that transforms every probabilistic state to a local gadget of deterministic states) for perfect-observation stochastic games to perfect-observation deterministic games for parity objectives was presented in [11,5]. This gadget, however, requires perfect observation for both players. We extend this reduction and present a local gadget-based polynomial reduction of partial-observation stochastic games to three-player partial-observation deterministic games, where player 1 has partial observation, the other two players have perfect observation, and player 3 is helpful to player 1. The crux of the proof is to show that the local reduction allows to infer properties about recurrent classes and end components (which are global properties). In the second step we present a polynomial reduction of the three-player games problem to the emptiness problem of alternating tree automata. We also remark that the new model of three-player games we introduce for the intermediate step of the reduction maybe also of independent interest for modeling of other applications.

Related works. The undecidability of the qualitative-analysis problem for partial-observation stochastic parity games with infinite-memory strategies follows from [3]. For partially observable Markov decision processes (POMDPs), which is a special case of partial-observation stochastic games where player 2 does not have any choices, the qualitative-analysis problem for parity objectives with finite-memory strategies was shown to be EXPTIME-complete [6]. For partial-observation stochastic games the almost-sure winning problem was shown to be EXPTIME-complete for Büchi objectives (both for finite-memory and infinite-memory strategies) [10,7]. Finally, for partial-observation stochastic parity games the almost-sure winning problem under finite-memory strategies was shown to be decidable in 2EXPTIME in [20].

Summary and discussion. The results for the qualitative analysis of various models of partial-observation stochastic parity games with finite-memory strategies for player 1 is summarized in Table 1. We explain the results of the table. The results of the first row follows from [6] and the results for the second row are the results of our contributions. In the most general case both players have partial observation. If we consider partial-observation stochastic games where both players have partial observation, then the results of the table are derived as follows: (a) If we consider infinite-memory strategies for player 2, then the problem remains undecidable as when player 1 is non-existent we obtain POMDPs as a special case. The non-elementary lower bound follows from the results of [7] where the lower bound was shown for reachability objectives where finite-memory strategies suffice for player 1 (against both finite and infinite-memory strategies for player 2). (b) If we consider finite-memory strategies for player 2, then the decidability of the problem is open, but we obtain the non-elementary lower bound on memory from the results of [7] for reachability objectives.

2 Partial-Observation Stochastic Parity Games

We consider partial-observation stochastic parity games where player 1 has partial observation and player 2 has perfect observation. We consider parity objectives, and for

Table 1. Complexity and memory bounds for qualitative analysis of partial-observation stochastic parity games with finite-memory strategies for player 1. The new results are boldfaced.

Game Models	Complexity	Memory bounds
POMDPs	EXPTIME-complete [6]	Exponential [6]
Player 1 partial and player 2 perfect (finite- or infinite-memory for player 2)	EXPTIME-complete	Exponential
Both players partial infinite-memory for player 2	Undecidable [3]	Non-elementary [7] (Lower bound)
Both players partial finite-memory for player 2	Open (??)	Non-elementary [7] (Lower bound)

almost-sure winning under finite-memory strategies for player 1 present a polynomial reduction to sure winning in three-player parity games where player 1 has partial observation, player 3 has perfect observation and is helpful towards player 1, and player 2 has perfect observation and is adversarial to player 1. A similar reduction also works for positive winning. We then show in the following section how to solve the sure winning problem for three-player games using alternating parity tree automata.

2.1 Basic Definitions

We start with basic definitions related to partial-observation stochastic parity games.

Partial-observation stochastic games. We consider slightly different notation (though equivalent) to the classical definitions, but the slightly different notation helps for more elegant and explicit reduction. We consider partial-observation stochastic games as a tuple $G = (S_1, S_2, S_P, A_1, \delta, E, \mathcal{O}, \text{obs})$ as follows: $S = S_1 \cup S_2 \cup S_P$ is the state space partitioned into player-1 states (S_1), player-2 states (S_2), and probabilistic states (S_P); and A_1 is a finite set of actions for player 1. Since player 2 has perfect observation, she chooses edges instead of actions. The transition function is as follows: $\delta : S_1 \times A_1 \rightarrow S_2$ that given a player-1 state in S_1 and an action in A_1 gives the next state in S_2 (which belongs to player 2); and $\delta : S_P \rightarrow \mathcal{D}(S_1)$ given a probabilistic state gives the probability distribution over the set of player-1 states. The set of edges is as follows: $E = \{(s, t) \mid s \in S_P, t \in S_1, \delta(s)(t) > 0\} \cup E'$, where $E' \subseteq S_2 \times S_P$. The observation set \mathcal{O} and observation mapping obs are standard, i.e., $\text{obs} : S \rightarrow \mathcal{O}$. Note that player 1 plays after every three steps (every move of player 1 is followed by a move of player 2, then a probabilistic choice). In other words, first player 1 chooses an action, then player 2 chooses an edge, and then there is a probability distribution over states where player 1 again chooses and so on.

Three-player non-stochastic turn-based games. We consider three-player partial-observation (non-stochastic turn-based) games as a tuple $G = (S_1, S_2, S_3, A_1, \delta, E, \mathcal{O}, \text{obs})$ as follows: S is the state space partitioned into player-1 states (S_1), player-2 states (S_2), and player-3 states (S_3); and A_1 is a finite set of actions for player 1. The transition function is as follows: $\delta : S_1 \times A_1 \rightarrow S_2$ that given a player-1 state in S_1 and an action in A_1 gives the next state (which belongs to player 2). The set of edges is as follows: $E \subseteq (S_2 \cup S_3) \times S$. Hence in these games player 1 chooses an action,

and the other players have perfect observation and choose edges. We only consider the sub-class where player 1 plays in every k -steps, for a fixed k . The observation set \mathcal{O} and observation mapping obs are again standard.

Plays and strategies. A *play* in a partial-observation stochastic game is an infinite sequence of states $s_0 s_1 s_2 \dots$ such that the following conditions hold for all $i \geq 0$: (i) if $s_i \in S_1$, then there exists $a_i \in A_1$ such that $s_{i+1} = \delta(s_i, a_i)$; and (ii) if $s_i \in (S_2 \cup S_P)$, then $(s_i, s_{i+1}) \in E$. The function obs is extended to sequences $\rho = s_0 \dots s_n$ of states in the natural way, namely $\text{obs}(\rho) = \text{obs}(s_0) \dots \text{obs}(s_n)$. A strategy for a player is a recipe to extend the prefix of a play. Formally, player-1 strategies are functions $\sigma : S^* \cdot S_1 \rightarrow A_1$; and player-2 (and analogously player-3 strategies) are functions: $\pi : S^* \cdot S_2 \rightarrow S$ such that for all $w \in S^*$ and $s \in S_2$ we have $(s, \pi(w \cdot s)) \in E$. We consider only observation-based strategies for player 1, i.e., for two play prefixes ρ and ρ' if the corresponding observation sequences match ($\text{obs}(\rho) = \text{obs}(\rho')$), then the strategy must choose the same action ($\sigma(\rho) = \sigma(\rho')$); and the other players have all strategies. The notations for three-player games are similar.

Finite-memory strategies. A player-1 strategy uses *finite-memory* if it can be encoded by a deterministic transducer $\langle M, m_0, \sigma_u, \sigma_n \rangle$ where M is a finite set (the memory of the strategy), $m_0 \in M$ is the initial memory value, $\sigma_u : M \times \mathcal{O} \rightarrow M$ is the memory-update function, and $\sigma_n : M \rightarrow A_1$ is the next-move function. The *size* of the strategy is the number $|M|$ of memory values. If the current observation is o , and the current memory value is m , then the strategy chooses the next action $\sigma_n(m)$, and the memory is updated to $\sigma_u(m, o)$. Formally, $\langle M, m_0, \sigma_u, \sigma_n \rangle$ defines the strategy σ such that $\sigma(\rho \cdot s) = \sigma_n(\hat{\sigma}_u(m_0, \text{obs}(\rho) \cdot \text{obs}(s)))$ for all $\rho \in S^*$ and $s \in S_1$, where $\hat{\sigma}_u$ extends σ_u to sequences of observations as expected. This definition extends to infinite-memory strategies by not restricting M to be finite.

Parity objectives. An *objective* for Player 1 in G is a set $\varphi \subseteq S^\omega$ of infinite sequences of states. A play ρ *satisfies* the objective φ if $\rho \in \varphi$. For a play $\rho = s_0 s_1 \dots$ we denote by $\text{Inf}(\rho)$ the set of states that occur infinitely often in ρ , that is, $\text{Inf}(\rho) = \{s \mid s_j = s \text{ for infinitely many } j\}$. For $d \in \mathbb{N}$, let $p : S \rightarrow \{0, 1, \dots, d\}$ be a *priority function*, which maps each state to a nonnegative integer priority. The *parity objective* $\text{Parity}(p)$ requires that the minimum priority that occurs infinitely often be even. Formally, $\text{Parity}(p) = \{\rho \mid \min\{p(s) \mid s \in \text{Inf}(\rho)\} \text{ is even}\}$. Parity objectives are a canonical way to express ω -regular objectives [24].

Almost-sure winning and positive winning. An *event* is a measurable set of plays. For a partial-observation stochastic game, given strategies σ and π for the two players, the probabilities of events are uniquely defined [25]. For a parity objective $\text{Parity}(p)$, we denote by $\mathbb{P}_s^{\sigma, \pi}(\text{Parity}(p))$ the probability that $\text{Parity}(p)$ is satisfied by the play obtained from the starting state s when the strategies σ and π are used. The *almost-sure* (resp. *positive*) winning problem under finite-memory strategies asks, given a partial-observation stochastic game, a parity objective $\text{Parity}(p)$, and a starting state s , whether there exists a finite-memory observation-based strategy σ for player 1 such that against all strategies π for player 2 we have $\mathbb{P}_s^{\sigma, \pi}(\text{Parity}(p)) = 1$ (resp. $\mathbb{P}_s^{\sigma, \pi}(\text{Parity}(p)) > 0$). The almost-sure and positive winning problems are also referred to as the qualitative-analysis problems for stochastic games.

Sure winning in three-player games. In three-player games once the starting state s and strategies σ, π , and τ of the three players are fixed we obtain a unique play, which we denote as $\rho_s^{\sigma, \pi, \tau}$. In three-player games we consider the following *sure* winning problem: given a parity objective $\text{Parity}(p)$, sure winning is ensured if there exists a finite-memory observation-based strategy σ for player 1, such that in the two-player perfect-observation game obtained after fixing σ , player 3 can ensure the parity objective against all strategies of player 2. Formally, the sure winning problem asks whether there exist a finite-memory observation-based strategy σ for player 1 and a strategy τ for player 3, such that for all strategies π for player 2 we have $\rho_s^{\sigma, \pi, \tau} \in \text{Parity}(p)$.

Remark 1 (Equivalence with standard model). We remark that for the model of partial-observation stochastic games studied in literature the two players simultaneously choose actions, and a probabilistic transition function determine the probability distribution of the next state. In our model, the game is turn-based and the probability distribution is chosen only in probabilistic states. However, it follows from the results of [8] that the models are equivalent: by the results of [8, Section 3.1] the interaction of the players and probability can be separated without loss of generality; and [8, Theorem 4] shows that in presence of partial observation, concurrent games can be reduced to turn-based games in polynomial time. Thus the turn-based model where the moves of the players and stochastic interaction are separated is equivalent to the standard model. Moreover, for a perfect-information player choosing an action is equivalent to choosing an edge in a turn-based game. Thus the model we consider is equivalent to the standard partial-observation game models.

Remark 2 (Pure and randomized strategies). In this work we only consider pure strategies. In partial-observation games, randomized strategies are also relevant as they are more powerful than pure strategies. However, for finite-memory strategies the almost-sure and positive winning problem for randomized strategies can be reduced in polynomial time to the problem for finite-memory pure strategies [7,20]. Hence without loss of generality we only consider pure strategies.

2.2 Reduction of Partial-Observation Stochastic Games to Three-Player Games

In this section we present a polynomial-time reduction for the almost-sure winning problem in partial-observation stochastic parity games to the sure winning problem in three-player parity games.

Reduction. Let us denote by $[d]$ the set $\{0, 1, \dots, d\}$. Given a partial-observation stochastic parity game graph $G = (S_1, S_2, S_P, A_1, \delta, E, \mathcal{O}, \text{obs})$ with a parity objective defined by priority function $p : S \rightarrow [d]$ we construct a three-player game graph $\overline{G} = (\overline{S}_1, \overline{S}_2, \overline{S}_3, A_1, \overline{\delta}, \overline{E}, \mathcal{O}, \text{obs})$ together with priority function \overline{p} . The construction is specified as follows.

1. For every nonprobabilistic state $s \in S_1 \cup S_2$, there is a corresponding state $\overline{s} \in \overline{S}$ such that (i) $\overline{s} \in \overline{S}_1$ if $s \in S_1$, else $\overline{s} \in \overline{S}_2$; (ii) $\overline{p}(\overline{s}) = p(s)$ and $\overline{\text{obs}}(\overline{s}) = \text{obs}(s)$; (iii) $\overline{\delta}(\overline{s}, a) = \overline{t}$ where $t = \delta(s, a)$, for $s \in S_1$ and $a \in A_1$; and (iv) $(\overline{s}, \overline{t}) \in \overline{E}$ iff $(s, t) \in E$, for $s \in S_2$.

2. Every probabilistic state $s \in S_P$ is replaced by the gadget shown in Figure 1 for illustration. In the figure, square-shaped states are player-2 states (in \overline{S}_2), and circle-shaped (or ellipsoid-shaped) states are player-3 states (in \overline{S}_3). Formally, from the state \overline{s} with priority $p(s)$ and observation $\text{obs}(s)$ (i.e., $\overline{p}(\overline{s}) = p(s)$ and $\overline{\text{obs}}(\overline{s}) = \text{obs}(s)$) the players play the following three-step game in \overline{G} .
 - In state \overline{s} player 2 chooses a successor $(\tilde{s}, 2k)$, for $2k \in \{0, 1, \dots, p(s) + 1\}$.
 - For every state $(\tilde{s}, 2k)$, we have $\overline{p}((\tilde{s}, 2k)) = p(s)$ and $\overline{\text{obs}}((\tilde{s}, 2k)) = \text{obs}(s)$. For $k \geq 1$, in state $(\tilde{s}, 2k)$ player 3 chooses between two successors: state $(\hat{s}, 2k-1)$ with priority $2k-1$ and same observation as s , or state $(\hat{s}, 2k)$ with priority $2k$ and same observation as s , (i.e., $\overline{p}((\hat{s}, 2k-1)) = 2k-1$, $\overline{p}((\hat{s}, 2k)) = 2k$, and $\overline{\text{obs}}((\hat{s}, 2k-1)) = \overline{\text{obs}}((\hat{s}, 2k)) = \text{obs}(s)$). The state $(\tilde{s}, 0)$ has only one successor $(\hat{s}, 0)$, with $\overline{p}((\hat{s}, 0)) = 0$ and $\overline{\text{obs}}((\hat{s}, 0)) = \text{obs}(s)$.
 - Finally, in each state (\hat{s}, k) the choice is between all states \overline{t} such that $(s, t) \in E$, and it belongs to player 3 (i.e., in \overline{S}_3) if k is odd, and to player 2 (i.e., in \overline{S}_2) if k is even. Note that every state in the gadget has the same observation as s .

We denote by $\overline{G} = \text{Tr}_{\text{as}}(G)$ the three-player game, where player 1 has partial-observation, and both player 2 and player 3 have perfect-observation, obtained from a partial-observation stochastic game. Observe that in \overline{G} there are exactly four steps between two player 1 moves.

Observation sequence mapping. Note that since in our partial-observation games first player 1 plays, then player 2, followed by probabilistic states, repeated ad infinitum, wlog, we can assume that for every observation $o \in \mathcal{O}$ we have either (i) $\text{obs}^{-1}(o) \subseteq S_1$; or (ii) $\text{obs}^{-1}(o) \subseteq S_2$; or (i) $\text{obs}^{-1}(o) \subseteq S_P$. Thus we partition the observations as $\mathcal{O}_1, \mathcal{O}_2$, and \mathcal{O}_P . Given an observation sequence $\kappa = o_0 o_1 o_2 \dots o_n$ in G corresponding to a finite prefix of a play, we inductively define the sequence $\overline{\kappa} = \overline{h}(\kappa)$ in \overline{G} as follows: (i) $\overline{h}(o_0) = o_0$ if $o_0 \in \mathcal{O}_1 \cup \mathcal{O}_2$, else $o_0 o_0 o_0$; (ii) $\overline{h}(o_0 o_1 \dots o_n) = \overline{h}(o_0 o_1 \dots o_{n-1}) o_n$ if $o_n \in \mathcal{O}_1 \cup \mathcal{O}_2$, else $\overline{h}(o_0 o_1 \dots o_{n-1}) o_n o_n o_n$. Intuitively the mapping takes care of the two extra steps of the gadgets introduced for probabilistic states. The mapping is a bijection, and hence given an observation sequence $\overline{\kappa}$ of a play prefix in \overline{G} we consider the inverse play prefix $\kappa = \overline{h}^{-1}(\overline{\kappa})$ such that $\overline{h}(\kappa) = \overline{\kappa}$.

Strategy mapping. Given an observation-based strategy $\overline{\sigma}$ in \overline{G} we consider a strategy $\sigma = \text{Tr}_{\text{as}}(\overline{\sigma})$ as follows: for an observation sequence κ corresponding to a play prefix in G we have $\sigma(\kappa) = \overline{\sigma}(\overline{h}(\kappa))$. The strategy σ is observation-based (since $\overline{\sigma}$ is observation-based). The inverse mapping $\text{Tr}_{\text{as}}^{-1}$ of strategies from G to \overline{G} is analogous. Note that for σ in G we have $\text{Tr}_{\text{as}}(\text{Tr}_{\text{as}}^{-1}(\sigma)) = \sigma$. Let $\overline{\sigma}$ be a finite-memory strategy with memory M for player 1 in the game \overline{G} . The strategy $\overline{\sigma}$ can be considered as a memoryless strategy, denoted as $\overline{\sigma}^* = \text{MemLess}(\overline{\sigma})$, in $\overline{G} \times M$ (the synchronous product of \overline{G} with M). Given a strategy (pure memoryless) $\overline{\pi}$ for player 2 in the 2-player game $\overline{G} \times M$, a strategy $\pi = \text{Tr}_{\text{as}}(\overline{\pi})$ in the partial-observation stochastic game $G \times M$ is defined as: $\pi((s, m)) = (t, m')$, if and only if $\overline{\pi}((\overline{s}, m)) = (\overline{t}, m')$; for all $s \in S_2$.

End components. Given an MDP, a set U is an end component in the MDP if the subgraph induced by U is strongly connected, and for all probabilistic states in U all outgoing edges end up in U (i.e., U is closed for probabilistic states). The *key property* about MDPs that is used in our proofs is a result established by [12,13] that given an MDP, for all strategies, with probability 1 the set of states visited infinitely often is an

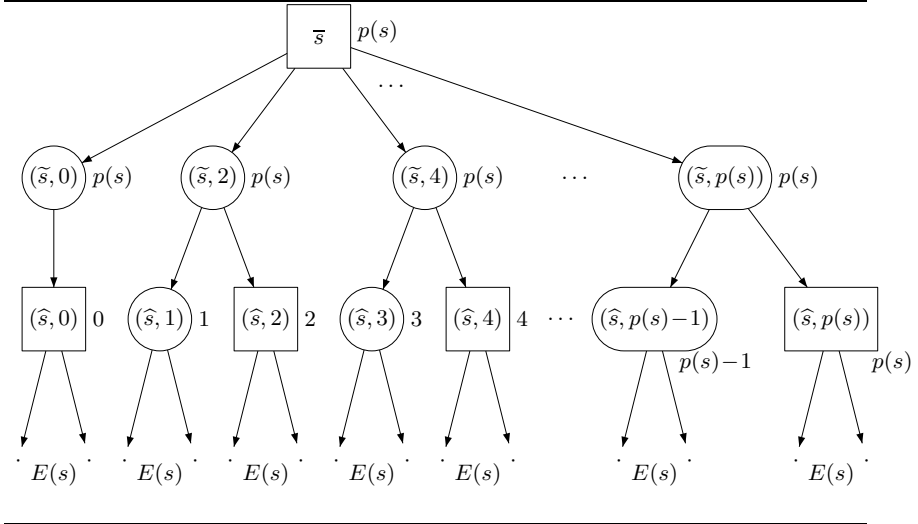


Fig. 1. Reduction gadget when $p(s)$ is even

end component. The key property allows us to analyze end components of MDPs and from properties of the end component conclude properties about all strategies.

The key lemma. We now present our main lemma that establishes the correctness of the reduction. Since the proof of the lemma is long we split the proof into two parts.

Lemma 1. *Given a partial-observation stochastic parity game G with parity objective $\text{Parity}(p)$, let $\overline{G} = \text{Tr}_{\text{as}}(G)$ be the three-player game with the modified parity objective $\text{Parity}(\overline{p})$ obtained by our reduction. Consider a finite-memory strategy $\overline{\sigma}$ with memory M for player 1 in \overline{G} . Let us denote by $\overline{G}_{\overline{\sigma}}$ the perfect-observation two-player game played over $\overline{G} \times M$ by player 2 and player 3 after fixing the strategy $\overline{\sigma}$ for player 1. Let*

$$\overline{U}_1^{\overline{\sigma}} = \{(\overline{s}, m) \in \overline{S} \times M \mid \text{player 3 has a sure winning strategy for } \text{Parity}(\overline{p}) \text{ from } (\overline{s}, m) \text{ in } \overline{G}_{\overline{\sigma}}\};$$

and let $\overline{U}_2^{\overline{\sigma}} = (\overline{S} \times M) \setminus \overline{U}_1^{\overline{\sigma}}$ be the set of sure winning states for player 2 in $\overline{G}_{\overline{\sigma}}$. Consider the strategy $\sigma = \text{Tr}_{\text{as}}(\overline{\sigma})$, and the sets $U_1^{\sigma} = \{(s, m) \in S \times M \mid (\overline{s}, m) \in \overline{U}_1^{\overline{\sigma}}\}$; and $U_2^{\sigma} = (S \times M) \setminus U_1^{\sigma}$. The following assertions hold.

1. For all $(s, m) \in U_1^{\sigma}$, for all strategies π of player 2 we have $\mathbb{P}_{(s,m)}^{\sigma, \pi}(\text{Parity}(p)) = 1$.
2. For all $(s, m) \in U_2^{\sigma}$, there exists a strategy π of player 2 such that $\mathbb{P}_{(s,m)}^{\sigma, \pi}(\text{Parity}(p)) < 1$.

We first present the proof for part 1 and then for part 2.

Proof (of Lemma 1: part 1). Consider a finite-memory strategy $\overline{\sigma}$ for player 1 with memory M in the game \overline{G} . Once the strategy $\overline{\sigma}$ is fixed we obtain the two-player finite-state perfect-observation game $\overline{G}_{\overline{\sigma}}$ (between player 3 and the adversary player 2). Recall the sure winning sets $\overline{U}_1^{\overline{\sigma}}$ for player 3, and $\overline{U}_2^{\overline{\sigma}} = (\overline{S} \times M) \setminus \overline{U}_1^{\overline{\sigma}}$ for player 2, respectively, in $\overline{G}_{\overline{\sigma}}$. Let $\sigma = \text{Tr}_{\text{as}}(\overline{\sigma})$ be the corresponding strategy in G . We denote by $\overline{\sigma}^* = \text{MemLess}(\overline{\sigma})$ and σ^* the corresponding memoryless strategies of $\overline{\sigma}$ in $\overline{G} \times M$ and σ in

$G \times M$, respectively. We show that all states in U_1^σ are almost-sure winning, i.e., given σ , for all $(s, m) \in U_1^\sigma$, for all strategies π for player 2 in G we have $\mathbb{P}_{(s,m)}^{\sigma,\pi}(\text{Parity}(p)) = 1$ (recall $U_1^\sigma = \{(s, m) \in S \times M \mid (\bar{s}, m) \in \bar{U}_1^\sigma\}$). We also consider explicitly the MDP $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ to analyze strategies of player 2 on the synchronous product, i.e., we consider the player-2 MDP obtained after fixing the memoryless strategy σ^* in $G \times M$, and then restrict the MDP to the set U_1^σ .

Two key components. The proof has two key components. First, we argue that all end components in the MDP restricted to U_1^σ are winning for player 1 (have min priority even). Second we argue that given the starting state (s, m) is in U_1^σ , almost-surely the set of states visited infinitely often is an end component in U_1^σ against all strategies of player 2. These two key components establish the desired result.

Winning end components. Our first goal is to show that every end component C in the player-2 MDP $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ is winning for player 1 for the parity objective, i.e., the minimum priority of C is even. We argue that if there is an end component C in $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ that is winning for player 2 for the parity objective (i.e., minimum priority of C is odd), then against any memoryless player-3 strategy $\bar{\tau}$ in $\bar{G}_{\bar{\sigma}}$, player 2 can construct a cycle in the game $(\bar{G} \times M \upharpoonright \bar{U}_1^\sigma)_{\bar{\sigma}^*}$ that is winning for player 2 (i.e., minimum priority of the cycle is odd) (note that given the strategy $\bar{\sigma}$ is fixed, we have finite-state perfect-observation parity games, and hence in the enlarged game we can restrict ourselves to memoryless strategies for player 3). This gives a contradiction because player 3 has a sure winning strategy from the set \bar{U}_1^σ in the 2-player parity game $\bar{G}_{\bar{\sigma}}$. Towards contradiction, let C be an end component in $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ that is winning for player 2, and let its minimum odd priority be $2r - 1$, for some $r \in \mathbb{N}$. Then there is a memoryless strategy π' for player 2 in the MDP $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ such that C is a bottom scc (or a terminal scc) in the Markov chain graph of $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*, \pi'}$. Let $\bar{\tau}$ be a memoryless for player 3 in $(\bar{G} \times M \upharpoonright \bar{U}_1^\sigma)_{\bar{\sigma}^*}$. Given $\bar{\tau}$ for player 3 and strategy π' for player 2 in $G \times M$, we construct a strategy $\bar{\pi}$ for player 2 in the game $(\bar{G} \times M \upharpoonright \bar{U}_1^\sigma)_{\bar{\sigma}^*}$ as follows. For a player-2 state in C , the strategy $\bar{\pi}$ follows the strategy π' , i.e., for a state $(s, m) \in C$ with $s \in S_2$ we have $\bar{\pi}((\bar{s}, m)) = (\bar{t}, m')$ where $(t, m') = \pi'((s, m))$. For a probabilistic state in C we define the strategy as follows (i.e., we now consider a state $(s, m) \in C$ with $s \in S_P$):

- if for some successor state $((\bar{s}, 2\ell), m')$ of (\bar{s}, m) , the player-3 strategy $\bar{\tau}$ chooses a successor $((\bar{s}, 2\ell - 1), m'')$ in C at the state $((\bar{s}, 2\ell), m')$, for $\ell < r$, then the strategy $\bar{\pi}$ chooses at state (\bar{s}, m) the successor $((\bar{s}, 2\ell), m')$; and
- otherwise the strategy $\bar{\pi}$ chooses at state (\bar{s}, m) the successor $((\bar{s}, 2r), m')$, and at $((\bar{s}, 2r), m')$ it chooses a successor shortening the distance (i.e., chooses a successor with smaller breadth-first-search distance) to a fixed state (\bar{s}^*, m^*) of priority $2r - 1$ of C (such a state (s^*, m^*) exists in C since C is strongly connected and has minimum priority $2r - 1$); and for the fixed state of priority $2r - 1$ the strategy chooses a successor (\bar{s}, m') such that $(s, m') \in C$.

Consider an arbitrary cycle in the subgraph $(\bar{G} \times M \upharpoonright \bar{C})_{\bar{\sigma}, \bar{\pi}, \bar{\tau}}$ where \bar{C} is the set of states in the gadgets of states in C . There are two cases. (Case 1): If there is at least one state $((\bar{s}, 2\ell - 1), m)$, with $\ell \leq r$ on the cycle, then the minimum priority on the cycle is odd, as even priorities smaller than $2r$ are not visited by the construction as C

does not contain states of even priorities smaller than $2r$. (Case 2): Otherwise, in all states choices shortening the distance to the state with priority $2r - 1$ are taken and hence the cycle must contain a priority $2r - 1$ state and all other priorities on the cycle are $\geq 2r - 1$, so $2r - 1$ is the minimum priority on the cycle. Hence a winning end component for player 2 in the MDP contradicts that player 3 has a sure winning strategy in \overline{G}_σ from \overline{U}_1^σ . Thus it follows that all end components are winning for player 1 in $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$.

Almost-sure reachability to winning end-components. Finally, we consider the probability of staying in U_1^σ . For every probabilistic state $(s, m) \in (S_P \times M) \cap U_1^\sigma$, all of its successors must be in U_1^σ . Otherwise, player 2 in the state (\bar{s}, m) of the game \overline{G}_σ can choose the successor $(\tilde{s}, 0)$ and then a successor to its winning set \overline{U}_2^σ . This again contradicts the assumption that (\bar{s}, m) belong to the sure winning states \overline{U}_1^σ for player 3 in \overline{G}_σ . Similarly, for every state $(s, m) \in (S_2 \times M) \cap U_1^\sigma$ we must have all its successors are in U_1^σ . For all states $(s, m) \in (S_1 \times M) \cap U_1^\sigma$, the strategy σ chooses a successor in U_1^σ . Hence for all strategies π of player 2, for all states $(s, m) \in U_1^\sigma$, the objective $\text{Safe}(U_1^\sigma)$ (which requires that only states in U_1^σ are visited) is ensured almost-surely (in fact surely), and hence with probability 1 the set of states visited infinitely often is an end component in U_1^σ (by key property of MDPs). Since every end component in $(G \times M \upharpoonright U_1^\sigma)_{\sigma^*}$ has even minimum priority, it follows that the strategy σ is an almost-sure winning strategy for the parity objective $\text{Parity}(p)$ for player 1 from all states $(s, m) \in U_1^\sigma$. This concludes the proof for first part of the lemma. ■

Proof (of Lemma 1: part 2). Consider a memoryless sure winning strategy $\bar{\pi}$ for player 2 in \overline{G}_σ from the set \overline{U}_2^σ . Let us consider the strategies $\sigma = \text{Tr}_{\text{as}}(\bar{\sigma})$ and $\pi = \text{Tr}_{\text{as}}(\bar{\pi})$, and consider the Markov chain $G_{\sigma, \pi}$. Our proof shows the following two properties to establish the claim: (1) in the Markov chain $G_{\sigma, \pi}$ all bottom sccs (the recurrent classes) in U_2^σ have odd minimum priority; and (2) from all states in U_2^σ some recurrent class in U_2^σ is reached with positive probability. This establishes the desired result of the lemma.

No winning bottom scc for player 1 in U_2^σ . Assume towards contradiction that there is a bottom scc C contained in U_2^σ in the Markov chain $G_{\sigma, \pi}$ such that the minimum priority in C is even. From C we construct a winning cycle (minimum priority is even) in \overline{U}_2^σ for player 3 in the game \overline{G}_σ given the strategy $\bar{\pi}$. This contradicts that $\bar{\pi}$ is a sure winning strategy for player 2 from \overline{U}_2^σ in \overline{G}_σ . Let the minimum priority of C be $2r$ for some $r \in \mathbb{N}$. The idea is similar to the construction of part 1. Given C , and the strategies $\bar{\sigma}$ and $\bar{\pi}$, we construct a strategy $\bar{\tau}$ for player 3 in \overline{G} as follows: For a probabilistic state (s, m) in C :

- if $\bar{\pi}$ chooses a state $((\tilde{s}, 2\ell - 2), m')$, with $\ell \leq r$, then $\bar{\tau}$ chooses the successor $((\hat{s}, 2\ell - 2), m')$;
- otherwise $\ell > r$ (i.e., $\bar{\pi}$ chooses a state $((\tilde{s}, 2\ell - 2), m')$ for $\ell > r$), then $\bar{\tau}$ chooses the state $((\hat{s}, 2\ell - 1), m')$, and then a successor to shorten the distance to a fixed state with priority $2r$ (such a state exists in C); and for the fixed state of priority $2r$, the strategy $\bar{\tau}$ chooses a successor in C .

Similar to the proof of part 1, we argue that we obtain a cycle with minimum even priority in the graph $(\overline{G} \times M \upharpoonright \overline{U}_2^\sigma)_{\bar{\sigma}, \bar{\pi}, \bar{\tau}}$. Consider an arbitrary cycle in the subgraph

$(\overline{G} \times M \upharpoonright \overline{C})_{\overline{\sigma}, \overline{\pi}, \overline{\tau}}$ where \overline{C} is the set of states in the gadgets of states in C . There are two cases. (Case 1): If there is at least one state $((\tilde{s}, 2\ell - 2), m)$, with $\ell \leq r$ on the cycle, then the minimum priority on the cycle is even, as odd priorities strictly smaller than $2r + 1$ are not visited by the construction as C does not contain states of odd priorities strictly smaller than $2r + 1$. (Case 2): Otherwise, in all states choices shortening the distance to the state with priority $2r$ are taken and hence the cycle must contain a priority $2r$ state and all other priorities on the cycle are $\geq 2r$, so $2r$ is the minimum priority on the cycle. Thus we obtain cycles winning for player 3, and this contradicts that $\overline{\pi}$ is a sure winning strategy for player 2 from \overline{U}_2^σ . Thus it follows that all recurrent classes in U_2^σ in the Markov chain $G_{\sigma, \pi}$ are winning for player 2.

Not almost-sure reachability to U_1^σ . We now argue that given σ and π there exists no state in U_2^σ such that U_1^σ is reached almost-surely. This would ensure that from all states in U_2^σ some recurrent class in U_2^σ is reached with positive probability and establish the desired claim since we have already shown that all recurrent classes in U_2^σ are winning for player 2. Given σ and π , let $X \subseteq U_2^\sigma$ be the set of states such that the set U_1^σ is reached almost-surely from X , and assume towards contradiction that X is non-empty. This implies that from every state in X , in the Markov chain $G_{\sigma, \pi}$, there is a path to the set U_1^σ , and from all states in X the successors are in X . We construct a strategy $\overline{\tau}$ in the three-player game \overline{G}_σ against strategy $\overline{\pi}$ exactly as the strategy constructed for winning bottom scc, with the following difference: instead of shortening distance to a fixed state of priority $2r$ (as for winning bottom scc's), in this case the strategy $\overline{\tau}$ shortens distance to \overline{U}_1^σ . Formally, given X , the strategies $\overline{\sigma}$ and $\overline{\pi}$, we construct a strategy $\overline{\tau}$ for player 3 in \overline{G} as follows: For a probabilistic state (s, m) in X :

- if $\overline{\pi}$ chooses a state $((\tilde{s}, 2\ell), m')$, with $\ell \geq 1$, then $\overline{\tau}$ chooses the state $((\tilde{s}, 2\ell - 1), m')$, and then a successor to shorten the distance to the set \overline{U}_1^σ (such a successor exists since from all states in X the set \overline{U}_1^σ is reachable).

Against the strategy of player 3 in \overline{G}_σ either (i) \overline{U}_1^σ is reached in finitely many steps, or (ii) else player 2 infinitely often chooses successor states of the form $(\tilde{s}, 0)$ with priority 0 (the minimum even priority), i.e., there is a cycle with a state $(\tilde{s}, 0)$ which has priority 0. If priority 0 is visited infinitely often, then the parity objective is satisfied. This ensures that in \overline{G}_σ player 3 can ensure either to reach \overline{U}_1^σ in finitely many steps from some state in \overline{U}_2^σ against $\overline{\pi}$, or the parity objective is satisfied without reaching \overline{U}_1^σ . In either case this implies that against $\overline{\pi}$ player 3 can ensure to satisfy the parity objective (by reaching \overline{U}_1^σ in finitely many steps and then playing a sure winning strategy from \overline{U}_1^σ , or satisfying the parity objective without reaching \overline{U}_1^σ by visiting priority 0 infinitely often) from some state in \overline{U}_2^σ , contradicting that $\overline{\pi}$ is a sure winning strategy for player 2 from \overline{U}_2^σ . Thus we have a contradiction, and obtain the desired result. ■

Lemma 1 establishes the desired correctness result as follows: (1) If $\overline{\sigma}$ is a finite-memory strategy such that in \overline{G}_σ player 3 has a sure winning strategy, then by part 1 of Lemma 1 we obtain that $\sigma = \text{Tr}_{\text{as}}(\overline{\sigma})$ is almost-sure winning. (2) Conversely, if σ is a finite-memory almost-sure winning strategy, then consider a strategy $\overline{\sigma}$ such that $\sigma = \text{Tr}_{\text{as}}(\overline{\sigma})$ (i.e., $\overline{\sigma} = \text{Tr}_{\text{as}}^{-1}(\sigma)$). By part 2 of Lemma 1, given the finite-memory

strategy $\bar{\sigma}$, player 3 must have a sure winning strategy in $\overline{G}_{\bar{\sigma}}$, otherwise we have a contradiction that σ is almost-sure winning. Thus we have the following theorem.

Theorem 1 (Polynomial reduction). *Given a partial-observation stochastic game graph G with a parity objective $\text{Parity}(p)$ for player 1, we construct a three-player game $\overline{G} = \text{Tr}_{\text{as}}(G)$ with a parity objective $\text{Parity}(\bar{p})$, where player 1 has partial-observation and the other two players have perfect-observation, in time $O((n + m) \cdot d)$, where n is the number of states of the game, m is the number of transitions, and d the number of priorities of the priority function p , such that the following assertion holds: there is a finite-memory almost-sure winning strategy σ for player 1 in G iff there exists a finite-memory strategy $\bar{\sigma}$ for player 1 in \overline{G} such that in the game $\overline{G}_{\bar{\sigma}}$ obtained given $\bar{\sigma}$, player 3 has a sure winning strategy for $\text{Parity}(\bar{p})$. The game graph $\text{Tr}_{\text{as}}(G)$ has $O(n \cdot d)$ states, $O(m \cdot d)$ transitions, and \bar{p} has at most $d + 1$ priorities.*

Remark 3 (Positive winning). We have presented the details of the reduction for almost-sure winning, and a very similar reduction works for positive winning (see [1]).

3 Solving Sure Winning for Three-player Parity Games

In this section we present the solution for sure winning in three-player non-stochastic parity games. We start with the basic definitions.

3.1 Basic Definitions

We first present a model of partial-observation concurrent three-player games, where player 1 has partial observation, and player 2 and player 3 have perfect observation. Player 1 and player 3 have the same objective and they play against player 2. Three-player turn-based games model (of Section 2) can be treated as a special case of this model (see [1, Remark 3] for details).

Partial-observation three-player concurrent games. Given alphabets A_i of actions for player i ($i = 1, 2, 3$), a partial-observation three-player concurrent game (for brevity, *three-player game* in sequel) is a tuple $G = \langle S, s_0, \delta, \mathcal{O}, \text{obs} \rangle$ where: (i) S is a finite set of states and $s_0 \in S$ is the initial state; (ii) $\delta : S \times A_1 \times A_2 \times A_3 \rightarrow S$ is a deterministic transition function that, given a current state s , and actions $a_1 \in A_1, a_2 \in A_2, a_3 \in A_3$ of the players, gives the successor state $s' = \delta(s, a_1, a_2, a_3)$ of s ; and (iii) \mathcal{O} is a finite set of observations and obs is the observation mapping (as in Section 2).

Strategies. Define the set Σ of *strategies* $\sigma : \mathcal{O}^+ \rightarrow A_1$ of player 1 that, given a sequence of past observations, return an action for player 1. Equivalently, we sometimes view a strategy of player 1 as a function $\sigma : S^+ \rightarrow A_1$ satisfying $\sigma(\rho) = \sigma(\rho')$ for all $\rho, \rho' \in S^+$ such that $\text{obs}(\rho) = \text{obs}(\rho')$, and say that σ is *observation-based*. A strategy of player 2 (resp. player 3) is a function $\pi : S^+ \rightarrow A_2$ (resp., $\tau : S^+ \rightarrow A_3$) without any restriction. We denote by Π (resp. Γ) the set of strategies of player 2 (resp. player 3).

Sure winning. Given strategies σ, π, τ of the three players in G , the *outcome play* from s_0 is the infinite sequence $\rho_{s_0}^{\sigma, \pi, \tau} = s_0 s_1 \dots$ such that for all $j \geq 0$, we have $s_{j+1} = \delta(s_j, a_j, b_j, c_j)$ where $a_j = \sigma(s_0 \dots s_j)$, $b_j = \pi(s_0 \dots s_j)$, and $c_j = \tau(s_0 \dots s_j)$.

Given a game $G = \langle S, s_0, \delta, \mathcal{O}, \text{obs} \rangle$ and a parity objective $\varphi \subseteq S^\omega$, the sure winning problem asks to decide if $\exists \sigma \in \Sigma \cdot \exists \tau \in \Gamma \cdot \forall \pi \in \Pi : \rho_{s_0}^{\sigma, \tau} \in \varphi$. It will follow from our result that if the answer to the sure winning problem is yes, then there exists a witness finite-memory strategy σ for player 1.

3.2 Alternating Tree Automata

In this section we recall the definitions of alternating tree automata, and present the solution of the sure winning problem for three-player games with parity objectives by a reduction to the emptiness problem of alternating parity tree automata.

Trees. Given an alphabet Ω , an Ω -labeled tree (T, V) consists of a prefix-closed set $T \subseteq \mathbb{N}^*$ (i.e., if $x \cdot d \in T$ with $x \in \mathbb{N}^*$ and $d \in \mathbb{N}$, then $x \in T$), and a mapping $V : T \rightarrow \Omega$ that assigns to each node of T a letter in Ω . Given $x \in \mathbb{N}^*$ and $d \in \mathbb{N}$ such that $x \cdot d \in T$, we call $x \cdot d$ the *successor* in direction d of x . The node ε is the *root* of the tree. An *infinite path* in T is an infinite sequence $\pi = d_1 d_2 \dots$ of directions $d_i \in \mathbb{N}$ such that every finite prefix of π is a node in T .

Alternating tree automata. Given a parameter $k \in \mathbb{N} \setminus \{0\}$, we consider input trees of rank k , i.e. trees in which every node has at most k successors. Let $[k] = \{0, \dots, k-1\}$, and given a finite set U , let $\mathcal{B}^+(U)$ be the set of positive Boolean formulas over U , i.e. formulas built from elements in $U \cup \{\text{true}, \text{false}\}$ using the Boolean connectives \wedge and \vee . An *alternating tree automaton* over alphabet Ω is a tuple $\mathcal{A} = \langle S, s_0, \delta \rangle$ where: (i) S is a finite set of states and $s_0 \in S$ is the initial state; and (ii) $\delta : S \times \Omega \rightarrow \mathcal{B}^+(S \times [k])$ is a transition function. Intuitively, the automaton is executed from the initial state s_0 and reads the input tree in a top-down fashion starting from the root ε . In state s , if $a \in \Omega$ is the letter that labels the current node x of the input tree, the behavior of the automaton is given by the formulas $\psi = \delta(s, a)$. The automaton chooses a *satisfying assignment* of ψ , i.e. a set $Q \subseteq S \times [k]$ such that the formula ψ is satisfied when the elements of Q are replaced by true, and the elements of $(S \times [k]) \setminus Q$ are replaced by false. Then, for each $\langle s_1, d_1 \rangle \in Q$ a copy of the automaton is spawned in state s_1 , and proceeds to the node $x \cdot d_1$ of the input tree. In particular, it requires that $x \cdot d_1$ belongs to the input tree. For example, if $\delta(s, a) = (\langle s_1, 0 \rangle \wedge \langle s_2, 0 \rangle) \vee (\langle s_3, 0 \rangle \wedge \langle s_4, 1 \rangle \wedge \langle s_5, 1 \rangle)$, then the automaton should either spawn two copies that process the successor of x in direction 0 (i.e., the node $x \cdot 0$) and that enter the respective states s_1 and s_2 , or spawn three copies of which one processes $x \cdot 0$ and enters state s_3 , and the other two process $x \cdot 1$ and enter the states s_4 and s_5 respectively.

Runs. A run of \mathcal{A} over an Ω -labeled input tree (T, V) is a tree (T_r, r) labeled by elements of $T \times S$, where a node of T_r labeled by (x, s) corresponds to a copy of the automaton proceeding the node x of the input tree in state s . Formally, a *run* of \mathcal{A} over an input tree (T, V) is a $(T \times S)$ -labeled tree (T_r, r) such that $r(\varepsilon) = (\varepsilon, s_0)$ and for all $y \in T_r$, if $r(y) = (x, s)$, then the set $\{\langle s', d' \rangle \mid \exists d \in \mathbb{N} : r(y \cdot d) = (x \cdot d, s')\}$ is a satisfying assignment for $\delta(s, V(x))$. Hence we require that, given a node y in T_r labeled by (x, s) , there is a satisfying assignment $Q \subseteq S \times [k]$ for the formula $\delta(s, a)$ where $a = V(x)$ is the letter labeling the current node x of the input tree, and for all states $\langle s', d' \rangle \in Q$ there is a (successor) node $y \cdot d$ in T_r labeled by $(x \cdot d', s')$.

Given an accepting condition $\varphi \subseteq S^\omega$, we say that a run (T_r, r) is *accepting* if for all infinite paths $d_1 d_2 \dots$ of T_r , the sequence $s_1 s_2 \dots$ such that $r(d_i) = (\cdot, s_i)$ for all

$i \geq 0$ is in φ . The *language* of \mathcal{A} is the set $L_k(\mathcal{A})$ of all input trees of rank k over which there exists an accepting run of \mathcal{A} . The emptiness problem for alternating tree automata is to decide, given \mathcal{A} and parameter k , whether $L_k(\mathcal{A}) = \emptyset$.

3.3 Solution of the Sure Winning Problem for Three-player Games

Theorem 2. *Given a three-player game $G = \langle S, s_0, \delta, \mathcal{O}, \text{obs} \rangle$ and a parity objective φ , the problem of deciding whether $\exists \sigma \in \Sigma \cdot \exists \tau \in \Gamma \cdot \forall \pi \in \Pi : \rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$ is EXPTIME-complete.*

Proof. The EXPTIME-hardness follows from EXPTIME-hardness of two-player partial-observation games with reachability objective [23].

We prove membership in EXPTIME by a reduction to the emptiness problem for alternating tree automata, which is solvable in EXPTIME for parity objectives [17,18,19]. The reduction is as follows. Given a game $G = \langle S, s_0, \delta, \mathcal{O}, \text{obs} \rangle$ over alphabet of actions A_i ($i = 1, 2, 3$), we construct the alternating tree automaton $\mathcal{A} = \langle S', s'_0, \delta' \rangle$ over alphabet Ω and parameter $k = |\mathcal{O}|$ (we assume that $\mathcal{O} = [k]$) where: (i) $S' = S$, and $s'_0 = s_0$; (ii) $\Omega = A_1$; and (iii) δ' is defined by $\delta'(s, a_1) = \bigvee_{a_3 \in A_3} \bigwedge_{a_2 \in A_2} \langle \delta(s, a_1, a_2, a_3), \text{obs}(\delta(s, a_1, a_2, a_3)) \rangle$ for all $s \in S$ and $a_1 \in \Omega$. The acceptance condition φ of the automaton is the same as the objective of the game G . We prove that $\exists \sigma \in \Sigma \cdot \exists \tau \in \Gamma \cdot \forall \pi \in \Pi : \rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$ if and only if $L_k(\mathcal{A}) \neq \emptyset$. We use the following notation. Given a node $y = d_1 d_2 \dots d_n$ in a $(T \times S)$ -labeled tree (T_r, r) , consider the prefixes $y_0 = \varepsilon$, and $y_i = d_1 d_2 \dots d_i$ (for $i = 1, \dots, n$). Let $\bar{r}_2(y) = s_0 s_1 \dots s_n$ where $r(y_i) = (\cdot, s_i)$ for $0 \leq i \leq n$, denote the corresponding state sequence of y .

1. *Sure winning implies non-emptiness.* First, assume that for some $\sigma \in \Sigma$ and $\tau \in \Gamma$, we have $\forall \pi \in \Pi : \rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$. From σ , we define an input tree (T, V) where $T = [k]^*$ and $V(\gamma) = \sigma(\text{obs}(s_0) \cdot \gamma)$ for all $\gamma \in T$ (we view σ as a function $[k]^+ \rightarrow \Omega$, since $[k] = \mathcal{O}$ and $\Omega = A_1$). From τ , we define a $(T \times S)$ -labeled tree (T_r, r) such that $r(\varepsilon) = (\varepsilon, s_0)$ and for all $y \in T_r$, if $r(y) = (x, s)$ and $\bar{r}_2(y) = \rho$, then for $a_1 = \sigma(\text{obs}(s_0) \cdot x) = V(x)$, for $a_3 = \tau(s_0 \cdot \rho)$, for every s' in the set $Q = \{s' \mid \exists a_2 \in A_2 : s' = \delta(s, a_1, a_2, a_3)\}$, there is a successor $y \cdot d$ of y in T_r labeled by $r(y \cdot d) = (x \cdot \text{obs}(s'), s')$. Note that $\{\langle s', \text{obs}(s') \rangle \mid s' \in Q\}$ is a satisfying assignment for $\delta'(s, a_1)$ and $a_1 = V(x)$, hence (T_r, r) is a run of \mathcal{A} over (T, V) . For every infinite path ρ in (T_r, r) , consider a strategy $\pi \in \Pi$ consistent with ρ . Then $\rho = \rho_{s_0}^{\sigma, \pi, \tau}$, hence $\rho \in \varphi$ and the run (T_r, r) is accepting, showing that $L_k(\mathcal{A}) \neq \emptyset$.
2. *Non-emptiness implies sure winning.* Second, assume that $L_k(\mathcal{A}) \neq \emptyset$. Let $(T, V) \in L_k(\mathcal{A})$ and (T_r, r) be an accepting run of \mathcal{A} over (T, V) . From (T, V) , define a strategy σ of player 1 such that $\sigma(s_0 \cdot \rho) = V(\text{obs}(\rho))$ for all $\rho \in S^*$. Note that σ is indeed observation-based. From (T_r, r) , we know that for all nodes $y \in T_r$ with $r(y) = (x, s)$ and $\bar{r}_2(y) = \rho$, the set $Q = \{\langle s', d' \rangle \mid \exists d \in \mathbb{N} : r(y \cdot d) = (x \cdot d', s')\}$ is a satisfying assignment of $\delta'(s, V(x))$, hence there exists $a_3 \in A_3$ such that for all $a_2 \in A_2$, there is a successor of y labeled by $(x \cdot \text{obs}(s'), s')$ with $s' = \delta(s, a_1, a_2, a_3)$ and $a_1 = \sigma(s_0 \cdot \rho)$. Then define $\tau(s_0 \cdot \rho) = a_3$. Now, for all strategies $\pi \in \Pi$ the outcome $\rho_{s_0}^{\sigma, \pi, \tau}$ is a path in (T_r, r) , and hence $\rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$. Therefore $\exists \sigma \in \Sigma \cdot \exists \tau \in \Gamma \cdot \forall \pi \in \Pi : \rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$. ■

The nonemptiness problem for an alternating tree automaton \mathcal{A} with parity condition can be solved by constructing an equivalent nondeterministic parity tree automaton \mathcal{N} (such that $L_k(\mathcal{A}) = L_k(\mathcal{N})$), and then checking emptiness of \mathcal{N} . The construction proceeds as follows [19]. The nondeterministic automaton \mathcal{N} guess a labeling of the input tree with a memoryless strategy for the alternating automaton \mathcal{A} . As \mathcal{A} has n states and k directions, there are (k^n) possible strategies. A nondeterministic parity word automaton with n states and d priorities can check that the strategy works along every branch of the tree. An equivalent deterministic parity word automaton can be constructed with (n^n) states and $O(d \cdot n)$ priorities [4]. Thus, \mathcal{N} can guess the strategy labeling and check the strategies with $O((k \cdot n)^n)$ states and $O(d \cdot n)$ priorities. The nonemptiness of \mathcal{N} can then be checked by considering it as a (two-player perfect-information deterministic) parity game with $O((k \cdot n)^n)$ states and $O(d \cdot n)$ priorities [15]. This games can be solved in time $O((k \cdot n)^{d \cdot n^2})$ [14]. Moreover, since memoryless strategies exist for parity games [14], if the nondeterministic parity tree automaton is nonempty, then it accepts a regular tree that can be encoded by a transducer with $((k \cdot n)^n)$ states. Thus, the nonemptiness problem for alternating tree automaton with parity condition can be decided in exponential time, and there exists a transducer to witness nonemptiness that has exponentially many states.

Theorem 3. *Given a three-player game $G = \langle S, s_0, \delta, \mathcal{O}, \text{obs} \rangle$ with n states (and $k \leq n$ observations for player 1) and parity objective φ defined by d priorities, the problem of deciding whether $\exists \sigma \in \Sigma \cdot \exists \tau \in \Gamma \cdot \forall \pi \in \Pi : \rho_{s_0}^{\sigma, \pi, \tau} \in \varphi$ can be solved in time exponential time. Moreover, memory of exponential size is sufficient for player 1.*

Remark 4. By our reduction to alternating parity tree automata and the fact that if an alternating parity tree automaton is non-empty, there is a regular witness tree for non-emptiness it follows that strategies for player 1 can be restricted to finite-memory without loss of generality. This ensures that we can solve the problem of the existence of finite-memory almost-sure winning (resp. positive winning) strategies in partial-observation stochastic parity games (by Theorem 1 of Section 2) also in EXPTIME, and EXPTIME-completeness of the problem follows since the problem is EXPTIME-hard even for reachability objectives for almost-sure winning [10] and safety objectives for positive winning [9].

Theorem 4. *Given a partial-observation stochastic game and a parity objective φ defined by d priorities, the problem of deciding whether there exists a finite-memory almost-sure (resp. positive) winning strategy for player 1 is EXPTIME-complete. Moreover, if there is an almost-sure (resp. positive) winning strategy, then there exists one that uses memory of at most exponential size.*

Remark 5. As mentioned in Remark 2 the EXPTIME upper bound for qualitative analysis of partial-observation stochastic parity games with finite-memory randomized strategies follows from Theorem 4. The EXPTIME lower bound and the exponential lower bound on memory requirement for finite-memory randomized strategies follows from the results of [10,9] for reachability and safety objectives (even for POMDPs).

References

1. Full version of the paper. CoRR, abs/1401.3289 (2014)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. *JACM* 49, 672–713 (2002)
3. 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)
4. Cai, Y., Zhang, T.: Determinization complexities of ω automata, Technical report (2013), <http://theory.stanford.edu/~tingz/tcs.pdf>
5. Chatterjee, K.: Stochastic ω -regular Games. PhD thesis, UC Berkeley (2007)
6. Chatterjee, K., Chmelik, M., Tracol, M.: What is decidable about partially observable Markov decision processes with omega-regular objectives. In: CSL (2013)
7. Chatterjee, K., Doyen, L.: Partial-observation stochastic games: How to win when belief fails. In: LICS, pp. 175–184. IEEE (2012)
8. Chatterjee, K., Doyen, L., Gimbert, H., Henzinger, T.A.: Randomness for free. CoRR abs/1006.0673 (Full version) (2010); Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 246–257. Springer, Heidelberg (2010)
9. Chatterjee, K., Doyen, L., Henzinger, T.A.: Qualitative analysis of partially-observable Markov decision processes. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 258–269. Springer, Heidelberg (2010)
10. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Algorithms for omega-regular games of incomplete information. *Logical Methods in Computer Science* 3(3:4) (2007)
11. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
12. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. *JACM* 42(4), 857–907 (1995)
13. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford Univ. (1997)
14. Emerson, E.A., Jutla, C.: Tree automata, mu-calculus and determinacy. In: FOCS, pp. 368–377. IEEE (1991)
15. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: STOC, pp. 60–65 (1982)
16. McNaughton, R.: Infinite games played on finite graphs. *Annals of Pure and Applied Logic* 65, 149–184 (1993)
17. Muller, D.E., Saoudi, A., Schupp, P.E.: Alternating automata, The weak monadic theory of the tree, and its complexity. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 275–283. Springer, Heidelberg (1986)
18. Muller, D.E., Schupp, P.E.: Alternating automata on infinite trees. *TCS* 54, 267–276 (1987)
19. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *TCS* 141(1&2), 69–107 (1995)
20. Nain, S., Vardi, M.Y.: Solving partial-information stochastic parity games. In: LICS, pp. 341–348. IEEE (2013)
21. Paz, A.: Introduction to probabilistic automata. Academic Press (1971)
22. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190. ACM (1989)
23. Reif, J.H.: The complexity of two-player games of incomplete information. *JCSS* 29, 274–301 (1984)
24. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, Beyond Words, vol. 3, ch. 7, pp. 389–455. Springer (1997)
25. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state systems. In: Proc. of FOCS, pp. 327–338 (1985)

On Negotiation as Concurrency Primitive II: Deterministic Cyclic Negotiations

Javier Esparza¹ and Jörg Desel²

¹ Fakultät für Informatik, Technische Universität München, Germany

² Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Germany

Abstract. We continue our study of negotiations, a concurrency model with multi-party negotiation as primitive. In a previous paper [7] we have provided a correct and complete set of reduction rules for sound, acyclic, and (weakly) deterministic negotiations. In this paper we extend this result to all deterministic negotiations, including cyclic ones. We also show that this set of rules allows one to decide soundness and to summarize negotiations in polynomial time.

1 Introduction

Negotiation has long been identified as a paradigm for process interaction [5]. It has been applied to different problems (see e.g. [17,2]), and studied on its own [15]. However, there is only little research on negotiations from a concurrency-theoretic point of view. Some works model the behaviour of a negotiation party using business process languages or Petri net, and model negotiation protocols as the concurrent composition of the parties [4,18,16]. In contrast, in [7] we have introduced a formalism that considers each elementary (multiparty) negotiation a single atom (graphically represented by a node) and model a distributed negotiation as a composition of atoms.

Observationally, an elementary negotiation (an atom) is an interaction in which several partners come together to agree on one out of a number of possible outcomes (a synchronized nondeterministic choice). Each possible outcome has associated a state-transformer. Negotiation partners enter the atom in certain states, and leave it in the states obtained by applying to these states the state-transformer of the outcome agreed upon. Atoms are combined into more complex, distributed negotiations by means of a next-atoms function that determines, for each atom, negotiating agent, and outcome, the set of atoms the agent is ready to engage in next if the atom ends with that outcome.

Like in workflow nets [1], distributed negotiations can be *unsound* because of deadlocks or livelocks. The *soundness* problem consists of deciding if a given negotiation is sound. Moreover, a sound negotiation is equivalent to a single atom whose state transformation function determines the possible final states of all parties as a function of their initial states. The *summarization problem* consists of computing such an atomic negotiation, called a *summary*.

Negotiations can simulate 1-safe Petri nets (see the arXiv version of [7]), which proves that the soundness problem and (a decision version of) the summarization problem are, unsurprisingly, PSPACE-complete. We have studied in [7] two subclasses: *deterministic* and *weakly deterministic* negotiations. Both have limited expressive power in comparison to general negotiations, but have natural semantic justifications (see [7]). Only deterministic negotiations are relevant for this paper. Loosely speaking, a negotiation is deterministic if, for each agent and each outcome of an atomic negotiation, the next-atom function yields only one next atom, i.e., each agent can always engage in one atom only.

We have shown in [7] that the soundness and summarization problems for *acyclic* deterministic negotiations can be solved in polynomial time. The algorithm progressively reduces the graphical representation of a negotiation to a simpler one by means of reduction rules. Each rule preserves soundness and summaries (i.e., the negotiation before the application of the rule is sound iff the negotiation after the application is sound, and both have the same summary). Reduction rules have been extensively applied to Petri nets or workflow nets, but most of this work has been devoted to the liveness or soundness problems [3,12,13,11,6], and many rules do not preserve summaries.

In [7] we conjectured that the addition of a simple rule allowing one to reduce trivial cycles yields a complete set of rules for all sound deterministic negotiations. In this paper we prove this result, and we show that the number of rule applications required to summarize a negotiation is still polynomial. While the new rule is very simple, the proof of our result is involved. It is structured in several sections, and some technical proofs have been moved to an extended version of this paper [8]. Section 2 presents the main definitions of [7] in compact form. Section 3 introduces our reduction rules. Section 4 proves that the rules summarize all sound deterministic negotiations. Section 5 proves that the summarization of a sound negotiation requires a polynomial number of steps.

2 Negotiations: Syntax and Semantics

We fix a finite set A of *agents*. Each agent $a \in A$ has a (possibly infinite) nonempty set Q_a of *internal states*. We denote by Q_A the cartesian product $\prod_{a \in A} Q_a$. A *transformer* is a left-total relation $\tau \subseteq Q_A \times Q_A$. Given $S \subseteq A$, we say that a transformer τ is an *S-transformer* if, for each $a_i \notin S$, $\left((q_{a_1}, \dots, q_{a_i}, \dots, q_{a_{|A|}}), (q'_{a_1}, \dots, q'_{a_i}, \dots, q'_{a_{|A|}}) \right) \in \tau$ implies $q_{a_i} = q'_{a_i}$. So an *S-transformer* only transforms the internal states of agents in S .

Definition 1. A negotiation atom, or just an atom, is a triple $n = (P_n, R_n, \delta_n)$, where $P_n \subseteq A$ is a nonempty set of parties, R_n is a finite, nonempty set of outcomes, and δ_n is a mapping assigning to each outcome r in R_n a P_n -transformer $\delta_n(r)$.

Intuitively, if the states of the agents before a negotiation n are given by a tuple q and the outcome of the negotiation is r , then the agents change their states to q' for some $(q, q') \in \delta_n(r)$.

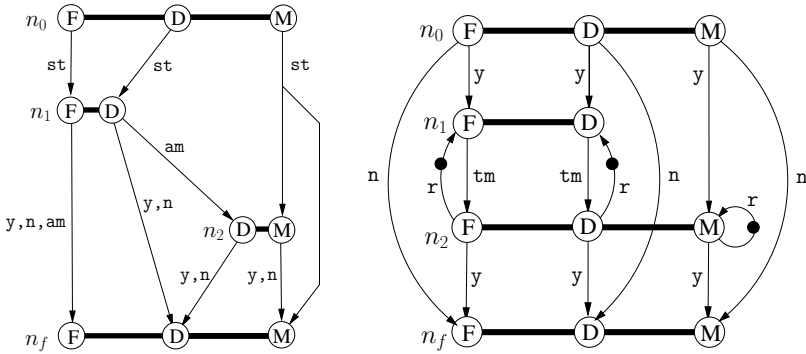


Fig. 1. Acyclic and cyclic negotiations

For a simple example, consider a negotiation atom n_{FD} with parties F (Father) and D (teenage Daughter). The goal of the negotiation is to determine whether D can go to a party, and the time at which she must return home. The possible outcomes are **yes** and **no**. Both sets Q_F and Q_D contain a state \perp plus a state t for every time $T_1 \leq t \leq T_2$ in a given interval $[T_1, T_2]$. Initially, F is in state t_f and D in state t_d . The transformer $\delta_{n_{FD}}$ is given by:

$$\begin{aligned} \delta_{n_{fd}}(\text{yes}) &= \{((t_f, t_d), (t, t)) \mid t_f \leq t \leq t_d \vee t_d \leq t \leq t_f\} \\ \delta_{n_{fd}}(\text{no}) &= \{((t_f, t_d), (\perp, \perp))\} \end{aligned}$$

2.1 Combining Atomic Negotiations

A negotiation is a composition of atoms. We add a *transition function* \mathcal{X} that assigns to every triple (n, a, r) consisting of an atom n , a party a of n , and an outcome r of n a set $\mathcal{X}(n, a, r)$ of atoms. Intuitively, this is the set of atomic negotiations agent a is ready to engage in after the atom n , if the outcome is r .

Definition 2. Given a finite set of atoms N , let $T(N)$ denote the set of triples (n, a, r) such that $n \in N$, $a \in P_n$, and $r \in R_n$. A negotiation is a tuple $\mathcal{N} = (N, n_0, n_f, \mathcal{X})$, where $n_0, n_f \in N$ are the initial and final atoms, and $\mathcal{X}: T(N) \rightarrow 2^N$ is the transition function. Further, \mathcal{N} satisfies the following properties:

- (1) every agent of \mathcal{A} participates in both n_0 and n_f ;
- (2) for every $(n, a, r) \in T(N)$: $\mathcal{X}(n, a, r) = \emptyset$ iff $n = n_f$.

Negotiations are graphically represented as shown in Figure 1. For each atom $n \in N$ we draw a black bar; for each party a of P_n we draw a white circle on the bar, called a *port*. For each $(n, a, r) \in T(N)$, we draw a hyperarc leading from the port of a in n to all the ports of a in the atoms of $\mathcal{X}(n, a, r)$, and label it by r . Figure 1 shows two Father-Daughter-Mother negotiations. On the left, Daughter and Father negotiate with possible outcomes **yes** (y), **no** (n), and **ask_mother** (am). If the outcome is the latter, then Daughter and Mother negotiate with outcomes **yes**, **no**. In the negotiation on the right, Father, Daughter and Mother

negotiate with outcomes **yes** and **no**. If the outcome is **yes**, then Father and Daughter negotiate a return time (atom n_1) and propose it to Mother (atom n_2). If Mother approves (outcome **yes**), then the negotiation terminates, otherwise (outcome **r**) Daughter and Father renegotiate the return time. For the sake of brevity we do not describe the transformers of the atoms.

Definition 3. *The graph associated to a negotiation $\mathcal{N} = (N, n_0, n_f, \mathcal{X})$ is the directed graph with vertices N and edges $\{(n, n') \mid \exists (n, a, r) \in T(N): n' \in \mathcal{X}(n, a, r)\}$. \mathcal{N} is acyclic if its graph has no cycles, otherwise it is cyclic.*

The negotiation on the left of Figure 1 is acyclic, the one the right is cyclic.

2.2 Semantics

A *marking* of a negotiation $\mathcal{N} = (N, n_0, n_f, \mathcal{X})$ is a mapping $\mathbf{x}: A \rightarrow 2^N$. Intuitively, $\mathbf{x}(a)$ is the set of atoms that agent a is currently ready to engage in next. The *initial* and *final* markings, denoted by \mathbf{x}_0 and \mathbf{x}_f respectively, are given by $\mathbf{x}_0(a) = \{n_0\}$ and $\mathbf{x}_f(a) = \emptyset$ for every $a \in A$.

A marking \mathbf{x} *enables* an atom n if $n \in \mathbf{x}(a)$ for every $a \in P_n$, i.e., if every party of n is currently ready to engage in it. If \mathbf{x} enables n , then n can take place and its parties agree on an outcome r ; we say that (n, r) *occurs*. Abusing language, we will call this pair also an outcome. The occurrence of (n, r) produces a next marking \mathbf{x}' given by $\mathbf{x}'(a) = \mathcal{X}(n, a, r)$ for $a \in P_n$, and $\mathbf{x}'(a) = \mathbf{x}(a)$ for $a \in A \setminus P_n$. We write $\mathbf{x} \xrightarrow{(n,r)} \mathbf{x}'$ to denote this, and call it a *small step*.

By this definition, $\mathbf{x}(a)$ is always either $\{n_0\}$ or equals $\mathcal{X}(n, a, r)$ for some atom n and outcome r . The marking \mathbf{x}_f can only be reached by the occurrence of (n_f, r) (r being a possible outcome of n_f), and it does not enable any atom. Any other marking that does not enable any atom is considered a *deadlock*.

Reachable markings are graphically represented by tokens (dots) on arcs (on forking points of hyperarcs, respectively). Figure 1 shows on the right a marking in which F and D are ready to engage in n_1 and M is ready to engage in n_2 .

We write $\mathbf{x}_1 \xrightarrow{\sigma}$ to denote that there is a sequence of small steps

$$\mathbf{x}_1 \xrightarrow{(n_1, r_1)} \mathbf{x}_2 \xrightarrow{(n_2, r_2)} \dots \xrightarrow{(n_{k-1}, r_{k-1})} \mathbf{x}_k \xrightarrow{(n_k, r_k)} \mathbf{x}_{k+1} \dots$$

such that $\sigma = (n_1, r_1) \dots (n_k, r_k) \dots$. If $\mathbf{x}_1 \xrightarrow{\sigma}$, then σ is an *occurrence sequence* enabled by \mathbf{x}_1 . If σ is finite, then we write $\mathbf{x}_1 \xrightarrow{\sigma} \mathbf{x}_{k+1}$ and call \mathbf{x}_{k+1} *reachable* from \mathbf{x}_1 . If \mathbf{x}_1 is the initial marking, then we call σ *initial occurrence sequence*. If moreover \mathbf{x}_{k+1} is the final marking, then σ is a *large step*.

A negotiation can be associated an equivalent Petri net with the same occurrence sequences (see [7], arXiv version). However, the Petri net can be exponentially larger than the negotiation.

2.3 Soundness

Following [1], we introduce a notion of well-formedness of a negotiation:

Definition 4. A negotiation is sound if (a) every atom is enabled at some reachable marking, and (b) every occurrence sequence from the initial marking is either a large step or can be extended to a large step.

The negotiations of Figure 1 are sound. However, if we set in the left negotiation $\mathcal{X}(n_0, \mathbf{M}, \mathbf{st}) = \{n_{\text{DM}}\}$ instead of $\mathcal{X}(n_0, \mathbf{M}, \mathbf{st}) = \{n_{\text{DM}}, n_f\}$, then the occurrence sequence $(n_0, \mathbf{st})(n_{\text{FD}}, \mathbf{yes})$ leads to a deadlock.

Definition 5. Given a negotiation $\mathcal{N} = (N, n_0, n_f, \mathcal{X})$, we attach to each outcome r of n_f a summary transformer $\langle \mathcal{N}, r \rangle$ as follows. Let E_r be the set of large steps of \mathcal{N} that end with (n_f, r) . We define $\langle \mathcal{N}, r \rangle = \bigcup_{\sigma \in E_r} \langle \sigma \rangle$, where for $\sigma = (n_1, r_1) \dots (n_k, r_k)$ we define $\langle \sigma \rangle = \delta_{n_1}(r_1) \dots \delta_{n_k}(r_k)$ (each $\delta_{n_i}(r_i)$ is a relation on Q_A ; concatenation is the usual concatenation of relations).

$\langle \mathcal{N}, r \rangle(q_0)$ is the set of possible final states of the agents after the negotiation concludes with outcome r , if their initial states are given by q_0 .

Definition 6. Two negotiations \mathcal{N}_1 and \mathcal{N}_2 over the same set of agents are equivalent if they are either both unsound, or if they are both sound, have the same final outcomes (outcomes of the final atom), and $\langle \mathcal{N}_1, r \rangle = \langle \mathcal{N}_2, r \rangle$ for every final outcome r . If \mathcal{N}_1 and \mathcal{N}_2 are equivalent and \mathcal{N}_2 consists of a single atom then \mathcal{N}_2 is the summary of \mathcal{N}_1 .

According to this definition, all unsound negotiations are equivalent: if soundness fails, we do not care about the rest. However, an unsound negotiation can have occurrence sequences from the initial to the final marking, and two unsound (and thus equivalent) negotiations may have different such occurrence sequences.

Definition 7. A negotiation \mathcal{N} is deterministic if for every $(n, a, r) \in T(N)$ there is an atom n' such that $\mathcal{X}(n, a, r) = \{n'\}$

Graphically, a negotiation is deterministic if there are no proper hyperarcs. The negotiation on the left of Figure 1 is not deterministic (it contains a proper hyperarc for Mother), while the one on the right is deterministic. In the sequel, we often assume that a negotiation is sound and deterministic, and abbreviate “sound and deterministic negotiation” to SDN. For deterministic negotiations we write $\mathcal{X}(n, a, r) = n'$ instead of $\mathcal{X}(n, a, r) = \{n'\}$.

3 Reduction Rules for Deterministic Negotiations

We present three equivalence-preserving reduction rules for negotiations. Two of them were already introduced in [7] (in a slightly more general version), while the iteration rule is new. Here we only consider deterministic negotiations.

A *reduction rule*, or just a rule, is a binary relation on the set of negotiations. Given a rule R , we write $\mathcal{N}_1 \xrightarrow{R} \mathcal{N}_2$ for $(\mathcal{N}_1, \mathcal{N}_2) \in R$. A rule R is *correct* if it preserves equivalence, i.e., if $\mathcal{N}_1 \xrightarrow{R} \mathcal{N}_2$ implies $\mathcal{N}_1 \equiv \mathcal{N}_2$. This implies that \mathcal{N}_1 is sound iff \mathcal{N}_2 is sound. Given a set of rules $\mathcal{R} = \{R_1, \dots, R_k\}$, we denote

by \mathcal{R}^* the reflexive and transitive closure of $R_1 \cup \dots \cup R_k$. We call \mathcal{R} *complete with respect to a class of negotiations* if, for every negotiation \mathcal{N} in the class, there is a negotiation \mathcal{N}' consisting of a single atom such that $\mathcal{N} \xrightarrow{\mathcal{R}^*} \mathcal{N}'$. We describe rules as pairs of a *guard* and an *action*; $\mathcal{N}_1 \xrightarrow{R} \mathcal{N}_2$ holds if \mathcal{N}_1 satisfies the guard and \mathcal{N}_2 is a possible result of applying the action to \mathcal{N}_1 .

Merge rule. Intuitively, the *merge rule* merges two outcomes with identical next enabled atoms into one single outcome.

Definition 8. *Merge rule*

Guard: N contains an atom n with two distinct outcomes $r_1, r_2 \in R_n$ such that $\mathcal{X}(n, a, r_1) = \mathcal{X}(n, a, r_2)$ for every $a \in A_n$.

Action: (1) $R_n \leftarrow (R_n \setminus \{r_1, r_2\}) \cup \{r_f\}$, where r_f is a fresh name.
 (2) For all $a \in P_n$: $\mathcal{X}(n, a, r_f) \leftarrow \mathcal{X}(n, a, r_1)$.
 (3) $\delta(n, r_f) \leftarrow \delta(n, r_1) \cup \delta(n, r_2)$.

Shortcut rule. Intuitively, the *shortcut rule* merges the outcomes of two atoms that can occur one after the other into one single outcome with the same effect. The examples in Figure 6 illustrate the definition (ignore the big circles for the moment): in both negotiations the outcome (n, r'_f) is a “shortcut” of the outcome (n, r) followed by (n', r') .

Definition 9. Given atoms n, n' , we say that (n, r) unconditionally enables n' if $P_n \supseteq P_{n'}$ and $\mathcal{X}(n, a, r) = n'$ for every $a \in P_{n'}$.

Observe that, if (n, r) unconditionally enables n' , then, for every marking \mathbf{x} that enables n , the marking \mathbf{x}' given by $\mathbf{x} \xrightarrow{(n,r)} \mathbf{x}'$ enables n' . Moreover, n' can only be disabled by its own occurrence.

Definition 10. *Shortcut rule for deterministic negotiations*

Guard: N contains an atom n with an outcome r and an atom n' , $n' \neq n$, such that (n, r) unconditionally enables n' .

Action: (1) $R_n \leftarrow (R_n \setminus \{r\}) \cup \{r'_f \mid r' \in R_{n'}\}$, where r'_f are fresh names.
 (2) For all $a \in P_{n'}$, $r' \in R_{n'}$: $\mathcal{X}(n, a, r'_f) \leftarrow \mathcal{X}(n', a, r')$.
 For all $a \in P \setminus P_{n'}$, $r' \in R_{n'}$: $\mathcal{X}(n, a, r'_f) \leftarrow \mathcal{X}(n, a, r)$.
 (3) For all $r' \in R_{n'}$: $\delta_n(r'_f) \leftarrow \delta_n(r)\delta_{n'}(r')$.
 (4) If $\mathcal{X}^{-1}(n') = \emptyset$ after (1)-(3), then remove n' from N , where $\mathcal{X}^{-1}(n') = \{(\tilde{n}, \tilde{a}, \tilde{r}) \in T(N) \mid n' \in \mathcal{X}(\tilde{n}, \tilde{a}, \tilde{r})\}$.

Iteration rule. Loosely speaking, the *iteration rule* replaces the iteration of a negotiation by one single atom with the same effect.

Definition 11. *Iteration rule*

Guard: N contains an atom n with an outcome r such that $\mathcal{X}(n, a, r) = n$ for every party a of n .

Action: (1) $R_n \leftarrow \{r'_f \mid r' \in R_n \setminus \{r\}\}$.
 (2) For every $r'_f \in R_n$: $\delta_n(r'_f) \leftarrow \delta_n(r)^* \delta_n(r')$.

Proposition 1. *If the application of the shortcut, merge or iteration rule to a deterministic negotiation \mathcal{N} yields negotiation \mathcal{N}' then \mathcal{N}' is deterministic, too.*

Theorem 1. *The merge, shortcut, and iteration rules are correct.*

Proof. Correctness of the merge and iteration rules is obvious. The correctness of a more general version of the shortcut rule is proved in [7]¹. \square

4 Completeness

In [7] we show that every sound and weakly deterministic acyclic negotiation can be summarized to a single atom, and that in the deterministic case the number of rule applications is polynomial (actually, [7] provides a sharper bound than the one in this theorem):

Theorem 2 ([7]). *Every sound deterministic acyclic negotiation \mathcal{N} can be reduced to a single atom by means of $|N|^2 + |\text{Out}(\mathcal{N})|$ applications of the merge and shortcut rules, where N is the set of atoms of \mathcal{N} , and $\text{Out}(\mathcal{N})$ is the set of all outcomes of all atoms of \mathcal{N} .*

In the rest of the paper we prove that, surprisingly, the addition of the very simple iteration rule suffices to extend this result to cyclic deterministic negotiations, although with a higher exponent. The argument is complex and requires a detailed analysis of the structure of SDNs.

In this section we present the completeness proof, while the complexity result is presented in the next. We illustrate the reduction algorithm by means of an example. Figure 2 (a) shows a cyclic SDN similar to the Father-Daughter-Mother negotiation on the right of Figure 1. We identify an “almost acyclic” fragment, given by atom n_2 with outcome a and atom n_4 with outcome b . Intuitively, “almost acyclic” means that the fragment can be obtained by “merging” the initial and final atoms of an acyclic SDN; in our example, this is the acyclic SDN shown in Figure 2 (b). This acyclic SDN can be summarized using the shortcut and merge rules. If we apply the same sequence of rules to the fragment mentioned before (with the exception of the last rule, which reduces a negotiation with two different atoms and one single outcome to an atomic negotiation) we obtain the negotiation shown in (c). The self-loop can now be eliminated with the help of the iteration rule, and the procedure can be iterated: we again identify an “almost acyclic” fragment, (d) shows the corresponding acyclic SDN. Its reduction yields the the negotiation shown in (e). The self-loops are eliminated by the iteration rule, yielding an acyclic negotiation, which can be summarized.

In order to prove completeness we must show that every cyclic SDN contains at least one almost acyclic fragment, which is non-trivial. The proof has three parts: We first show that every cyclic SDN has a *loop*: an occurrence sequence from some reachable marking \mathbf{x} back to \mathbf{x} . Then we show that each minimal

¹ The rule of [7] has an additional condition in the guard which is always true for deterministic negotiations.

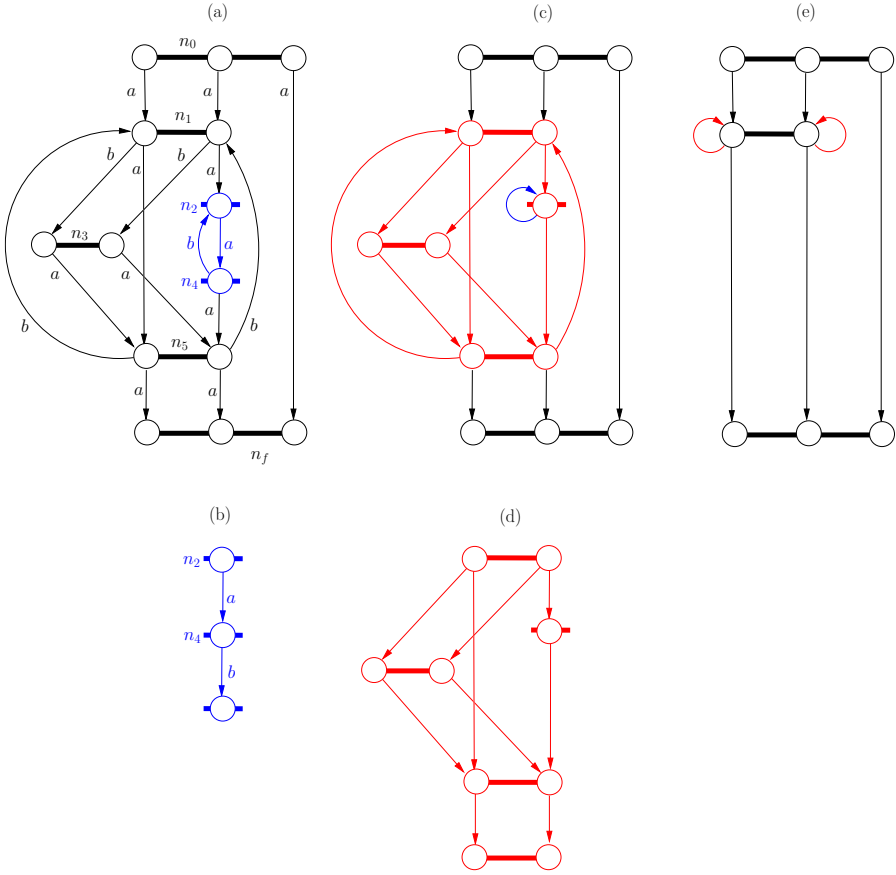


Fig. 2. The reduction procedure

loop has a *synchronizer*: an atom involving each agent that is party of any atom of the loop. Finally we show how to use synchronizers to identify a nonempty and almost acyclic fragment.

4.1 Lassos and Loops

Definition 12. A lasso of a negotiation is a pair (ρ, σ) of occurrence sequences such that σ is not the empty sequence and $\mathbf{x}_0 \xrightarrow{\rho} \mathbf{x} \xrightarrow{\sigma} \mathbf{x}$ for some marking \mathbf{x} . A loop is an occurrence sequence σ such that (ρ, σ) is a lasso for some ρ . A minimal loop is a loop σ satisfying the property that there is no other loop σ' such that the set of atoms in σ' is a proper subset of the set of atoms in σ .

Observe that lassos and loops are behavioural notions, i.e., structures of the reachability graph of a negotiation. The following result establishes relations between loops and cycles, where cycles are defined on the graph of a negotiation.

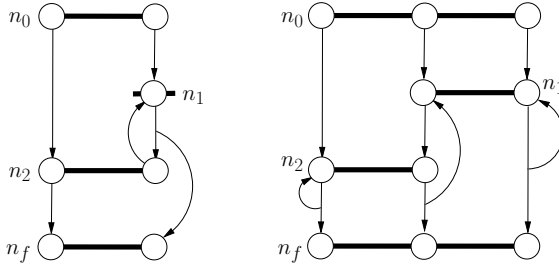


Fig. 3. Two sound and cyclic negotiations

Lemma 1. (1) Every cyclic SDN has a loop.
 (2) The set of atoms of a minimal loop generates a strongly connected subgraph of the graph of the considered negotiation.

Proof. See [8]. □

4.2 Synchronizers

Definition 13. A loop $\sigma = (n_1, r_1) \dots (n_k, r_k)$ is synchronized if there is an atom n_i in σ such that $P_j \subseteq P_i$ for $1 \leq j \leq k$, i.e., every party of every atom in the loop is also a party of n_i . We call n_i a synchronizer of the loop. An atom is a synchronizer of a negotiation if it is a synchronizer of at least one of its loops.

Each loop $\mathbf{x} \xrightarrow{(n,r)} \mathbf{x}$ is synchronized. In the graph of a negotiation, such a loop appears as a self-loop, i.e., as an edge from atom n to atom n .

Some of the loops of the SDN shown in Figure 2 (a) are $(n_1, a) (n_2, a) (n_4, a) (n_5, b)$, $(n_1, b) (n_3, a) (n_5, b)$, and $(n_2, a) (n_4, b)$. The first loop is synchronized by (n_1, a) and by (n_5, b) , the two others are synchronized by all their outcomes.

The main result of this paper is strongly based on the following lemma.

Lemma 2. Every minimal loop of a SDN is synchronized.

Proof. See [8] □

The negotiation on the left in Figure 3 shows that Lemma 1(1) holds only in the deterministic case. It is sound and cyclic, but has no loops, because the only big step is $n_0 n_1 n_2 n_1 n_f$ (all atoms have only one outcome, whose name is omitted).

Lemma 2 does not hold for arbitrary (i.e., non-deterministic) sound negotiations. For the negotiation on the right of Figure 3 (the name of the outcome is again omitted), the sequence $n_1 n_2$ is a loop without synchronizers.

4.3 Fragments

We assign to each atom n of an SDN a “fragment” \mathcal{F}_n as follows: we take all the loops synchronized by n , and (informally) define \mathcal{F}_n as the atoms and outcomes that appear in these loops. Figure 4 (a) and (c) show \mathcal{F}_{n_1} and \mathcal{F}_{n_2} for the SDN of Figure 2. Since a cyclic SDN has at least one loop and hence also a minimal one, and since every loop has a synchronizer, at least one of the fragments of a cyclic SDN is nonempty.

Given a fragment \mathcal{F}_n , let \mathcal{N}_n denote the negotiation obtained by, intuitively, “splitting” the atom n into an initial and a final atom. Figure 4 (b) and (d) show the “splittings” \mathcal{N}_{n_1} and \mathcal{N}_{n_2} of \mathcal{F}_{n_1} and \mathcal{F}_{n_2} . Not all fragments are almost acyclic. For instance, \mathcal{N}_{n_1} is not acyclic, and so \mathcal{F}_{n_1} is not almost acyclic. However, we prove that if a fragment is not almost acyclic, then it contains a smaller fragment (for instance, \mathcal{F}_{n_1} contains \mathcal{F}_{n_2}). This shows that every minimal fragment is almost acyclic.

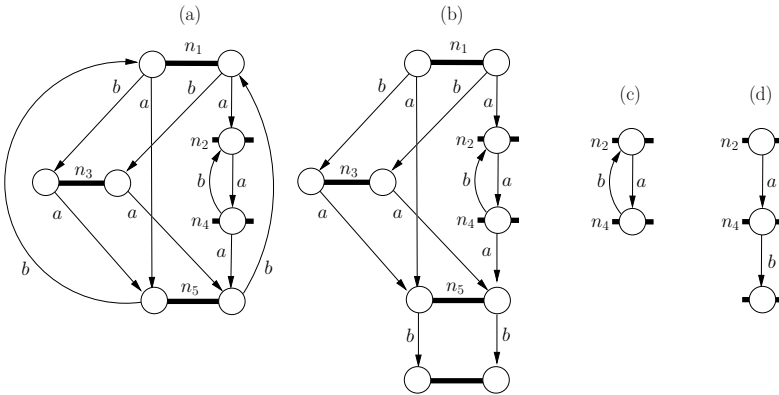


Fig. 4. Fragments of the SDN of Figure 2(a) and their “splittings”

Definition 14. Let \mathcal{L} be a set of loops of \mathcal{N} . Abusing language, we write $(n, r) \in \mathcal{L}$ resp. $n \in \mathcal{L}$ to denote that (n, r) resp. n appears in some loop of \mathcal{L} . The projection of an atom $n = (P_n, R_n, \delta_n) \in \mathcal{L}$ onto \mathcal{L} is the atom $n_{\mathcal{L}} = (P_{\mathcal{L}}, R_{\mathcal{L}}, \delta_{\mathcal{L}})$, where $P_{\mathcal{L}} = P_n$, $R_{\mathcal{L}} = \{r \mid (n, r) \in \mathcal{L}\}$, and $\delta_{\mathcal{L}}((n_{\mathcal{L}}, r)) = \delta((n, r))$ for every $(n, r) \in \mathcal{L}$.

Definition 15. Let s be an atom of a negotiation \mathcal{N} , and let \mathcal{L} be the set of loops synchronized by s . The s -fragment of \mathcal{N} is the pair $\mathcal{F}_s = (F_s, \mathcal{X}_s)$, where $F_s = \{n_{\mathcal{L}} \mid n \in \mathcal{L}\}$ and $\mathcal{X}_s(n_{\mathcal{L}}, a, r) = \mathcal{X}(n, a, r)$ for every $a \in P_{\mathcal{L}}$ and $r \in R_{\mathcal{L}}$.

The s -negotiation of \mathcal{N} is the negotiation $\mathcal{N}_s = (N_s, n_{s0}, n_{sf}, \mathcal{X}'_s)$, where N_s contains the atoms of F_s plus a fresh atom n_{sf} ; $n_{s0} = s_{\mathcal{L}}$; and

$$\mathcal{X}'_s(n_{\mathcal{L}}, a, r) = \begin{cases} \mathcal{X}(n, a, r) & \text{if } \mathcal{X}(n, a, r) \neq s \\ n_{sf} & \text{otherwise} \end{cases}$$

Lemma 3. *A cyclic SDN contains an atom n such that \mathcal{N}_n is an acyclic SDN.*

Proof. See [8] □

The example on the left of Figure 3 shows that this result does not hold for the non-deterministic case.

4.4 The Reduction Procedure

We can now formulate a reduction procedure to summarize an arbitrary SDN.

Input: a deterministic negotiation \mathcal{N}_0 ;

- 1 $\mathcal{N} \leftarrow$ result of exhaustively applying the merge rule to \mathcal{N}_0 ;
- 2 **while** \mathcal{N} is cyclic **do**
- 3 select $s \in N$ such that \mathcal{N}_s is acyclic;
- 4 apply to \mathcal{N} the sequence of rules used to summarize \mathcal{N}_s (but the last);
- 5 apply the iteration rule to s ;
- 6 exhaustively apply the merge rule
- 7 apply the reduction sequence of Theorem 2

Theorem 3. *The reduction procedure returns a summary of \mathcal{N}_0 iff \mathcal{N}_0 is sound.*

Proof. By induction on the number k of atoms of \mathcal{N} that synchronize at least one loop. If $k = 0$, then by Lemma 1 and 2 \mathcal{N} is acyclic, and the result follows from Theorem 2. If $k > 0$, then by Lemma 3 \mathcal{N} contains an almost acyclic fragment \mathcal{F}_s , and so \mathcal{N}_s is acyclic. Since the sequence of rules of line 4 summarizes \mathcal{N}_s , its application to \mathcal{N} ends with a negotiation having a unique self-loop-outcome on s . After removing this outcome with the iteration rule in line 5, we obtain a SDN with $k - 1$ synchronizers, which can be summarized by induction hypothesis (line 6 is not necessary for completeness, but required for the complexity result of the next section). □

5 Complexity

We analyze the number of rule applications required by the reduction procedure. Let $\mathcal{N}_i = (\mathcal{N}_i, n_{0i}, n_{fi}, \mathcal{X}_i)$ be the negotiation before the i -th execution of the while-loop. We next collect some basic properties of the sequence $\mathcal{N}_1, \mathcal{N}_2, \dots$

Lemma 4. *For every $i \geq 1$: (a) $\mathcal{N}_{i+1} \subseteq \mathcal{N}_i$; (b) the merge rule cannot be applied to \mathcal{N}_i ; and (c) \mathcal{N}_{i+1} has fewer synchronizers than \mathcal{N}_i .*

In particular, by (c) the while loop is executed at most $|\mathcal{N}_1| = |\mathcal{N}_0|$ times.

Proof. (a) and (b) follow immediately from the definitions of the rules and the reduction algorithm. For (c), we observe that every synchronizer of \mathcal{N}_{i+1} is a synchronizer of \mathcal{N}_i , but the atom s selected at the i -th loop execution is not a synchronizer of \mathcal{N}_{i+1} , because all loops synchronized by s are collapsed to self-loops on s during the i -th iteration of the loop, and then removed by the iteration rule. □

By Theorem 2, during the i -th iteration of the while-loop line 4 requires at most $|N_i|^2 + |Out(N_i)|$ rule applications. Line 5 only requires one application. Now, let N'_i be the negotiation obtained after the execution of line 5. The number of rule applications of line 6 is clearly bounded by the number of outcomes of $Out(N'_i)$. For the total number of rule applications $Appl(N_0)$ we then obtain

$$\begin{aligned}
 Appl(N_0) &\leq \sum_{i=1}^{|N_0|} (|N_i|^2 + |Out(N_i)| + 1 + |Out(N'_i)|) && \text{Lemma 4(c) and} \\
 & && \text{Theorem 2} \\
 &\leq \sum_{i=1}^{|N_0|} (|N_0|^2 + 1 + |Out(N_i)| + |Out(N'_i)|) && \text{Lemma 4(a)} \\
 &\in \mathcal{O}(|N_0|^3 + |N_0| \sum_{i=1}^{|N_0|} (|Out(N_i)| + |Out(N'_i)|)) && (*)
 \end{aligned}$$

However, we cannot yet bound $Appl(N_0)$ by a polynomial in $|N_0|$ and $|Out(N_0)|$, because, in principle, the number of outcomes of N_i or N'_i might grow exponentially with i . Indeed, the shortcut rule can increase the number of outcomes. Consider the degenerate negotiation \mathcal{N} with only one agent shown in Figure 5(a). \mathcal{N} has one single loop, namely $(n_1, a) (n_3, a) (n_4, b)$. The corresponding fragment \mathcal{F}_{n_1} consists of the atoms and outcomes of this loop, and \mathcal{N}_{n_1} is shown below \mathcal{N} . The negotiation \mathcal{N}_{n_1} can be summarized by three applications of the shortcut rule, shown in the lower row of the figure. The upper row shows the result of application of the same rules to \mathcal{N} .

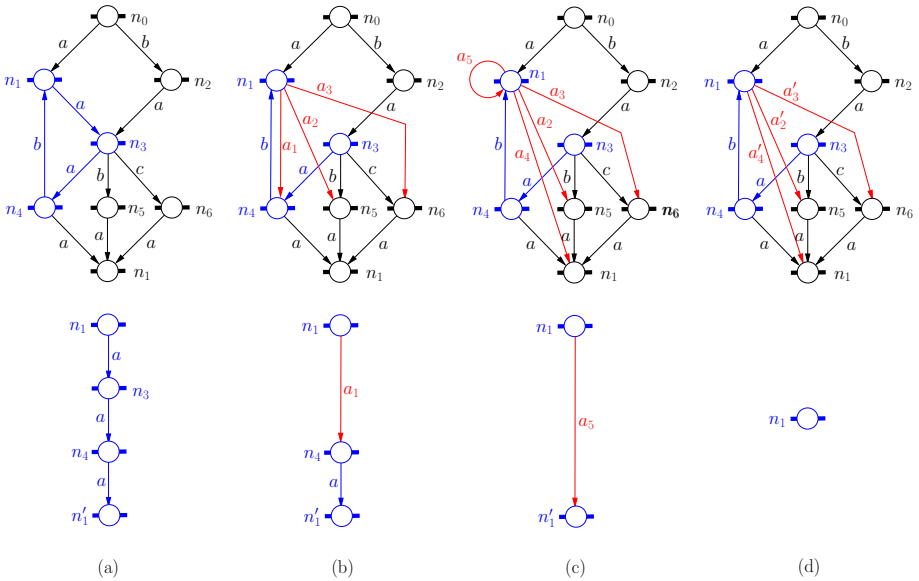


Fig. 5. Reducing an SND with one agent

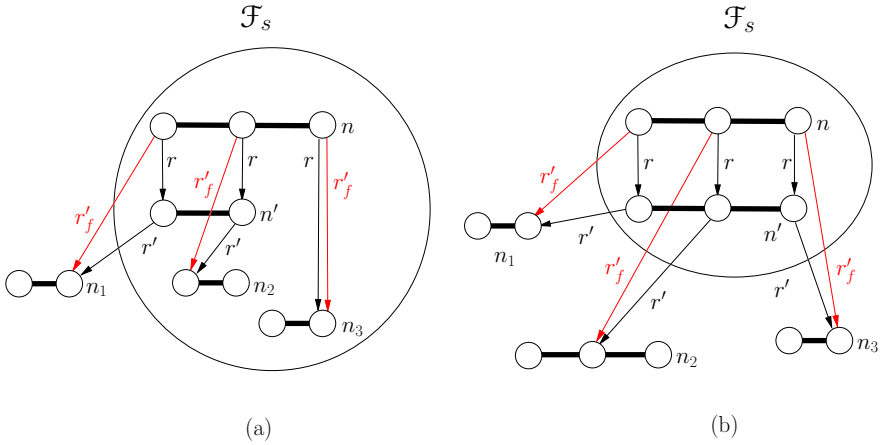


Fig. 6. Exits of SNDs

The first application removes n_3 from \mathcal{N}_{n_1} but not from \mathcal{N} , because n_3 has more than one input arc in \mathcal{N} (Figure 5(b)). Moreover, the rule adds three outcomes to \mathcal{N} (outgoing arcs of n_1). The second application removes n_4 from \mathcal{N}_{n_1} but not from \mathcal{N} , and adds two new outcomes (n_1, a_4) and (n_1, a_5) (Figure 5(c)). The third application removes n'_1 from \mathcal{N}_{n_1} ; in \mathcal{N} it is replaced by an application of the iteration rule, yielding the negotiation at the top of Figure 5(d), which has two outcomes more than the initial one.

To solve this problem we introduce *sources*, *targets* and *exits*.

5.1 Sources, Targets, and Exits

Definition 16. Let $\mathcal{N} = (N, n_0, n_f, \mathcal{X})$ be a negotiation, and let (n, r) be an outcome. The source of (n, r) is n . The target of (n, r) is the partial function $A \rightarrow N$ that assigns to every party $a \in P_n$ the atom $\mathcal{X}(n, a, r)$, and is undefined for every $a \in A \setminus P_n$. The set of targets of \mathcal{N} , denoted by $Ta(\mathcal{N})$, contains the targets of all outcomes of \mathcal{N} .

Consider the reduction process from \mathcal{N}_i to \mathcal{N}_{i+1} . It proceeds by applying to \mathcal{N}_i the same sequence of rules that summarizes an acyclic negotiation \mathcal{N}_s . This sequence progressively reduces the fragment \mathcal{F}_s until it consists of self-loops on the atom s , which can then be reduced by the iteration rule. However, the sequence also produces new outcomes of s that leave \mathcal{F}_s , and that become outcomes of \mathcal{N}_{i+1} not present in \mathcal{N}_i . Consider for instance Figure 6(a), which sketches an application of the shortcut rule. The outcome (n, r) unconditionally enables n' , whose outcome (n', r') makes the left agent leave \mathcal{F}_s . The target of (n, r'_f) assigns the agents of the negotiations to atoms n_1, n_2 and n_3 , respectively. This target is different from the targets of the other atoms in the figure.

We investigate the sources and targets of outcomes that leave \mathcal{F}_s . We call them *exits* of \mathcal{F}_s .

Definition 17. Let \mathcal{F}_s be a fragment of \mathcal{N} . An exit of \mathcal{F}_s is an outcome $(n, r) \in \text{Out}(\mathcal{N})$ such that $n \in F_s$ but $(n, r) \notin \text{Out}(\mathcal{F}_s)$.

The following lemma presents a key property of the exits of fragments of SDNs: the occurrence of an exit (n, r) of \mathcal{F}_s forces all agents of P_s to leave the fragment \mathcal{F}_s . In other words: all agents of P_s are parties of n , and the occurrence of (n, r) does not lead any agent back to an atom of \mathcal{F}_s .

Lemma 5. Let \mathcal{F}_s be a fragment of a SDN \mathcal{N} , and let (e, r_e) be an exit of \mathcal{F}_s . Then e has the same agents as s (i.e., e is also a synchronizer of \mathcal{F}_s), and $\mathcal{X}(e, a, r_e) \notin F_s$ for every agent a of e .

Proof. See [8]. □

In particular, the situation of Figure 6(a) cannot occur, and so in SDNs the correct picture for the application of the shortcut rule to exits is the one of Figure 6(b): the exit n' has the same agents as the synchronizer s . Moreover, the new target of (s, r'_f) equals the already existing target of (n', r') . So Lemma 5 leads to the following bound on the number of targets of \mathcal{N}_i :

Lemma 6. For every $1 \leq i \leq |N_0|$: $Ta(\mathcal{N}_i) \subseteq Ta(N_0)$.

Proof. See [8]. □

We use this lemma to bound $\text{Out}(\mathcal{N}'_i)$.

Lemma 7. For every $1 \leq i \leq |N_0|$: $|\text{Out}(\mathcal{N}'_i)| \in \mathcal{O}(|N_0|^2 \cdot |\text{Out}(N_0)|)$.

Proof. We first give an upper bound for $|\text{Out}(\mathcal{N}_i)|$. Since the merge rule cannot be applied to \mathcal{N}_i , no two outcomes of \mathcal{N}_i have the same source and the same target, and so $|\text{Out}(\mathcal{N}_i)| \leq |N_i| \cdot |Ta(\mathcal{N}_i)|$.

By Lemma 6, $|\text{Out}(\mathcal{N}_i)| \leq |N_0| \cdot |\text{Out}(N_0)|$.

Now we consider $|\text{Out}(\mathcal{N}'_i)|$. Each outcome of $\text{Out}(\mathcal{N}'_i) \setminus \text{Out}(\mathcal{N}_i)$ has some atom of \mathcal{F}_s as source, and is generated by some exit of \mathcal{F}_s . So the number of such outcomes is at most the product of the numbers of nodes of \mathcal{F}_s and the number of exits of \mathcal{F}_s . Since these numbers are bounded by $|N_i|$ and $|\text{Out}(\mathcal{N}_i)|$ respectively, we get $|\text{Out}(\mathcal{N}'_i)| \leq |\text{Out}(\mathcal{N}_i)| + |N_i| \cdot |\text{Out}(\mathcal{N}_i)|$. The result now follows from $|\text{Out}(\mathcal{N}_i)| \leq |N_0| \cdot |\text{Out}(N_0)|$ and Lemma 4(a). □

Finally, combining (*) and Lemma 7 we get

Theorem 4. Let \mathcal{N}_0 be an SDN. Then $\text{Appl}(\mathcal{N}_0) \in \mathcal{O}(|N_0|^4 \cdot \text{Out}(\mathcal{N}_0))$.

We conjecture that a more detailed complexity analysis can improve this bound to at least $\mathcal{O}(|N_0|^3 \cdot \text{Out}(\mathcal{N}_0))$, but this is beyond the scope of this paper.

6 Conclusions

We have continued the analysis of negotiations started in [7]. We have provided a set of three reduction rules that can summarize all and only the sound deterministic negotiations. Moreover, the number of rule applications is polynomial in the size of the negotiation.

The completeness and polynomiality proofs turned out to be quite involved. At the same time, we think they provide interesting insights. In particular, the completeness proofs shows how in deterministic negotiations soundness requires to synchronize all agents at least once in every loop. It also shows that, intuitively, loops must be properly nested. Intuitively, sound deterministic negotiations are *necessarily* well structured, in the sense of structured programming.

Our rules generalize the rules used to transform finite automata into regular expressions by eliminating states [14]. Indeed, deterministic negotiations can be seen as a class of communicating deterministic automata, and thus our result becomes a generalization of Kleene's theorem to a concurrency model. In future work we plan to investigate the connection to other concurrent Kleene theorems in the literature like e.g. [9,10].

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. *J. Circuits, Syst. and Comput.* 08(01), 21–66 (1998)
2. Atdelzater, T., Atkins, E.M., Shin, K.G.: QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers* 49(11), 1170–1183 (2000)
3. Berthelot, G.: Transformations and decompositions of nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 359–376. Springer, Heidelberg (1987)
4. Chen, Y., Peng, Y., Finin, T., Labrou, Y., Cost, S., Chu, B., Sun, R., Wilhelm, B.: A negotiation-based multi-agent system for supply chain management. In: Proceedings of Agents 1999 Workshop on Agent Based Decision-Support for Managing the Internet-Enabled Supply-Chain, pp. 15–20 (1999)
5. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20(1), 63–109 (1983)
6. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge University Press, New York (1995)
7. Esparza, J., Desel, J.: On negotiation as concurrency primitive. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 440–454. Springer, Heidelberg (2013); (Extended version in arXiv:1307.2145)
8. Esparza, J., Desel, J.: On negotiation as concurrency primitive II: Deterministic cyclic negotiations. Technical report, Technische Universität München, Germany. Available via arxiv.org (2014)
9. Gastin, P., Petit, A., Zielonka, W.: An extension of Kleene's and Ochmanski's theorems to infinite traces. *Theor. Comput. Sci.* 125(2), 167–204 (1994)
10. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.* 204(6), 920–956 (2006)

11. Genrich, H.J., Thiagarajan, P.S.: A theory of bipolar synchronization schemes. *Theor. Comput. Sci.* 30, 241–318 (1984)
12. Haddad, S.: A reduction theory for coloured nets. In: Rozenberg, G. (ed.) APN 1989. LNCS, vol. 424, pp. 209–235. Springer, Heidelberg (1990)
13. Haddad, S., Pradat-Peyre, J.-F.: New efficient Petri nets reductions for parallel programs verification. *Parallel Processing Letters* 16(1), 101–116 (2006)
14. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
15. Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Wooldridge, M.J., Sierra, C.: Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation* 10(2), 199–215 (2001)
16. Simon, C.: *Negotiation Processes – The Semantic Process Language and Applications*. Shaker, Aachen (2008)
17. Winsborough, W.H., Seamons, K.E., Jones, V.E.: Automated trust negotiation. In: *Proceedings of the DARPA Information Survivability Conference and Exposition, DISCEX 2000*, vol. 1, pp. 88–102. IEEE (2000)
18. Xu, H., Shatz, S.M.: An agent-based Petri net model with application to seller/buyer design in electronic commerce. In: *Proceedings of the 5th International Symposium on Autonomous Decentralized Systems*, pp. 11–18. IEEE (2001)

On Asymmetric Unification and the Combination Problem in Disjoint Theories

Serdar Erbatur¹, Deepak Kapur^{2,*}, Andrew M. Marshall³,
Catherine Meadows⁴, Paliath Narendran^{5,**}, and Christophe Ringeissen⁶

¹ Università degli Studi di Verona, Italy

² University of New Mexico, Albuquerque, NM, USA

³ ASEE, Washington, DC, USA

⁴ Naval Research Laboratory, Washington, DC, USA

⁵ University at Albany–SUNY, Albany, NY, USA

⁶ LORIA – INRIA Nancy-Grand Est, Nancy, France

Abstract. Asymmetric unification is a new paradigm for unification modulo theories that introduces irreducibility constraints on one side of a unification problem. It has important applications in symbolic cryptographic protocol analysis, for which it is often necessary to put irreducibility constraints on portions of a state. However many facets of asymmetric unification that are of particular interest, including its behavior under combinations of disjoint theories, remain poorly understood. In this paper we give a new formulation of the method for unification in the combination of disjoint equational theories developed by Baader and Schulz that both gives additional insights into the disjoint combination problem in general, and furthermore allows us to extend the method to asymmetric unification, giving the first unification method for asymmetric unification in the combination of disjoint theories.

1 Introduction

We examine the disjoint combination problem in the newly developed paradigm of asymmetric unification. This new unification problem was developed based on newly identified requirements arising from symbolic cryptographic protocol analysis [8]. Its application involves unification-based exploration of a space in which the states obey rich equational theories that can be expressed as a decomposition $R \uplus E$, where R is a set of rewrite rules that is confluent, terminating and coherent modulo E . However, in order to apply state space reduction techniques, it is usually necessary for at least part of this state to be in normal form, and to remain in normal form even after unification is performed. This requirement can be expressed as an *asymmetric* unification problem $\{s_1 =^\downarrow t_1, \dots, s_n =^\downarrow t_n\}$ where the $=^\downarrow$ denotes a unification problem with the restriction that any unifier leaves the right-hand side of each equation irreducible (see Definition 4).

* Partially supported by the NSF grant CNS-0905222.

** Partially supported by the NSF grant CNS-0905286.

The concept of asymmetric unification has its genesis in the unification method that is commonly used in symbolic analysis of cryptographic protocols. Here, two different requirements must be satisfied. The first is to have a generic unification algorithm that can be applied to a large class of equational theories that are encountered in cryptographic protocol analysis. The second is to guarantee that certain terms always be in normal form with respect to R (see Section 1.1), so that it is possible to apply state space reduction techniques. This is done by decomposing the theory into $R \uplus E$ so that R has the *finite variant property* [6] with respect to E , i.e., for any term t there is a finite set of irreducible variants $V(t)$ of pairs (u, σ) , where u is a term and σ is a substitution, so that for each $(u, \sigma) \in V(t)$ we have $t\sigma \downarrow =_E u$ and for any substitution τ there is a $(u, \sigma) \in V(t)$ and a substitution ρ such that $t\tau \downarrow =_E u\rho$. In other words, the set of variants gives a complete representation of the irreducible forms of t under any substitution. A unification problem is then solved by computing the variants of each side and unifying those modulo E . This approach to unification is used in a number of tools, including ProVerif [4], OFMC [16], Maude-NPA [10], and Tamarin [15]. More recently, it has been formalized in a procedure known as *folding variant narrowing* [12], which terminates if and only if the terms being unified have a finite number of variants.

Although variant narrowing is sound and complete for theories with the finite variant property, it is not optimally efficient. In [7] it is pointed out that the issue can often be addressed by computing the set of variants of only one side of a unification problem $s =^? t$, replacing it with a new asymmetric problem $s =^\downarrow t_1, \dots, s =^\downarrow t_n$. One may then apply more efficient special-purpose asymmetric unification algorithms that satisfy the irreducibility constraints. Recent work on asymmetric algorithms for exclusive-or [14], [8] and free Abelian groups [14] indicate that such algorithms can lead to significant enhancement of performance.

Although asymmetric unification has the potential of playing an important role in cryptographic protocol analysis, and possibly other unification-based state exploration as well, it is still not that well understood. Until the development of special-purpose algorithms for exclusive-or and free Abelian group theories mentioned above, the only known asymmetric unification algorithm was variant narrowing. Since then, some better understanding has been developed. For Example, we know that asymmetric unification is strictly harder than “symmetric” unification. In particular, there are theories for which symmetric unification is decidable and asymmetric unification is not. Still, there are many questions that remain to be answered. One of the most important of these unanswered questions is the problem of asymmetric unification in a combination of theories, in particular how to produce an algorithm for the combined theory by combining algorithms for the separate theories. This is particularly significant for cryptographic protocol analysis. Cryptographic protocols generally make use of more than one cryptoalgorithm. Often, these cryptoalgorithms can be described in terms of disjoint equational theories. In the case in which the algorithm used is variant narrowing, the problem is straightforward. If the combination of two

theories with the finite variant property also has the finite variant property, then one applies variant narrowing. However, in attempting to combine theories with special-purpose algorithms, the path is less clear. This is an important point with respect to efficiency since, as pointed out above, special-purpose asymmetric algorithms have the promise of being more efficient than variant narrowing.

In this paper we take the first step to solving this problem, by showing that the combination method for the unification problem in disjoint equational theories developed by Baader and Schulz in [2] can be modified and extended to the asymmetric unification paradigm, thus providing the first general combination method for this new paradigm. The only restrictions on this new method are those inherited from the asymmetric unification problem and those inherited from Baader and Schulz. From [2] we require that the algorithms being combined solve *the asymmetric unification with linear constant restriction problem*, although we show this reduces to solving the *general* asymmetric unification problem.

Do to space restrictions some proof details have been omitted. Please see the technical report version for full proofs [9].

1.1 Preliminaries

We use the standard notation of equational unification [3] and term rewriting systems [1]. The set of Σ -terms, denoted by $T(\Sigma, \mathcal{X})$, is built over the signature Σ and the (countably infinite) set of variables \mathcal{X} . The terms $t|_p$ and $t[u]_p$ denote respectively the subterm of t at the position p , and the term t having u as subterm at position p . The symbol of t occurring at the position p (resp. the top symbol of t) is written $t(p)$ (resp. $t(\epsilon)$). The set of positions of a term t is denoted by $Pos(t)$, the set of non variable positions for a term t over a signature Σ is denoted by $Pos(t)_\Sigma$. A Σ -rooted term is a term whose top symbol is in Σ . The set of variables of a term t is denoted by $Var(t)$. A term is *ground* if it contains no variables. A term t is *linear* if each variable of t occurs only once in t .

A Σ -substitution σ is an endomorphism of $T(\Sigma, \mathcal{X})$ denoted by $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ if there are only finitely many variables x_1, \dots, x_n not mapped to themselves. We call the *domain* of σ the set of variables $\{x_1, \dots, x_n\}$ and the *range* of σ the set of terms $\{t_1, \dots, t_n\}$. Application of a substitution σ to a term t (resp. a substitution ϕ) may be written $t\sigma$ (resp. $\phi\sigma$).

Given a first-order signature Σ , and a set E of Σ -axioms (i.e., pairs of Σ -terms, denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity. By a slight abuse of terminology, E will be often called an equational theory. An axiom $l = r$ is *variable-preserving* if $Var(l) = Var(r)$. An axiom $l = r$ is *linear* (resp. *collapse-free*) if l and r are linear (resp. non-variable terms). An equational theory is *variable-preserving* (resp. *linear/collapse-free*) if all its axioms are variable-preserving (resp. linear/collapse-free). An equational theory E is *finite* if for each term t , there are finitely many terms s such that $t =_E s$.

A Σ -equation is a pair of Σ -terms denoted by $s =^? t$. An E -unification problem is a set of Σ -equations, $\mathcal{S} = \{s_1 =^? t_1, \dots, s_m =^? t_m\}$. The set of variables of \mathcal{S} is denoted by $Var(\mathcal{S})$.

A solution to \mathcal{S} , called an E -unifier, is a substitution σ such that $s_i\sigma =_E t_i\sigma$ for all $1 \leq i \leq m$. A substitution σ is *more general modulo E* than θ on a set of variables V , denoted as $\sigma \leq_E^V \theta$, if there is a substitution τ such that $x\sigma\tau =_E x\theta$ for all $x \in V$. Two substitutions θ_1 and θ_2 are *equivalent modulo E* on a set of variables V , denoted as $\theta_1 =_E^V \theta_2$, if and only if $x\theta_1 =_E x\theta_2$ for all $x \in V$. A *complete set of E -unifiers* of \mathcal{S} is a set of substitutions denoted by $CSU_E(\mathcal{S})$ such that each $\sigma \in CSU_E(\mathcal{S})$ is an E -unifier of \mathcal{S} , and for each E -unifier θ of \mathcal{S} , there exists $\sigma \in CSU_E(\mathcal{S})$ such that $\sigma \leq_E^{Var(\mathcal{S})} \theta$.

Equational unification problems are classified based on the function symbols that appear in them, i.e., their signature (Sig). An E -unification problem S is *elementary* if and only if $Sig(S) = Sig(E)$. S is called an E -unification problem *with constants* if $Sig(S) \setminus Sig(E)$ contains only free constants. Finally, if there are uninterpreted function symbols in $Sig(S) \setminus Sig(E)$, S is called a general E -unification problem.

Let E_1 and E_2 be two equational theories built over the disjoint signatures Σ_1 and Σ_2 . The elements of Σ_i will be called *i -symbols*. A term t is an *i -term* if and only if it is of the form $t = f(t_1, \dots, t_n)$ for an i -symbol f or t is a variable. An i -term is *pure* (or an *i -pure term*) if it only contains i -symbols and variables. An equation $s =^? t$ is *i -pure* (or just *pure*) iff there exists an i such that s and t are i -pure terms or variables. A subterm s of an i -term t is called an *alien subterm* (or just *alien*) of t iff it is a non-variable j -term, $j \neq i$, such that every proper superterm of s in t is an i -term. A unification problem S is an *i -pure problem* if all equations in S are i -pure.

Definition 1. Let Γ be an E -unification problem, let \mathcal{X} denote the set of variables occurring in Γ and \mathcal{C} the set of free constants occurring in Γ . For a given linear ordering $<$ on $\mathcal{X} \cup \mathcal{C}$, and for each $c \in \mathcal{C}$ define the set V_c as $\{x \mid x \text{ is a variable with } x < c\}$. An E -unification problem with linear constant restriction (LCR) is an E -unification problem with constants, Γ , where each constant c in Γ is equipped with a set V_c of variables. A solution of the problem is an E -unifier σ of Γ such that for all c, x with $x \in V_c$, the constant c does not occur in $x\sigma$. We call σ an E -unifier with linear constant restriction.

A *rewrite rule* is an ordered pair $l \rightarrow r$ such that $l, r \in T(\Sigma, \mathcal{X})$ and $l \notin \mathcal{X}$. We use R to denote a term rewrite system which is defined as a set of rewrite rules. The rewrite relation on $T(\Sigma, \mathcal{X})$, written $t \rightarrow_R s$, hold between t and s iff there exists a non-variable $p \in Pos_\Sigma(t)$, $l \rightarrow r \in R$ and a substitution σ , such that $t|_p = l\sigma$ and $s = t[r\sigma]_p$. The relation $\rightarrow_{R/E}$ on $T(\Sigma, \mathcal{X})$ is $=_E \circ \rightarrow_R \circ =_E$. The relation $\rightarrow_{R,E}$ on $T(\Sigma, \mathcal{X})$ is defined as: $t \rightarrow_{R,E} t'$ if there exists a position $p \in Pos_\Sigma(t)$, a rule $l \rightarrow r \in R$ and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The transitive (resp. transitive and reflexive) closure of $\rightarrow_{R,E}$ is denoted by $\rightarrow_{R,E}^+$ (resp. $\rightarrow_{R,E}^*$). A term t is $\rightarrow_{R,E}$ *irreducible* (or in R, E -*normal form*) if there is no term t' such that $t \rightarrow_{R,E} t'$. If $\rightarrow_{R,E}$ is confluent and terminating we denote the irreducible version of a term, t , by $t \rightarrow_{R,E}^!$ or $t \downarrow_{R,E}$.

Definition 2. A rewrite rule $l \rightarrow r$ is duplicating if r contains more occurrences of some variable than l ; otherwise, $l \rightarrow r$ is non-duplicating. We say that R is non-duplicating if every $l \rightarrow r \in R$ is non-duplicating

Definition 3. We call (Σ, E, R) a decomposition of an equational theory Δ over a signature Σ if $\Delta = R \uplus E$ and R and E satisfy the following conditions:

1. E is variable preserving, i.e., for each $s = t$ in E we have $\text{Var}(s) = \text{Var}(t)$.
2. E has a finitary and complete unification algorithm. That is, an algorithm that produces a finite complete set of unifiers.
3. For each $l \rightarrow r \in R$ we have $\text{Var}(r) \subseteq \text{Var}(l)$.
4. R is confluent and terminating modulo E , i.e., the relation $\rightarrow_{R/E}$ is confluent and terminating.
5. $\rightarrow_{R,E}$ is E -coherent, i.e., $\forall t_1, t_2, t_3$ if $t_1 \rightarrow_{R,E} t_2$ and $t_1 =_E t_3$ then $\exists t_4, t_5$ such that $t_2 \rightarrow_{R,E}^* t_4$, $t_3 \rightarrow_{R,E}^+ t_5$, and $t_4 =_E t_5$.

This definition is inherited directly from [8] where Asymmetric unification and the corresponding theory decomposition are first defined. The last restrictions ensure that $s \rightarrow_{R/E}^! t$ iff $s \rightarrow_{R,E}^! t$ (see [11], [8]).

Definition 4 (Asymmetric Unification). Given a decomposition (Σ, E, R) of an equational theory, a substitution σ is an asymmetric R, E -unifier of a set \mathcal{S} of asymmetric equations $\{s_1 =^\downarrow t_1, \dots, s_n =^\downarrow t_n\}$ iff for each asymmetric equations $s_i =^\downarrow t_i$, σ is an $(E \cup R)$ -unifier of the equation $s_i =^? t_i$ and $(t_i \downarrow_{R,E})\sigma$ is in R, E -normal form. A set of substitutions Ω is a complete set of asymmetric R, E -unifiers of \mathcal{S} (denoted $CSAU_{R \cup E}(\mathcal{S})$ or just $CSAU(\mathcal{S})$ if the background theory is clear) iff: (i) every member of Ω is an asymmetric R, E -unifier of \mathcal{S} , and (ii) for every asymmetric R, E -unifier θ of \mathcal{S} there exists a $\sigma \in \Omega$ such that $\sigma \leq_E^{Var(\mathcal{S})} \theta$.

Example 1. Let $R = \{x \oplus 0 \rightarrow x, x \oplus x \rightarrow 0, x \oplus x \oplus y \rightarrow y\}$ and E be the AC theory for \oplus . Consider the equation $y \oplus x =^\downarrow x \oplus a$, the substitution $\sigma_1 = \{y \mapsto a\}$ is an asymmetric solution but, $\sigma_2 = \{x \mapsto 0, y \mapsto a\}$ is not.

Definition 5 (Asymmetric Unification with Linear Constant Restriction). Let \mathcal{S} be a set of asymmetric equations with some LCR. A substitution σ is an asymmetric R, E -unifier of \mathcal{S} with LCR iff σ is an asymmetric solution to \mathcal{S} and σ satisfies the LCR.

2 Combining Asymmetric Unification Algorithms

Here we modify and extend the method for unification in the union of disjoint equational theories, developed by Baader and Schulz [2], to the combination of asymmetric unification algorithms in the union of disjoint equational theories.

Problem Description: Let Δ_1 and Δ_2 denote two equational theories with disjoint signatures Σ_1 and Σ_2 . Let Δ be the combination, $\Delta = \Delta_1 \cup \Delta_2$, of the two theories having signature $\Sigma_1 \cup \Sigma_2$. Let A_i , $i \in \{1, 2\}$, be an asymmetric Δ_i -unification with linear constants restriction algorithm. We then give an algorithm which uses A_1 and A_2 to solve the *elementary* asymmetric unification problem over Δ . Recall that elementary implies that terms can only contain symbols in the signature of the theory or variables. But this is not restrictive, if we wish to have additional free functional symbols, these function symbols define a new empty theory and lead to another combination. Therefore, in what follows we will assume that a problem, Γ_0 , in the combined theory Δ , is an elementary asymmetric Δ -unification problem. In order to satisfy the requirements for asymmetric unification we make the following assumptions.

Restrictions: for each constituent theory (Σ_i, Δ_i) :

1. There is a decomposition $\Delta_i = R_i \uplus E_i$ and $u \rightarrow_{R_i, E_i}^! v$ iff $u \rightarrow_{R_i/E_i}^! v$ (see note (2) below).
2. E_i is collapse-free and there exists a finitary E_i -unification algorithm.
3. There exists a finitary complete asymmetric Δ_i -unification algorithm with linear constants restriction, A_i (see note (3) below).
4. Variables are \rightarrow_{R_i, E_i} -normal forms.
5. Each R_i is non-duplicating.

Notes on the Restrictions:

1. All Restrictions, except (3), are due to the asymmetric unification definition.
2. The definition of decomposition requires that \rightarrow_{R_i/E_i} be confluent and terminating. Thus, if $u \rightarrow_{R_i, E_i}^! v$ iff $u \rightarrow_{R_i/E_i}^! v$, we have that \rightarrow_{R_i, E_i} is also confluent and terminating.
3. We show in Section 2.5 that there exists an asymmetric Δ_i -unification algorithm with linear constants restriction if there exists a *general* asymmetric Δ_i -unification algorithm.

According to our Restrictions, E_1 and E_2 are both variable preserving and collapse-free. Consequently, we have the following property:

Lemma 1. $t \neq_{E_1 \cup E_2} s$, if t is a non-variable i -term and s is a non-variable j -term, $j \neq i$.

2.1 Rewriting in the Combined Theory

The definition of asymmetric unification in the combined theory Δ , where $\Delta = \Delta_1 \cup \Delta_2$, requires us to not only find Δ -unifiers but also decide if a term is in $\rightarrow_{(R_1 \cup R_2), (E_1 \cup E_2)}$ normal form. Therefore, we need to first ensure the modularity of rewriting, i.e., ensure we can compute $\rightarrow_{(R_1 \cup R_2), (E_1 \cup E_2)}$ -normal forms.

Consider now the combined theory (Σ, Δ) , where $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Delta = \Delta_1 \cup \Delta_2$. Let $\mathcal{R} = R_1 \cup R_2$ and $\mathcal{E} = E_1 \cup E_2$. Therefore, $\rightarrow_{\mathcal{R}, \mathcal{E}}$ denotes $\rightarrow_{R_1 \cup R_2, E_1 \cup E_2}$.

Theorem 1. $\rightarrow_{\mathcal{R}, \mathcal{E}} = \rightarrow_{R_1, E_1} \cup \rightarrow_{R_2, E_2}$

Proof. Follows from the the fact that $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $\mathcal{E} = E_1 \cup E_2$ is variable preserving and collapse-free. \square

The relation \rightarrow_{R_i, E_i} is decidable for each sub-theory due to the assumption that \rightarrow_{R_1, E_1} is convergent. Therefore we obtain the following corollary to Theorem 1.

Corollary 1. *The relation $\rightarrow_{\mathcal{R}, \mathcal{E}}$ is decidable.*

Note, with respect to termination, $R_1 \cup R_2$ is non-duplicating, this is due to the disjoint theories and the fact that each R_i is non-duplicating by assumption. Since $R_1 \cup R_2$ is non-duplicating termination is obtained due to the results of [17], where it is shown that non-duplicating implies termination in the combination of terminating rewrite systems. Next we would like to know that $\rightarrow_{\mathcal{R}, \mathcal{E}}$ is complete with respect to $\rightarrow_{\mathcal{R}/\mathcal{E}}$, i.e., $u \rightarrow_{R_i, E_i}^! v$ iff $u \rightarrow_{R_i/E_i}^! v$, which is not true in general. For this to be true we need to know that $\rightarrow_{\mathcal{R}, \mathcal{E}}$ is \mathcal{E} -coherent, which implies the result (see [11]).

Lemma 2. *If there exist terms t_0, t_1 and t_2 such that $t_0 \leftrightarrow_{\mathcal{E}}^* t_2$ and $t_0 \rightarrow_{\mathcal{R}, \mathcal{E}} t_1$ then there exists a term t_3 such that $t_2 \rightarrow_{\mathcal{R}, \mathcal{E}} t_3$.*

Proof. This can be shown via an induction argument relying on the fact that \rightarrow_{R_i, E_i} is coherent modulo E_i . See [9] for the full proof. \square

Theorem 2. *$\rightarrow_{\mathcal{R}, \mathcal{E}}$ is \mathcal{E} -coherent.*

Proof. If $t_0 \rightarrow_{\mathcal{R}, \mathcal{E}} t_1$ and $t_0 =_{\mathcal{E}} t_2$, then by Lemma 2, there exists a term, t_3 , such that $t_2 \rightarrow_{\mathcal{R}, \mathcal{E}} t_3$. Thus, $t_1 \leftarrow_{R_1 \cup R_2 \cup E_1 \cup E_2} t_3$. Now the combined system has the properties (normal form variables, E_i collapse-free, and disjoint signatures) such that the Church-Rosser result in [13] applies. This implies the existence of terms t_4 and t_5 such that $t_1 \rightarrow_{\mathcal{R}, \mathcal{E}}^* t_4$, $t_3 \rightarrow_{\mathcal{R}, \mathcal{E}}^* t_5$ and $t_4 =_{\mathcal{E}} t_5$. \square

Therefore, based on Corollary 1 and Theorem 2, $u \rightarrow_{\mathcal{R}, \mathcal{E}}^* v$ iff $u \rightarrow_{\mathcal{R}/\mathcal{E}}^* v$ which implies the following:

Theorem 3. *$t =_{\mathcal{R} \cup \mathcal{E}} s$ iff $t \downarrow_{\mathcal{R}, \mathcal{E}} =_{\mathcal{E}} s \downarrow_{\mathcal{R}, \mathcal{E}}$*

2.2 Asymmetry in the Projection of Terms

Now that we have established the modular results for rewriting we can use the well defined normal forms to define projections onto pure terms. Later we will use the bijection defined below to prove that if the original problem has a solution then there exists solutions to the pure sub-problems. This is accomplished by mapping the combined solution into two pure solutions. In order for this to work we also need to ensure that equality modulo E and asymmetric restrictions are maintained after the mapping is applied. Let \mathcal{X} and \mathcal{Y} be disjoint sets of variables that are countably infinite. Let $T(\Sigma, \mathcal{X})$ be the set of $\Sigma_1 \cup \Sigma_2$ -terms over \mathcal{X} . We define a bijection

$$\pi : (T(\Sigma, \mathcal{X}) \downarrow_{\mathcal{R}, \mathcal{E}})_{/=_{\mathcal{E}}} \rightarrow \mathcal{Y} \tag{1}$$

The bijection π induces two mappings π_1 and π_2 of terms in $T(\Sigma, \mathcal{X})$ to terms in $T(\Sigma, \mathcal{Y})$ as follows. For each $x \in \mathcal{X}$, $x^{\pi_1} := \pi(x)$. If $t = f(t_1, \dots, t_n)$ for a 1-symbol f , then $t^{\pi_1} := f(t_1^{\pi_1}, \dots, t_n^{\pi_1})$. If t is a 2-term then $t^{\pi_1} := y$ where $y = \pi([s]_{\mathcal{E}})$ for the unique $\rightarrow_{\mathcal{R}, \mathcal{E}}$ -irreducible term s of t , where $[s]_{\mathcal{E}}$ denotes the equivalence class of s modulo \mathcal{E} . The mapping π_2 is defined analogously.

Given a substitution σ , σ^{π_i} denotes the abstraction defined by $\sigma^{\pi_i}(x) = (\sigma(x))^{\pi_i}$, for all x is the domain of σ . These two mapping can be seen as projections from mixed terms into pure terms. More informally, we can view an i -abstraction as method for converting a mixed term into a pure term by replacing the alien subterms with variables. Recall that we assume that variables are \rightarrow_{R_i, E_i} -irreducible and thus by modularity $\rightarrow_{\mathcal{R}, \mathcal{E}}$ -irreducible. As in [2] we can also define the inverse, π^{-1} of π as a substitution that maps the variables $y \in \mathcal{Y}$ back to terms $\pi^{-1}(y)$ and is the identity on all other variables. Note that, $\pi^{-1}(t^{\pi_i}) =_{\mathcal{E}} t$, if t is in \mathcal{R}, \mathcal{E} -normal form or an i -term with normal form aliens.

Theorem 4. *Let s and t be i -pure terms. Let t and σ be in \mathcal{R}, \mathcal{E} -normal form, such that $s\sigma =_{\Delta} t\sigma$. Then $s\sigma^{\pi_i} =_{\Delta_i} t\sigma^{\pi_i}$ and $t\sigma$ is in \mathcal{R}, \mathcal{E} -normal form iff $t\sigma^{\pi_i}$ is in R_i, E_i -normal form.*

Proof. This result follows from the disjoint signatures and the fact that \mathcal{E} is variable preserving. See [9] for the full proof details. □

2.3 Asymmetric Unification with Linear Constant Restriction

We present the combination Algorithm, *AsymComb*, in Figure 1. Let us first give a rough, intuitive overview of the steps. First, equations are purified using variable abstraction and splitting (steps 1 and 2). This ensures that the original problem is separated into pure problems which can be solved by the algorithms for the pure theories. Next, a variable identification is non-deterministically chosen, allowing for the testing all the ways the variables may be equated to other variables. Then, a linear ordering and theory indices are non-deterministically chosen. Note that a shared variable can only “belong” exclusively to one theory. Since we don’t know beforehand what variable belongs to which theory the non-deterministic selections allow us to check all the possibilities. In addition, each solution to the original problem will correspond to one or more linear ordering among the variables. Next, the problem is split into two pure problems where the linear ordering defines a linear constant restriction. The pure problems are solved by asymmetric unification algorithms with linear constant restriction. The solutions returned by the sub-algorithms are combined into solutions for the original problem. The Algorithm *AsymComb* (cf. Figure 1) must also ensure that we only combine a specific type of unifier, which ensures asymmetry.

The notions of identification, theory indexes and linear constant restrictions, have all been used in [2] (see Section 1.1 for definitions). In order to handle the asymmetry restriction we introduce two additional notions, which ensure pure problem solutions having these properties will result in asymmetric combined solutions.

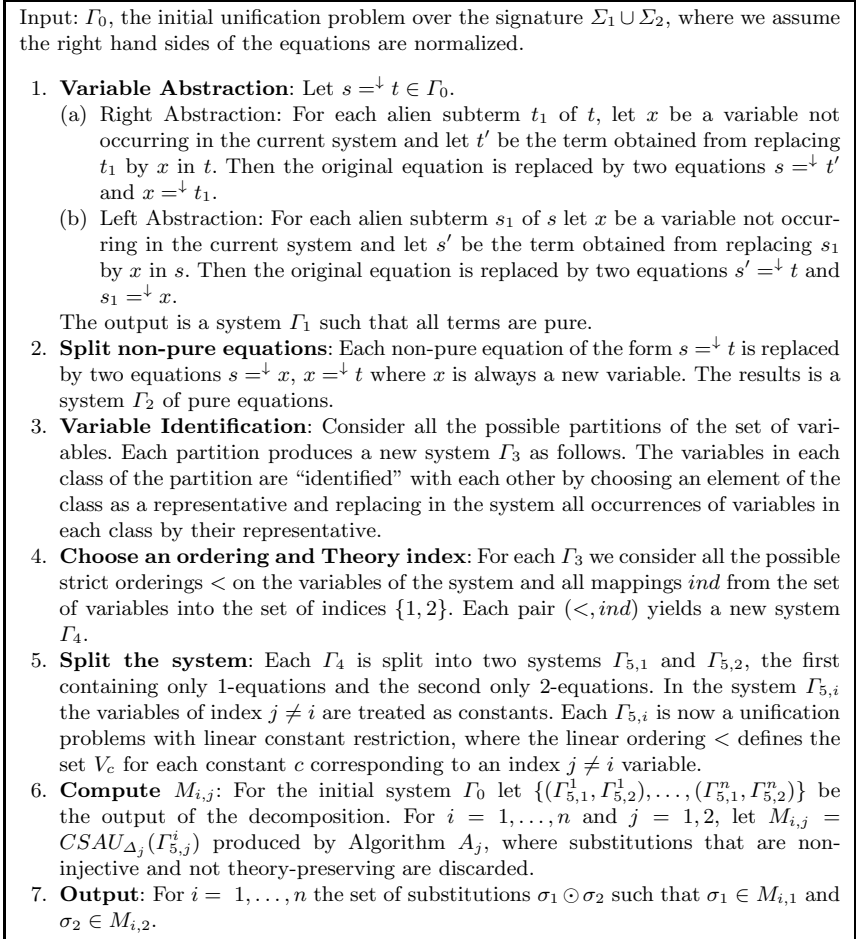


Fig. 1. Algorithm *AsymComb*

Definition 6. (*Injective*)

A substitution, σ_i , is said to be injective modulo Δ_i if for any two variables x, y in the domain of σ_i , we have that $x\sigma_i =_{\Delta_i} y\sigma_i$ if and only if $x = y$.

Definition 7. (*Theory Preserving*)

A substitution σ_i , solving an i -pure problem Γ_i , is said to be theory preserving if for any variable x of index i in the domain of σ_i , $x\sigma_i$ is not a variable of index $j \neq i$.

With respect to the combination algorithm this definition basically states that a substitution σ_i , which solves an i -pure problem, Γ_i , produced by Algorithm *AsymComb* (cf. Figure 1), is theory-preserving if for all $x \in Dom(\sigma_i)$, $x\sigma_i \neq c$ where c is a free constant. This is due to the fact that for the pure sub-problems produced by the combination algorithm the only free constants

will be those corresponding to shared variables of a different index. *Note, the definition of theory-preserving does not restrict σ_i from sending an i -variable x to a non-variable i -term whose leafs are j -variables.* Thus, if $x\sigma_i = t$ and t is an i -term, then t may contain j -variables. In addition, since the Algorithm *AsymComb* assigns the variable indexes, it can always check the substitutions returned by the algorithms for the pure theories to ensure that they are injective and theory-preserving.

Definition 8. *Let σ_1 and σ_2 be unifiers with linear constant restriction for $\Gamma_{5,1}$ and $\Gamma_{5,2}$ such that $\Gamma_{5,i}$ is the set of i -pure equations from Γ_4 and $<$ is the corresponding linear ordering. The combined solution $\sigma_1 \odot \sigma_2$ is defined by induction on $<$:*

Let x be the least variable with respect to the ordering $<$ from step 4 and let i be its index. Since the solution σ_i of $\Gamma_{5,i}$ satisfies the constant restriction induced by $<$, $x\sigma_i$ does not contain any variables of index $j \neq i$. We define $x(\sigma_1 \odot \sigma_2)$ to be $x\sigma_i$.

Let x be an arbitrary variable of index i and let y_1, \dots, y_m be the variables of index $j \neq i$ occurring in $x\sigma_i$. Again, due to the constant restriction, the variables y_1, \dots, y_m have to be smaller than x . This implies that $y_1(\sigma_1 \odot \sigma_2), \dots, y_m(\sigma_1 \odot \sigma_2)$ are already defined. The term $x(\sigma_1 \odot \sigma_2)$ is obtained from $x\sigma_i$ by replacing y_k by $y_k(\sigma_1 \odot \sigma_2)$, and we define $x(\sigma_1 \odot \sigma_2)$ to be $x\sigma_i(\sigma_1 \odot \sigma_2)$.

Lemma 3. *(Baader-Schulz [2])*

The combined unifier $\sigma_1 \odot \sigma_2$ from Definition 8 is a unifier of Γ_4 .

Example 2. Let $\Delta_1 = R_1 \cup E_1$, where $R_1 = \{e(x, d(x, y)) \rightarrow y, d(x, e(x, y)) \rightarrow y\}$ and $E_1 = \emptyset$. Let $\Delta_2 = R_2 \cup E_2$, where $R_2 = \{x \oplus 0 \rightarrow x, x \oplus x \rightarrow 0, x \oplus x \oplus y \rightarrow y\}$ and $E_2 = \{x \oplus y = y \oplus x, (x \oplus y) \oplus z = x \oplus (y \oplus z)\}$. Let $\Delta = \Delta_1 \cup \Delta_2$.

Consider the initial problem Γ_0 consisting of the following: $\{x_0 \oplus x_1 \oplus x_2 = \downarrow x_3 \oplus x_4, e(x_1, d(0, x_5)) = \downarrow x_2 \oplus x_0, e(x_1, d(x_0, e(x_2, x_6))) = \downarrow e(x_7, x_5)\}$

Let us now examine the action of Algorithm *AsymComb* (cf. Figure 1) on Γ_0 and how it would find a particular asymmetric solution. The first 2 steps produce the set of pure equations $\Gamma_2: \{x_0 \oplus x_1 \oplus x_2 = \downarrow x_3 \oplus x_4, e(x_1, d(z_0, x_5)) = \downarrow z_1, 0 = \downarrow z_0, z_1 = \downarrow x_2 \oplus x_0, e(x_1, d(x_0, e(x_2, x_6))) = \downarrow e(x_7, x_5)\}$.

The next step considers the set of variable partitions, one of which is the following partition $\{\{x_0, x_3\}, \{x_2, x_4\}, \{x_5, z_1\}, \{x_1, z_0, x_7\}, \{x_6\}\}$. Choosing a representative for each set and doing the replacement the Algorithm would produce the following Γ_3 from that partition: $\{x_0 \oplus x_1 \oplus x_2 = \downarrow x_0 \oplus x_2, e(x_1, d(x_1, x_5)) = \downarrow x_5, 0 = \downarrow x_1, x_5 = \downarrow x_2 \oplus x_0, e(x_1, d(x_0, e(x_2, x_6))) = \downarrow e(x_1, x_5)\}$.

The next step considers the possible pairs of variable orderings and theory indexes. One pair that would be produced is the following: $x_6 > x_5 > x_2 > x_1 > x_0$, index-1 = $\{x_0, x_1, x_2, x_5\}$ and index-2 = $\{x_6\}$.

Next Γ_4 is produced from that pair and split into pure sets to produce $\Gamma_{5,1}$ and $\Gamma_{5,2}$. Let us denote a variable, y , being treated as a constant as \mathbf{y} . Then, $\Gamma_{5,1}$ is the following set of equations: $\{x_0 \oplus x_1 \oplus x_2 = \downarrow x_0 \oplus x_2, 0 = \downarrow x_1, x_5 = \downarrow x_2 \oplus x_0\}$ and $\Gamma_{5,2}$ is the following set of equations: $\{e(\mathbf{x}_1, d(\mathbf{x}_1, \mathbf{x}_5)) = \downarrow \mathbf{x}_5, e(\mathbf{x}_1, d(\mathbf{x}_0, e(\mathbf{x}_2, x_6))) = \downarrow e(\mathbf{x}_1, \mathbf{x}_5)\}$

Next $\Gamma_{5,1}$ is solved with A_1 and $\Gamma_{5,2}$ with A_2 , where the linear constant restriction is obtained via the linear ordering and theory index. The last step is to combine each pair of substitutions (σ_1, σ_2) into a substitution σ , where σ_i is an injective and theory-preserving asymmetric with LCR solution to $\Gamma_{5,i}$ returned by A_i . One such pair is $\sigma_1 = \{x_1 \mapsto 0, x_5 \mapsto x_2 \oplus x_0\}$ and $\sigma_2 = \{x_6 \mapsto d(\mathbf{x}_2, e(\mathbf{x}_0, \mathbf{x}_5))\}$. Applying Definition 8 we get the following solution, $\{x_1 \mapsto 0, x_3 \mapsto x_0, x_4 \mapsto x_2, x_5 \mapsto x_2 \oplus x_0, x_6 \mapsto d(x_2, e(x_0, x_2 \oplus x_0)), x_7 \mapsto 0\}$, which is an asymmetric solution to Γ_0 (existential variables z_0, z_1 are removed).

Before presenting the proof details lets us briefly point out the main differences between Algorithm *AsymComb* (cf. Figure 1) and the algorithm of [2]. While the general framework of the two algorithms is similar there are several key differences. First, we do not consider general theories. Due to the restrictions inherited from the definition of asymmetric unification we must consider theories with specific structure, namely the decomposition. This requires new results for showing the correctness of the algorithm and new results for showing that the required properties for asymmetric unification are maintained. Second, we must identify the specific unifiers which satisfy the asymmetry. We accomplish this by identifying two key properties, theory preservation (Definition. 7) and injectivity (Definition. 6).

2.4 Correctness

We show in this section that the Algorithm *AsymComb* (cf. Figure 1) is both sound and complete for the decision problem. In addition, we show that the algorithm produces a complete set of asymmetric unifiers.

Lemma 4. *Assume that σ_1 and σ_2 are pure, injective, theory-preserving and R_i, E_i -normalized unifiers modulo respectively $\Delta_1 = R_1 \uplus E_1$ and $\Delta_2 = R_2 \uplus E_2$ and they satisfy the same linear constant restriction. Then, the substitution $\sigma = \sigma_1 \odot \sigma_2$ satisfies the following properties: (1) σ is an injective substitution modulo $\Delta_1 \cup \Delta_2$. (2) σ is \mathcal{R}, \mathcal{E} -normalized.*

Proof. We proceed by induction on the linear ordering.

Base case: Let v be the smallest variable, say of index i . Then, σ is clearly injective and \mathcal{R}, \mathcal{E} -normalized for variables smaller or equal to v , since $v\sigma = v\sigma_i$ is R_i, E_i -normalized, and so also \mathcal{R}, \mathcal{E} -normalized.

Inductive case: Assume the the properties holds for variables smaller than a variable y of index i . To show that (1) holds, assume by contradiction that there exists a variable x strictly smaller than y such that $x\sigma =_{\Delta_1 \cup \Delta_2} y\sigma$. Since σ is \mathcal{R}, \mathcal{E} -normalized for variables smaller than y , we have that $x\sigma =_{\Delta_1 \cup \Delta_2} y\sigma$ implies $x\sigma^{\pi_i} =_{\Delta_i} y\sigma^{\pi_i}$. Since σ is injective for variables smaller than y , there exists a renaming ρ such that $x\sigma^{\pi_i}\rho = x\sigma_i$ and $y\sigma^{\pi_i}\rho = y\sigma_i$. Therefore $x\sigma_i =_{\Delta_i} y\sigma_i$, which is a contradiction. Consider now the property (2): if $y\sigma$ is \mathcal{R}, \mathcal{E} -reducible, then $(y\sigma)^{\pi_i}$ is R_i, E_i -reducible, which means that $y\sigma_i\rho$ and $y\sigma_i$ are R_i, E_i -reducible too, which contradicts the assumption that σ_i is an R_i, E_i -normalized substitution. \square

Lemma 5. *Let Γ_0 be a solvable asymmetric Δ -unification problem, where $\Delta = \Delta_1 \cup \Delta_2$. Assume there exists a pair $(\Gamma_{5,1}, \Gamma_{5,2})$ produced by the Algorithm AsymComb (cf. Figure 1) on Γ_0 and a pair (σ_1, σ_2) such that $\sigma_i \in CSAU_{\Delta_i}(\Gamma_{5,i})$ for $i = 1, 2$.*

Then, there exists pairs $(\Gamma'_{5,1}, \Gamma'_{5,2})$ produced by the Algorithm AsymComb on Γ_0 and a pair (ϕ_1, ϕ_2) such that ϕ_i is injective and theory-preserving, $\phi_i \in CSAU_{\Delta_i}(\Gamma'_{5,i})$ for $i = 1, 2$, and $\phi_1 \odot \phi_2 \leq_{\Delta}^{Var(\Gamma_0)} \sigma_1 \odot \sigma_2$.

Proof. Construct $(\Gamma'_{5,1}, \Gamma'_{5,2})$ and (σ'_1, σ'_2) : Let Γ_4 be the conjunction of $\Gamma_{5,1}$ and $\Gamma_{5,2}$. Then there exists a linear ordering, $<$, and a theory index, ind . From Γ_4 we can construct a new Γ'_4 as follows: If there exists x, y in the domain of σ_i such that $x\sigma_i =_{\Delta_i} y\sigma_i$ we add $x = y$ to the variable identification. If there exists variables x, y such that x is an index i variable, y is an index j variable and $x\sigma_i = y$, we replace all x with y in the variable identification. ind and $<$ remain the same. The result of these steps is a new Γ'_4 , which also gives us a new pair $(\Gamma'_{5,1}, \Gamma'_{5,2})$. We can now define the new pair of substitutions (σ'_1, σ'_2) as follows: Let $Dom(\sigma'_i) = Var(\Gamma'_{5,i})$. $\forall x \in Dom(\sigma'_i)$, $x\sigma'_i = x\sigma_i$ and is the identity on all other variables.

Show that σ'_1 and σ'_2 are theory-preserving and injective unifiers of $\Gamma'_{5,1}$ and $\Gamma'_{5,2}$: This follows from the construction of Γ'_4 , where variables violating the definitions have been removed.

Show that $\forall x \in Var(\Gamma_0) \ x\sigma =_{\Delta} x\sigma'$: First, by the definition of σ' for all $x \in Dom(\sigma')$, $x\sigma' = x\sigma$. Therefore, we need only consider the variables removed by the variable identification step. From Definition 8, for any variable x in the initial system replaced by a variable y during the identification step, $x\sigma := y\sigma$. Since any identifications occurring in the definition of Γ_4 must also occur in Γ'_4 , $x\sigma' := y\sigma' = y\sigma = x\sigma$. Now consider the variable identifications added to construct Γ'_4 but not existing in Γ_4 . If $x = y$ is added because $x\sigma_i = y\sigma_i$, without loss of generality assume x is replaced by y , then $x\sigma' := y\sigma' = y\sigma =_{\Delta} x\sigma$. Lastly if x is replaced by y because $x\sigma_i = y$, then $x\sigma' := y\sigma = x\sigma$.

To complete the proof, there exists $\phi_i \in CSAU_{\Delta}(\Gamma'_{5,i})$ such that $\phi_i \leq_{\Delta}^{Var(\Gamma_0)} \sigma'_i$ for $i = 1, 2$. By the definition of \odot , we have that $\phi_1 \odot \phi_2 \leq_{\Delta}^{Var(\Gamma_0)} \sigma'_1 \odot \sigma'_2 = \sigma'$, and $\sigma' =_{\Delta}^{Var(\Gamma_0)} \sigma$. Therefore, $\phi_1 \odot \phi_2 \leq_{\Delta}^{Var(\Gamma_0)} \sigma_1 \odot \sigma_2 = \sigma$. □

Lemma 6. *For each asymmetric unifier of a problem Γ_0 , there exists a pair $(\Gamma_{5,1}, \Gamma_{5,2})$ computed by the Algorithm AsymComb (cf. Figure 1), where for each $\Gamma_{5,i}$ there exist a substitution τ_i which asymmetrically solves $\Gamma_{5,i}$.*

Proof. The construction given in [2] can be used here with modifications to account for asymmetry and the projection π (Equation 1), see [9]. □

Lemma 7. *Let Γ_0 be an asymmetric Δ -unification problem. For each asymmetric unifier τ of Γ_0 there exists a pair $(\Gamma_{5,1}, \Gamma_{5,2})$ in the output set of the Algorithm AsymComb (cf. Figure 1), and a pair of substitutions (σ_1, σ_2) with each $\sigma_i \in CSAU_{\Delta_i}(\Gamma_{5,i})$. Such that $\sigma = \sigma_1 \odot \sigma_2$ is an injective asymmetric solution to Γ_0 with $\sigma \leq_{\Delta}^{Var(\Gamma_0)} \tau$.*

Proof. From Lemma 6 we have that given τ there exists a pair $(\Gamma_{5,1}^1, \Gamma_{5,2}^1)$ and substitutions (τ_1^1, τ_2^1) such that τ_i^1 asymmetrically solves $\Gamma_{5,i}^1$. Now, if τ_i^1 is an asymmetric solution to $\Gamma_{5,i}^1$, there exists a substitution τ_i^2 produced by the algorithm A_i such that $\tau_i^2 \in CSAU_{\Delta_i}(\Gamma_{5,i}^1)$ and $\tau_i^2 \leq_{\Delta_i}^{Var(\Gamma_{5,i})} \tau_i^1$. Furthermore, as in Lemma 5, by the definition of \odot , we have that $\tau^2 = \tau_1^2 \odot \tau_2^2 \leq_{\Delta}^{Var(\Gamma_0)} \tau_1^1 \odot \tau_2^1 = \tau$

From Lemma 5, there exists a pair $(\Gamma_{5,1}^2, \Gamma_{5,2}^2)$ produced by AsymComb and a pair (σ_1, σ_2) such that σ_i is *injective* and *theory-preserving*, $\sigma_i \in CSAU_{\Delta_i}(\Gamma_{5,i}^2)$ and $\sigma = \sigma_1 \odot \sigma_2 \leq_{\Delta}^{Var(\Gamma_0)} \tau^2$. By Lemma 4, σ is an injective asymmetric solution to Γ_4 . Finally, $\sigma \leq_{\Delta}^{Var(\Gamma_0)} \tau^2$ and $\tau^2 \leq_{\Delta}^{Var(\Gamma_0)} \tau$, and so $\sigma \leq_{\Delta}^{Var(\Gamma_0)} \tau$. \square

We can now show the the Algorithm *AsymComb* (cf. Figure 1) is correct, i.e., both sound and complete.

Theorem 5. *Let Γ_0 be a combined asymmetric unification problem. Γ_0 is asymmetrically unifiable if and only if the Algorithm AsymComb (cf. Figure 1) returns a combined substitution.*

Proof. From Lemma 3, the substitutions returned are unifiers. From Lemma 4 the substitutions are asymmetric. Completeness follows from Lemma 7. \square

Now we can consider the complete set of unifiers.

Theorem 6. *Let Γ_0 be an asymmetric Δ -unification problem. Then, for every $\tau \in CSAU(\Gamma_0)$, Algorithm AsymComb (cf. Figure 1) produces an injective substitution σ such that $\sigma \leq_{\Delta}^{Var(\Gamma_0)} \tau$.*

Proof. For any problem, Γ_0 , the Algorithm AsymComb will try every combination of variable identification, theory index and linear ordering, i.e. every possible pair of sub-problems $(\Gamma_{5,1}, \Gamma_{5,2})$. Furthermore, the Algorithm AsymComb will combine every pair, (σ_1, σ_2) , of injective and theory preserving solutions such that $\sigma_i \in CSAU_{\Delta_i}(\Gamma_{5,i})$, $i \in \{1, 2\}$. Thus, the result follows from Lemma 7. \square

2.5 Obtaining Linear Constant Restriction Algorithms

If one has a *general* asymmetric unification algorithm an algorithm that respects an LCR can be obtained. The construction is similar to the one given in [2]. Given Γ , an asymmetric unification problem with a linear constant restriction, we construct a general unification problem Γ' such that Γ is solvable iff Γ' is solvable. Let $<$ denote the linear ordering. Let \mathcal{X} denote the variables of Γ and let C denote the set of all free constants in Γ . Now, we construct Γ' as follows: The free constants in Γ are treated as variables in Γ' . For each free constant c of Γ we add a new free function symbol f_c which has arity $|V_c|$. Recall that $V_c = \{x \in \mathcal{X} | x < c\}$. $\Gamma' = \Gamma \cup \{c =^{\downarrow} f_c(x_1, \dots, x_n) \mid c \in C \text{ and } V_c = \{x_1, \dots, x_n\}\}$

Theorem 7. *The Asymmetric E-unification problem with linear constant restriction, Γ , is solvable iff the general Asymmetric E-unification problem Γ' is solvable.*

Proof. The same proof used in [2] can here with only a modification for asymmetric equations. A full proof is given in [9]. \square

3 Conclusions

With respect to efficiency, the combination algorithm provides a significant first step to more efficient methods since, unlike a narrowing approach, we can now combine efficient special purpose asymmetric unification algorithms. In addition, it should be possible to improve the efficiency of the current algorithm. We are currently studying the question of improving the efficiency.

Briefly, the only theories that are currently known to have asymmetric unification algorithms are those with the *finite variant property* [6], in which case a general algorithm known as *folding variant narrowing* [12] applies. This is a sizable class, including many, but not all, theories of interest to cryptographic protocol analysis (see [6]). In many cases known characterizations of theories with the finite variant property [12], [5] depend on conditions on E and R that can be checked without further reference to Σ , and so for these cases the finite variant property still holds after the addition of uninterpreted function symbols. Thus general asymmetric unification algorithms exist. Moreover, the earlier mentioned special-purpose algorithms for exclusive-or and free Abelian groups [14], [8] are also general asymmetric algorithms. In [14] and [8] a strategy is presented for converting symmetric unification algorithms to asymmetric ones. This opens up an avenue for the development of special-purpose general asymmetric unification algorithms for theories with and without the finite variant property as well, to which our results would also apply.

There exists an interesting connection between Asymmetric unification and Disunification. Consider a disunification problem $s \neq t$ in the theory $\Delta = E \cup R$ over signature Σ . We can simulate this problem using asymmetric unification. First, let f and g be new function symbols added to Σ . Let $f(x, x) \rightarrow g(x)$ be a new rule added to R . Now $s \neq t$ can be simulated by $\{s =^\downarrow u, t =^\downarrow v, w =^\downarrow f(u, v)\}$. Although there is some connection between the two problems they may still be independent and resolving this is an interesting open problem.

References

1. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
2. Baader, F., Schulz, K.U.: Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation* 21(2), 211–243 (1996)
3. Baader, F., Snyder, W.: Unification Theory. In: Robinson, J.A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, pp. 445–532. Elsevier and MIT Press (2001)
4. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: *Proceedings of the 14th IEEE Workshop on Computer Security Foundations, CSFW 2001*, pp. 82–96. IEEE Computer Society (2001)
5. Bouchard, C., Gero, K.A., Lynch, C., Narendran, P.: On Forward Closure and the Finite Variant Property. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) *FroCoS 2013. LNCS*, vol. 8152, pp. 327–342. Springer, Heidelberg (2013)
6. Comon-Lundh, H., Delaune, S.: The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In: Giesl, J. (ed.) *RTA 2005. LNCS*, vol. 3467, pp. 294–307. Springer, Heidelberg (2005)

7. Erbatur, S., Escobar, S., Kapur, D., Liu, Z., Lynch, C., Meadows, C., Meseguer, J., Narendran, P., Santiago, S., Sasse, R.: Effective Symbolic Protocol Analysis via Equational Irreducibility Conditions. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 73–90. Springer, Heidelberg (2012)
8. Erbatur, S., Escobar, S., Kapur, D., Liu, Z., Lynch, C.A., Meadows, C., Meseguer, J., Narendran, P., Santiago, S., Sasse, R.: Asymmetric Unification: A New Unification Paradigm for Cryptographic Protocol Analysis. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 231–248. Springer, Heidelberg (2013)
9. Erbatur, S., Kapur, D., Marshall, A.M., Meadows, C., Narendran, P., Ringeissen, C.: On Asymmetric Unification and the Combination Problem in Disjoint Theories. INRIA Research Report (2014), <http://hal.inria.fr/>
10. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2009)
11. Escobar, S., Meseguer, J., Sasse, R.: Variant Narrowing and Equational Unification. *Electronic Notes Theor. Comput. Science* 238(3), 103–119 (2009)
12. Escobar, S., Sasse, R., Meseguer, J.: Folding Variant Narrowing and Optimal Variant Termination. *J. Log. Algebr. Program.* 81(7-8), 898–928 (2012)
13. Jouannaud, J.-P., Toyama, Y.: Modular Church-Rosser Modulo: The Complete Picture. *Int. J. Software and Informatics* 2(1), 61–75 (2008)
14. Liu, Z.: Dealing Efficiently with Exclusive OR, Abelian Groups and Homomorphism in Cryptographic Protocol Analysis. PhD thesis, Clarkson University (2012)
15. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013)
16. Mödersheim, S.: Models and methods for the automated analysis of security protocols. PhD thesis, ETH Zurich (2007)
17. Rusinowitch, M.: On Termination of the Direct sum of Term-Rewriting Systems. *Information Processing Letters* 26, 65–70 (1987)

Axiomatizing Bisimulation Equivalences and Metrics from Probabilistic SOS Rules[★]

Pedro R. D'Argenio¹, Daniel Gebler², and Matias David Lee¹

¹ Universidad Nacional de Córdoba & CONICET, Argentina

² VU University Amsterdam, The Netherlands

Abstract. Probabilistic transition system specifications (PTSS) provide structural operational semantics for reactive probabilistic labeled transition systems. Bisimulation equivalences and bisimulation metrics are fundamental notions to describe behavioral relations and distances of states, respectively. We provide a method to generate from a PTSS a sound and ground-complete equational axiomatization for strong and convex bisimilarity. The construction is based on the method of Aceto, Bloom and Vaandrager developed for non-deterministic transition system specifications. The novelty in our approach is to employ many-sorted algebras to axiomatize separately non-deterministic choice, probabilistic choice and their interaction. Furthermore, we generalize this method to axiomatize the strong and convex metric bisimulation distance of PTSS.

1 Introduction

Structural operational semantics (SOS for short) [20] is a powerful tool to provide semantics to programming languages. In SOS, process behavior is described using transition systems and the behavior of a composite process is given in terms of the behavior of its components. Based on this technique, different meta-properties have been studied. They state general properties on process operations by only inspecting the format of the rules that define the semantics of this operator. Among them, congruence and other compositionality properties stand out. (See [19] for an overview.)

However, there are properties that are better understood from an axiomatic point of view, by regarding the language as a signature equipped with an equational theory (see e.g. [18,3]). This is a different way to understand the language that brings new insights on the behavior of its operators and processes. General properties, such as associativity, distributivity, or reduction to basic operators, or specific ones, can be easily derived with equational reasoning, which is also used for the verification of systems.

In [1], Aceto, Bloom and Vaandrager link these two approaches by providing an algorithm to derive an equational theory for any language whose semantics is defined in terms of SOS rules that meet the GSOS format [7]. This equational theory is sound and ground-complete for bisimulation equivalence [18]. For recent work in the area, see [2,11] and references therein.

[★] Supported by ANPCYT PICT 2012-1823, SeCyT-UNC 05/B497 and 05/BP02, Erasmus Mundus Action 2 Lot 13A EU Mobility Programme 2010-2401/001-001-EMA2 and EU 7FP grant agreement 295261 (MEALS).

The above mentioned results are set in traditional non-deterministic semantics. However, in the modeling and programming of systems, the interaction of non-determinism and probabilities arises naturally, for example, in the sampling of a random number or in the occurrence of an externally induced fault. Therefore, modeling and programming languages need to have operations whose semantics include probabilistic behavior.

For probabilistic languages, SOS theories have also been developed in which not only congruence properties are considered, but also non-expansiveness, which is a concept that arises naturally when measuring distances in the probabilistic behavior of two processes (see, e.g., [9,12] and references therein). Moreover, equational theories for probabilistic languages have been developed (see [5,15] and references therein).

In this work we lift the result of [1] to languages with probabilistic operations. The input of our algorithm is an SOS system (more precisely, a PTSS) in PGSOS format (actually, it is a generalization of the Segala-GSOS format [6]) and the output is a sound and ground-complete equational theory for strong bisimulation equivalence. Having this aim, we came across with additional contributions, more precisely:

1. In Sec. 3, we generalize the PGSOS format to two-sorted signatures in order to syntactically denote states and distributions. By doing so, operations can be parameterized on distributions, and moreover, we can neatly express open terms in the rules of the PTSS. While the syntax somehow resembles the alternating model of probabilistic processes, we continue the research line of [9,17,12] and let PTSS have models in Segala's probabilistic automata. We show that strong bisimulation equivalence [16] and convex bisimulation equivalence (also called probabilistic bisimulation) [21] are congruences for any operation whose semantics is defined in PGSOS format.
2. In Sec. 4, we provide an algorithm that takes a PGSOS system, and produces an equational theory that is sound and ground-complete for strong bisimulation equivalence. We show ground-completeness for semantically well-founded PGSOS systems, and we indicate how this result can be extended to arbitrary PGSOS. We show how our algorithm easily extends to derive a sound and ground-complete equational theories for convex bisimulation equivalence.
3. As a by-product we needed to define a two-sorted calculus for finite probabilistic processes equipped with two sound and ground-complete equational theories, one for each bisimulation equivalence. This calculus is adapted from [5]. (See Sec. 2.)
4. In Sec. 5, we provide an equational theory for the basic calculus that captures exactly the notion of (strong) bisimilarity metric [10]. The equational theory is sound in the sense that, whenever the equality between the distance of two processes and the distance of two other processes (or a particular value) can be calculated with the calculus, it can also be calculated semantically in the probabilistic transition system. We show that it is also ground-complete (i.e. the inverse implication holds for closed terms).
5. Also in Sec. 5, we modify the previous algorithm to derive a sound and ground-complete equational theory for bisimilarity metric from a given PGSOS system.

2 Preliminaries

Let $S = \{s, d\}$ be a set denoting two sorts. States of the transition system will be of sort $s \in S$ and distributions over states of sort $d \in S$. We let σ range over the sorts

in S . We write S -sorted families X as pairs (X_s, X_d) with the first element X_s denoting the member of sort s and the second element X_d denoting the member of sort d . An S -sorted signature is a structure (F, ar) , where (i) F is a set of *function names*, and (ii) $\text{ar}: F \rightarrow (S^* \times S)$ is the *arity function*. The rank of $f \in F$ is the number of arguments of f , defined by $\text{rk}(f) = n$ if $\text{ar}(f) = \sigma_1 \dots \sigma_n \rightarrow \sigma$. (We write “ $\sigma_1 \dots \sigma_n \rightarrow \sigma$ ” instead of “ $(\sigma_1 \dots \sigma_n, \sigma)$ ”). Function f is a *constant* if $\text{rk}(f) = 0$. To simplify the presentation we will write an S -sorted signature (F, ar) as a pair of disjoint signatures (Σ, Γ) where Σ is the set of operations that map to s and Γ is the set of operations that map to d .

Let $(\mathcal{V}, \mathcal{D})$ be an infinite set of S -sorted variables where $\mathcal{V}, \mathcal{D}, F$ are all mutually disjoint. We use x, y, z (with possible sub- or sup-indexes) to range over \mathcal{V} , μ, ν to range over \mathcal{D} and ζ to range over $\mathcal{V} \cup \mathcal{D}$. The S -sorted set of Σ -terms over $(V, D) \subseteq (\mathcal{V}, \mathcal{D})$, notation $(T(\Sigma, V), T(\Gamma, D))$, is the smallest set satisfying: (i) $V \subseteq T(\Sigma, V)$, (ii) $D \subseteq T(\Gamma, D)$, (iii) $f(t_1, \dots, t_{\text{rk}(f)}) \in T(\Sigma, V)$, if $\text{ar}(f) = \sigma_1 \dots \sigma_n \rightarrow \sigma$, $\sigma = s$, $t_i \in T(\Sigma, V)$ whenever $\sigma_i = s$, and $t_i \in T(\Gamma, D)$ whenever $\sigma_i = d$, and (iv) $f(t_1, \dots, t_{\text{rk}(f)}) \in T(\Gamma, D)$, if instead $\sigma = d$. $(T(\Sigma, \mathcal{V}), T(\Gamma, \mathcal{D}))$ is the set of all *open terms* and is denoted by $(\mathbb{T}(\Sigma), \mathbb{T}(\Gamma))$. $(T(\Sigma, \emptyset), T(\Gamma, \emptyset))$ is the set of all *closed terms* and is denoted by $(\mathbb{C}(\Sigma), \mathbb{C}(\Gamma))$. $\text{Var}(t) \subseteq (\mathcal{V}, \mathcal{D})$ denotes the S -sorted set of variables in term t . We let ξ range over terms of both sorts $T(\Sigma) \cup T(\Gamma)$.

Let $\Delta(T(\Sigma))$ denote the set of all (discrete) probability distributions on $T(\Sigma)$. We let π range over $\Delta(T(\Sigma))$ and ψ range over $\Delta(T(\Sigma)) \cup T(\Gamma)$. For each $t \in T(\Sigma)$, let δ_t denote the *Dirac distribution*, i.e., $\delta_t(t) = 1$ and $\delta_t(t') = 0$ if t and t' are not syntactically equal. For $X \subseteq T(\Sigma)$ we define $\pi(X) = \sum_{t \in X} \pi(t)$. The convex combination $\sum_{i \in I} p_i \pi_i$ of a family $\{\pi_i\}_{i \in I}$ of probability distributions with $p_i \in (0, 1]$ and $\sum_{i \in I} p_i = 1$ is defined by $(\sum_{i \in I} p_i \pi_i)(t) = \sum_{i \in I} (p_i \pi_i(t))$.

We fix the signature to describe probability distributions of finite support by $\Gamma_\Delta = (F_\Delta, \text{ar}_\Delta)$ with $F_\Delta = \{\delta, \oplus_p \mid p \in \mathbb{Q} \cap (0, 1)\}$, and $\text{ar}_\Delta(\delta) = s \rightarrow d$ and $\text{ar}_\Delta(\oplus_p) = dd \rightarrow d$. Given an arbitrary S -sorted signature with $\Sigma_s = (F_s, \text{ar}_s)$, the operations that map to sort s and all function symbols in F_s and F_Δ are disjoint. We define the *probabilistic lifting* of Σ_s as an S -sorted signature (Σ, Γ) with $\Sigma = \Sigma_s$ and $\Gamma = (F_d, \text{ar}_d)$ extending Γ_Δ such that for each $f \in F_s$ there is a new distinct function symbol $\mathbf{f} \in F_d$ with $\text{ar}(\mathbf{f}) = d \dots d \rightarrow d$ and $\text{rk}(\mathbf{f}) = \text{rk}(f)$. (Operators in boldface are probabilistically lifted.)

The algebra associated with a probabilistically lifted signature (Σ, Γ) is defined as follows. For sort s , it is the freely generated algebra $T(\Sigma)$. For sort d , it is defined by the carrier $\Delta(T(\Sigma))$ and the following interpretation: $\llbracket \delta(t) \rrbracket = \delta_t$ for $t \in T(\Sigma)$, $\llbracket \theta_1 \oplus_p \theta_2 \rrbracket = p \llbracket \theta_1 \rrbracket + (1-p) \llbracket \theta_2 \rrbracket$ for $\theta_1, \theta_2 \in T(\Gamma)$, $\llbracket \mathbf{f}(\theta_1, \dots, \theta_{\text{rk}(f)}) \rrbracket (f(\xi_1, \dots, \xi_{\text{rk}(f)})) = \prod_{\sigma_j = s} \llbracket \theta_j \rrbracket (\xi_j)$ if for all $\sigma_j = d$, θ_j and ξ_j are syntactically equal, and, in any other case, $\llbracket \mathbf{f}(\theta_1, \dots, \theta_{\text{rk}(f)}) \rrbracket (t) = 0$.

A *substitution* is an S -indexed family of maps $(\rho_s, \rho_d): (\mathcal{V}, \mathcal{D}) \rightarrow (\mathbb{T}(\Sigma), \mathbb{T}(\Gamma))$. A substitution is closed if it maps each variable to a closed term. A substitution extends to a mapping from terms to terms as usual.

3 Probabilistic Transition System Specifications

Probabilistic transition systems (PTSs) generalize labelled transition systems by allowing for probabilistic choices in the transitions. We consider non-deterministic PTSs

(Segala-type systems) [21] with countable state spaces. A *probabilistic labeled transition system* (PTS) is a triple $(T(\Sigma), A, \rightarrow)$, where Σ is a signature specifying only functions with target sort s , A is a countable set of actions, and $\rightarrow \subseteq T(\Sigma) \times A \times \mathcal{A}(T(\Sigma))$ is a transition relation. We write $t \xrightarrow{a} \pi$ for $(t, a, \pi) \in \rightarrow$. Satisfaction is defined by $\rightarrow \models t \xrightarrow{a} \pi$ if $t \xrightarrow{a} \pi \in \rightarrow$, and $\rightarrow \not\models t \xrightarrow{a} \pi$ if $t \xrightarrow{a} \pi \notin \rightarrow$ for all $\pi \in \mathcal{A}(T(\Sigma))$.

We specify PTSs by means of transition system specifications [20,7,14]. We generalize the probabilistic GSOS format of [6] with operators of sort s with arguments of either sort s or d . From now on, (Σ, Γ) denotes a probabilistically lifted signature.

Definition 1 (PGSOS-rule). A PGSOS-rule has the form:

$$\frac{\{x_i \xrightarrow{a_{i,m}} \mu_{i,m} \mid i \in I, m \in M_i\} \quad \{x_i \xrightarrow{b_{i,n}} \nu_{i,n} \mid i \in I, n \in N_i\}}{f(\zeta_1, \dots, \zeta_{\text{rk}(f)}) \xrightarrow{a} \theta}$$

with $f \in F$ a function symbol, I, M_i, N_i are finite index sets, $a_{i,m}, b_{i,n}, a \in A$ are actions, $x_i \in \mathcal{V}$, $\zeta_i \in \mathcal{V} \cup \mathcal{D}$, $\mu_{i,m} \in \mathcal{D}$ are variables, $\theta \in \mathbb{T}(\Gamma)$ a distribution term, and satisfying the following constraints:

1. all $\mu_{i,m}$ and ζ_j , for $i \in I, m \in M_i$ and $j \in \{1, \dots, \text{rk}(f)\}$, are pairwise different;
2. $\{x_i \mid i \in I\} \subseteq \{\zeta_1, \dots, \zeta_{\text{rk}(f)}\}$;
3. $\text{Var}(\theta) \subseteq \{\mu_{i,m} \mid i \in I, m \in M_i\} \cup \{\zeta_1, \dots, \zeta_{\text{rk}(f)}\}$.

A *probabilistic transition system specification* in PGSOS format (PTSS) is a structure $P = (\Sigma, A, R)$ where Σ is a probabilistically lifted signature, A is a finite set of labels and R is a finite set of PGSOS rules. 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 $\text{conc}(r)$. Given a positive literal $t \xrightarrow{a} \theta$ and a closed substitution ρ , $\llbracket t \xrightarrow{a} \theta \rrbracket_\rho$ denotes the transition $\rho(t) \xrightarrow{a} \llbracket \rho(\theta) \rrbracket$. For negative literals, $\llbracket t \xrightarrow{a} \theta \rrbracket_\rho$ denotes $\rho(t) \not\xrightarrow{a} \theta$. A *supported model* of P is a PTS $(T(\Sigma), A, \rightarrow)$ satisfying that $t \xrightarrow{a} \pi \in \rightarrow$ iff there is a rule $r \in R$ with a substitution ρ such that all premises of r hold, i.e. $\rightarrow \models \llbracket \text{prem}(r) \rrbracket_\rho$, and the conclusion instantiates to $t \xrightarrow{a} \pi$, i.e. $\llbracket \text{conc}(r) \rrbracket_\rho = t \xrightarrow{a} \pi$. Each PTSS has a supported model which is, moreover, unique.

A set $X \subseteq T(\Sigma)$ is closed with respect to a binary relation $R \subseteq T(\Sigma) \times T(\Sigma)$ if $R(X) \subseteq X$ where $R(X) = \{t' \in T(\Sigma) \mid \exists t \in X. t R t'\}$. A relation $R \subseteq T(\Sigma) \times T(\Sigma)$ on terms of sort s lifts to a relation $\overline{R} \subseteq \mathcal{A}(T(\Sigma)) \times \mathcal{A}(T(\Sigma))$ on distributions over terms of sort s by $\pi \overline{R} \pi'$ iff $\pi(X) = \pi'(X)$ for all $X \subseteq T(\Sigma)$ that are closed with respect to R .

Definition 2 ([16]). Let $(T(\Sigma), A, \rightarrow)$ be a PTS. A symmetric relation $R \subseteq T(\Sigma) \times T(\Sigma)$ is a strong bisimulation if whenever $t R t'$ and $t \xrightarrow{a} \pi$, there exists a transition $t' \xrightarrow{a} \pi'$ such that $\pi \overline{R} \pi'$. Strong bisimilarity \sim is defined as the union of all strong bisimulations.

The convex closure $cl(D)$ of a set of distributions $D \subseteq \mathcal{A}(T(\Sigma))$ is the least subset of $\mathcal{A}(T(\Sigma))$ which contains D and is closed under convex combination. A combined transition $t \xrightarrow{a}_c \pi$ is given whenever $\pi \in cl(\{\pi' \mid t \xrightarrow{a} \pi'\})$.

Definition 3 ([21]). Let $(T(\Sigma), A, \rightarrow)$ be a PTS. A symmetric relation $R \subseteq T(\Sigma) \times T(\Sigma)$ is a convex bisimulation if whenever $t R t'$ and $t \xrightarrow{a} \pi$, there exists a combined transition

$t' \xrightarrow{a}_c \pi'$ such that $\pi \overline{R} \pi'$. Convex bisimilarity \sim_c is defined as the union of all convex bisimulations.

A crucial property of process description languages to ensure compositional modeling is the compatibility of process operators with the chosen behavioral relation. In algebraic terms the compatibility of an equivalence R with an operator f is expressed by the congruence property which is defined as $f(\xi_1, \dots, \xi_{rk(f)}) R f(\xi'_1, \dots, \xi'_{rk(f)})$ whenever $\xi_i R \xi'_i$ with $\xi_i, \xi'_i \in T(\Sigma)$ if $\sigma_i = s$ and $\xi_i \overline{R} \xi'_i$ with $\xi_i, \xi'_i \in T(\Gamma)$ if $\sigma_i = d$. The PGSOS rule format ensures that both strong and convex bisimilarity are congruences.

Theorem 1. *Let $P = (\Sigma, A, R)$ be a PTSS in PGSOS format. Then, both strong and convex bisimilarity are congruences for all operators defined by P .*

4 Axiomatization of Bisimulation Equivalences

The technique to derive an axiomatization for PGSOS operators follows the same strategy as in [1]. It starts with a given axiomatization of a basic calculus which is a probabilistic extension of CCS similar to the one studied in [5]. Then, according to the rules, axioms are provided for any other operator so that these operators can be eliminated, in the sense that every closed term can be equated to another closed term in the basic calculus. To introduce these new axioms, operators are split in three classes: distinctive, smooth, and non-smooth. *Distinctive operators* are well-behaved operators that distribute with summation and the probabilistic operators \oplus_p and δ . The defining rules for distinctive operators can be directly mapped into axioms. *Smooth operators* are a generalization of distinctive operators in the sense that the set of rules defining the semantics of a smooth operator can be split in disjoint sets, each one of them satisfying the conditions of distinctive operators. Thus a smooth operator can be represented as a non-deterministic sum of distinctive operators. For each *non-smooth operator*, a new smooth operator is introduced that, when properly instantiated, shows the same behavior as the original non-smooth operator. Precisely the equality between these terms is introduced as a new axiom. This section presents these results and provides an algorithm that, given a PTSS P in PGSOS format, generates an axiom system for all operators in P that is sound and ground-complete for strong bisimilarity. We close the section with an explanation on how the technique extends to convex bisimilarity.

Axiomatizing Finite Probabilistic Trees. Let Σ_{CCS} be the signature of the (recursion free) *basic probabilistic CCS* defined by constant 0 of sort s , binary operation $+$ with $\text{ar}(+) = ss \rightarrow s$ and prefix operators a with $\text{ar}(a) = d \rightarrow s$ for all $a \in A$. We write $a.\theta$ for $a(\theta)$ with $\theta \in \mathbb{T}(\Gamma)$. The PTSS $P_{CCS} = (\Sigma_{CCS}, A, R)$ is given by the following rules R :

$$\frac{}{a.\mu \xrightarrow{a} \mu} \quad \frac{x \xrightarrow{a} \mu}{x + y \xrightarrow{a} \mu} \quad \frac{y \xrightarrow{a} \mu}{x + y \xrightarrow{a} \mu} \quad (1)$$

A closed term $t \in T(\Sigma)$ is in *normal form* if either $t = 0$ or $t = \sum_{i \in I} a_i.\theta_i$ with $\theta_i \in T(\Gamma)$ in normal form. A closed term $\theta \in T(\Gamma)$ is in normal form if $\theta = \bigoplus_{i \in I} p_i \delta(t_i)$, with $t_i \in T(\Sigma)$ in normal form and $\sum_{i \in I} p_i = 1$. Here, $\bigoplus_{i \in \{1..n\}} p_i \theta_i$ is a shorthand for $\theta_1 \oplus \frac{p_1}{\sum_{j=1}^n p_j} (\theta_2 \oplus \frac{p_2}{\sum_{j=2}^n p_j} (\dots (\theta_{n-1} \oplus \frac{p_{n-1}}{\sum_{j=n-1}^n p_j} \theta_n) \dots))$, and $\sum_{i \in \{1..n\}} t_i$ is a shorthand for $t_1 + \dots + t_n$.

Table 1. Axiomatization of strong and convex bisimilarity of CCS

$x + y = y + x$	(N1)	$(\mu_1 \oplus_p \mu_2) + \mu_3 = (\mu_1 + \mu_3) \oplus_p (\mu_2 + \mu_3)$	(NP1)
$(x + y) + z = x + (y + z)$	(N2)	$\mu_1 + (\mu_2 \oplus_p \mu_3) = (\mu_1 + \mu_2) \oplus_p (\mu_1 + \mu_3)$	(NP2)
$x + 0 = x$	(N3)	$\delta(x) + \delta(y) = \delta(x + y)$	(NP3)
$x + x = x$	(N4)		
		$a.\mu_1 + a.\mu_2 = a.\mu_1 + a.\mu_2 + a.(\mu_1 \oplus_p \mu_2)$	(CC)
$\mu \oplus_p \mu = \mu$	(P1)		
$\mu_1 \oplus_p \mu_2 = \mu_2 \oplus_{1-p} \mu_1$	(P2)	$\mu_1 \oplus_{p_1} (\mu_2 \oplus_{\frac{p_2}{1-p_1}} \mu_3) = (\mu_1 \oplus_{\frac{p_1}{p_1+p_2}} \mu_2) \oplus_{p_1+p_2} \mu_3$	(P3)

Let E_{CCS} be the set of equations of Table 1 without equation CC. The axioms N1–N4 are standard for non-deterministic choice of reactive systems [18]. The axioms P1–P3 are standard for probabilistic choice [5]. Moreover, axioms NP1–NP3 allow one to normalize distribution terms in a similar way to the normalization of state terms by axioms N1–N4. The axiomatization of [5] did not require those axioms because distribution terms were assumed to be already in normal form.

Equational reasoning over many-sorted algebras [13] requires non-empty carrier sets. For E_{CCS} and all its following extensions this holds since $0 \in T(\Sigma)$ and $\delta(0) \in T(\Gamma)$. A set of S -sorted equations E over signature Σ is a sound and ground-complete axiomatization of strong bisimilarity of P if for all $t, t' \in T(\Sigma)$, $E \vdash t = t'$ iff $t \sim t'$.

In order to show ground-completeness of E_{CCS} we require that the axiomatization is normalizing for both sort s and d , i.e. that for each closed term $\xi \in T(\Sigma) \cup T(\Gamma)$ there is a closed term $\xi' \in T(\Sigma) \cup T(\Gamma)$ in normal form such that $E_{CCS} \vdash \xi = \xi'$. The proof of the next lemma follows as usual by transforming the axiom system into a term rewriting system, showing that it is strongly normalizing modulo commutativity and associativity, and that the normal form is indeed of the expected form.

Lemma 1. *The axiom system E_{CCS} is normalizing.*

The proof of soundness for axioms involving state terms follows standard lines: for each axiom we find a bisimulation relation that shows its instances are valid with respect to bisimilarity. For axioms on distribution terms we prove that both sides of the equation represent exactly the same distribution. Ground-completeness is proven by first reducing to normal form and then showing that, for two bisimilar state terms in normal form, the transfer properties induce a proof using the axioms. Similarly, two distribution terms in normal form that represent the same distribution up to bisimulation, can be reduced to the same term using the axioms.

Theorem 2. *E_{CCS} is sound and ground-complete for strong bisimilarity.*

In order to derive axioms for systems with rules including negative premises, following [1], we introduce the family of one-step restriction operators ∂_H^1 , where $H \subseteq A$, $\text{ar}(\partial_H^1) = s \rightarrow s$, and whose semantics is given by

$$\frac{x \xrightarrow{a} \mu}{\partial_H^1(x) \xrightarrow{a} \mu} \quad a \notin H \quad (2)$$

Table 2. Axioms for ∂_H^1

$\partial_H^1(x + y) = \partial_H^1(x) + \partial_H^1(y)$	(H1)	$\partial_H^1(0) = 0$	(H4)
$\partial_H^1(a.\mu) = a.\mu$ if $a \notin H$	(H2)	$\partial_H^1(\mu_1 \oplus_p \mu_2) = \partial_H^1(\mu_1) \oplus_p \partial_H^1(\mu_2)$	(H5)
$\partial_H^1(a.\mu) = 0$ if $a \in H$	(H3)	$\partial_H^1(\delta(x)) = \delta(\partial_H^1(x))$	(H6)

$\partial_H^1(t)$ represents the inability to perform any action $a \in H$ in the next step, otherwise behaving as t . The signature Σ_{CCS° extends Σ_{CCS} with operators ∂_H^1 . $P_{CCS^\circ} = (\Sigma_{CCS^\circ}, A, R_{CCS^\circ})$ is the PTSS whose set of rules R_{CCS° extends R_{CCS} with the family of rules given in (2).

Let E_{CCS° extend E_{CCS} with equations in Table 2. H1–H4 are standard for the one-step restriction operator [1]. H5 and H6 propagate the one-step restriction operation to each single term in the support of a distribution. Hence, restriction distributes over probabilistic choices and Dirac embedding. Soundness of H1–H4 is proven in the same way as for the non-probabilistic case [1]. Soundness of H5 and H6 is proven by showing that both sides of each axiom represent exactly the same distribution. ∂_H^1 can be eliminated in the sense that for each closed term $\xi \in T(\Sigma_{CCS^\circ}) \cup T(\Gamma_{CCS^\circ})$ there is a closed term $\xi' \in T(\Sigma_{CCS}) \cup T(\Gamma_{CCS})$ such that $E_{CCS^\circ} \vdash \xi = \xi'$. This can be proven by induction on the height of a term. (Notice that, when read from left to right, axioms H1, H5, and H6 “push” operator ∂_H^1 inside the term, while axioms H2–H4, remove it.) Using elimination and Thm. 2, ground-completeness follow immediately.

Theorem 3. E_{CCS° is sound and ground-complete for strong bisimilarity.

Probabilistically Lifted Operators. The semantics of all probabilistically lifted operators is defined following the same scheme. Thus, the axioms for these operators are defined similarly regardless of whether the original operator is distinctive, smooth or non-smooth. There are actually two types of axioms that explain how a lifted operation interacts with the probabilistic operations \oplus_p and δ .

Definition 4. Let f be an operator with arity $\text{ar}(f) = \sigma_1 \dots \sigma_{\text{rk}(f)} \rightarrow s$. We associate with f the axiom system E_f consisting of the following equations:

1. Probabilistic distributivity laws: For each position i of f , such that $\sigma_i = s$, and for each $p \in \mathbb{Q} \cap (0, 1)$ we have the equations

$$f(\mu_1, \dots, \mu'_i \oplus_p \mu''_i, \dots, \mu_{\text{rk}(f)}) = f(\mu_1, \dots, \mu'_i, \dots, \mu_{\text{rk}(f)}) \oplus_p f(\mu_1, \dots, \mu''_i, \dots, \mu_{\text{rk}(f)})$$

2. Dirac distributivity laws: We have the equation

$$f(\theta_1, \dots, \theta_{\text{rk}(f)}) = \delta(f(\zeta_1, \dots, \zeta_{\text{rk}(f)}))$$

with $\theta_i = \delta(\zeta_i)$, $\zeta_i \in \mathcal{V}$ if $\sigma_i = s$ and $\theta_i = \zeta_i$, $\zeta_i \in \mathcal{D}$ if $\sigma_i = d$.

The soundness of these laws follows immediately from the semantics of \oplus_p , δ and the lifted operator, and using rational arithmetic. (Rational arithmetic is completely axiomatized for ground terms, which are the only ones we use, see e.g. [8]).

Smooth and Distinctive Operators. A smooth rule is a rule that, whenever a variable is tested in a positive literal, then it is the only literal that tests that variable and the tested variable does not occur in the target of the conclusion. A smooth operator is an operator defined only by smooth rules. A distinctive operator is a smooth operator in which the hypotheses of each pair of rules differ in at least one literal.

Definition 5. A PGSOS rule is smooth if it has the form

$$\frac{\{x_i \xrightarrow{a_i} \mu_i \mid i \in I\} \quad \{x_j \xrightarrow{b_{j,n}} \nu_j \mid j \in J, n \in N_j\}}{f(\zeta_1, \dots, \zeta_{\text{rk}(f)}) \xrightarrow{a} \theta} \quad (3)$$

where I and J are disjoint sets s.t. $I \cup J = \{i \in \{1, \dots, \text{rk}(f)\} \mid \zeta_i \in \mathcal{V}\}$, and $x_i \notin \text{Var}(\theta)$ if $i \in I$. An operator f is smooth if all its defining rules are smooth.

A smooth operator f is distinctive if (i) each f -defining rule tests the same set of arguments I positively, and (ii) for every two different f -defining rules there is some argument $\zeta_i \in \mathcal{V}$ tested positively by both rules, but with a different action.

Notice that $+$ is smooth, but it is not distinctive since, e.g., x is tested positively in the first rule (actually, a set of rules), but not in the second one. Instead, ∂_H^1 is distinctive.

We introduce a new operator that we will use in our examples. Assume that each action a may fail with probability $p_a \in [0, 1)$. In case of failure, the occurrence of a is ignored and the system remains in the same state, otherwise, it proceeds normally. The new operator $\text{sc}(t)$ is a safe controller that minimizes the probability of failure of process t . Its semantics is given by the rules

$$\frac{x \xrightarrow{a} \mu \quad \{x \xrightarrow{b} \nu \mid p_b < p_a, p_a \neq 0\}}{\text{sc}(x) \xrightarrow{a} \delta(x) \oplus_{p_a} \text{sc}(\mu)}, \text{ if } p_a > 0 \quad \frac{x \xrightarrow{a} \mu}{\text{sc}(x) \xrightarrow{a} \text{sc}(\mu)}, \text{ if } p_a = 0$$

sc is a variant of the ACP-style priority operator and it is not smooth since the rule on the left tests x in the positive literal but also in the negative literal, and, moreover, x appears in the target of the conclusion.

Let $\text{pos}(r) = I$ (resp. $\text{neg}(r) = J$) be those positions which are positively (resp. negatively) tested by rule r (considering r as in (3)). Let $\text{pact}(r, i) = \{a_i \mid i \in I\}$ (resp. $\text{nact}(r, i) = \{b_{i,n} \mid n \in N_i\}$) be those actions for which x_i is positively (resp. negatively) tested by rule r . Note that if $N_i = \emptyset$ then $\text{nact}(r, i) = \emptyset$. A position i of operator f is *positive* if $i \in \text{pos}(r)$ for all rules r defining f .

Definition 6. Let f be a distinctive operator with arity $\text{ar}(f) = \sigma_1 \dots \sigma_{\text{rk}(f)} \rightarrow s$. Let $\zeta_i \in \mathcal{V}$ if $\sigma_i = s$ and $\zeta_i \in \mathcal{D}$ if $\sigma_i = d$ for $1 \leq i \leq \text{rk}(f)$. We associate with f the axiom system E_f consisting of the following equations:

1. Non-deterministic distributivity laws: For each positive position i of f , we have

$$f(\zeta_1, \dots, \zeta_i' + \zeta_i'', \dots, \zeta_{\text{rk}(f)}) = f(\zeta_1, \dots, \zeta_i', \dots, \zeta_{\text{rk}(f)}) + f(\zeta_1, \dots, \zeta_i'', \dots, \zeta_{\text{rk}(f)})$$

2. Action laws: For each f -defining rule r (as in (3)), we have the equation

$$\rho(f(\zeta_1, \dots, \zeta_{\text{rk}(f)})) = a.\rho(\theta)$$

with $\theta = \text{trgt}(r)$ the target of r and substitution ρ defined by

$$\rho(\zeta) = \begin{cases} a_i \cdot \mu_i & \text{if } \zeta = x_i \text{ with } i \in \text{pos}(r) \\ \partial_H^1(x_i) & \text{if } \zeta = x_i \text{ with } i \in \text{neg}(r) \text{ and } H = \text{nact}(r, i) \neq \emptyset \\ \zeta & \text{otherwise.} \end{cases}$$

3. Inaction laws: We have the equations

$$\rho(f(\zeta_1, \dots, \zeta_{\text{rk}(f)})) = 0$$

for all sort-respecting substitutions ρ mapping into terms of the form 0 , x , $a \cdot \mu$, $b \cdot \mu + x$, or μ , such that for every f -defining rule r there is some position i with sort $\sigma_i = s$ satisfying one of the following conditions: (i) if $i \in \text{pos}(r)$, then either $\rho(\zeta_i) = 0$ or $\rho(\zeta_i) = a \cdot \mu_i$ with $a \notin \text{pact}(r, i)$, or (ii) if $i \in \text{neg}(r)$, then $\rho(\zeta_i) = b \cdot \mu_i + x$ with $b \in \text{nact}(r, i)$.

The fact that all rules of a distinctive operator f test positively the same positions guarantees the soundness of the non-deterministic distributivity law. There is one action law for each rule of f . The action law describes the execution of an action by pushing the executing action to the “head” of the term. The conditions of its associated rule are properly encoded in each operand of f . Contrarily to the action law, an inaction law traverses every f -defining rule ensuring through the operands that at least one of the conditions of each rule does not hold.

Soundness of the axioms in E_f can be proven regardless of the PTSS containing operator f as long as the set of rules defining the semantics of f is the same for any PTSS. That is, if f is defined in a PTSS P , E_f is sound for any disjoint extension of P .

Definition 7. Let $P = (\Sigma, A, R)$ and $P' = (\Sigma', A, R')$ be two PTSSs in PGSOS format. P' is a disjoint extension of P , notation $P' \sqsupseteq P$, iff $\Sigma \subseteq \Sigma'$, $R \subseteq R'$ and R' introduces no new rule for any operation in Σ .

Then, we have the following theorem.

Theorem 4. Let $P = (\Sigma, A, R)$ be a PTSS in PGSOS format, s.t. $P \sqsupseteq P_{\text{CCS}^\circ}$ and $\Sigma_{sd} = \Sigma - \Sigma_{\text{CCS}^\circ}^\circ$ is a collection of distinctive operators. Let E_P be the axiom system consisting of E_{CCS° and $E_f \cup E_f$, for each $f \in \Sigma_{sd}$. Then, for every disjoint extension $P' \sqsupseteq P$ in PGSOS format, the axiom system E_P is sound for strong bisimilarity on P' .

Notice that the set of rules R defining a smooth operator f in a PTSS P can always be partitioned into sets R_1, \dots, R_m , such that f is distinctive when considering only the rules in R_i . Let f_i be fresh operators with arity $\text{ar}(f_i) = \text{ar}(f)$ and let R'_i be the same set of rule as R_i only that the operator in the source of each rules is renamed to f_i . Consider the disjoint extension $P' \sqsupseteq P$ with all fresh operators f_i and rules in $R'_1 \cup \dots \cup R'_m$ added to the signature and set of rules of P , respectively. Then, it should be clear that the distinctive law

$$f(\zeta_1, \dots, \zeta_{\text{rk}(f)}) = f_1(\zeta_1, \dots, \zeta_{\text{rk}(f)}) + \dots + f_m(\zeta_1, \dots, \zeta_{\text{rk}(f)}) \quad (4)$$

is sound for bisimilarity. Thus, a smooth operator f is axiomatized by the non-deterministic choice over the distinctive variants of f .

Table 3. Axiomatization of \mathbf{sc} (redundant laws, such as $\overline{\mathbf{sc}}(0, a.\mu) = 0$, are omitted)

$$\begin{array}{ll}
\overline{\mathbf{sc}}(x_1 + x_2, y) = \overline{\mathbf{sc}}(x_1, y) + \overline{\mathbf{sc}}(x_2, y) & \overline{\mathbf{sc}}(a.\mu, y) = a.\mathbf{sc}(\mu) \quad \text{if } p_a = 0 \\
\overline{\mathbf{sc}}(a.\mu, \delta_H^1(x)) = a.(\delta(\delta_H^1(x)) \oplus_{p_a} \mathbf{sc}(\mu)) & \overline{\mathbf{sc}}(0, y) = 0 \\
\quad \text{if } p_a > 0, \text{ with } H = \{b \mid p_b < p_a\} & \overline{\mathbf{sc}}(a.\mu, b.v + y) = 0 \quad \text{if } p_b < p_a \\
\overline{\mathbf{sc}}(\mu_1 \oplus_p \mu_2, \nu) = \overline{\mathbf{sc}}(\mu_1, \nu) \oplus_p \overline{\mathbf{sc}}(\mu_2, \nu) & \overline{\mathbf{sc}}(\delta(x), \delta(y)) = \delta(\overline{\mathbf{sc}}(x, y)) \\
\overline{\mathbf{sc}}(\mu, \nu_1 \oplus_p \nu_2) = \overline{\mathbf{sc}}(\mu, \nu_1) \oplus_p \overline{\mathbf{sc}}(\mu, \nu_2) &
\end{array}$$

Theorem 5. *Let $P = ((\Sigma, \Gamma), A, R)$ be a PTSS in PGSOS format, s.t. $P \sqsupseteq P_{CCS^\partial}$ and $f \in \Sigma$ be a smooth operator. There is a disjoint extension $P' = ((\Sigma', \Gamma'), A, R')$ of P with m distinctive smooth operations f_1, \dots, f_m such that $\mathbf{ar}(f_i) = \mathbf{ar}(f)$ for $1 \leq i \leq m$ and (4) is sound for strong bisimulation in any disjoint extension of P' .*

Non-smooth Operators. An operator that is not smooth has a rule in which a variable that is tested in a positive literal either is tested in a second literal or it appears in the target of a conclusion. In this case we proceed by constructing a smooth version of the operator with one argument for each kind of use of the variable that breaks smoothness (actually, one argument for each positive test plus an additional one if the variable is tested negatively or it appears on the target of the conclusion of a rule). Thus, for the unary operator \mathbf{sc} , we introduce a binary operator $\overline{\mathbf{sc}}$, the first argument related to the positive literal and the other related to the negative test and the occurrence in the target of the rule. So $\overline{\mathbf{sc}}$ is defined by the rules

$$\frac{x \xrightarrow{a} \mu \quad \{y \xrightarrow{b} \cdot \mid p_b < p_a\}}{\overline{\mathbf{sc}}(x, y) \xrightarrow{a} \delta(y) \oplus_{p_a} \mathbf{sc}(\mu)}, \text{ if } p_a > 0 \quad \frac{x \xrightarrow{a} \mu}{\overline{\mathbf{sc}}(x, y) \xrightarrow{a} \mathbf{sc}(\mu)}, \text{ if } p_a = 0$$

It should be clear that $\mathbf{sc}(x) = \overline{\mathbf{sc}}(x, x)$. Moreover, notice that $\overline{\mathbf{sc}}$ is smooth. (In fact, it is also distinctive.) The premise on the second rule could have alternatively tested on y rather than x , in which case, $\overline{\mathbf{sc}}$ would have also been smooth but not distinctive.

In general, given a non-smooth operator f , we define a new smooth operator f' by extending its arity as explained above, and proceeding as following: for each rule r of f we introduce a new rule r' for f' such that, if we intend to equate $f(\vec{\zeta}) = f'(\vec{\zeta}')$, and $f(\vec{\zeta})$ and $f'(\vec{\zeta}')$ are the sources of r and r' , respectively, $r[\vec{\zeta}/\vec{\zeta}]$ and $r'[\vec{\zeta}'/\vec{\zeta}']$ have to be identical with the exception of their sources. (Here, $[\vec{\zeta}/\vec{\zeta}]$ denotes the usual substitution of variables.) Notice that this results in a one to one correspondence between the rules of f and those of f' . Then, we have the following theorem.

Theorem 6. *Let P be a PTSS in PGSOS format, s.t. $P \sqsupseteq P_{CCS^\partial}$. Let $f \in \Sigma_P$ be a non-smooth operator. Then there is a disjoint extension $P' \sqsupseteq P$ with a smooth operator f' s.t. the equation $f(\zeta_1, \dots, \zeta_{\mathbf{rk}(f)}) = f'(\zeta'_1, \dots, \zeta'_{\mathbf{rk}(f)})$, where ζ_i , $1 \leq i \leq \mathbf{rk}(f)$ are all different variables and $\{\zeta'_1, \dots, \zeta'_{\mathbf{rk}(f)}\} \subseteq \{\zeta_1, \dots, \zeta_{\mathbf{rk}(f)}\}$, is sound for strong bisimulation in every disjoint extension of P' .*

As an example, we complete the axiomatization of \mathbf{sc} with the axioms for $\overline{\mathbf{sc}}$ which can be derived using Definitions 4 and 5. They are given in Table 3.

As a result of the previous theorems, we obtain the algorithm of Fig. 1 that, given a PTSS P_i in PGSOS format, generates an equational theory E_o that captures the behavior of all operations in P_i and is sound for strong bisimilarity.

Input: a PTSS P_i in PGSOS format

Output: a PTSS P_o in PGSOS format, with $P_o \sqsupseteq P_i$, and an equational theory E_o that is sound for strong bisimilarity in all disjoint extensions of P_o .

1. If necessary, complete P_i so that it disjointly extends CCS^δ .
2. For each non-smooth operator of P_i , extend the system with a smooth version according to Thm. 6 and add all the corresponding equations to CCS^δ .
3. For each smooth non-distinctive operator $f \notin \Sigma_{CCS^\delta}$ in the resulting PTSS, apply the construction of Thm. 5 and extend the PTSS with the distinctive operators f_1, \dots, f_m and the respective rules. Add also the resulting instances of axiom (4).
4. Add all equations associated to the distinctive operators in the resulting system (but not in Σ_{CCS^δ}) according to Def. 6.
5. Finally, for every operator not in Σ_{CCS^δ} add the equation for their respective lifted version according to Def. 4.

Fig. 1. Algorithm to generate an axiomatization for P_i

The fact that the set of rules of P_i (and hence also P_o) is finite guarantees that the equational theory E_o is *head-normalizing* for all operations of P_o , that is, every closed term of P_o can be proven equal to a term of the form 0 , $\sum_{i \in I} a_i \cdot \theta_i$ or $\bigoplus_{j \in J} p_j \delta(t_j)$, with $\theta_i \in T(\Gamma_{P_o})$ and $t_j \in T(\Sigma_{P_o})$, within the equational theory E_o . The construction of head-normal forms is the key towards proving ground-completeness. In fact, notice that if the semantics of a term $t \in T(\Sigma_{P_o})$ is a finite tree, then all operators can be eliminated in E_o (i.e., there is a term $t' \in T(\Sigma_{CCS})$, s.t., $E_o \vdash t = t'$). However this is not the case in general. Consider the constant operator nwf whose semantics is defined by the rule $\text{nwf} \xrightarrow{a} \delta(\text{nwf}) \oplus_{\frac{1}{2}} \delta(0)$. Using the action law, axiom $\text{nwf} = a \cdot (\delta(\text{nwf}) \oplus_{\frac{1}{2}} \delta(0))$ is derived, in which the elimination process will never terminate.

In order to guarantee ground-completeness, we adapt the notion of semantic well-foundedness of [1] to our setting. A term $t \in T(\Sigma_P)$ is *semantically well founded* in P if there is no infinite sequence $t_0 a_0 \theta_0 t_1 a_1 \theta_1 \dots$ of terms $t_i \in T(\Sigma_P)$ and $\theta_i \in T(\Gamma_P)$ and actions $a_i \in A$, such that $t_i \xrightarrow{a_i} \theta_i$ is derivable in P and $\llbracket \theta_i \rrbracket(t_{i+1}) > 0$, for all $i \geq 0$. P is *semantically well founded* if all its terms are. Now, if P_o is semantically well founded (which is the case if P_i is semantically well-founded and $P_i \sqsupseteq P_{CCS}$), E_o has an elimination theorem. As a consequence, we have the following theorem.

Theorem 7. *Let P_i be the input and P_o and E_o be the outputs of the algorithm in Fig. 1. If P_i is semantically well-founded with $P_i \sqsupseteq P_{CCS}$, then the equational theory E_o is ground-complete for strong bisimulation in P_o .*

Ground completeness can be extended to semantically non well-founded PTSS in PGSOS format by also using the approximation induction principle (AIP) [4]. We omit the details here. The proof follows closely the lines of [1].

Axiomatization of Convex Bisimulation. Equation CC of Table 1 was introduced in [5] which proved it sound for convex bisimilarity. The equational theory resulting from extending E_{CCS} with CC is ground-complete for CCS modulo convex bisimilarity. The proof of this result proceeds very much like the one in [5].

Since all the axioms generated by the algorithm in Fig. 1 are sound for strong bisimilarity, they are also sound for convex bisimilarity. Since they also provide elimination for semantically well founded terms, we have the following result:

Theorem 8. *Let P_i be a semantically well-founded PTSS in PGSOS format with $P_i \sqsupseteq P_{CCS}$. Let the PTSS P_o and the equational theory E_o be the outputs of the algorithm in Fig. 1. Then, (i) $E_o \cup \{\text{CC}\}$ is sound for convex bisimulation in any disjoint extension of P_o , and (ii) it is ground-complete in P_o .*

5 Axiomatization of Bisimilarity Metric

In the previous section we developed an equational theory for bisimulation equivalence. Now we shift our focus to bisimilarity pseudometrics and develop an equational theory that characterizes the bisimulation distance.

Axiomatization of Finite Probabilistic Trees. A 1-bounded pseudometric on $T(\Sigma)$ is a function $d: T(\Sigma) \times T(\Sigma) \rightarrow [0, 1]$ such that (i) $d(t, t) = 0$; (ii) $d(t, t') = d(t', t)$; and (iii) $d(t, t'') \leq d(t, t') + d(t', t'')$ for all $t, t', t'' \in T(\Sigma)$. Pseudometrics are used to formalize the notion of *behavioral distance* between terms.

A *matching* $\omega \in \Delta(T(\Sigma) \times T(\Sigma))$ for $(\pi, \pi') \in \Delta(T(\Sigma)) \times \Delta(T(\Sigma))$ is a distribution satisfying $\sum_{t' \in T(\Sigma)} \omega(t, t') = \pi(t)$ and $\sum_{t \in T(\Sigma)} \omega(t, t') = \pi'(t')$ for all $t, t' \in T(\Sigma)$. We denote by $\mathcal{Q}(\pi, \pi')$ the set of all matchings for (π, π') . The *Kantorovich pseudometric* $\mathbf{K}(d): \Delta(T(\Sigma)) \times \Delta(T(\Sigma)) \rightarrow [0, 1]$ lifts a pseudometric $d: T(\Sigma) \times T(\Sigma) \rightarrow [0, 1]$ on state terms to distributions:

$$\mathbf{K}(d)(\pi, \pi') = \min_{\omega \in \mathcal{Q}(\pi, \pi')} \sum_{t, t' \in T(\Sigma)} d(t, t') \cdot \omega(t, t') \quad (5)$$

for $\pi, \pi' \in \Delta(T(\Sigma))$. Note that $\mathbf{K}(d)(\delta_t, \delta_{t'}) = d(t, t')$ for all $t, t' \in T(\Sigma)$. The *Hausdorff pseudometric* $\mathbf{H}(\hat{d}): P(\Delta(T(\Sigma))) \times P(\Delta(T(\Sigma))) \rightarrow [0, 1]$ lifts a pseudometric $\hat{d}: \Delta(T(\Sigma)) \times \Delta(T(\Sigma)) \rightarrow [0, 1]$ on distributions to *sets* of distributions:

$$\mathbf{H}(\hat{d})(\Pi_1, \Pi_2) = \max \left\{ \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \hat{d}(\pi_1, \pi_2), \sup_{\pi_2 \in \Pi_2} \inf_{\pi_1 \in \Pi_1} \hat{d}(\pi_2, \pi_1) \right\} \quad (6)$$

for $\Pi_1, \Pi_2 \subseteq \Delta(T(\Sigma))$ whereby $\inf \emptyset = 1$ and $\sup \emptyset = 0$.

Definition 8 ([10]). *Let $(T(\Sigma), A, \rightarrow)$ be a PTS. A 1-bounded pseudometric d on $T(\Sigma)$ is a bisimulation metric if for all $t, t' \in T(\Sigma)$ with $d(t, t') < 1$, whenever there is a transition $t \xrightarrow{a} \pi$ then there exists a transition $t' \xrightarrow{a} \pi'$ such that $\mathbf{K}(d)(\pi, \pi') \leq d(t, t')$*

We order bisimulation metrics $d_1 \sqsubseteq d_2$ iff $d_1(t, t') \leq d_2(t, t')$ for all $t, t' \in T(\Sigma)$. The smallest bisimulation metric, notation \mathfrak{d} , is called *bisimilarity metric* and assigns to each pair of processes their least possible distance. Strong bisimilarity is the kernel of the bisimilarity metric [10], i.e. $\mathfrak{d}(t, t') = 0$ iff $t \sim t'$.

Let E_{CCS}^m be the system of equations in Table 4. The equations consider two kind of symbols for metrics: one on state terms (**d**) and the other on distribution terms (**d**). Axioms D1–D4 correspond to conditions (i) and (ii) of the definition of a pseudometric. Axioms MN and MP lift the axioms for bisimulation to metrics. In a way, they state that two bisimilar terms should have the same distance to a third term. From Def. 8, it can be

Table 4. Axiomatization of bisimilarity metric of CCS. (We assume $\min \emptyset = 1$.)

$$\mathbf{d}(x, x) = 0 \quad (\text{D1}) \quad \mathbf{d}(\mu, \mu) = 0 \quad (\text{D3})$$

$$\mathbf{d}(x, y) = \mathbf{d}(y, x) \quad (\text{D2}) \quad \mathbf{d}(\mu, \nu) = \mathbf{d}(\nu, \mu) \quad (\text{D4})$$

$$\mathbf{d}(t, x) = \mathbf{d}(t', x) \quad \text{where } t = t' \text{ is one of axioms N1–N4} \quad (\text{MN})$$

$$\mathbf{d}(\theta, \mu) = \mathbf{d}(\theta', \mu) \quad \text{where } \theta = \theta' \text{ is one of axioms NP1–NP3 or P1–P3} \quad (\text{MP})$$

$$\mathbf{d}(0, a.\mu + x) = 1 \quad (\text{H1})$$

$$\mathbf{d}\left(\sum_{i \in I} a_i.\mu_i, \sum_{j \in J} b_j.\nu_j\right) = \max \left\{ \max_{i \in I} \min_{j \in J, a_i = b_j} \mathbf{d}(\mu_i, \nu_j), \max_{j \in J} \min_{i \in I, a_i = b_j} \mathbf{d}(\mu_i, \nu_j) \right\} \quad (\text{H2})$$

$$\mathbf{d}\left(\bigoplus_{i \in I} p_i \delta(x_i), \bigoplus_{j \in J} q_j \delta(y_j)\right) = \min_{\omega \in \Omega(I, J)} \sum_{i \in I, j \in J} \mathbf{d}(x_i, y_j) \cdot \omega(i, j) \quad (\text{K})$$

$$\text{where } \Omega(I, J) = \{\omega : I \times J \rightarrow [0, 1] \mid \forall i \in I : \omega(i, J) = p_i, \forall j \in J : \omega(I, j) = q_j\}$$

easily seen that d is a bisimulation metric whenever $\max_{a \in A} \mathbf{H}(\mathbf{K}(d))(\{\pi \mid t \xrightarrow{a} \pi\}, \{\pi' \mid t' \xrightarrow{a} \pi'\}) \leq d(t, t')$. This is captured by H1 and H2. The equality in the axioms is due to the fact that we aim to characterize only the bisimilarity metric \mathfrak{d} . Finally, axiom K corresponds to the definition of the Kantorovich pseudometric. We also need the following general rules that should be considered together with the usual inference rules of equational logic. For all $f : \sigma_1 \dots \sigma_{\text{rk}(f)} \rightarrow s$ and $g : \sigma_1 \dots \sigma_{\text{rk}(g)} \rightarrow d$, we have

$$\frac{\{\mathbf{d}(\zeta_i, \zeta'_i) = 0, \mathbf{d}(\zeta_j, \zeta'_j) = 0 \mid 1 \leq i, j \leq \text{rk}(f), \sigma_i = s, \sigma_j = d\}}{\mathbf{d}(f(\zeta_1, \dots, \zeta_{\text{rk}(f)}), z) = \mathbf{d}(f(\zeta'_1, \dots, \zeta'_{\text{rk}(f)}), z)} \quad (\text{S1})$$

$$\frac{\{\mathbf{d}(\zeta_i, \zeta'_i) = 0, \mathbf{d}(\zeta_j, \zeta'_j) = 0 \mid 1 \leq i, j \leq \text{rk}(f), \sigma_i = s, \sigma_j = d\}}{\mathbf{d}(g(\zeta_1, \dots, \zeta_{\text{rk}(g)}), z) = \mathbf{d}(g(\zeta'_1, \dots, \zeta'_{\text{rk}(g)}), z)} \quad (\text{S2})$$

These rules ensure that $E_{CCS}^m \vdash \mathbf{d}(t, t'') = \mathbf{d}(t', t'')$ whenever $E_{CCS} \vdash t = t'$ and similarly for distribution terms.

Let \mathfrak{d} be the bisimilarity metric and $\mathbf{K}(\mathfrak{d})$ its Kantorovich lifting. Let ρ be a closed substitution. We define $\llbracket \mathbf{d}(t, t') \rrbracket_\rho = \mathfrak{d}(\rho(t), \rho(t'))$ and $\llbracket \mathbf{d}(\theta, \theta') \rrbracket_\rho = \mathbf{K}(\mathfrak{d})(\llbracket \rho(\theta) \rrbracket, \llbracket \rho(\theta') \rrbracket)$ for $t, t' \in \mathbb{T}(\Sigma_{CCS})$ and $\theta, \theta' \in \mathbb{T}(\Gamma_{CCS})$. We lift $\llbracket _ \rrbracket_\rho$ to arithmetic terms containing expressions of the form $\mathbf{d}(t, t')$ or $\mathbf{d}(\theta, \theta')$ in the obvious way (e.g. $\llbracket \min_{i \in I} \text{expr}_i \rrbracket_\rho = \min_{i \in I} \llbracket \text{expr}_i \rrbracket_\rho$). E_{CCS}^m is sound for \mathfrak{d} in the sense that, whenever $E_{CCS}^m \vdash \text{expr} = \text{expr}'$ (meaning that $\text{expr} = \text{expr}'$ can be proved using axioms in E_{CCS}^m and arithmetic), then $\llbracket \text{expr} \rrbracket_\rho = \llbracket \text{expr}' \rrbracket_\rho$ for every closed substitution ρ . Soundness should be clear for all the axioms except maybe for H2. By definition of bisimulation metric, the right-hand side is smaller than or equal to the left-hand side interpreting them on any closed substitution. Equality follows from the fact that \mathfrak{d} is the *smallest* bisimulation metric.

Besides, E_{CCS}^m is also ground-complete for \mathfrak{d} , in the sense that, for any (closed) arithmetic expressions expr and expr' possibly containing closed terms of the form $\mathbf{d}(t, t')$ or $\mathbf{d}(\theta, \theta')$ with $t, t' \in \mathbb{T}(\Sigma_{CCS})$ and $\theta, \theta' \in \mathbb{T}(\Gamma_{CCS})$, $\llbracket \text{expr} \rrbracket = \llbracket \text{expr}' \rrbracket$ implies $E_{CCS}^m \vdash \text{expr} = \text{expr}'$. Notice that by arithmetic, this is a direct consequence of the following claims: (i) for all closed state terms $t, t' \in \mathbb{T}(\Sigma_{CCS})$ and $p \in [0, 1]$, if $\mathfrak{d}(t, t') = p$ then $E_{CCS}^m \vdash \mathbf{d}(t, t') = p$, and (ii) for all closed distribution terms $\theta, \theta' \in \mathbb{T}(\Gamma_{CCS})$, if $\mathbf{K}(\mathfrak{d})(\llbracket \theta \rrbracket, \llbracket \theta' \rrbracket) = p$, $E_{CCS}^m \vdash \mathbf{d}(\theta, \theta') = p$. The proof of these claims follows by reducing closed terms involved in $\mathbf{d}(t, t')$ and $\mathbf{d}(\theta, \theta')$ to normal form using

axioms D1–D4, MN, and MP (and rules S1 and S2), and then inductively applying H1, H2, K and arithmetic calculations to reach the expected value.

Theorem 9. E_{CCS}^m is sound and ground-complete for the bisimilarity metric \mathfrak{d} .

Axiomatization of Bisimilarity Metric of PGSOS. The algorithm of Fig. 1 can be modified to provide axioms for bisimilarity metric to any operator defined in PGSOS as follows. Instead of adding the axioms in E_{CCS} , add the axioms in E_{CCS}^m , and for each equation $t_1 = t_2$ (resp. $\theta = \theta'$) added by the algorithm in Fig. 1, add instead $\mathfrak{d}(t_1, x) = \mathfrak{d}(t_2, x)$ (resp. $\mathfrak{d}(\theta, \mu) = \mathfrak{d}(\theta', \mu)$).

Soundness of the axioms introduced by the algorithm is straightforward: we know that $t_1 \sim t_2$ implies $d(t_1, t_2) = 0$ and hence $d(t_1, t) = d(t_2, t)$ can be calculated from properties (ii) and (iii) in the definition of pseudometric (similarly for distribution terms).

We already observed that E_{CCS}^m is normalizing. Besides, it can be shown that the axiom system generated by the new algorithm is head-normalizing. Then, for every semantically well founded closed term t there is a t' in normal form such that $\mathfrak{d}(t, t'') = \mathfrak{d}(t', t'')$ for every t'' . Using this elimination result ground-completeness follows.

Theorem 10. Let P_i be a PTSS in PGSOS format and let the PTSS P_o and the equational theory E_o be the outputs of the algorithm in Fig. 1 modified as before. Then, (i) E_o is sound for the bisimilarity metric \mathfrak{d} in any disjoint extension of P_o , and (ii) it is ground-complete in P_o , provided P_o is semantically well founded.

6 Concluding Remarks

As we pointed out in [9], the use of literals as a triple $t \xrightarrow{a} \theta$ in PTSS rules (rather than the old fashion quadruple $t \xrightarrow{a,p} t'$ that partially specifies a probabilistic jump) paves the way for generalizing the theory transition system specification to the probabilistic setting. We went further in this paper and defined a two-sorted signature that leads to a rigorous and clear definition of the distribution term in the target of positive literals. Moreover, this also fits nicely with the introduction of the equational theory.

This setting allows us to borrow the strategies of [1] to obtain the algorithm of Fig. 1 and prove its correctness (Thm. 7). This is particularly facilitated by the introduction of the operators mapping into sort d , and particularly by the fact that all probabilistically lifted operators distribute with respect to \oplus_p and δ . The generalization of the algorithm to behavioral equivalences weaker than strong bisimilarity and whose equational theories contain E_{CCS} , is simple as demonstrated with convex bisimilarity (Thm. 8).

The result that convex bisimilarity is a congruence for all operators defined with PGSOS rules (Thm. 1) is new in this paper and, to our knowledge, it is actually the first time that a general congruence theorem is proved for *convex* bisimilarity. Here, we insist on the advantages of a good definition: this result is a direct consequence of the fact that strong bisimilarity is a congruence and this is so because the definition of combined transition can be encoded with a set of PGSOS rules (then a strong bisimulation in the extended PTSS is also a convex bisimulation).

We remark that the axiomatization E_{CCS}^m of bisimilarity metric is new in this paper. Axiom scheme H2 can be translated into a set of axioms that only include binary sum

by introducing an auxiliary operator. However we have been unable so far to find a set of axioms that only use binary \oplus_p operators in order to replace the axiom scheme K.

References

1. Aceto, L., Bloom, B., Vaandrager, F.: Turning SOS rules into equations. *Inf. Comput.* 111(1), 1–52 (1994)
2. Aceto, L., Goriac, E.-I., Ingolfssdottir, A., Mousavi, M.R., Reniers, M.A.: Exploiting algebraic laws to improve mechanized axiomatizations. In: Heckel, R. (ed.) *CALCO 2013*. LNCS, vol. 8089, pp. 36–50. Springer, Heidelberg (2013)
3. Baeten, J.C.M., Basten, T., Reniers, M.A.: *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, New York (2009)
4. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: On the consistency of Koomen’s fair abstraction rule. *TCS* 51, 129–176 (1987)
5. Bandini, E., Segala, R.: Axiomatizations for probabilistic bisimulation. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) *ICALP 2001*. LNCS, vol. 2076, pp. 370–381. Springer, Heidelberg (2001)
6. Bartels, F.: *On Generalised Coinduction and Probabilistic Specification Formats*. Ph.D. thesis, VU University Amsterdam (2004)
7. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can’t be traced. *J. ACM* 42, 232–268 (1995)
8. Contejean, E., Marché, C., Rabehasaina, L.: Rewrite systems for natural, integral, and rational arithmetic. In: Comon, H. (ed.) *RTA 1997*. LNCS, vol. 1232, pp. 98–112. Springer, Heidelberg (1997)
9. D’Argenio, P.R., Lee, M.D.: Probabilistic transition system specification: Congruence and full abstraction of bisimulation. In: Birkedal, L. (ed.) *FOSSACS 2012*. LNCS, vol. 7213, pp. 452–466. Springer, Heidelberg (2012)
10. Desharnais, J., Jagadeesan, R., Gupta, V., Panangaden, P.: The metric analogue of weak bisimulation for probabilistic processes. In: *Proc. LICS 2002*, pp. 413–422. IEEE (2002)
11. Gazda, M., Fokkink, W.: Turning GSOS rules into equations for linear time-branching time semantics. *The Computer Journal* 56(1), 34–44 (2013)
12. Gebler, D., Tini, S.: Compositionality of approximate bisimulation for probabilistic systems. In: *Proc. EXPRESS/SOS 2013*. EPTCS, vol. 120, pp. 32–46 (2013)
13. Goguen, J.A., Meseguer, J.: Completeness of many-sorted equational logic. *SIGPLAN Not.* 17(1), 9–17 (1982)
14. Groote, J.F., Vaandrager, F.: Structured operational semantics and bisimulation as a congruence. *Inf. Comput.* 100, 202–260 (1992)
15. Hennessy, M.: Exploring probabilistic bisimulations, part I. *Formal Aspects of Computing* 24(4-6), 749–768 (2012)
16. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* 94, 1–28 (1991)
17. Lee, M.D., Gebler, D., D’Argenio, P.R.: Tree rules in probabilistic transition system specifications with negative and quantitative premises. In: *Proc. EXPRESS/SOS 2012*. EPTCS, vol. 89, pp. 115–130 (2012)
18. Milner, R.: *Communication and Concurrency*. Prentice-Hall (1989)
19. 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)
20. 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)
21. Segala, R.: *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, MIT (1995)

Generalized Synchronization Trees

James Ferlez^{1,2}, Rance Cleaveland^{1,3}, and Steve Marcus^{1,2}

¹ The Institute for Systems Research, University of Maryland

² Department of Electrical and Computer Engineering, University of Maryland

³ Department of Computer Science, University of Maryland

Abstract. This paper develops a generalized theory of synchronization trees. In their original formulation, synchronization trees modeled the behavior of nondeterministic discrete-time reactive systems and served as the foundational theory upon which process algebra was based. In this work, a more general notion of tree is proposed that is intended to support the modeling of systems with a variety of notions of time, including continuous and hybrid versions. (Bi)simulation is also studied, and it is shown that two notions that coincide in the discrete setting yield different relations in the generalized framework. A CSP-like parallel composition operator for generalized trees is defined as a means of demonstrating the support for compositionality the new framework affords.

1 Introduction

Research into process algebra has been highly influential in the mathematical study of system modeling [4]. Such algebras include a collection of operators for assembling more complex systems from smaller ones, as well as notions of behavioral equivalence and refinement for determining when two systems are indistinguishable behaviorally and when one system is an elaboration of another. This principled approach to compositional modeling has inspired the development of a wealth of mechanisms for combining systems in practically interesting yet mathematically well-founded ways for event-driven systems.

Synchronization trees, as proposed originally by Milner [15], played a pivotal role in the development of process algebra. In any algebraic theory, carrier sets must be specified; operators in the algebra are then interpreted as constructions over the elements from these sets. Synchronization trees play this role in traditional process algebras. Intuitively, a synchronization tree encodes the behavior of a system: nodes in the tree correspond to states, with edges, which are labeled by events, representing execution steps. Composition operators may then be interpreted as constructions on these trees, with the result of the construction representing the behavior of the composite system. These constructions in turn may be specified co-inductively via inference rules [5]. The simplicity and flexibility of synchronization trees led several researchers to formalize Milner's original notion using different mathematical machinery [1,3,17] and indeed helped inspire seminal work on co-induction in computing [12,18].

Synchronization trees are intended to model discrete systems that evolve by engaging in atomic events and changing state. For systems with non-discrete

behavior, a similarly general yet simple model for defining composition has arguably yet to emerge. Some researchers use transition systems to model such behavior [6,9], while others adopt category-theoretic [10,11] and trajectory-based models [14,19]. However, neither case has yielded the rich results for composition operators that can be found in discrete process algebra. By contrast, others have recently recognized that generalizing the notion of a tree is a profitable approach [7,8]. However, in [8], generalized trees appear only as a consequence of other system models, whereas in [7], the notions of bisimulation and CSP-parallel composition are not generalized to non-discrete behavior.

The purpose of this paper is to propose a new tree-based model of system behavior, which we call *generalized synchronization trees*, that is intended to play the same role in a generalized-time setting that synchronization trees play for discrete time. Our goal is to provide the foundation for a general, flexible theory of composition for systems that include components with a variety of different notions of time, including continuous, discrete, and their hybrids. Generalized synchronization trees also represent system behavior and subsume traditional synchronization trees, but include a flexible mechanism for modeling non-discrete models of time as well. In this paper, we define these trees and study notions of equivalence (bisimulation) and refinement (simulation) in a logical (i.e. non-real-time) setting. Our results show in particular that definitions of these behavioral relations that coincide in the discrete setting differ in the generalized setting. We also show how our trees subsume some existing models of hybrid behavior, and we initiate the study of composition operations in this theory by showing how CSP parallel composition can be extended to this framework.

The remainder of the paper is structured as follows. The next section presents mathematical preliminaries, while the section following gives the definition of generalized synchronization trees and some examples. Section 4 then studies different notions of bisimulation for this model, and the section after shows how our trees may be used to model systems in an existing hybrid process algebra. The next section considers parallel composition in the setting of our new trees, and the final section concludes with directions for future work.

2 Preliminaries

This section presents basic background on partial and total orders and reviews a classical definition of tree in this setting.

Definition 1 (Partial Order). *Let P be a set, and let $\preceq \subseteq P \times P$ be a binary relation on P . Then \preceq is a partial order on P if the following hold.*

1. \preceq is reflexive: for all $p \in P$, $p \preceq p$.
2. \preceq is anti-symmetric: for all $p_1, p_2 \in P$, if $p_1 \preceq p_2$ and $p_2 \preceq p_1$ then $p_1 = p_2$.
3. \preceq is transitive: for all $p_1, p_2, p_3 \in P$, if $p_1 \preceq p_2$ and $p_2 \preceq p_3$ then $p_1 \preceq p_3$.

We abuse terminology and refer to $\langle P, \preceq \rangle$ as a partial order if \preceq is a partial order over set P . We write $p_1 \prec p_2$ if $p_1 \preceq p_2$ and $p_1 \neq p_2$ and $p_2 \succeq p_1$ if $p_1 \preceq p_2$. We adapt the usual interval notation for numbers to partial orders as follows.

$$[p_1, p_2] \triangleq \{p \in P \mid p_1 \preceq p \preceq p_2\}$$

$$(p_1, p_2) \triangleq \{p \in P \mid p_1 \prec p \prec p_2\}$$

Half-open intervals, e.g. $[p_1, p_2)$ and $(p_1, p_2]$, have the obvious definitions.

Definition 2 (Upper/Lower Bounds). Fix partial order $\langle P, \preceq \rangle$ and $P' \subseteq P$.

1. $p \in P$ is an upper (lower) bound of P' if for every $p' \in P'$, $p' \preceq p$ ($p \preceq p'$).
2. $p \in P$ is the least upper (greatest lower) bound of P' if p is an upper (lower) bound of P' and for every upper (lower) bound p' of P' , $p \preceq p'$ ($p' \preceq p$).

When set P' has a least upper bound (greatest lower bound) we sometimes use $\sup P'$ ($\inf P'$) to denote this element.

Definition 3 (Total Order). Let $\langle P, \preceq \rangle$ be a partial order. Then \preceq is a total or linear order on P if for every $p_1, p_2 \in P$, either $p_1 \preceq p_2$ or $p_2 \preceq p_1$.

If $\langle P, \preceq \rangle$ is a partial order and $P' \subseteq P$, then we sometimes abuse notation and write $\langle P', \preceq \rangle$ for the partial order obtained by restricting \preceq to elements in P' . We say that P' is *totally ordered* by \preceq if \preceq is a total order for P' . We refer to P' as a *linear subset* of P in this case. Trees may now be defined as follows [13].

Definition 4 (Tree [13]). A tree is partial order $\langle P, \preceq \rangle$ such that for each $p \in P$, the set $\{p' \in P \mid p' \preceq p\}$ is totally ordered by \preceq . If there is also a $p_0 \in P$ such that $p_0 \preceq p$ for all $p \in P$, then p_0 is called the root of the tree, and $\langle P, \preceq, p_0 \rangle$ is said to be a rooted tree.

In [7], these structures are referred to as prefix orders. The distinguishing feature of a tree implies that \preceq defines a notion of ancestry. In a rooted tree, the root is an ancestor of every node, so every node has a unique “path” to the root. Since the subsequent development is modeled on synchronization trees, we will consider only rooted trees in the sequel.

We conclude this section by discussing a notion of discreteness for trees.

Definition 5 (Discrete Tree). A tree $\langle P, \preceq, p_0 \rangle$ is discrete if and only if for every p , the set $[p_0, p]$ is finite.

The following alternative characterization of discreteness is sometimes useful.

Proposition 1. A tree $\langle P, \preceq, p_0 \rangle$ is discrete if and only the following all hold.

1. For every $p \neq p_0$, $\sup[p_0, p) \in [p_0, p)$.
2. For every $p \in P$ and $p' \succ p$, $\inf(p, p'] \in (p, p']$.
3. Every nonempty linear subset P' of P has a greatest lower bound.

3 Generalized Synchronization Trees

This section defines, and gives examples of, generalized synchronization trees.

3.1 Traditional Synchronization Trees

Milner introduced the notion of synchronization tree in [15]. The following quotes the definition given on p. 16 of that reference.

“A Synchronization Tree (ST) of sort L is a rooted, unordered, finitely branching tree each of whose arcs is labelled by a member of $L \cup \{\tau\}$.”¹

(Milner also indicates that such trees may be of infinite depth. Note that because of its reference to “arcs”, Milner’s definition implies that synchronization trees must be discrete in the sense of Definition 5.) Intuitively, the set L of labels contains externally visible actions that systems may engage in; τ denotes a designated internal action. A tree then represents the full behavior of a system; the root represents the start state, while edges represent discrete computation steps and branching represents nondeterminism. Milner shows how the basic composition operators, including choice and parallel composition, of his Calculus of Communicating Systems (CCS) may be interpreted as constructions on these trees. Throughout his presentation Milner consciously follows a traditionally algebraic approach, making CCS one of the earliest process algebras.

Process algebras are often given a semantics in terms of *labeled transition systems*; a ST may be seen as the “unrolling” of such a system.

Milner also defined a notion of *strong equivalence* (now called *bisimulation equivalence*) on systems in order to equate synchronization trees that, while not isomorphic, nevertheless represent the same behavior. The definitions below are adapted from [16]; recall that if R is a binary relation on a set $S \times T$ then R^{-1} , the inverse of R , is binary relation on $T \times S$ defined by $R^{-1} \triangleq \{\langle t, s \mid \langle s, t \rangle \in R\}$.

Definition 6 (Simulation and Bisimulation for Synchronization Trees).

Let L be a set of labels, and let ST_L be the set of STs whose labels come from L .

1. Let $T, T' \in ST_L$ and $a \in L$. Then $T \xrightarrow{a} T'$ if there is an edge labeled by a from the root of T to the root of T' .
2. Relation $R \subseteq ST_L \times ST_L$ is a simulation if, whenever $\langle T_1, T_2 \rangle \in R$ and $T_1 \xrightarrow{a} T'_1$, then there exists T'_2 such that $T_2 \xrightarrow{a} T'_2$ and $\langle T'_1, T'_2 \rangle \in R$.
3. Relation $R \subseteq ST_L \times ST_L$ is a bisimulation if both R and R^{-1} are simulations.
4. Let $T_1, T_2 \in ST_L$. Then T_1 is simulated by T_2 (notation $T_1 \sqsubseteq T_2$) if there is a simulation relation R with $\langle T_1, T_2 \rangle \in R$.
5. Let $T_1, T_2 \in ST_L$. Then T_1 and T_2 are bisimulation equivalent, or bisimilar (notation $T_1 \sim T_2$), if there is a bisimulation R with $\langle T_1, T_2 \rangle \in R$.

¹ Milner also introduces the notion of *rigid synchronization tree*, which limit edge labels to the set L . We elide this distinction, as we do not consider τ in this paper.

The relation \sqsubseteq is often called *the simulation preorder*. It can be shown that the simulation preorder (bisimulation equivalence) itself is a simulation (bisimulation) relation, and indeed is the unique largest such relation.

Milner's definition of synchronization tree may be seen as semi-formal in that trees are not formally defined. Other authors [1,3,17,20] subsequently developed the underlying mathematics fully, and in the process helped justify, as coinductive constructions, the composition operations given by Milner on infinite trees.

3.2 Generalized Synchronization Trees

The impact of Milner's work is hard to overstate; process algebra is a major field of study in computing, and the notions of simulation and bisimulation have had a substantial influence on other areas such as control-system modeling and systems biology, where the focus is on continuous, rather than discrete, behavior. However, the rich array of composition operators, and associated elegant metatheoretical results [2,5] found in traditional process algebra have yet to emerge in these more general contexts. Our motivation for generalized synchronization trees is to provide a flexible framework analogous to synchronization trees over which composition operations may be easily defined, and their algebraic properties studied, for this more general setting.

Synchronization trees are intended to model discrete systems that evolve via the execution of atomic actions. This phenomenon is evident in the fact that trees have edges that are labeled by these actions; each node in a tree is thus at most finitely many transitions from the root. For systems that have continuous as well as discrete dynamics, synchronization trees offer a problematic foundation for system modeling, since the notion of continuous trajectory is missing.

Generalized synchronization trees are intended to provide support for discrete, continuous, and hybrid notions of computation, where nondeterminism (branching) may also be discrete, continuous, or both.

Definition 7 (Generalized Synchronization Tree). *Let L be a set of labels. Then a generalized synchronization tree (GST) is a tuple $\langle P, \preceq, p_0, \mathcal{L} \rangle$, where:*

1. $\langle P, \preceq, p_0 \rangle$ is a tree in the sense of Definition 4; and
2. $\mathcal{L} \in P \setminus \{p_0\} \rightarrow L$ is a (possibly partial) labeling function.

A GST differs from a synchronization tree in two respects. On the one hand, the tree structure is made precise by reference to Definition 4. On the other hand, labels are attached to (non-root) nodes, rather than edges; indeed, a GST may not in general have a readily identifiable notion of edge.

In the rest of this section we show how different classes of systems may be encoded as GSTs. These examples contain different mixtures of discrete / continuous time and discrete / continuous nondeterminism (called "choice").

Example 1 (Labeled Transition Systems as GSTs). Let $T = \langle X, L, \rightarrow, x_0 \rangle$ be a labeled transition system with state set X , label set L , transition relation $\rightarrow \subseteq X \times L \times X$, and initial state x_0 . Then the behavior of T starting from

x_0 may be encoded as a discrete GST. Define an *execution* e of T to be a sequence of transitions (formally, an element of \rightarrow^*) such that if $e = \langle x, \ell, x' \rangle \cdot e'$ then $x = x_0$ (i.e. the first transition is always from the start state), and if $e = e_1 \cdot \langle x_1, \ell_1, x'_1 \rangle \cdot \langle x_2, \ell_2, x'_2 \rangle \cdot e_2$ then $x'_1 = x_2$ (i.e. the next transition always starts from the target state of the last transition). Let E_T be the set of all executions of T ; note that ϵ , the empty sequence of transitions, is in E_T , and that \preceq_T , the prefix ordering on \rightarrow^* , is a partial order on E_T such that $\epsilon \preceq_T e$ for all $e \in E_T$. Finally, if $e = e' \cdot \langle x, \ell, x' \rangle$, define $\mathcal{L}_T(e)$ to be ℓ (i.e. the label of the last transition in e). It is easy to see that $G_T = \langle E_T, \preceq_T, \epsilon, \mathcal{L}_T \rangle$ is a GST, and that G_T is discrete in the sense of Definition 5.

The previous construction is the classical “unrolling” method for generating trees from LTSs, and is generally associated with discrete-time modeling. Perhaps surprisingly, however, it is also applicable to formalisms that model continuous behavior via transition systems. For example, Hybrid Process Algebra (HyPA) [6] is a compositional algebraic framework for modeling hybrid systems that permit instantaneous jumps in their continuous model variables; the signature of HyPA reflects this by including reinitialization operators, flow clauses and disrupt operators. The behavior of HyPA terms depends on the values of the continuous model variables; transitions are enabled only for certain valuations of these variables, and transitions can also alter the model variables when they execute. Thus, the operational semantics, which is specified in the traditional SOS style, defines transitions for HyPA-term / variable-valuation pairs. Two types of transitions are in fact defined; zero-duration, discrete-action “jumps”, and finite-duration continuous flows. The results is a labeled transition system where each state (location) is specified by a HyPA term and a valuation of the model variables. These labeled transition systems are coined *hybrid transition systems* in [6]. As labeled transition systems, hybrid transition systems can be represented as GSTs using the construction above; these GSTs are also discrete, interestingly, even though the phenomena being modeled in HyPA are not.

Difference equations with inputs also represent models that semantically give rise to transition systems. Such models often arise in the description of control systems when the quantities of interest – states and inputs, for example – are sampled in time. Such difference equations typically take the form

$$x_{k+1} = f(x_k, u_k),$$

where x_k represents state at the k^{th} sampling interval, u_k represents an input arriving between time k and $k + 1$, and f is a function computing the new state based on the existing state and this input. These systems can be represented as labeled transition systems, with states given by the x and transitions labeled by u defined by f , so the above construction yields discrete GSTs in this case, too.

Example 2 (Differential Equations with Inputs as GSTs). Continuous-time, continuous - choice systems have traditionally been the standard problem considered by control theorists; these systems usually take the form of an ordinary differential equation (ODE) with inputs. A simple example of this class of systems is one derived from Newton’s laws of motion. For example, consider an object

with mass m that is both confined to travel in a straight line and affected by a time-varying external force u (from a motor, say). If we let x_1 represent the position of the object and we let x_2 represent its velocity, then Newton's laws can be used to derive the following state equations for the object:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u \quad (1)$$

where all the variables are functions of time and \dot{x} represents the time-derivative of x . Recently, Willems *et al.* [19] have suggested that continuous-time systems be treated as a collection of time trajectories (or *behaviors*) instead of a set of state equations. If we suppose that time starts from 0, then the previous system is completely described by the set of all pairs of functions

$$\mathcal{B} \triangleq \left\{ \langle u, x \rangle \in (\mathcal{R}^{\geq 0} \rightarrow \mathcal{R}) \times (\mathcal{R}^{\geq 0} \rightarrow \mathcal{R}^2) : u \text{ is locally integrable and} \right. \\ \left. \exists x_0 \in \mathcal{R}^2 \text{ s.t. } x(t) = \exp(At)x_0 + \int_0^t \exp(A(t-\tau))Bu(\tau)d\tau \quad \forall t \geq 0 \right\} \quad (2)$$

where $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}$ and the function \exp should be interpreted as the matrix exponential. The function u is assumed to be locally integrable so that the subsequent integral is well defined for all t .

This behavioral treatment of continuous-time systems facilitates the construction of GSTs using a generalized notion of "prefix". To help with this, we define a notion of truncation for functions defined on $\mathcal{R}^{\geq 0}$: given a function $f \in \mathcal{B}$, we define $f|_t$ to be the restriction of the function f to the set $[0, t]$. Now we can define a GST from \mathcal{B} as follows.

- Let $P = \{\langle t, f \rangle : t \in \mathcal{R}^{\geq 0} \text{ and } f = x|_t \text{ for some } x \in \mathcal{B}\} \cup \{p_0\}$ where p_0 is a distinguished element not in the first set.
- The partial order \preceq is defined in the following way: let $p_0 \preceq p$ for all $p \in P$ and for $p_1 \triangleq \langle t_1, f_1 \rangle$, $p_2 \triangleq \langle t_2, f_2 \rangle$ let $p_1 \preceq p_2$ if and only if $t_1 \leq t_2$ and $f_1 = f_2|_{t_1}$.
- The labeling function $\mathcal{L} : P \setminus \{p_0\} \rightarrow \mathcal{R}$ is defined so that $\mathcal{L}(\langle t, f \rangle) = \pi_1 f(t)$.

We close this section by remarking on correctness issues for the translations just given; in what sense are they "right"? In the absence of notions of equivalence, this question is hard to answer. The next section helps remedy the situation by defining (bi)simulation for GSTs; this permits us to revisit HyPA, for example, in Section 5 in order to give a different, more satisfactory translation of HyPA terms into GSTs.

4 (Bi)Simulations for Generalized Synchronization Trees

Section 2 defined standard notions of equivalence (bisimulation) and refinement (simulation) on synchronization trees. The goal of this section is to study adaptations of these notions for generalized synchronization trees. In the process of

doing so, we highlight a subtlety that arises because of GSTs' capability of modeling non-discrete time. As the notions of simulation and bisimulation are closely linked (see Definition 6), in what follows we focus our attention on simulation.

4.1 Simulations for Generalized Synchronization Trees

Simulation relations in Definition 6 rely on a notion, labeled edges, that synchronization trees possess but GSTs do not. However, an intuition underlying the simulation relation is that if $T_1 \sqsubseteq T_2$, then every "execution step" of T_1 can be matched by T_2 in such a way that the resulting trees are still related. This observation provides a starting point for simulations on GSTs; rather than relying on edges to define computation, use the notion *trajectory* instead.

Definition 8 (Trajectory). *Let $\langle P, \preceq, p_0, \mathcal{L} \rangle$ be a GST, and let $p \in P$. Then a trajectory from p is a linear subset $P' \subseteq P$ such that for all $p' \in P'$:*

1. $p' \succ p$ and
2. $(p, p'] \subseteq P'$.

A trajectory from a node p in a GST is a path that starts from p , but for technical reasons, does not include p . A trajectory can be bounded with a maximal element as in the case of the interval $(p, p']$, or it can be bounded with a least upper bound as in the case of (p, p') . It is also possible for a trajectory to be bounded without a least upper bound or even unbounded.

Trajectories are analogous to computations and thus will form the basis of the simulation relations given below. In order to determine when two trajectories "match", we introduce the concept of *order-equivalence*.

Definition 9 (Order Equivalence). *Let $\langle P, \preceq_P, p_0, \mathcal{L}_P \rangle$ and $\langle Q, \preceq_Q, q_0, \mathcal{L}_Q \rangle$ be GSTs, and T_p, T_q be trajectories from $p \in P$ and $q \in Q$ respectively. Then T_p and T_q are order-equivalent if there exists a bijection $\lambda \in T_p \rightarrow T_q$ such that:*

1. $p_1 \preceq_P p_2$ if and only if $\lambda(p_1) \preceq_Q \lambda(p_2)$ for all $p_1, p_2 \in T_p$, and
2. $\mathcal{L}_P(p) = \mathcal{L}_Q(\lambda(p))$ for all $p \in T_p$.

When λ has this property, we say that λ is an order equivalence from T_p to T_q .

Two trajectories that are order-equivalent can be seen as possessing the same "content", as given by the labeling functions of the trees, in the same "order". Note that in general, the bijections used to relate two order-equivalent trajectories need not be unique, although when the trees in question are discrete, they must be. The first notion of simulation may now be given as follows.

Definition 10 (Weak Simulation for GSTs). *Let $G_1 = \langle P, \preceq_P, p_0, \mathcal{L}_P \rangle$ and $G_2 = \langle Q, \preceq_Q, q_0, \mathcal{L}_Q \rangle$ be GSTs. Then $R \subseteq P \times Q$ is a weak simulation from G_1 to G_2 if, whenever $\langle p, q \rangle \in R$ and $p' \succeq p$, then there is a $q' \succeq q$ such that:*

1. $\langle p', q' \rangle \in R$, and
2. Trajectories $(p, p']$ and $(q, q']$ are order-equivalent.

$G_1 \sqsubseteq_w G_2$ if there is a weak simulation R from G_1 to G_2 with $\langle p_0, q_0 \rangle \in R$.

Weak bisimulation equivalence can be defined easily. Call a weak simulation R from G_1 to G_2 a weak bisimulation if R^{-1} is a weak simulation from G_2 to G_1 . Then $G_1 \sim_w G_2$ iff there is a weak bisimulation R with $\langle p_0, q_0 \rangle \in R$.

Weak simulation appears to be the natural extension of simulation to GSTs: for one node to be simulated by another, each bounded trajectory from the first node must be appropriately “matched” by an equivalent trajectory from the second node. However, one may impose a stronger condition on the trajectories emanating from related nodes, as follows.

Definition 11 (Strong Simulation for GSTs). Let $G_1 = \langle P, \preceq_P, p_0, \mathcal{L}_P \rangle$ and $G_2 = \langle Q, \preceq_Q, q_0, \mathcal{L}_Q \rangle$ be GSTs. Then $R \subseteq P \times Q$ is a strong simulation from G_1 to G_2 if, whenever $\langle p, q \rangle \in R$ and T_p is a trajectory from p , there is a trajectory T_q from q and bijection $\lambda \in T_p \rightarrow T_q$ such that:

1. λ is an order equivalence from T_p to T_q , and
2. $\langle p', \lambda(p') \rangle \in R$ for all $p' \in T_p$.

$G_1 \sqsubseteq_s G_2$ if there is a strong simulation R from G_1 to G_2 with $\langle p_0, q_0 \rangle \in R$.

Strong simulations strengthen weak ones by requiring that matching trajectories also pass through nodes that are related by the simulation relation, and by also considering potentially unbounded trajectories as well as bounded ones.

4.2 Relating Strong and Weak Simulations

This section now considers the relationships between weak and strong simulation. The first result indicates that the latter is indeed stronger than the former.

Theorem 1. Let G_1 and G_2 be GSTs with $G_1 \sqsubseteq_s G_2$. Then $G_1 \sqsubseteq_w G_2$.

The proof follows from the fact that every strong simulation is a weak simulation.

The next result, coupled with the previous one, establishes that for discrete trees, the two simulation orderings in fact coincide.

Theorem 2. Suppose that G_1 and G_2 are discrete GSTs, and that $G_1 \sqsubseteq_w G_2$. Then $G_1 \sqsubseteq_s G_2$.

The proof uses induction on transitions to show that any weak simulation is also strong.

We now show that $\sqsubseteq_w / \sqsubseteq_s$ coincides with the simulation ordering, \sqsubseteq , given for synchronization trees (i.e. discrete, finite-branching GSTs) in Definition 6. The next definition defines a notion of \xrightarrow{a} for discrete GSTs.

Definition 12 (Transitions for Discrete GSTs). Let $G = \langle P, \preceq, p_0, \mathcal{L} \rangle$ be a GST, with $p, p' \in P$.

1. p' is an immediate successor of p if $p' \succ p$ and there exists no $p'' \in P$ such that $p' \succ p'' \succ p$.

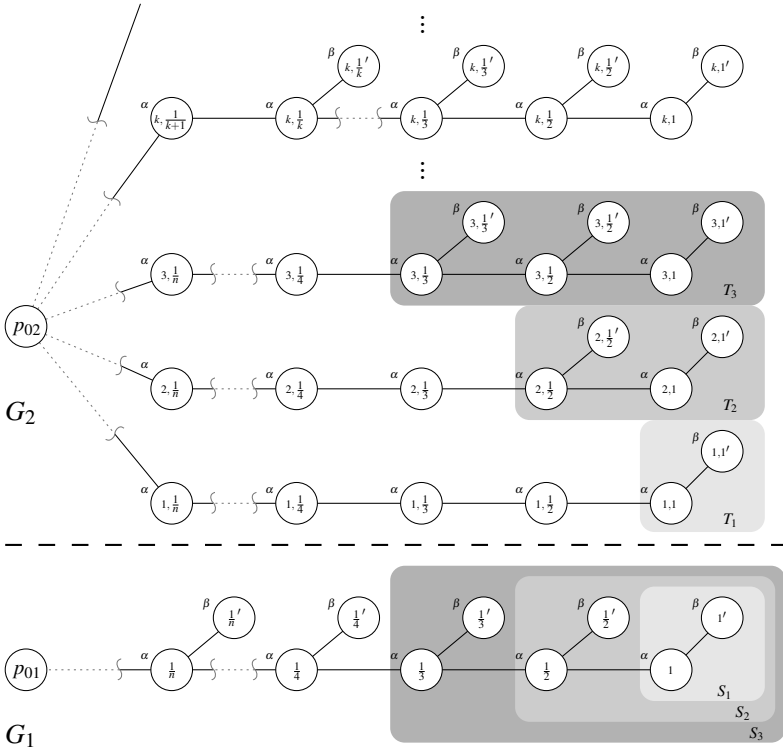


Fig. 1. Visualization of the GSTs used in proof of Theorem 4

2. $G[p, \text{ the subtree of } G \text{ rooted at } p, \text{ is } \langle P', \preceq', p, \mathcal{L}' \rangle$, where $P' = \{p' \in P \mid p \preceq p'\}$, and \preceq' / \mathcal{L}' are the restrictions of \preceq and \mathcal{L} to $P' / P' / \{p\}$.
3. Let $G' = \langle P', \preceq', p'_0, \mathcal{L}' \rangle$ be a GST. Then $G \xrightarrow{a} G'$ exactly when $p'_0 \in P$, p'_0 is an immediate successor of p_0 , $G' = G[p'_0$, and $\mathcal{L}(p') = a$.

Intuitively, $G \xrightarrow{a} G'$ if G' is an immediate subtree of G and the root of G' labeled by a . Based on this notion, Definition 6 may now be applied to discrete GSTs. We have the following.

Theorem 3. Let G_1, G_2 be discrete GSTs. Then the following are equivalent.

1. $G_1 \sqsubseteq G_2$.
2. $G_1 \sqsubseteq_w G_2$.
3. $G_1 \sqsubseteq_s G_2$.

One might hope that \sqsubseteq_w and \sqsubseteq_s would coincide for general GSTs, thereby obviating the need for two notions. Unfortunately, this is not the case.

Theorem 4. There exist GSTs G_1 and G_2 such that $G_1 \sqsubseteq_w G_2$ but $G_1 \not\sqsubseteq_s G_2$.

Proof. Consider the GSTs depicted in Figure 1. It turns out that the trees G_1 and G_2 are such that $G_1 \sqsubseteq_w G_2$, but $G_1 \not\sqsubseteq_s G_2$.

Both G_1 and G_2 use a label set $\{\alpha, \beta\}$, and both are discrete except for their start states. Each tree is constructed from a basic “time axis” $T = \{1/n \mid n \in \mathcal{N} \setminus \{0\}\} \cup \{0\}$, with the usual ordering \leq . The start state G_1 corresponds to time 0; each subsequent node is labeled by α if there is an edge to the next time point, or β if the node is maximal. In traditional synchronization-tree terms, each (non-start) node has an outgoing labeled α and another labeled β . G_2 is similar to G_1 except it contains an infinite number of branches from the start state, with branch k only enabling β transitions for the last k nodes.

The shading in the diagram illustrates a weak simulation that may be constructed and used to show that the start states in G_1 and G_2 are indeed related by a weak simulation. Intuitively, every trajectory from the start node of G_1 leads to a node from which a finite number of α s are possible, with a β possible at each step also. This trajectory can be matched by one from the start state of G_2 that leads to a branch from which enough β -less nodes can be bypassed.

On the other hand, no strong simulation can be constructed relating the start node of G_1 with G_2 . The basic intuition is that any trajectory leadings from the start node of G_1 has β s enabled at every intermediate node, and this behavior does not exist in any trajectory leading from the start node of G_2 . \square

The preceding result suggests that simulation is more nuanced for GSTs than for synchronization trees. One naturally may wonder which of the two notions proposed in this section is the “right” one. Our perspective is that the strong notion possesses a sense of invariance that one might expect for simulation; if one system is simulated by the other then any execution of the former can be “traced” by the latter following only related states. In this sense, strong simulation may be seen to have stronger intuitive justifications than the weaker one.

5 Constructing GSTs and Implications for Bisimulation

This section shows how discrete GSTs can be constructed from HyPA terms so that bisimulation on GSTs (recall that weak and strong bisimulation coincide for discrete trees) corresponds exactly with a congruence for HyPA terms in [6].

In Example 1, reference was made to the operational semantics of HyPA being given as a hybrid transition system; the states in the transition were HyPA-term / variable-valuation pairs. Bisimulation may be defined as usual for such a transition system. The authors of [6] then consider different definitions of bisimulation for terms alone. One obvious candidate defines terms \mathbf{p} and \mathbf{q} to be bisimilar iff for all variable valuations ν , $\langle \mathbf{p}, \nu \rangle$ and $\langle \mathbf{q}, \nu \rangle$ are bisimilar as states in the HyPA hybrid transition system. Unfortunately this relation is not a congruence for HyPA terms; the problem resides in the fact that parallel processes within terms can interfere with the variable valuations produced by another parallel process. To fix this problem, the authors introduce another relation, *robust bisimulation*, show it to be a congruence for HyPA, then establish that it is the same as another relation, *stateless bisimulation*, given in the same paper.

In the rest of this section we show how to construct GSTs from HyPA terms in such as way that two HyPA terms are statelessly bisimilar iff the corresponding

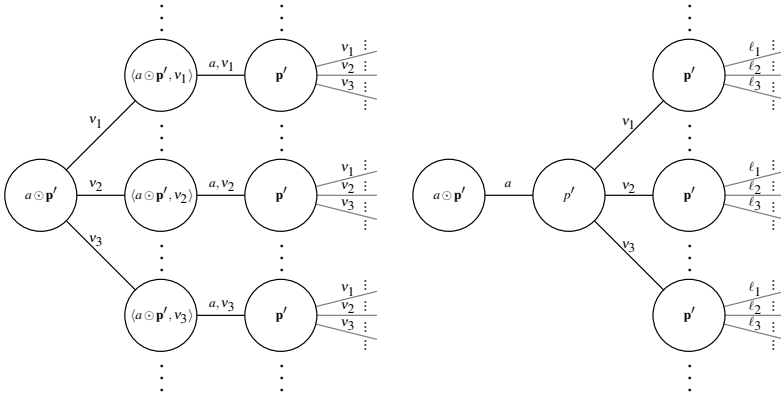


Fig. 2. Constructing GSTs from HyPA Terms

GSTs are bisimilar. We begin by reviewing the definition of stateless bisimulation. $\mathcal{T}(\mathcal{V}_p)$ is the set of HyPA terms that use only the recursive process variables \mathcal{V}_p , Val is the set of valuations for the (continuous) model variables, the transition $\xrightarrow{\ell}$ represents either a discrete action or a continuous flow (depending on ℓ) and \checkmark is a set of “terminating” states.

Definition 13 (Stateless bisimulation [6]). *Given a hybrid transition system with state space $\mathcal{T}(\mathcal{V}_p) \times Val$, a **stateless bisimulation relation** is a binary relation $R \subseteq \mathcal{T}(\mathcal{V}_p) \times \mathcal{T}(\mathcal{V}_p)$ such that for all $\nu, \nu' \in Val$ and $\mathbf{p}R\mathbf{q}$,*

- $\langle \mathbf{p}, \nu \rangle \in \checkmark$ implies $\langle \mathbf{q}, \nu \rangle \in \checkmark$,
- $\langle \mathbf{q}, \nu \rangle \in \checkmark$ implies $\langle \mathbf{p}, \nu \rangle \in \checkmark$,
- $\langle \mathbf{p}, \nu \rangle \xrightarrow{\ell} \langle \mathbf{p}', \nu' \rangle$ implies $\exists \mathbf{q}' \in \mathcal{T}(\mathcal{V}_p)$ such that $\langle \mathbf{q}, \nu \rangle \xrightarrow{\ell} \langle \mathbf{q}', \nu' \rangle \wedge \mathbf{p}'R\mathbf{q}'$ and
- $\langle \mathbf{q}, \nu \rangle \xrightarrow{\ell} \langle \mathbf{q}', \nu' \rangle$ implies $\exists \mathbf{p}' \in \mathcal{T}(\mathcal{V}_p)$ such that $\langle \mathbf{p}, \nu \rangle \xrightarrow{\ell} \langle \mathbf{p}', \nu' \rangle \wedge \mathbf{p}'R\mathbf{q}'$.

For simplicity we ignore \checkmark in what follows. The construction of GST $G_{\mathbf{p}}$ from HyPA term \mathbf{p} is exemplified in Figure 2. First, let the root of the $G_{\mathbf{p}}$ be identified with \mathbf{p} . Then for each valuation ν of the model variables, create one successor node of \mathbf{p} that is identified with $\langle \mathbf{p}, \nu \rangle$ and label these nodes as ν . Since each $\langle \mathbf{p}, \nu \rangle$ is a hybrid transition system state, the node has transitions of form $\langle \mathbf{p}, \nu \rangle \xrightarrow{\ell} \langle \mathbf{p}', \nu' \rangle$ for some ℓ ; for each such $\langle \mathbf{p}', \nu' \rangle$, make a node for \mathbf{p}' labeled by ℓ . Now repeat this procedure (coinductively) from \mathbf{p}' to obtain discrete GST $G_{\mathbf{p}}$. We now have the following (recall that weak and strong bisimilarity coincide for discrete GSTs).

Theorem 5. *Let \mathbf{p} and \mathbf{q} be HyPA terms. Then \mathbf{p} and \mathbf{q} are statelessly bisimilar iff $G_{\mathbf{p}}$ and $G_{\mathbf{q}}$ are bisimilar.*

There are yet other ways to represent HyPA processes as GSTs. For example, the behavioral systems in Example 2 suggest that if HyPA processes are regarded in terms of execution trajectories (i.e. functions of time), yet a different

GST construction can be obtained. It should be noted that such a construction necessarily has a great deal more granularity, as the resulting GSTs would be non-discrete. Consequently, our previous results about strong bisimulation would likely have significant ramifications for such a GST construction.

6 Composition of GSTs

One of the motivations for this work is to provide a framework for defining composition operators for systems having discrete / non-discrete behavior. In this section, we illustrate the potential of GSTs for this purpose by showing how a version of CSP parallel composition may be defined as a GST construction.

The parallel composition operator we consider is notated $|S|$, where S is a set of action labels. Given two (discrete) systems P and Q , $P|S|Q$ interleaves the executions of P and Q , with the following exception: actions in S must be performed by *both* P and Q in order to for $P|S|Q$ to perform them. The precise semantics of the operator may be given via the following SOS rules.

$$\boxed{\frac{P \xrightarrow{a} P' \quad a \notin S}{P|S|Q \xrightarrow{a} P'|S|Q}} \quad \boxed{\frac{Q \xrightarrow{a} Q' \quad a \notin S}{P|S|Q \xrightarrow{a} P|S|Q'}} \quad \boxed{\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q' \quad a \in S}{P|S|Q \xrightarrow{a} P'|S|Q'}}$$

Defining this operator in the GST setting requires first identifying the non-discrete analog of “interleaved execution.” Recall that for a GST, the analog of an execution is a (bounded) trajectory (cf. Definition 8). Interleaving two such trajectories can then be formalized as a linearization of the partial order obtained by taking the union of the trajectories. To formalize these ideas, first recall that if two partial orders $\langle P, \preceq_P \rangle$ and $\langle Q, \preceq_Q \rangle$ are disjoint (i.e. $P \cap Q = \emptyset$), then $\langle P \cup Q, \preceq_P \cup \preceq_Q \rangle$ is also a partial order. Interleavings can now be defined as follows.

Definition 14 (S-synchronized Interleaving). *Let $G_1 = \langle P_1, \preceq_1, p_1, \mathcal{L}_1 \rangle$ and $G_2 = \langle P_2, \preceq_2, p_2, \mathcal{L}_2 \rangle$ be GSTs, and WLOG assume that $P_1 \cap P_2 = \emptyset$. Also let T_1 and T_2 be trajectories (cf. Definition 8) from the roots of G_1 and G_2 , respectively. Also let $S \subseteq L$. Then total order $\langle Q, \preceq_Q \rangle$ is an S -synchronized interleaving of T_1 and T_2 iff there exists a monotonic bijection $\lambda \in \{p \in T_1 \mid \mathcal{L}_1(p) \in S\} \rightarrow \{p \in T_2 \mid \mathcal{L}_2(p) \in S\}$ such that the following hold.*

1. $\mathcal{L}_1(p) = \mathcal{L}_2(\lambda(p))$ for all $p \in T_1$ such that $\mathcal{L}_1(p) \in S$.
2. $Q = \{p \in T_1 \mid \mathcal{L}_1(p) \notin S\} \cup \{p \in T_2 \mid \mathcal{L}_2(p) \notin S\} \cup \{p, \lambda(p) \mid \mathcal{L}_1(p) \in S\}$.
3. Define $\pi_1 \in Q \rightarrow (T_1 \cup T_2)$ by $\pi_1(p) = p$ if $p \in T_1 \cup T_2$ and $\pi_1(\langle p'_1, p'_2 \rangle) = p'_1$ otherwise, and similarly for π_2 . Then, $\pi_1(q) \preceq_1 \pi_1(q')$ or $\pi_2(q) \preceq_2 \pi_2(q')$ implies $q \preceq_Q q'$.

We write $I_S(T_1, T_2)$ for the set of S -synchronized interleavings of T_1 and T_2 .

Intuitively, an S -synchronized interleaving of two trajectories from different (disjoint) trees is a total ordering on the union of the execution that respects the individual orderings from each of the trees in isolation while requiring synchronization on events in S . The bijection λ in the definition is used to identify the

synchronization partners in the trajectories. We may now define the CSP parallel composition construction on GST as follows.

Definition 15. Let $G_1 = \langle P_1, \preceq_1, p_1, \mathcal{L}_1 \rangle$ and $G_2 = \langle P_2, \preceq_2, p_2, \mathcal{L}_2 \rangle$ be GSTs with $P_1 \cap P_2 = \emptyset$. Then the GST $G_1 |S| G_2 = \langle Q, \preceq_Q, q_0, \mathcal{L}_Q \rangle$ is given by:

1. $Q = \{\langle p_1, p_2 \rangle\} \cup \{T \mid T \in I_S(T_1, T_2) \text{ for some trajectories } T_1 = (p_1, p'_1] \text{ of } G_1, T_2 = (p_2, p'_2] \text{ of } G_2\}$.
2. $q \preceq_Q q'$ iff $q = \langle p_1, p_2 \rangle$, or $q = \langle r, \preceq_r \rangle$, $q' = \langle r', \preceq_{r'} \rangle$, and $\preceq_r \subseteq \preceq_{r'}$.
3. $q_0 = \langle p_1, p_2 \rangle$.
4. Let $q \in Q$ and let $p' = \sup(q)$. Then define \mathcal{L}_Q according to

$$\mathcal{L}_Q(q) = \begin{cases} \mathcal{L}_1(p') & \text{if } p' \in P_1 \\ \mathcal{L}_2(p') & \text{if } p' \in P_2 \\ \mathcal{L}_2(p'_1) & \text{if } p' = \langle p'_1, p'_2 \rangle \end{cases}$$

To justify this construction, the following theorem shows that the definition coincides with the standard one for discrete systems (i.e. those modeled using labeled transition systems).

Theorem 6. Let G_T be the GST associated with a labeled transition system (LTS) T as given in Section 3. Then we have the following for LTSs T_1 and T_2 .

$$G_{(T_1 |S| T_2)} \sim_w G_{T_1} |S| G_{T_2}.$$

This result establishes that for discrete-time systems, the GST construction coincides with the standard one for labeled transition systems. However, Theorem 6 depends on the prohibition of unbounded trajectories in part 1 of Definition 15. This suggests that there is a non-trivial interplay between the properties of the parallel composition operator and the trajectories that are permitted in Definition 15. We regard this as an interesting direction for future research.

7 Conclusions and Directions for Future Research

This paper has defined Generalized Synchronization Trees, which are intended to provide a modeling framework for composition operations on systems that may contain non-discrete time. Like Milner’s synchronization trees, GSTs are also trees, but are based on earlier, non-inductive definitions of these structures that permit discrete as well as non-discrete behavior to be modeled uniformly. The work then considers notions of simulation and bisimulation for GSTs, establishing that definitions that coincide in the purely discrete setting of synchronization trees nevertheless differ in the generalized setting. It is then shown how a hybrid process algebra can be captured cleanly in our formalism, and also how a notion of parallel composition may be interpreted at a construction on GSTs.

There are numerous directions for future work. The framework in this paper makes no mention of real-time; indeed the definitions of simulation given in this

paper impose no restrictions on preserving duration information when matching up trajectories. This is by design, as one of the interesting observations to emerge is that one can have continuous as well as discrete notions of logical time. Nevertheless, enhancing the framework to accommodate metric notions of time would permit the embedding of various hybrid and real-time models into trees and the development of general notions of composition as a result. Describing other composition operations, and studying their congruence properties *vis à vis* (bi)simulation would yield useful insights into the algebra of GSTs. Finally, developing parsimonious mechanisms *à la* SOS rules for defining composition operations coinductively would simplify their definition and open up insights into the meta-theory of GSTs.

References

1. Abramsky, S.: A domain equation for bisimulation. *Information and Computation* 92(2), 161–218 (1991)
2. Aceto, L., Bloom, B., Vaandrager, F.: Turning SOS rules into equations. *Information and Computation* 111(1), 1–52 (1994)
3. Aczel, P.: Non-Well-Founded Sets. CSLI Lecture Notes, vol. 14. Center for the Study of Language and Information (1988)
4. Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.): *Handbook of Process Algebra*. North-Holland, Amsterdam (2001)
5. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. *J. ACM* 42(1), 232–268 (1995)
6. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. *The Journal of Logic and Algebraic Programming* 62(2), 191–245 (2005)
7. Cuijpers, P.J.L.: Prefix Orders as a General Model of Dynamics. In: 9th International Workshop on Developments in Computational Models, CONCUR (2013)
8. Davoren, J.M., Tabuada, P.: On Simulations and Bisimulations of General Flow Systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 145–158. Springer, Heidelberg (2007)
9. Girard, A., Pappas, G.J.: Approximate bisimulation: A bridge between computer science and control theory. *Eur. J. Control* 17(5-6), 568–578 (2011)
10. Haghverdi, E., Tabuada, P., Pappas, G.J.: Unifying Bisimulation Relations for Discrete and Continuous Systems. In: Proceedings of the International Symposium MTNS 2002, South (2002)
11. Haghverdi, E., Tabuada, P., Pappas, G.J.: Bisimulation relations for dynamical, control, and hybrid systems. *Theoretical Comput Science* 342, 229–261 (2005)
12. Jacobs, B., Rutten, J.: A tutorial on (co)algebras and (co)induction. *EATCS Bulletin* 62, 62–222 (1997)
13. Jech, T.: *Set Theory*. Academic Press (1978)
14. Julius, A.A., van der Schaft, A.J.: Bisimulation as congruence in the behavioral setting. In: Proceedings of the 44th IEEE Conference on Decision and Control and 2005 European Control Conference, pp. 814–819 (2005)
15. Milner, R.: *A Calculus of Communication Systems*. LNCS, vol. 92. Springer, Heidelberg (1980)

16. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
17. Rounds, W.C.: On the relationships between Scott domains, synchronization trees, and metric spaces. *Information and Control* 66(1-2), 6–28 (1985)
18. Sangiorgi, D.: An introduction to bisimulation and coinduction. Cambridge University Press, Cambridge (2012)
19. Willems, J.C.: On interconnections, control, and feedback. *IEEE Transactions on Automatic Control* 42(3), 326–339 (1997)
20. Winskel, G.: Synchronization Trees. *Theoretical Computer Science* 34(1-2), 33–82 (1984)

Bisimulations for Communicating Transactions^{*}

(Extended Abstract)

Vasileios Koutavas, Carlo Spaccasassi^{**}, and Matthew Hennessy

Trinity College Dublin

Abstract. We develop a theory of bisimulations for a simple language containing communicating transactions, obtained by dropping the isolation requirement of standard transactions. Such constructs have emerged as a useful programming abstraction for distributed systems.

In systems with communicating transactions actions are tentative, waiting for certain transactions to commit before they become permanent. Our theory captures this by making bisimulations history-dependent, in that actions performed by transactions need to be recorded. The main requirement on bisimulations is the systems being compared need to match up exactly in the permanent actions but only those.

The resulting theory is fully abstract with respect to a natural contextual equivalence and, as we show in examples, provides an effective verification technique for comparing systems with communicating transactions.

1 Introduction

Communicating transactions, obtained by dropping the isolation requirement of standard transactions, is a novel and powerful programming construct for distributed systems. For example, it can be used to simplify the programming of complex concurrent consensus scenarios, avoiding the use of locks and explicit error handling [16]. Variants of such constructs have been proposed as extensions to programming languages [5,7,11,16] and process calculi [2,1,3]. However, before they can be adopted in mainstream programming, significant research is needed in efficient implementation strategies [6,11,16], programming paradigms [1,7], and viable verification techniques [8]. The last concern is the topic of this paper.

Bisimulations [12] provide an elegant and effective proof technique for proving equivalences between processes in a variety of settings (e.g., [15,14]). They are essentially defender strategies in a game where a challenger attempts to discover a difference in the extensional behaviour of two processes, while the defender tries to refute these attempts [17]. For example, consider the standard CCS processes

$$P_1 = a.(b.\mathbf{0} + c.\mathbf{0}) \qquad Q_1 = a.b.\mathbf{0} + a.c.\mathbf{0} \qquad (1)$$

^{*} This research was supported by SFI project SFI 06 IN.1 1898.

^{**} Supported by MSR (MRL 2011-039).

which can perform the action a followed by either b or c , with a slight difference in when this choice is taken. The challenger chooses Q_1 to perform the action a , with residual $Q'_1 = b.\mathbf{0}$ to which the defender must respond with a matching a action from P_1 ; the only possibility is for P_1 to perform a with residual $P'_1 = b.\mathbf{0} + c.\mathbf{0}$. But now the challenger chooses P'_1 to perform the c action, to which the defender has no response. The defender loses the game. In fact there is no possible winning strategy for the defender in this game and thus no bisimulation containing the pair (P_1, Q_1) . Therefore P_1 and Q_1 are deemed to be behaviourally distinct.

However, the appropriate notion of the bisimulation game for a language with communicating transactions or similar constructs is *a priori* unclear. An objective criterion for a potential bisimulation game is relation with contextual equivalence. If the existence of a winning defender's strategy in a game over two systems implies that the systems are contextually equivalent then this game is a *sound* verification technique. If the absence of such a strategy implies that the two systems are contextually inequivalent then the game is a *complete* technique. The main result of this paper is a weak bisimulation theory for a simple language containing communicating transactions, $TCCS^m$, which provides a sound and complete proof methodology with respect to a natural contextual equivalence.

$TCCS^m$ is obtained from CCS [12] by essentially adding one construct $\llbracket P \triangleright_k Q \rrbracket$ for communicating transactions, and a new command co for committing them. Here k is the name of the transaction, P the body which is expected to be completed in its entirety or not at all, and Q is the alternative, to be executed in case the transaction is aborted. However it should be emphasised that if an abort occurs not only is Q launched but P and all its effects on the environment are rolled back. For example consider the systems with p fresh from R and S :

$$\begin{aligned} P_2 &= \nu p. \llbracket a.p.\text{co}.R \triangleright_k \mathbf{0} \rrbracket \parallel \llbracket b.\bar{p}.\text{co}.S \triangleright_l \mathbf{0} \rrbracket \\ Q_2 &= \llbracket a.b.\text{co}.(R \mid S) + b.a.\text{co}.(R \mid S) \triangleright_m \mathbf{0} \rrbracket \end{aligned} \quad (2)$$

Here P_2 consists of two independent transactions which co-operate by synchronising on a private channel p . If after this synchronisation the left-hand transaction aborts then the effect of the a action, which is a communication with the environment, must be rolled back. But the synchronisation on p must also be undone, and therefore, because of the all-or-nothing nature of transactions, the effects of the b action in the right-hand transaction will be rolled back. Indeed in our reduction semantics, given in Sect. 2, this right-hand transaction is also aborted. Because of the synchronisation on p , the destiny of both transactions is conjoined. For this reason we should be able to demonstrate that P_2 is behaviourally equivalent to the single transaction Q_2 .

The standard bisimulation game outlined above cannot be easily extended to $TCCS^m$. Many actions, such as a, b in (2) above, are tentative, in the sense that their effect can only be considered to be permanent when the transactions k and l commit, if ever. To underline this consider the processes:

$$P_3 = \llbracket a.(b.\text{co} + c.\mathbf{0}) \triangleright_k \mathbf{0} \rrbracket \quad Q_3 = \llbracket a.b.\text{co} \triangleright_l \mathbf{0} \rrbracket \quad (3)$$

Here P_3 commits only after performing the a, b . It would be unreasonable for the challenger, after the a action, to demand a response to c . Not only is this action

tentative on the completion of transaction k but also if c is performed then k will *never* commit; so the defender should be able to ignore this challenge.

Our approach is to play the bisimulation game on *configurations*, of the form $\mathcal{C} = (H \triangleright P)$ where P is a system and H a *history* of all the tentative actions taken so far in the game by P . These are of the form $k(a)$, where k is the name of a transaction which needs to commit before the action becomes a permanent a . When playing bisimulation moves, the histories of both systems being scrutinised must remain *consistent*, in that permanent actions in the respective histories must match exactly (for simplicity old permanent actions are not garbage collected). The crucial aspect of this new game is that when a system commits a transaction k , and only then, all tentative actions in its history dependent on k are made permanent. This consistency requirement then forces a response in which the corresponding actions match exactly.

For example consider the following variation on (1), using transactions:

$$P_4 = \llbracket a.(b.\text{co} + c.\text{co}) \triangleright_l \mathbf{0} \rrbracket \quad Q_4 = \llbracket a.b.\text{co} + a.c.\text{co} \triangleright_k \mathbf{0} \rrbracket \quad (4)$$

Replaying the game from (1), where the challenger first chooses a from Q_4 and then c from P_4 with the same responses, we reach the configurations

$$\mathcal{C}_4 = (k(a), k(c) \triangleright \llbracket \text{co} \triangleright_k \mathbf{0} \rrbracket) \quad \mathcal{D}_4 = (l(a), l(b) \triangleright \llbracket \text{co} \triangleright_l \mathbf{0} \rrbracket)$$

At this stage the two histories are still consistent as they contain no permanent actions. However, now there is no possible response when the challenger chooses the commit move $\mathcal{C}_4 \rightarrow \mathcal{C}'_4 = (a, c \triangleright \mathbf{0})$. This is a silent move from \mathcal{C}_4 in which transaction k commits, making the two actions in the history permanent. There are various ways in which \mathcal{D}_4 can try to respond but all lead to an inconsistent history. Thus, with our version of bisimulations P_4 and Q_4 are not bisimilar.

Note that such a successful attack by the challenger cannot be mounted for (3) above. After one round in the game we have the configurations

$$\mathcal{C}_3 = (k(a) \triangleright \llbracket b.\text{co} + c.\mathbf{0} \triangleright_k \mathbf{0} \rrbracket) \quad \mathcal{D}_3 = (l(a) \triangleright \llbracket b.\text{co} \triangleright_l \mathbf{0} \rrbracket)$$

and since \mathcal{D}_3 has no possible c actions the challenger might request a response to the action $\mathcal{C}_3 \rightarrow \mathcal{C}'_3 = (k(a), k(c) \triangleright \llbracket \mathbf{0} \triangleright_k \mathbf{0} \rrbracket)$. However the tentative $k(c)$ recorded in the history will never become permanent and thus the defender can successfully respond with any tentative action of \mathcal{D}_3 which will also never become permanent. To ensure that such responses can always be made, our bisimulations will allow *any* configuration to make the degenerate silent move $(H \triangleright P) \rightarrow (H, k(\star) \triangleright P)$, where \star is a reserved symbol. This defender move represents a weak idle move whose validity is postponed until after k commits. So \mathcal{C}_3 's move above can be matched by \mathcal{D}_3 playing this degenerate move followed by the abort of the transaction. Later we prove that P_3 and Q_3 are indeed bisimilar.

Using recursion we can write *restarting transactions* as $\text{rec}X.\llbracket P \blacktriangleright X \rrbracket$. Here $\llbracket P \blacktriangleright X \rrbracket$ is an uninitiated transaction which has yet to be allocated a name. These transactions can abort and re-run internal steps, thus branching differences in initial silent actions can be hidden from the challenger. Consider a compiler performing common subexpression elimination, transforming P_5 to Q_5 :

$$P_5 = \text{rec}X. \llbracket \tau.b.\text{co} + \tau.c.\text{co} \blacktriangleright X \rrbracket \quad Q_5 = \text{rec}X. \llbracket \tau.(b.\text{co} + c.\text{co}) \blacktriangleright X \rrbracket \quad (5)$$

As we will show, all moves of the challenger can be matched by the defender in the bisimulation game. The interesting scenario is when (after initiating the transactions) the challenger picks the right τ action from P_5 and the defender responds with the τ action from Q_5 . We then get the configurations:

$$\mathcal{C}_5 = (\varepsilon \triangleright \llbracket c.\text{co} \triangleright_k P_5 \rrbracket) \quad \mathcal{D}_5 = (\varepsilon \triangleright \llbracket b.\text{co} + c.\text{co} \triangleright_l Q_5 \rrbracket)$$

The challenger then picks the b action from \mathcal{D}_5 . The defender responds with a silent abort of \mathcal{C}_5 which will reinstate P_5 , re-initialise the transaction, and select the left τ action in \mathcal{P}_5 followed by the b action. This would lead to the configurations $\mathcal{C}'_5 = (k'(b) \triangleright \llbracket \text{co} \triangleright_{k'} P_5 \rrbracket)$ and $\mathcal{D}'_5 = (l(b) \triangleright \llbracket \text{co} \triangleright_l Q_5 \rrbracket)$. We will in fact prove that this optimisation is sound in our setting, although it is not sound in the case of P_4 , Q_4 , even if we used restarting transactions.

In the remainder of this extended abstract we explain the language $TCCS^m$ (Sect. 2), which is a simplification of that used in previous work [4] in that we do not consider nested transactions, simplifying technical development. Inspired by cJoin [1], when transactions co-operate by synchronising on an action they are virtually *merged* by acquiring the same name. Thus in P_2 from (2), when synchronisation occurs on the private channel p , the residual will be a term equivalent to the single transaction $\nu p. \llbracket \text{co}.(R \mid S) \triangleright_n \mathbf{0} \mid \mathbf{0} \rrbracket$.

This is followed by an exposition of our history dependent bisimulations (Sect. 3). As we have already stated, these bisimulations demand the appropriate matching of all actions, even those dependent on transactions which can never commit. We also give a variation, called *predictive bisimulations*, in which dependent actions need only be matched when the transaction on which they depend has some future possibility of committing (Sect. 4). We ultimately show that both equivalences coincide but the former is an easier proof technique while the latter is easier to prove sound. We then outline the proof of the main result of the paper, namely that these bisimulation equivalences coincide with contextual equivalence (Sect. 5).

2 The Language $TCCS^m$

The syntax for terms in the language is given in Fig. 1 where $a \in \text{Act}$ are actions, $\mu \in \text{Act} \uplus \{\tau\} \uplus \Omega$ are prefixes, and X ranges over a collection of *recursion variables*. Here $\omega \in \Omega$ are special actions which will be used to define contextual equivalence, while $(\bar{\tau}) : \text{Act} \rightarrow \text{Act}$ is a total bijection over Act , used in the standard manner to formalise CCS synchronisation between processes. The language contains all the standard constructs of CCS, with the result that CCS is a sub-language of $TCCS^m$. There are three extra operators, discussed in the Introduction. We assume the standard notion of free and bound occurrence of recursion variables, and only consider closed terms, those which contain no free occurrences of variables. We use the standard abbreviations associated

TCCS^m Syntax		
$P, Q, R ::= \sum \mu_i.P_i \mid P \mid Q \mid \nu a.P \mid X \mid \mathbf{rec} X.P$ $\mid \llbracket P \triangleright_k Q \rrbracket \mid \mathbf{co}.P \mid \llbracket P \blacktriangleright Q \rrbracket$		
CCS Transitions		
CCSSUM $\frac{}{\sum \mu_i.P_i \xrightarrow{\mu_i} P_i}$	CCSSYNC $\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	CCSREC $\frac{}{\mu X.P \xrightarrow{\tau} P[\mu X.P/X]}$
Transactional Transitions		
TRTAU $\frac{P \xrightarrow{\tau} P'}{\llbracket P \triangleright_k Q \rrbracket \xrightarrow{\tau} \llbracket P' \triangleright_k Q \rrbracket}$	TRSUM $\frac{}{\sum \mu_i.P_i \xrightarrow{k(a)}_{\varepsilon \mapsto k} \llbracket P_j \mid \mathbf{co} \triangleright_k \sum \mu_i.P_i \rrbracket} \mu_j = a$	
TRACT $\frac{P \xrightarrow{\alpha} P'}{\llbracket P \triangleright_l Q \rrbracket \xrightarrow{k(a)}_{l \mapsto k} \llbracket P' \triangleright_k Q \rrbracket} k \# l$	TRSYNC $\frac{P \xrightarrow{k(a)}_{\sigma_1} P' \quad Q \xrightarrow{k(\bar{a})}_{\sigma_2} Q' \quad \sigma_1 = \tilde{l}_1 \mapsto k \quad \sigma_2 = \tilde{l}_2 \mapsto k}{P \mid Q \xrightarrow{k(\tau)}_{(\tilde{l}_1, \tilde{l}_2) \mapsto k} P' \sigma_2 \mid Q' \sigma_1}$	
Propagation Transitions		
RESTR $\frac{P \xrightarrow{\alpha}_{\sigma} P'}{\nu a.P \xrightarrow{\alpha}_{\sigma} \nu a.P'} a \notin \alpha$	PARL $\frac{P \xrightarrow{\alpha}_{\sigma} P'}{P \mid Q \xrightarrow{\alpha}_{\sigma} P' \mid Q\sigma} \mathit{range}(\sigma) \# Q$	

Fig. 1. Communication and internal transitions (omitting symmetric rules)

with CCS, and write $s \# s'$ when the transaction names of the syntax object s are fresh from those in s' ; $\mathit{fn}(s)$ denotes the transaction names in s . Note that unlike previous work [3,4] transaction names are never bound and we do not require that all transaction names used in a term are distinct. Thus, we allow terms of the form $\llbracket P_1 \triangleright_k P_2 \rrbracket \mid R \mid \llbracket Q_1 \triangleright_k Q_2 \rrbracket$. Here k should be looked upon as a *distributed* transaction whose behaviour will be approximately the same as the centralised $\llbracket P_1 \mid Q_1 \triangleright_k P_2 \mid Q_2 \rrbracket$. The use of these distributed transactions will simplify considerably the exposition of the reduction semantics.

Definition 2.1. *A closed term is called well-formed if in every occurrence of $\llbracket P \triangleright_k Q \rrbracket$, $\llbracket P \blacktriangleright Q \rrbracket$, and $\mathbf{rec} X.P$, the subterms P and Q do not contain named transactions of the form $\llbracket - \triangleright_k - \rrbracket$. We refer to well-formed terms as processes.*

Note that dormant transactions *can* appear within other transactions and under recursion but they will be activated only when they end up at top-level. In the sequel we only consider well-formed terms.

The reduction semantics of the language is given as a binary relation between processes $P \rightarrow Q$. However this is defined indirectly in terms of three auxiliary relations, which will also be used in the formulation of bisimulations:

$$P \rightarrow Q \quad \text{when} \quad P \xrightarrow{\tau}_{\sigma} Q \quad \text{or} \quad P \xrightarrow{\beta} Q \quad \text{or} \quad P \xrightarrow{k(\tau)}_{\sigma} Q \quad (6)$$

TRNEW <hr style="border: 1px solid black;"/> $\frac{[[P \blacktriangleright Q]] \xrightarrow{\text{new } k} [[P \triangleright_k Q]]}{\text{TRBCAST}}$ $\frac{P \xrightarrow{\beta} P' \quad Q \xrightarrow{\beta} Q' \quad \beta \in \{\text{co } k, \text{ab } k\}}{P Q \xrightarrow{\beta} P' Q'}$	TRAB <hr style="border: 1px solid black;"/> $[[P \triangleright_k Q]] \xrightarrow{\text{ab } k} Q$ TRIGNORE $\frac{P \xrightarrow{\beta} P'}{P Q \xrightarrow{\beta} P' Q} \beta \# Q$	TRCO $\frac{P \rightsquigarrow_{\text{co}} P'}{[[P \triangleright_k Q]] \xrightarrow{\text{co } k} P'}$ TRRESTR $\frac{P \xrightarrow{\beta} P'}{\text{va. } P \xrightarrow{\beta} \text{va. } P'}$
--	---	---

Fig. 2. Transactional reconfiguration transitions

The first, $P \xrightarrow{\tau}_\sigma Q$, is essentially synchronisation between pure CCS processes. The second, $P \xrightarrow{\beta} Q$ (Fig. 2), where β ranges over $\text{co } k$, $\text{ab } k$ and $\text{new } k$, encode the creation of new named transactions, and commit/abort broadcast transitions (TRCO, TRAB, TRBCAST) which eliminate distributed transactions. The notation $P \rightsquigarrow_{\text{co}} P'$ means the execution of a top-level co in P and the replacement of all other top-level commits with τ -prefixes. We now concentrate on the third, $P \xrightarrow{k(\tau)}_\sigma Q$, or more generally $P \xrightarrow{k(\mu)}_\sigma Q$ where $\mu \in \text{Act} \cup \{\tau\}$.

Action $P \xrightarrow{k(a)}_\sigma Q$ should be viewed as the synchronisation between P and some transaction named k in the environment which can perform the complementary \bar{a} . Because this transaction is external the freshness side conditions in the rules of Fig. 1 ask that the name k is fresh with respect to P . Also, the effect of this synchronisation is that the future behaviour of P , or at least any transactions involved in the execution of a , is dependent on the eventual committing of k . This dependency is implemented by σ , a substitution renaming the responsible transaction in P to k . The essential rule in the generation of these judgements is TRACT in Fig. 1. For example, this rule ensures that we can derive $[[a.P_1 \triangleright_{l_1} Q_1]] \xrightarrow{k(a)}_{l_1 \mapsto k} [[P_1 \triangleright_k Q_1]]$ for any fresh k . The substitution recorded in the action is propagated by PARL into contexts. Note that by TRSUM, even pure CCS processes with no transactions can perform a $k(a)$ action; e.g., $a.P \xrightarrow{k(a)}_{\varepsilon \mapsto k} [[P | \text{co } \triangleright_k a.P]]$. This embeds P into the k -transaction; the distributed part of the k -transaction surrounding P is always ready to commit (hence the introduction of co). Note that this is a *communication-driven embedding*, which reduces the nondeterminism of embedding of previous work [3,4], making semantics more concrete [16]. Embedding leads to a uniform treatment of CCS processes and transactions, and a simple reduction semantics.

The conjoining of transactions is implemented in TRSYNC. Using it we infer:

$$[[a.P_1 \triangleright_{l_1} Q_1]] \mid [[\bar{a}.P_2 \triangleright_{l_2} Q_2]] \xrightarrow{k(\tau)}_{(l_1, l_2) \mapsto k} [[P_1 \triangleright_k Q_1]] \mid [[P_2 \triangleright_k Q_2]]$$

for any fresh k . Here the previously independent transactions l_1, l_2 have been merged into the transaction k (recorded in the substitution $(l_1, l_2) \mapsto k$). Note that this new transaction is distributed, in that its activity is divided in two. In order for it to commit the rules in Fig. 2 ensure that *both* components commit.

Example 2.2. Consider the process P_2 defined in (2) in the Introduction. Two applications of TRACT followed by rule RESTR gives the reduction from (6) above

$$P_2 \mid \bar{a}.b \rightarrow^* \nu p. \llbracket p.\text{co}.R \triangleright_{k_1} \mathbf{0} \rrbracket \mid \llbracket \bar{p}.\text{co}.S \triangleright_{k_2} \mathbf{0} \rrbracket = P'_2$$

where k_1 and k_2 are fresh names. The synchronisation rule TRSYNC then gives

$$P'_2 \rightarrow \nu p. \llbracket \text{co}.R \triangleright_k \mathbf{0} \rrbracket \mid \llbracket \text{co}.S \triangleright_k \mathbf{0} \rrbracket$$

where k is an arbitrary fresh name. Here the residual is a single transaction named k , albeit distributed. For it to commit both components have to commit: using TRBCAST this leads to the process $R \mid S$. \square

The semantics has a number of properties: it preserves well-formedness, generates only fresh transaction names and is equivariant. The properties about transaction names are important because they give us the liberty to pick fresh enough transaction names in proofs. To state these properties we use *renamings*, ranged over by r , which are bijective substitutions of the form $\{l_1/k_1, \dots, l_n/k_n\}$. The range of a *fresh* renaming r_{fr} has names not appearing in the proof.

Lemma 2.3 (Names). *Suppose $P \xrightarrow{l(\mu)}_{\sigma} Q$. Then*

1. l is fresh to P , Q is well-formed, and the substitution σ has the form $\tilde{k} \mapsto l$;
2. (Equivariance) $P r_{\text{fr}} \xrightarrow{l'(\mu)}_{\sigma r_{\text{fr}}} Q r_{\text{fr}}$, where $r_{\text{fr}}(l) = l'$. \square

Based on this semantics we give a natural contextual equivalence, using standard formulations [15]. We write \Rightarrow for the reflexive transitive closure of \rightarrow .

Definition 2.4 (Barb). $P \Downarrow \omega$ ($\omega \in \Omega$) if $\exists Q, Q'$ such that $P \Rightarrow Q \xrightarrow{\omega}_{\epsilon} Q'$.

Definition 2.5 (Reduction Barbed Equivalence (\cong_{rbe})). (\cong_{rbe}) is the largest relation for which $P \cong_{\text{rbe}} Q$ when:

1. $P \Downarrow \omega$ iff $Q \Downarrow \omega$,
2. if $P \rightarrow P'$ then there exists Q' such that $Q \Rightarrow Q'$ and $P' \cong_{\text{rbe}} Q'$,
3. if $Q \rightarrow Q'$ then there exists P' such that $P \Rightarrow P'$ and $P' \cong_{\text{rbe}} Q'$,
4. $P \mid R \cong_{\text{rbe}} Q \mid R$ for any R with $R \nmid P, Q$.

Here we consider contexts with fresh transaction names to enforce that observer transactions are distinct from process transactions before communication occurs. If this was not the case then transaction names would be observable: $\llbracket P \triangleright_k Q \rrbracket$ would not be equivalent to $\llbracket P \triangleright_l Q \rrbracket$ because by introducing the context $\llbracket \mathbf{0} \triangleright_k \mathbf{0} \rrbracket$ the k -transaction can no longer commit but l still can. Thus, *all* transaction names are considered local here; the side condition $R \nmid P, Q$ enforces this without the syntactic overhead of a ν -binder for all transaction names.

To see why in the above definition we use barbs from a distinct Ω consider:

$$P = \llbracket a.\text{co} \triangleright_k \mathbf{0} \rrbracket \qquad Q = a.\mathbf{0} + \tau.\mathbf{0}$$

Intuitively, we would expect P to have exactly the same behaviour as Q , eventually executing the single action a , or failing with a τ step. But if we allowed the barb $\Downarrow a$ in Thm. 2.5 then they would not be equivalent because $P \Downarrow a$ and $Q \not\Downarrow a$.

Example 2.6. P_4 and Q_4 from (4) in the Introduction are indeed inequivalent.

Assume $P_4 \cong_{\text{rbe}} Q_4$. Take $C_1 = \bar{a}$. Then $P_4 | C_1 \cong_{\text{rbe}} Q_4 | C_1$. We have:

$$Q_4 | C_1 \xrightarrow{l(\tau)} \llbracket b.\text{co} \triangleright_l \mathbf{0} \rrbracket | \llbracket \text{co} \triangleright_l C_1 \rrbracket = Q'_4 \quad \text{thus} \quad Q_4 | C_1 \rightarrow Q'_4$$

Process Q'_4 should be equivalent to $P_4 | C_1$ or one of its successors:

1. $P_4 | C_1$. Let $C_2 = \bar{c}.\omega$; then $Q'_4 | C_2 \not\Downarrow \omega$, $P_4 | C_1 | C_2 \Downarrow \omega$. Thus $P_4 | C_1 \not\cong_{\text{rbe}} Q'_4$.
2. $P'_4 = \llbracket b.\text{co} + c.\text{co} \triangleright_m \mathbf{0} \rrbracket | \llbracket \mathbf{0} \triangleright_m C_1 \rrbracket$. Again, $P'_4 | C_2 \Downarrow \omega$, thus $P'_4 \not\cong_{\text{rbe}} Q'_4$.
3. C_1 (after an abort). $Q'_4 | \bar{b}.\omega \Downarrow \omega$ but $C_1 | \bar{b}.\omega \not\Downarrow \omega$, so $C_1 \not\cong_{\text{rbe}} Q'_4$.

Thus $Q'_4 \not\cong_{\text{rbe}} P_4 | C_1$ or any later state (contradiction), and $P_4 \not\cong_{\text{rbe}} Q_4$. \square

Note that the difference in the branching structure of P_4 and Q_4 is not observable by the may- and must-testing equivalences [3,4]. These equivalences are characterised by so-called *clean traces*, which are traces in which all tentative actions are committed. If bisimulations were developed using such clean traces, P_4 and Q_4 would also be identified in the resulting bisimulation theory but it would not correspond to a natural definition of (\cong_{rbe}) .

3 Bisimulations

As mentioned in the Introduction, our bisimulations will be over configurations $(H \triangleright P)$ with a process P and a history H of the tentative interactions of P with its environment. An element of such a history can be a $k(a)$, a , ab , $k(\star)$, or \star . A past tentative action $k(a)$ that has not been committed or aborted is recorded as is in the history. If the k -transaction that performed this action commits, the action becomes a ; if k aborts, it becomes ab . Histories also record the trivial actions $k(\star)$ which can be performed by any process. If k commits, $k(\star)$ -recordings become \star , which terminates the bisimulation game in favour of the attacker; if k aborts they become ab . For technical convenience in proofs, elements in a history are uniquely indexed and permanent actions are not garbage-collected.

Definition 3.1 (History). A history H is a partial function from objects i of a countable set I to the set $\{a, \star, k(a), k(\star), \text{ab} \mid a \in \text{Act}\}$.

We often write histories as *lists*, omitting the indices of their elements. History composition, written as H_1, H_2 , is defined when $\text{dom}(H_1) \cap \text{dom}(H_2) = \emptyset$. We also let \hat{a} and \hat{b} range over $\text{Act} \cup \{\star\}$ and $\hat{\mu}$ range over $\text{Act} \cup \Omega \cup \{\tau, \star\}$. To express the effect of commits and aborts to histories we define the following operations.

Definition 3.2. $H \setminus_{\text{co}} k$ and $H \setminus_{\text{ab}} k$ are the lifting to lists of the operations:

$$\begin{array}{lll} (i \mapsto k(\hat{a})) \setminus_{\text{co}} k & = (i \mapsto \hat{a}) & (i \mapsto k(\hat{a})) \setminus_{\text{ab}} k & = (i \mapsto \text{ab}) \\ (i \mapsto l(\hat{a})) \setminus_{\text{co}} k & = (i \mapsto l(\hat{a})) & (i \mapsto l(\hat{a})) \setminus_{\text{ab}} k & = (i \mapsto l(\hat{a})) & \text{when } k \# l \\ (i \mapsto \hat{a}) \setminus_{\text{co}} k & = (i \mapsto \hat{a}) & (i \mapsto \hat{a}) \setminus_{\text{ab}} k & = (i \mapsto \hat{a}) \\ (i \mapsto \text{ab}) \setminus_{\text{co}} k & = (i \mapsto \text{ab}) & (i \mapsto \text{ab}) \setminus_{\text{ab}} k & = (i \mapsto \text{ab}) \end{array}$$

For the reasons we explained in the Introduction, weak bisimulations for $TCCS^m$ require configurations to agree on the committed actions in their histories, and only those actions. Soundness of our technique will establish this as a sufficient requirement for contextual equivalence between processes.

Definition 3.3 (Consistency). H_1 and H_2 are consistent when they have the same domain and for all $i \in I$, $a \in \text{Act}$: $H_1(i) = a$ iff $H_2(i) = a$.

History consistency is one of the two main requirements for weakly bisimilar configurations; the other is to have the same barbs. Thus the weak bisimulation game for $TCCS^m$ will be over transitions with three simple labels: $\zeta ::= \tau \mid k \mid \omega$ annotating internal (τ), tentative synchronisation (k), and barb (ω) transitions.

Definition 3.4 (Bisimulation Transitions). $C \xrightarrow{\zeta} C'$ is derived by the rules:

$$\begin{array}{lll}
(H \triangleright P) \xrightarrow{\tau} (\sigma(H) \triangleright Q) & \text{if } P \xrightarrow{\tau}_{\sigma} Q & (\text{LTS}\tau) \\
(H \triangleright P) \xrightarrow{\tau} (\sigma(H) \triangleright Q) & \text{if } P \xrightarrow{k(\tau)}_{\sigma} Q \text{ and } k \# H & (\text{LTS}k(\tau)) \\
(H \triangleright P) \xrightarrow{\tau} (H \triangleright Q) & \text{if } P \xrightarrow{\text{new } k}_{\sigma} Q \text{ and } k \# H & (\text{LTS}\text{new}) \\
(H \triangleright P) \xrightarrow{\tau} (H \setminus_{\text{co}} k \triangleright Q) & \text{if } P \xrightarrow{\text{co } k}_{\sigma} Q & (\text{LTS}\text{co}) \\
(H \triangleright P) \xrightarrow{\tau} (H \setminus_{\text{ab}} k \triangleright Q) & \text{if } P \xrightarrow{\text{ab } k}_{\sigma} Q & (\text{LTS}\text{ab}) \\
(H \triangleright P) \xrightarrow{k} (\sigma(H), k(a) \triangleright Q) & \text{if } P \xrightarrow{k(a)}_{\sigma} Q \text{ and } k \# H & (\text{LTS}k(a)) \\
(H \triangleright P) \xrightarrow{k} (H, k(\star) \triangleright P) & \text{if } k \# H, P & (\text{LTS}\star) \\
(H \triangleright P) \xrightarrow{\omega} (\sigma(H) \triangleright Q) & \text{if } P \xrightarrow{\omega}_{\sigma} Q & (\text{LTS}\omega)
\end{array}$$

We define $\xrightarrow{\zeta}$ to be $\xrightarrow{\tau}^*$ when $\zeta = \tau$, and $\xrightarrow{\tau} \xrightarrow{\zeta} \xrightarrow{\tau}$ otherwise.

The first five rules encode the $TCCS^m$ reduction semantics of (6) in Sect. 2, updating the history of the configurations accordingly. $\text{LTS}k(a)$ encodes the synchronisation between a transaction in the process and its environment, yielding a fresh transaction k ; this tentative action is recorded in the history. $\text{LTS}\omega$ encodes top-level barbs and $\text{LTS}\star$ records a trivial defender synchronisation move. Weak τ - and k -transitions can always be performed by the defender in the bisimulation game. Moreover, there are no top-level a -transitions because they can always be simulated by a $k(a)$ -transition followed by the commit of k .

Lemma 3.5. Suppose $P \xrightarrow{a}_{\varepsilon} Q$; then $P \xrightarrow{k(a)}_{\varepsilon \rightarrow k} P' \xrightarrow{\text{co } k}_{\sigma} Q$ and $(H \triangleright P) \xrightarrow{k} (H, k(a) \triangleright P') \xrightarrow{\tau} (H, a \triangleright Q)$ for some P' , any H , and any $k \# P, H$. \square

To keep histories and the bisimulation game finite in examples, the challenger of this bisimulation game performs all-but- \star transitions.

Definition 3.6. $C_1 \xrightarrow{\zeta} C_2$ is a challenger move if it is derived without using $\text{LTS}\star$.

We now give the definition for a weak bisimulation over the above transitions.

Definition 3.7 (Weak Bisimulation). A binary relation \mathcal{R} over configurations is a weak bisimulation when for all $C_1 \mathcal{R} C_2$:

$$\begin{aligned}
 \mathcal{R}_3 &\stackrel{\text{def}}{=} \{ ((\varepsilon \triangleright P_3), (\varepsilon \triangleright Q_3)), ((k(a) \triangleright \llbracket b.\text{co} + c.\mathbf{0} \triangleright_k \mathbf{0} \rrbracket), (k(a) \triangleright \llbracket b.\text{co} \triangleright_k \mathbf{0} \rrbracket)), \\
 &\quad ((k(a), k(b) \triangleright \llbracket \text{co} \triangleright_k \mathbf{0} \rrbracket)), (k(a), k(b) \triangleright \llbracket \text{co} \triangleright_k \mathbf{0} \rrbracket)), ((a, b \triangleright \mathbf{0}), (a, b \triangleright \mathbf{0})), \\
 &\quad ((k(a), k(c) \triangleright \llbracket \mathbf{0} \triangleright_k \mathbf{0} \rrbracket)), (\text{ab}, \text{ab} \triangleright \mathbf{0}), ((\text{ab}, \dots \triangleright \mathbf{0}), (\text{ab}, \dots \triangleright \mathbf{0})) \mid \text{any } k \} \\
 \mathcal{R}_2 &\stackrel{\text{def}}{=} \{ ((\varepsilon \triangleright P_2), (\varepsilon \triangleright Q_2)), ((\text{ab}, \dots \triangleright \mathbf{0}), (\text{ab}, \dots \triangleright \mathbf{0})), \\
 &\quad (k_1(a) \triangleright \nu p. \llbracket p.\text{co}.R \triangleright_{k_1} \mathbf{0} \rrbracket \mid \llbracket b.p.\text{co}.S \triangleright_{k_2} \mathbf{0} \rrbracket), (k_1(a) \triangleright \llbracket b.\text{co}.(R \mid S) \triangleright_{k_1} \mathbf{0} \rrbracket)), \\
 &\quad (k_2(b) \triangleright \nu p. \llbracket a.p.\text{co}.R \triangleright_{k_1} \mathbf{0} \rrbracket \mid \llbracket p.\text{co}.S \triangleright_{k_2} \mathbf{0} \rrbracket), (k_2(b) \triangleright \llbracket a.\text{co}.(R \mid S) \triangleright_{k_2} \mathbf{0} \rrbracket)), \\
 &\quad ((\mathbf{k}_1(a), \mathbf{k}_2(b) \triangleright \nu p. \llbracket p.\text{co}.R \triangleright_{k_1} \mathbf{0} \rrbracket \mid \llbracket p.\text{co}.S \triangleright_{k_2} \mathbf{0} \rrbracket), \\
 &\quad (k_2(a), k_2(b) \triangleright \llbracket \text{co}.(R \mid S) \triangleright_{k_2} \mathbf{0} \rrbracket)), \\
 &\quad ((\mathbf{k}_2(b), \mathbf{k}_1(a) \triangleright \nu p. \llbracket p.\text{co}.R \triangleright_{k_1} \mathbf{0} \rrbracket \mid \llbracket p.\text{co}.S \triangleright_{k_2} \mathbf{0} \rrbracket), \\
 &\quad (k_1(b), k_1(a) \triangleright \llbracket \text{co}.(R \mid S) \triangleright_{k_1} \mathbf{0} \rrbracket)), \\
 &\quad ((\mathbf{k}(x), \mathbf{k}(y) \triangleright \nu p. \llbracket \text{co}.R \triangleright_k \mathbf{0} \rrbracket \mid \llbracket \text{co}.S \triangleright_k \mathbf{0} \rrbracket), (k(x), k(y) \triangleright \llbracket \text{co}.(R \mid S) \triangleright_k \mathbf{0} \rrbracket)) \\
 &\quad ((x, y, H \triangleright \nu p. R \mid S), (x, y, H \triangleright R \mid S)) \\
 &\quad \mid \text{any } k, k_1, k_2, R, S, H \text{ and } (x, y) = (a, b) \text{ or } (b, a) \} \\
 \mathcal{R}_5 &\stackrel{\text{def}}{=} \{ ((H \triangleright P_5), (H \triangleright Q_5)), ((H, x \triangleright \mathbf{0}), (H, x \triangleright \mathbf{0})) \\
 &\quad ((H \triangleright \llbracket \tau.b.\text{co} + \tau.c.\text{co} \blacktriangleright P_5 \rrbracket), (H \triangleright \llbracket \tau.(b.\text{co} + c.\text{co}) \blacktriangleright Q_5 \rrbracket)), \\
 &\quad ((H \triangleright \llbracket \tau.b.\text{co} + \tau.c.\text{co} \triangleright_k P_5 \rrbracket), (H \triangleright \llbracket \tau.(b.\text{co} + c.\text{co}) \triangleright_l Q_5 \rrbracket)), \\
 &\quad ((H \triangleright \llbracket b.\text{co} \triangleright_k P_5 \rrbracket), (H \triangleright \llbracket (b.\text{co} + c.\text{co}) \triangleright_l Q_5 \rrbracket)), \\
 &\quad ((H \triangleright \llbracket c.\text{co} \triangleright_k P_5 \rrbracket), (H \triangleright \llbracket (b.\text{co} + c.\text{co}) \triangleright_l Q_5 \rrbracket)), \\
 &\quad ((H, k(x) \triangleright \llbracket \text{co} \triangleright_k P_5 \rrbracket), (H, k(x) \triangleright \llbracket \text{co} \triangleright_k Q_5 \rrbracket)), \\
 &\quad \mid \text{any } k, l, H = (\text{ab}, \dots), \text{ and } x = a \text{ or } b \}
 \end{aligned}$$

Fig. 3. Relations used to prove the equivalences in Ex(s). 3.10 to 3.12.

1. $\text{hist}(\mathcal{C}_1)$ and $\text{hist}(\mathcal{C}_2)$ are consistent,
2. if $\mathcal{C}_1 \xrightarrow{\zeta} \mathcal{C}'_1$ is a challenger move and $\zeta \# \mathcal{C}_2$ then $\exists \mathcal{C}'_2: \mathcal{C}_2 \xrightarrow{\zeta} \mathcal{C}'_2$ and $\mathcal{C}'_1 \mathcal{R} \mathcal{C}'_2$,
3. the converse of the preceding condition.

Condition $\zeta \# \mathcal{C}_2$ guarantees that the choice of fresh transaction names in ζ does not hinder the transition from \mathcal{C}_2 . Weak bisimilarity (\approx) is the largest weak bisimulation, and extends to processes $P \approx Q$ if $(\emptyset \triangleright P) \approx (\emptyset \triangleright Q)$. Bisimulation transitions and weak bisimulations are unaffected by fresh renaming. Thus, the name selected in a challenger move is unimportant.

Lemma 3.8 (ζ -Equivariance). *If $\mathcal{C} \xrightarrow{\zeta} \mathcal{C}'$ then $\mathcal{C}r_{\text{fr}} \xrightarrow{\zeta r_{\text{fr}}} \mathcal{C}'r_{\text{fr}}$.* □

Lemma 3.9 (Equivariance of \approx). *If $\mathcal{C} \approx \mathcal{D}$ then $\mathcal{C} \approx \mathcal{D}r$.* □

We close this section by showing the equivalence of the processes in the introduction by proving them weakly bisimilar. The soundness of our bisimulation technique establishes a proof of contextual equivalence between these processes.

Example 3.10. Recall P_3 and Q_3 from (3) in the Introduction. We show that $P_3 \approx Q_3$; i.e., $(\emptyset \triangleright P_3) \approx (\emptyset \triangleright Q_3)$. It suffices to check that \mathcal{R}_3 in Fig. 3 is a weak bisimulation. Related histories in \mathcal{R}_3 are consistent. The interesting case is when $(k(a) \triangleright \llbracket b.\text{co} + c.\mathbf{0} \triangleright_k \mathbf{0} \rrbracket) \xrightarrow{k'} (k'(a), k'(c) \triangleright \llbracket \mathbf{0} \triangleright_{k'} \mathbf{0} \rrbracket)$. The defender responds:

$(k(a) \triangleright \llbracket b.\text{co} \triangleright_k \mathbf{0} \rrbracket) \xrightarrow{k'}_{\text{LTS}_\star} (k'(a), k'(\star) \triangleright \llbracket b.\text{co} \triangleright_k \mathbf{0} \rrbracket) \xrightarrow{\tau}_{\text{LTSab}} (\mathbf{ab}, \mathbf{ab} \triangleright \mathbf{0})$
 and get $(k'(a), k'(c) \triangleright \llbracket \mathbf{0} \triangleright_{k'} \mathbf{0} \rrbracket) \mathcal{R}_3 (\mathbf{ab}, \mathbf{ab} \triangleright \mathbf{0})$. The rest is trivial, thus the defender always wins, therefore \mathcal{R}_3 is a weak bisimulation and $P_3 \approx Q_3$. \square

Example 3.11. Let us now prove $P_2 \approx Q_2$ from (2) in the Introduction. For this proof we construct relation \mathcal{R}_2 in Fig. 3. It is easy to verify that all histories related in \mathcal{R}_2 are consistent and all challenger moves can be matched by the defender. Here it is noteworthy that the two tentative actions a and b are recorded in the left-hand history under different transaction names (k_1 and k_2 , respectively) until the synchronisation on p merges the two transactions; these history annotations are highlighted in bold typeface. \square

Example 3.12. In our final example proof we show that $P_5 \approx Q_5$ from (5) in the Introduction. Here we construct relation \mathcal{R}_5 in Fig. 3. In this construction, H is a history with zero or more aborted actions; we add this to our configurations because restarting transactions can nondeterministically abort and restart. The proof that \mathcal{R}_5 is a weak bisimulation is again by an easy inspection of the moves of the challenger. The important move is when from the pair $((H \triangleright \llbracket b.\text{co} \triangleright_k P_5 \rrbracket), (H \triangleright \llbracket (b.\text{co} + c.\text{co}) \triangleright_l Q_5 \rrbracket))$ the challenger picks the transition $(H \triangleright \llbracket (b.\text{co} + c.\text{co}) \triangleright_l Q_5 \rrbracket) \xrightarrow{l'}$ $(H, l'(c) \triangleright \llbracket \text{co} \triangleright_{l'} Q_5 \rrbracket)$ and the defender:

$$\begin{array}{ll} ((H \triangleright \llbracket b.\text{co} \triangleright_k P_5 \rrbracket) & \xrightarrow{\tau}_{(\text{LTSab})} (H \triangleright P_5) \\ \xrightarrow{\tau}_{(\text{LTS}_\tau)} (H \triangleright \llbracket \tau.b.\text{co} + \tau.c.\text{co} \blacktriangleright P_5 \rrbracket) & \xrightarrow{\tau}_{(\text{LTS}_{\text{new}})} (H \triangleright \llbracket \tau.b.\text{co} + \tau.c.\text{co} \triangleright_k P_5 \rrbracket) \\ \xrightarrow{\tau}_{(\text{LTS}_{k(\tau)})} (H \triangleright \llbracket c.\text{co} \triangleright_{k'} P_5 \rrbracket) & \xrightarrow{l'}_{(\text{LTS}_{k(a)})} (H, l'(c) \triangleright \llbracket \text{co} \triangleright_{l'} P_5 \rrbracket) \end{array}$$

and get $(H, l'(c) \triangleright \llbracket \text{co} \triangleright_{l'} Q_5 \rrbracket) \mathcal{R}_5 (H, l'(c) \triangleright \llbracket \text{co} \triangleright_{l'} P_5 \rrbracket)$. \square

4 Predictive Bisimulations

In the previous section we showed that weak bisimulations provide an effective verification technique of equivalence. However, it does not enable a direct soundness proof. The difficulty is in proving weak bisimulation *compositional* (i.e., a congruence). Here we define *predictive bisimulations*, an alternative notion of bisimulations that allows us to give an indirect proof of soundness of (\approx) . First we explain the problem with directly proving (\approx) compositional.

Failing Proof (Compositionality) We need to prove \mathcal{R} a weak bisimulation, when

$$(H_1 \triangleright P \mid R) \mathcal{R} (H_2 \triangleright Q \mid R) \quad (7)$$

for any context R and $(H_1 \triangleright P) \approx (H_2 \triangleright Q)$. Let $(H_1 \triangleright P) \xrightarrow{k} (H_1, k(a) \triangleright P')$ and R can perform $k(\bar{a})$ to become R' . We have $(H_1 \triangleright P \mid R) \xrightarrow{\tau} (H_1 \triangleright P' \mid R')$. We need to show that there exist H'_2 , Q'' , and R'' such that $(H_2 \triangleright Q \mid R) \xrightarrow{\tau} (H'_2 \triangleright Q'' \mid R'')$ and $(H_1 \triangleright P' \mid R') \mathcal{R} (H'_2 \triangleright Q'' \mid R'')$. Weak bisimilarity gives

$$(H_2 \triangleright Q) \xrightarrow{k} (H_2, k(\hat{b}) \triangleright Q') \text{ and } (H_1, k(a) \triangleright P') \approx (H_2, k(\hat{b}) \triangleright Q')$$

for some \hat{b} , Q' , and the new parts $k(a)$ and $k(\hat{b})$ of the histories are consistent. However, consistency does not restrict the values of \hat{b} ; it can be a name different than a , or even \star , provided that k does not commit in any extension of the bisimulation game. When $\hat{b} = a$ we can complete the proof by taking $H'_2 = (H_2, k(\hat{a}))$, $Q'' = Q'$ and $R'' = R'$, showing $(H_1, k(a) \triangleright P' \mid R') \mathcal{R} (H_2, k(a) \triangleright Q' \mid R')$ because it is of the same shape as (7). However, when $\hat{b} = c \neq a$ or $\hat{b} = \star$ it is unclear how to proceed in this direct proof. \square

We instead prove soundness by defining (\approx_{prd}) and showing:

$$P \approx Q \text{ implies } P \approx_{\text{prd}} Q \text{ implies } P \cong_{\text{rbe}} Q \quad (8)$$

To show the second implication we need to prove (\approx_{prd}) compositional. The intuition of why this is possible is because (\approx_{prd}) only takes into account those challenger transitions that have the possibility to be committed later in the bisimulation game. This allows us to define a stronger definition of consistency which avoids the problematic cases of the above failed direct proof. Strong consistency is a reflexive, symmetric, and transitive relation.

Definition 4.1 (Strong Consistency (\simeq)). $H_1 \simeq H_2$ when:

$$(H_1(i) = \hat{a} \text{ iff } H_2(i) = \hat{a}) \quad \text{and} \quad (\exists k. H_1(i) = k(\hat{a}) \text{ iff } \exists l. H_2(i) = l(\hat{a}))$$

In a predictive bisimulation game the challenger only performs transitions within transactions when those transactions can commit later in the game. Thus, here we differentiate between τ - and $k(\tau)$ -transitions. Moreover, we emphasise that a challenger $k(a)$ move has to be matched with an identical defender move. Thus predictive bisimulations are over the actions $\eta ::= \tau \mid k(\tau) \mid k(a) \mid \omega$.

Definition 4.2 (Pred. Bisim. Transitions). $\mathcal{C} \xrightarrow{\eta} \mathcal{C}'$ is derived by the rules:

$$\begin{array}{ll} (H \triangleright P) \xrightarrow{\tau} (H \triangleright Q) & \text{if } P \xrightarrow{\tau}_{\varepsilon} Q \quad (\text{LTS}'\tau) \\ (H \triangleright P) \xrightarrow{k(\tau)} (\sigma(H) \triangleright Q) & \text{if } P \xrightarrow{k(\tau)}_{\sigma} Q \text{ and } k \# H \quad (\text{LTS}'k(\tau)) \\ (H \triangleright P) \xrightarrow{\tau} (H \triangleright Q) & \text{if } P \xrightarrow{\text{new } k} Q \text{ and } k \# H \quad (\text{LTS}'\text{new}) \\ (H \triangleright P) \xrightarrow{\tau} (H \setminus_{\text{co}} k \triangleright Q) & \text{if } P \xrightarrow{\text{co } k} Q \quad (\text{LTS}'\text{co}) \\ (H \triangleright P) \xrightarrow{\tau} (H \setminus_{\text{ab}} k \triangleright Q) & \text{if } P \xrightarrow{\text{ab } k} Q \quad (\text{LTS}'\text{ab}) \\ (H \triangleright P) \xrightarrow{k(a)} (\sigma(H), k(a) \triangleright Q) & \text{if } P \xrightarrow{k(a)}_{\sigma} Q \text{ and } k \# H \quad (\text{LTS}'k(a)) \\ (H \triangleright P) \xrightarrow{\omega} (\sigma(H) \triangleright \mathbf{0}) & \text{if } P \xrightarrow{\omega}_{\sigma} Q \quad (\text{LTS}'\omega) \end{array}$$

We define $\xrightarrow{\eta}$ to be $\left((\xrightarrow{\tau}) \cup (\xrightarrow{k(\tau)}) \right)^*$ when $\eta \in \{\tau, k(\tau)\}$, and $\xrightarrow{\tau} \xrightarrow{\eta} \xrightarrow{\tau}$ otherwise.

In a predictive bisimulation game, defender moves are weak η -transitions, with no need for $k(\star)$ -transitions. Challenger moves are *committable* transitions.

Definition 4.3 (Committable Transition). $\mathcal{C} \xrightarrow{\eta} \mathcal{C}'$ with $\eta \in \{\tau, \omega\}$ is *committable*; $\mathcal{C} \xrightarrow{k(\mu)} (H_1 \triangleright P_1)$ is *committable* when there exists $(H_1 \triangleright P_1) \xrightarrow{\eta_1} \dots \xrightarrow{\eta_{(n+1)}} (H_n \triangleright P_n)$ such that for any a and $i \notin \text{dom}(H_1)$:

$$(H_1, (i \mapsto k(a)) \triangleright P_1) \xrightarrow{\eta_1} \dots \xrightarrow{\eta_{(n+1)}} (H_n, (i \mapsto a) \triangleright P_n)$$

Weak predictive bisimulations are thus defined as follows.

Definition 4.4 (Weak Predictive Bisimulation). *A binary relation \mathcal{R} is a weak predictive bisimulation when for all $\mathcal{C}_1 \mathcal{R} \mathcal{C}_2$:*

1. *$\text{hist}(\mathcal{C}_1)$ and $\text{hist}(\mathcal{C}_2)$ are strongly consistent,*
2. *if $\mathcal{C}_1 \xrightarrow{\eta} \mathcal{C}'_1$ is a committable transition and $\text{ftn}(\eta) \# \mathcal{C}_2$ then $\exists \mathcal{C}'_2$ such that $\mathcal{C}_2 \xrightarrow{\eta} \mathcal{C}'_2$ and $\mathcal{C}'_1 \mathcal{R} \mathcal{C}'_2$, and its converse.*

Weak predictive bisimilarity (\approx_{prd}) is the largest such bisimulation and (\approx_{prd}) extends to processes in the same way as (\approx). The first part of (8) follows by:

Theorem 4.5. *Let $\mathcal{C} \approx \mathcal{C}'$ with strongly consistent histories; then $\mathcal{C} \approx_{\text{prd}} \mathcal{C}'$.*

Proof. The proof of this proposition relies on showing that strong consistency is closed under committable transitions of weakly bisimilar configurations. \square

To prove the second part of (8) we need to show that (\approx_{prd}) is compositional.

Theorem 4.6. *If $P \approx_{\text{prd}} Q$ and $\text{ftn}(R) \# P, Q$ then $P \mid R \approx_{\text{prd}} Q \mid R$.*

Proof. This relies on de-/re-composition of actions; e.g., we need to decompose $(H \triangleright P \mid R) \xrightarrow{\eta} (H \triangleright P' \mid R')$ into the constituent sub-actions from P, R with appropriate histories. This is facilitated by strongly consistent histories. \square

5 Full Abstraction

Using (\approx) we can prove soundness of our original bisimulation game.

Theorem 5.1 (Soundness). *If $P \approx Q$ then $P \cong_{\text{rbe}} Q$.*

Proof. In view of Thm. 4.5 it is sufficient to prove the result for (\approx_{prd}). The major step in this proof is already established in Thm. 4.6. \square

We prove completeness for \mathcal{L}_{Act} by first translating any history H into a process $\langle H \rangle$. Then we show that the transitions of a configuration $(H \triangleright P)$ examined by bisimulations can be modelled by reductions of the process $\langle H \rangle \mid P$, when put in parallel with the appropriate contexts. The translation of H is the parallel composition of the translation of each element in H according to:

$$\begin{aligned} \langle i \mapsto k(a) \rangle &= \llbracket \text{co} \mid \omega_{ai}^{\text{commit}} \triangleright_k \omega_i^{\text{abort}} \rrbracket & \langle i \mapsto a \rangle &= \omega_{ai}^{\text{commit}} & \langle i \mapsto \text{ab} \rangle &= \omega_i^{\text{abort}} \\ \langle i \mapsto k(\star) \rangle &= \llbracket \text{co} \triangleright_k \omega_i^{\text{abort}} \rrbracket & \langle i \mapsto \star \rangle &= \mathbf{0} \end{aligned}$$

A tentative $k(a)$ -action, recorded in the history as $(i \mapsto k(a))$, corresponds to a particular move of the bisimulation game—say the i th move. This is translated to a k -transaction which is ready to commit. When k commits because *all* distributed parts of k commit, a unique “success” barb $\omega_{ai}^{\text{commit}}$ becomes observable, signalling that the i th move in the bisimulation game was a synchronisation on a which became permanent. In the history this is recorded as $(i \mapsto a)$. If the k -transaction aborts then a unique ω_i^{abort} barb signals the abortion of the i th

move, corresponding to $(i \mapsto \mathbf{ab})$ in the history. The translation of a defender's $(i \mapsto k(\star))$ move is similar, with the exception that this is a no-action that has no "success" barb associated with it. A key proposition is that reductions of translated configurations model silent bisimulation transitions.

Proposition 5.2. $(H_1 \triangleright P) \xrightarrow{\tau} (H_2 \triangleright Q)$ iff $\langle H_1 \rangle | P \rightarrow \langle H_2 \rangle | Q$.

Moreover, the i th tentative k -transition in the bisimulation game is modelled by the reduction induced by the context:

$$\langle k \rangle^i = \llbracket \mathbf{co} \mid (\sum_{a \in \text{Act}} \bar{a}.\omega_{ai}^{\text{commit}}) + \tau.\mathbf{0} + \omega_i^{\text{before}} \triangleright_k \omega_i^{\text{abort}} \rrbracket$$

When synchronising with a process, this context becomes $\langle i \mapsto k(a) \rangle$ (for any a), modelling a tentative $k(a)$ -transition. It may also spontaneously become $\langle i \mapsto k(\star) \rangle$, modelling a $k(\star)$ -transition. In any case it loses the weak barb ω_i^{before} .

Proposition 5.3. Let $H_2 = \sigma(H_1)$, $(i \mapsto k(\hat{a}))$ and $k' \# k$, H_1, P , and $\text{ftn}(H_1) \subseteq \text{ftn}(P)$; then $(H_1 \triangleright P) \xrightarrow{k} (H_2 \triangleright Q)$ iff $\langle H_1 \rangle | P \mid \langle k' \rangle^i \rightarrow \langle H_2 \rangle | Q \not\Downarrow \omega_i^{\text{before}}$. \square

Theorem 5.4 (Completeness). If $P, Q \in \mathcal{L}_{\text{Act}}$ and $P \cong_{\text{rbe}} Q$ then $P \approx Q$.

Proof. Using the above propositions we show \mathcal{X} is a weak bisimulation, where $\mathcal{X} = \{((H \triangleright P), (H' \triangleright Q)) \mid H, H' \text{ cons.}, P, Q \in \mathcal{L}_{\text{Act}}, \langle H \rangle | P \cong_{\text{rbe}} \langle H' \rangle | Q\}$. \square

6 Conclusions

We presented a weak bisimulation theory for $TCCS^m$, a simple language with communicating transactions. In $TCCS^m$, two transactions that communicate are conjoined by being renamed to the same name forming a *distributed* version of cJoin's merged transactions [1]. When a transaction communicates with a non-transactional process, the latter is embedded in the former in line with the semantics of previous work [3]. Compared to that semantics, embedding and merging in $TCCS^m$ is communication-driven limiting nondeterminism. For simplicity this language has only single-level transactions, although we believe that the results of this paper can be adapted to a language with nested transactions.

The bisimulation equivalence is sound and complete with respect to a natural contextual equivalence which equates transactions that differ only in uncommitable actions; these are destined to eventually be rolled back. We motivated this with examples in the Introduction which we verified in Sect. 3 using our technique. In related work about reversible calculi [10,9], contextual equivalence can discriminate between transactions $\llbracket b + a.b.\mathbf{co} \triangleright_k \mathbf{0} \rrbracket$ and $\llbracket a.b.\mathbf{co} \triangleright_k \mathbf{0} \rrbracket$ by virtue of the fact that the former contains an extra uncommitable b -action.

Our bisimulations provide an effective verification technique for the aforementioned contextual equivalence which can be applied to related programming languages where uncommitable actions have no effect [5,7,11,16]. They can also serve as a verification technique for other forms of contextual equivalences, such as may- and must-testing equivalences [3,4]. Other bisimulation methods for

reversible computation [2,8,9,10] may also be used in these settings, but they are more fine-grained, discriminating even between the above two processes. Forward-reverse and hereditary history-preserving bisimulations [13] differentiate between forward and reverse transitions, which would discriminate between the processes P_5 and Q_5 shown in the Introduction capturing a possible compiler optimisation. To our knowledge, the bisimulation technique presented here is the only one that can be used to prove the correctness of such compiler optimisations.

References

1. Bruni, R., Melgratti, H.C., Montanari, U.: Nested commits for mobile calculi: extending Join. In: Levy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) TCS 2004. IFIP, vol. 155, pp. 563–576. Springer, Boston (2004)
2. Danos, V., Krivine, J.: Transactions in RCCS. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 398–412. Springer, Heidelberg (2005)
3. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 569–583. Springer, Heidelberg (2010)
4. de Vries, E., Koutavas, V., Hennessy, M.: Liveness of communicating transactions (Extended abstract). In: Ueda, K. (ed.) APLAS 2010. LNCS, vol. 6461, pp. 392–407. Springer, Heidelberg (2010)
5. Donnelly, K., Fluet, M.: Transactional events. In: ICFP, pp. 124–135. ACM (2006)
6. Effinger-Dean, L., Kehrt, M., Grossman, D.: Transactional events for ML. In: ICFP, pp. 103–114. ACM (2008)
7. Field, J., Varela, C.A.: Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments. In: POPL, pp. 195–208. ACM (2005)
8. Krivine, J.: A verification technique for reversible process algebra. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 204–217. Springer, Heidelberg (2013)
9. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Concurrent flexible reversibility. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 370–390. Springer, Heidelberg (2013)
10. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Reversing higher-order pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 478–493. Springer, Heidelberg (2010)
11. Lesani, M., Palsberg, J.: Communicating memory transactions. In: PPOPP, pp. 157–168. ACM (2011)
12. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
13. Phillips, I., Ulidowski, I.: Reversibility and models for concurrency. ENTCS 192(1), 93–108 (2007) (SOS 2007)
14. Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. In: LICS, pp. 293–302. IEEE Computer Society (2007)
15. Sangiorgi, D., Walker, D.: The π -calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
16. Spaccasassi, C., Koutavas, V.: Towards efficient abstractions for concurrent consensus. In: McCarthy, J. (ed.) TFP 2013. LNCS, vol. 8322, pp. 76–90. Springer, Heidelberg (2014)
17. Stirling, C.: Playing games and proving properties of concurrent systems. J. Comput. Sci. Technol. 13(6), 482 (1998)

Upper-Expectation Bisimilarity and Łukasiewicz μ -Calculus

Matteo Mio

Computer Laboratory, University of Cambridge, UK

Abstract. Several notions of bisimulation relations for probabilistic nondeterministic transition systems have been considered in the literature. We study a novel testing-based behavioral equivalence called *upper expectation bisimilarity* and, using standard results from functional analysis, we develop its coalgebraic and algebraic theory and provide a logical characterization in terms of an expressive probabilistic modal μ -calculus.

1 Introduction

Directed-graph structures are sufficient for modeling nondeterministic programs and concurrent systems but cannot directly represent other important aspects of computation, such as *probabilistic behavior*, *timed transitions* and other *quantitative* information one might need to express. *Probabilistic nondeterministic transition systems* (PNTS), also known in the literature as (simple) Segala systems [2], concurrent Markov chains and probabilistic automata [29], have been identified in the last two decades as convenient mathematical structures, generalizing standard nondeterministic transition systems (NTS), to provide operational semantics to probabilistic nondeterministic languages (see, e.g., [2,30]).

A central concept in the theory of programming languages and concurrent systems is the notion of *behavioral equivalence*. An equivalence relation $\simeq \subseteq S \times S$ between states of a system is, informally speaking, a behavioral equivalence if $s \simeq t$ implies that s and t satisfy the same class of properties of interest. Of course different classes of properties induce different notions of equivalence. The paradigmatic example of behavioral equivalence for ordinary NTS's is Milner and Park's *bisimilarity* [21]. Among its good properties, bisimilarity enjoys the following: **B1**) two states are bisimilar if and only if they satisfy the same properties expressed, e.g., in Kozen's modal μ -calculus [17] or in other (weaker but useful in practice) branching-time logics such as CTL, CTL* [1] and the basic modal logic **K** or its labeled version, the Hennessy–Milner modal logic. Furthermore, **B2**) bisimilarity is a congruence for a wide family of process algebras, all specified following one of the many rule formats (e.g., GSOS, tyft/tyxt, *etc*) and **B3**) enjoys a rich mathematical theory based on coinduction, playing a role of paramount importance in *coalgebra* [28]. Lastly, but importantly, **B4**) bisimilarity can also be explained in terms of Milner's standard metaphor of *push-button experiments* on systems [21]. Such experiments provide an abstract, yet intuitive, testing semantics for bisimilarity.

In the context of PNTS's several notions of behavioral equivalence, based on the technical machinery of coinduction, have been considered in the literature [29,20,31,3,9,10]. The one which has attracted most attention so far was introduced by Segala in [29] and is referred to in this paper as *standard bisimilarity*. This is a mathematically natural notion (cf. B3) and, indeed, it has been rediscovered using the methods of coalgebra [30]. In [29] another behavioral equivalence (strictly coarser than standard bisimilarity), which we refer to as *convex bisimilarity*, is introduced by Segala as a preferable notion because, as we shall discuss in Section 2, it is strongly motivated by the concept of *probabilistic scheduler*. Importantly, both equivalences are congruence relations (cf. B2) for the wide class of PGSOS process algebras, which includes virtually all Milner's CCS-style (probabilistic) process operators of practical interest [2].

Following a traditional approach based on interpreting formulas as *sets* of states (or probability distributions over states [10]), several Hennessy-Milner-style modal logics, capable of characterizing either standard or convex bisimilarity have been investigated (see, e.g., [29,20,10,11]). However all these logics (even when enriched with fixed-point operators as in [10]) lack the expressive power required to formulate many natural and practically useful properties of PNTS's such as, e.g., those expressible in the popular temporal logics PCTL and PCTL* of [4]. Unlike their non-probabilistic counterparts, CTL and CTL* [1], these two logics induce non-standard and different behavioral equivalences [31]:

$$\text{standard bisimilarity} \subsetneq \text{convex bisimilarity} \subsetneq \text{PCTL}^* \subsetneq \text{PTCL}$$

Despite significant efforts, the field still lacks a satisfactory logical framework (cf. B1) for expressing practically useful properties of systems and, at the same time, reasoning about behavior up-to satisfactory notions of behavioral equivalence. The problem is further complicated by the lack of a (widely accepted) testing semantics (cf. B4) for either standard bisimilarity, convex bisimilarity or any of the other proposed equivalences studied in the literature (e.g., [11,6,3]).

Starting from the late 90's, following an alternative *quantitative* approach based on interpreting formulas as maps $[[\phi]] : S \rightarrow \mathbb{R}$ from states to real values, fixed-point modal logics (quantitative μ -calculi) for PNTS's have been studied (see, e.g., [15,9,26]). The present paper contributes to a programme of ongoing research, one of whose overall aims is to investigate the extent to which quantitative μ -calculi play as fundamental a rôle in the probabilistic-nondeterministic setting as that of Kozen's μ -calculus in the nondeterministic setting.

Following the quantitative approach, the author has recently introduced in [22,24] the first fixed-point based logic, called Łukasiewicz μ -calculus ($\mathbb{L}\mu$), capable of encoding PCTL. It has been shown in [22,23] how this (and similar) logic can be given a *game semantics* generalizing the well-known parity-game semantics of Kozen's μ -calculus. Compositional verification techniques have been developed in [25], exploiting the elegant fixed-point semantics. Most recently, model checking algorithms have been investigated in [24].

Contributions. This paper adds to the mounting evidence that the \mathbb{R} -valued approach to temporal logics for PNTS's is a convenient framework by showing that

$L\mu$ captures what is arguably the preferred notion of bisimilarity for PNTS's: convex bisimilarity. A strong novelty of our approach is that to obtain this result one needs to take a fresh perspective on convex bisimilarity. To this end, a nonstandard behavioral equivalence called *upper expectation (UE) bisimilarity*, which naturally arises from a very abstract testing scenario based on \mathbb{R} -valued experiments (cf. property **B4**), is introduced. Unlike other related works, our experiments are not given by, e.g., the formulas of a given logic (such as the μ -calculus of [9]) nor by terms of some process algebra (see, e.g., [11]) but, instead, are modeled abstractly by functions $f : S \rightarrow \mathbb{R}$ from program states to real values.

We prove that UE-bisimilarity coincides with the behavioral equivalence induced by the expressive logic $L\mu$ (cf. property **B1**). Furthermore, we show that this logic characterizes the so-called *behavioral (Hausdorff) metric*, a concept of fundamental importance in the theory of approximation of probabilistic systems pioneered by Panangaden [27]. Thus the \mathbb{R} -valued approach to logics for PNTS's may be considered equally suitable as a mechanism for characterizing process equivalence as other non-quantitative logics advocated for this specific purpose.

As a main result we prove that, while UE-bisimilarity is generally coarser than convex bisimilarity, the two notions coincide on a wide class of systems:

$$\text{convex bisimilarity} = \text{UE-bisimilarity} = \text{\u0141ukasiewicz } \mu\text{-Calculus}$$

As a matter of fact, we argue that UE-bisimilarity constitute a natural relaxation of convex bisimilarity based on elementary topological considerations. Indeed, a very interesting feature of our work is that results from linear algebra and functional analysis are crucial to link UE-bisimilarity with the topological notions of *closedness*, *compactness* and *convexity*. This, in turn, allows the identification of UE-bisimilarity with the *coalgebraic* notion of cocongruence of the appropriate functor type (cf. property **B3**) and to relate it with Segala's convex bisimilarity.

In Section 5 we present two algebraic results about UE-bisimilarity: a sound and complete equational characterization of the behavior of PNTS's, obtained once again by applying a known representation theorem from functional analysis, and a congruence result with respect to the CCS-style process-algebra operators definable in the PGSOS format of [2] (cf. property **B2**).

Collectively our results provide strong mathematical foundations for UE-bisimilarity comparable with those of Milner and Park's bisimilarity and shed light on the mathematical nature of PNTS's and on the quantitative approach to temporal logics for PNTS's.

2 Coalgebra and PNTS's

We employ the basic language of coalgebra in the description of systems and behavioral equivalences. We refer to [28] for a gentle introduction to the subject.

Definition 1. *Let \mathbf{Set} be the category of sets and functions between them. The endofunctor \mathcal{P} (powerset) on \mathbf{Set} is defined as: $\mathcal{P}(X) = \{A \mid A \subseteq X\}$ and $(\mathcal{P}(f))(A) = f[A] = \{f(x) \mid x \in A\}$, for all sets X, Y and functions $f : X \rightarrow Y$.*

Definition 2. A nondeterministic transition system (NTS) is a F -coalgebra $(X, \alpha : X \rightarrow F(X))$ of the functor $F = \mathcal{P}$.

In applications one most often encounters *labeled* NTS's, or *Kripke structures*, i.e., NTS's endowed with a set P of propositional letters $p \in P$ interpreted as predicates. For the sake of simplicity we just consider plain NTS's and their probabilistic generalizations. The results we develop extend straightforwardly to the labeled and propositional extensions.

Coalgebra provides an abstract notion of behavioral equivalence of F -coalgebras called *cocongruence* which is based on the idea of lifting relations on the set X of states to relations on $F(X)$ [18]. The notion of cocongruence for the functor \mathcal{P} coincides with ordinary Milner and Park's bisimulation for NTS's [18].

Definition 3 ([30]). The endofunctor \mathcal{D} (discrete probability distributions) on **Set** is defined as: $\mathcal{D}(X) = \{\mu : X \rightarrow [0, 1] \mid \sum_x \mu(x) = 1\}$, and $(\mathcal{D}(f))(\mu) = f[\mu] = y \mapsto \mu(f^{-1}(y))$ where, for all sets $S \subseteq X$, we define $\mu(S) = \sum_{x \in S} \mu(x)$.

Note that the composite functor $\mathcal{P}\mathcal{D}$ maps a set X to the collection of all sets of discrete probability distributions on X .

Definition 4. A probabilistic nondeterministic transition system (PNTS) is a $\mathcal{P}\mathcal{D}$ -coalgebra $(X, \alpha : X \rightarrow \mathcal{P}\mathcal{D}(X))$. We write $x \rightarrow \mu$ to specify that $\mu \in \alpha(x)$.

The intended interpretation is that the system, at some state $x \in X$, can evolve by nondeterministically choosing one of the *accessible* distributions $\mu \in \alpha(x)$ and then continuing its execution from the state $y \in X$ with probability $\mu(y)$. PNTS's can be visualized, using graphs labeled with probabilities. For example the PNTS (X, α) having set of states $X = \{x, x_1, x_2\}$ and transition map $\alpha(x) = \{\mu_1, \mu_2\}$ and $\alpha(x_1) = \alpha(x_2) = \emptyset$, with $\mu_1(x_1) = \mu_2(x_2) = 0.2$ and $\mu_1(x_2) = \mu_2(x_1) = 0.8$, can be depicted as in Figure 1. The combination of nondeterministic choices immediately followed by probabilistic ones, allows the modeling of concurrent probabilistic programming languages in a natural way [2].

PNTS's with a definition equivalent to the coalgebraic one given above were introduced by Segala [29] who also defined two notions of bisimilarity for PNTS's. We refer to *standard bisimilarity* as the the stronger (i.e., finer) of the two:

Definition 5 (Standard Bisimulation). Given a PNTS (X, α) , a standard bisimulation is an equivalence relation $E \subseteq X \times X$ such that if $(x, y) \in E$ then

- if $x \rightarrow \mu$ then there exists ν such that $y \rightarrow \nu$ and $(\mu, \nu) \in \hat{E}$, and
- if $y \rightarrow \nu$ then there exists μ such that $x \rightarrow \mu$ and $(\mu, \nu) \in \hat{E}$,

where $(\mu, \nu) \in \hat{E}$ holds if $\mu(A) = \nu(A)$ (see Definition 3) for all sets $A \subseteq X$ which are unions of E -equivalence classes.

The definition looks technical but has a simple interpretation. Two states x, y of a PNTS (X, α) are standard-bisimilar if the two sets of reachable distributions $\alpha(x)$ and $\alpha(y)$ are equal modulo E , i.e., if $\{[\mu]_{\hat{E}} \mid \mu \in \alpha(x)\} = \{[\mu]_{\hat{E}} \mid \mu \in \alpha(y)\}$, where $(\mu, \nu) \in \hat{E}$ means that if one identifies E -related states, then μ and ν

become *equal*, i.e., they assign the same probabilities to all events $A \subseteq X/E$ (i.e., unions of equivalence classes). Using this argument one can show that the notion of standard bisimilarity coincides with that of cocongruence of \mathcal{PD} -coalgebras [30]. As an example, consider the two PNTS's rooted at x and y respectively, depicted in Figure 1, and assume the processes x_1 and x_2 to be observationally different¹. The processes x and y are not standard bisimilar. This is because y can lead to a probability distribution μ_3 which cannot be matched by x .

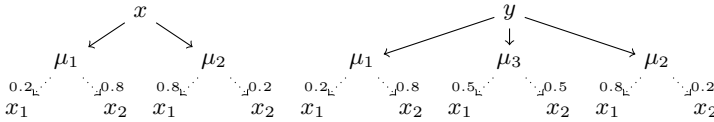


Fig. 1. Example of states (x, y) not standard bisimilar

It has been argued by Segala (see, e.g., [29]) that standard bisimilarity is too strict a behavioral equivalence when PNTS's are used to model nondeterministic probabilistic programs/systems. In this setting, the nondeterminism in the system is supposed to model all the possible choices which can be made by, e.g., an external *scheduler*. It is natural, however, to assume that schedulers can themselves use probabilistic methods to perform their choices. Thus, a *probabilistic scheduler* could choose to pick, from the state x , the successor distributions μ_1 and μ_2 with equal probability $\frac{1}{2}$, and consequently reach states x_1 and x_2 with equal probabilities $\frac{1}{2}(0.2 + 0.8) = 0.5$. Thus a scheduler, by choosing probabilistically between μ_1 and μ_2 , can mimic the choice of $\mu_3 = \frac{1}{2}\mu_1 + \frac{1}{2}\mu_2$.

Definition 6. Let X be a set. A convex combination of elements of $\mathcal{D}(X)$ is a distribution ν of the form $\nu(x) = \sum_{i=1}^n \lambda_i \cdot \mu_i(x)$, for $\mu_i \in \mathcal{D}(X)$, $\lambda_i \in [0, 1]$ and $\sum_i \lambda_i = 1$. The convex hull of $A \in \mathcal{PD}(X)$, denoted by $H(A)$, is the collection of all convex combinations of elements in A . The set A is convex if $A = H(A)$.

We are now ready to introduce the second notion of bisimilarity introduced by Segala [29,20], which we refer to² as *convex bisimilarity*.

Definition 7 (Convex Bisimulation). Given a PNTS (X, α) , a convex bisimulation is an equivalence relation $E \subseteq X \times X$ such that if $(x, y) \in E$ then

- if $x \rightarrow_C \mu$ then there exists ν such that $y \rightarrow_C \nu$ and $(\mu, \nu) \in \hat{E}$, and
- if $y \rightarrow_C \nu$ then there exists μ such that $x \rightarrow_C \mu$ and $(\mu, \nu) \in \hat{E}$,

where $x \rightarrow_C \mu$ holds if and only if $\mu \in H(\alpha(x))$.

¹ By this we mean that $(x_1, x_2) \notin E$ for all bisimulations E . Of course this can be implemented by adding structure to the system (e.g., a single edge from x_1 to μ_1). Our assumption thus simply abstracts away from such additional details.

² The adjective *probabilistic* is often adopted in the literature [29,31]. We prefer the adjective *convex* which transparently reflects the mathematics behind the notion.

Thus convex bisimilarity is obtained by replacing \rightarrow with \rightarrow_C in Definition 5. Probabilistic schedulers take the place of ordinary schedulers and can simulate any convex combination of the reachable probability distributions.

Remark 1. Note that Definition 7 remains unaltered if \rightarrow_C is replaced by \rightarrow (as originally proposed by Segala in [29]) in the left side of the two conditions.

If probabilistic schedulers are assumed, Example 1 shows how, as observed by Segala, standard-bisimilarity is too strict because it distinguishes between states that, under the intended interpretation, ought to be identified. This fact does not mean that cocongruence (i.e., standard bisimilarity) is not the “right” notion of behavioral equivalence for \mathcal{PD} -coalgebras. Rather, it suggests that \mathcal{PD} -coalgebras do not precisely model the class of systems we have in mind.

Definition 8. *The endofunctor $\mathcal{P}_c\mathcal{D}$ (convex sets of probability distributions) on **Set** is defined as follows: $\mathcal{P}_c\mathcal{D}(X) = \{A \mid A \in \mathcal{PD}(X) \text{ and } A = H(A)\}$ and $(\mathcal{D}(f))(A) = f[A] = \{f[\mu] \mid \mu \in A\}$, where $f[\mu]$ is defined as in Definition 3.*

Remark 2. The functoriality of $\mathcal{P}_c\mathcal{D}$ is well known. In fact more is true, and $\mathcal{P}_c\mathcal{D}$ carries a monad structure [32]. This property has been extensively studied, especially in the field of domain theory (see, e.g., [32,13]), and recognized as important in the setting of probabilistic-nondeterminism.

It is clear that a $\mathcal{P}_c\mathcal{D}$ -coalgebra (X, α) is just a particular kind of \mathcal{PD} -coalgebra such that, for all $x \in X$, the set $\alpha(x)$ is convex. Formally (see, e.g., [30] for an introduction to this concept) $\text{id} : \mathcal{PD} \rightarrow \mathcal{PD}$ is an injective natural transformation. Furthermore the convex hull operation gives us a natural way to convert a \mathcal{PD} -coalgebra into a $\mathcal{P}_c\mathcal{D}$ -coalgebra (formally, $H : \mathcal{PD} \rightarrow \mathcal{P}_c\mathcal{D}$ is a surjective natural transformation). Transforming a \mathcal{PD} -coalgebra (i.e., a PNTS) (X, α) into the $\mathcal{P}_c\mathcal{D}$ -coalgebra $(X, H_X \circ \alpha)$ precisely corresponds to the substitution of the arrow relation \rightarrow in Definition 5 with the relation \rightarrow_C in Definition 7. It is now straightforward to verify that an equivalence relation $E \subseteq X \times X$ is a convex bisimulation in the PNTS (X, α) if and only if E is a cocongruence (i.e., standard bisimilarity) in the $\mathcal{P}_c\mathcal{D}$ -coalgebra $(X, H_X \circ \alpha)$. Thus convex bisimilarity can be seen as standard bisimilarity *modulo* (H_X is surjective) the *behavioral equation* $A \approx B$ whenever $H(A) = H(B)$, for all $A, B \in \mathcal{PD}(X)$, capturing the the behavior of probabilistic schedulers.

Remark 3. Although $\mathcal{P}_c\mathcal{D}$ -coalgebras are the models naturally corresponding to convex bisimilarity, we can always work, concretely, with ordinary PNTS’s (i.e., \mathcal{PD} -coalgebras) (X, α) , perhaps represented as finite graphs, and tacitly replace α with $H_X \circ \alpha$ (i.e., \rightarrow with \rightarrow_C). This is convenient since, generally, the convex hull of a finite set is uncountable.

The discussion carried out in this section serves to clarify that no *a priori* categorical or coalgebraic argument exists supporting a notion of behavioral equivalence in favor of another, when modeling computing systems. Coalgebra provides, e.g., the mathematically deep notion of cocongruence for a functor, but the choice of an appropriate functor is part of the modeling process.

3 Upper Expectation Bisimilarity

We saw how convex bisimilarity naturally arises from the observation that schedulers may make probabilistic choices. We also discussed how it can be understood coalgebraically in terms of cocongruences of $\mathcal{P}_c\mathcal{D}$ -coalgebras. However it is not possible to claim, on the sole basis of these facts, that convex bisimilarity is a convenient notion of behavioral equivalence for PNTS's. Probabilistic schedulers constitute a good reason to consider two convex bisimilar states as behaviorally equivalent. But it is not clear why one might want to distinguish between two states that are not convex bisimilar. We illustrate the problem by means of the simple example of Figure 2. As usual we assume the three states x_1, x_2 and x_3 to be observationally distinct (cf. Footnote 1). The two states x and y are not convex bisimilar because μ_3 is not a convex combination of μ_1 and μ_2 . It is not

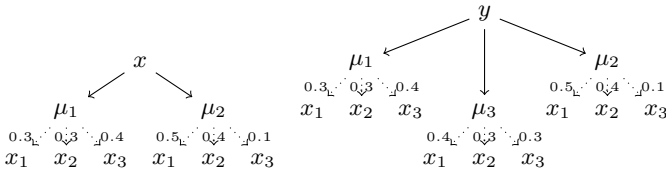


Fig. 2. Example of states (x, y) not convex bisimilar

simple, however, to find a *concrete*³ reason to distinguish between the two states. As a matter of fact, one can prove (see, e.g., [31]) that the states x and y satisfy the same properties formulated in the expressive logics PCTL and PCTL* of [4].

Remark 4. While modal logics, carefully crafted to capture convex bisimilarity (and even standard bisimilarity) can be defined [20,29,10,11], it is certainly interesting to look at the distinguishing power of popular temporal logics for PNTS's, (of which PCTL and PCTL* are main examples) capable of expressing branching properties of probabilistic concurrent systems useful in practice.

We now introduce a simple, yet realistic, experimental scenario which allows one to distinguish the two states x and y of Figure 2. Suppose we are allowed to make repeated experiments (in the sense of Milner's push-button metaphor [21]) on the PNTS's of Figure 1. After a sufficiently large number ($n \rightarrow \infty$) of experiments at, e.g., the state x , we observe that an event S (e.g., $S = \{x_1\}$, representing the occurrence of terminal state x_1) happened with *upper probability*⁴ $\frac{m}{n}$. We can then make the following reasonable assessment: the PNTS

³ This deliberately vague adjective could be seconded by, e.g., *experimental, testing-based, etc.*

⁴ Our scenario is of course based on the common frequentist interpretation of probabilities as limits of relative frequencies in a large number of trials. Technically, due to the nondeterminism involved, a (unique) limit may not exist. Thus, what is actually experimentally observed is that the sequence of relative frequencies eventually settles below an upper value (supremum limit) λ .

at state x can exhibit behavior S (i.e., end up in x_1) with *at least* probability $\frac{m}{n}$. It seems then natural to stipulate that two states x and y are equivalent if, for every event S , the state x can exhibit behavior (event) S with maximum probability λ if and only if y can. It is simple to verify (the relevant events S to be considered are the subsets of $\{x_1, x_2, x_3\}$) that the two states x and y of Figure 2 are equivalent in this sense.

However it is also realistic to assume that in the experiments carried on PNTS's, one is not just allowed to observe the occurrences of events and thus, after a sufficient number of experiments estimate their upper probabilities by means of relative frequencies. Rather, one is allowed to associate a real valued information $r_i \in \mathbb{R}$ to the outcome of each experiment i and, after n experiments, observe the average value $\frac{1}{n} \sum_{i=0}^n r_i$. This enhanced scenario is better explained by a simple example. Consider again the states x and y of Figure 2 and consider the function $g: \{x_1, x_2, x_3\} \rightarrow \mathbb{R}$ defined as $g(x_1)=0.6$, $g(x_2)=0$ and $g(x_3)=0.5$. The function g represents the experiment in the sense that if, after letting the scheduler choose a transition (i.e., pushing the button, in Milner's metaphor) the state x_i is reached, then the real number $g(x_i)$ is registered as result. Thus, for instance, if $\{x_1, x_2, x_3, x_2, x_1\}$ was the outcome of five experiments, the numerical sequence $\{0.6, 0.5, 0, 0, 0.6\}$ and its average $0.34 = \frac{1.70}{5}$ would be our observation. Note that the simpler observation of an event S can be modeled by the experiment $\chi_S : X \rightarrow \{0, 1\}$, i.e., by the characteristic function of S .

Definition 9. Let X be a set, $\mu \in \mathcal{D}(X)$ and $f : X \rightarrow \mathbb{R}$. The expected value of f under μ , written $E_\mu(f)$, is defined as $E_\mu(f) = \sum_x \mu(x)f(x)$.

The expected values of g under μ_1 , μ_2 and μ_3 are 0.38, 0.35 and 0.39, respectively, and this readily means that, for a sufficiently large number of trials, the average resulting from experiments g on state x is *necessarily* smaller or equal than 0.38, while in state y it *can be* strictly greater than 0.38 (and at most 0.39). Thus, it is *possible* that an agent, by means of a sufficiently large number of repeated experiments g , *may* be able to distinguish between the two states x and y . This discussion leads to the following definitions.

Definition 10. Let X be a set and $A \in \mathcal{PD}(X)$ a set of probability distributions on X . The upper expectation functional $ue_A : (X \rightarrow \mathbb{R}) \rightarrow \mathbb{R}$, mapping functions $X \rightarrow \mathbb{R}$ to real numbers, is defined as $ue_A(f) = \bigsqcup_{\mu \in A} E_\mu(f)$.

Thus, $ue_A(f)$ represents the maximum (supremum limit) expected value of f which *may* be achieved when choosing probability distributions in A .

Definition 11. Given a PNTS (X, α) , an upper expectation (UE) bisimulation is an equivalence relation $E \subseteq X \times X$ such that if $(x, y) \in E$ then the equality $ue_{\alpha(x)}(f) = ue_{\alpha(y)}(f)$ holds for all E -invariant functions $f : X \rightarrow \mathbb{R}$, i.e., such that if $(z, w) \in E$ then $f(z) = f(w)$.

We restrict to E -invariant experiments f following the idea that E -related (i.e., UE-bisimilar) states ought to be identified. The choice of considering upper expectations seems one-sided, but one could equally well choose lower expectations (le_A) observing that $le_A(f) = 1 - ue_A(1 - f)$. This is an instance of the

may/must duality well known in classical bisimulation theory, modal logic and concurrency theory [21]. It follows from our discussion that restricting Definition 11 to $\{0, 1\}$ -valued functions $g: X \rightarrow \{0, 1\}$ would weaken the definition to that of an equivalence relation not capable of, e.g., distinguishing the states x and y of Figure 2. This remark reveals that \mathbb{R} -valued experiments ($f: X \rightarrow \mathbb{R}$) generally provide more information than just observation of events ($g: X \rightarrow \{0, 1\}$).

The definition of UE-bisimulation gives reasons for distinguishing states based on the existence of a witnessing experiment f which *may* have an expected outcome in one state which *cannot* be matched by the other state.

3.1 Relation between Convex and UE-Bisimilarity

UE-bisimilarity arises naturally from the simple testing scenario discussed above and, as we shall discuss in Section 4, enjoys a remarkable natural connection with real-valued modal logics. It is thus worth to develop its theory and compare it with that of convex bisimilarity. In this section we show that the two notions coincide on a wide class of systems by means of an alternative characterization of UE-bisimilarity. Recall that $\mathcal{P}_c\mathcal{D}$ -coalgebras (cf. Remark 3) can be thought of as $\mathcal{P}\mathcal{D}$ -coalgebras modulo the behavioral equation $A \approx B$ if $H(A) = H(B)$, for $A, B \in \mathcal{P}\mathcal{D}(X)$. Following the same idea, to understand UE-bisimilarity coalgebraically one needs to consider the behavioral equation $A \approx B$ if $ue_A = ue_B$ (pointwise equality). As it turns out, the two equations coincide under very mild conditions. In rest of the paper we restrict attention to a fairly simple (yet important) class of PNTS's, as this greatly simplifies our discussion. In Remark 5 we briefly outline how our results can be generalized.

Convention 1. *We restrict attention to PNTS's having a finite state space, i.e., $\mathcal{P}\mathcal{D}$ -coalgebras $(X, \alpha: X \rightarrow \mathcal{P}\mathcal{D}(X))$ having a finite carrier set $X = \{x_1, \dots, x_n\}$ (endowed with the discrete topology). This allows one to view the space of functions $X \rightarrow \mathbb{R}$ as the Euclidean space \mathbb{R}^n and each $\mu \in \mathcal{D}(X)$ as the n -dimensional vector $\mu = [\mu(x_1), \dots, \mu(x_n)]$. Note that $\mathcal{D}(X)$ is a closed and bounded (hence compact [16]) subset of \mathbb{R}^n and that $\alpha(x)$, for $x \in X$, can be infinite.*

The following is a known result from functional analysis [14, §10.2] and optimization theory which generalizes to the setting described in Remark 5.

Theorem 1. *Let X be a finite set and $A, B \in \mathcal{P}\mathcal{D}(X)$. Then $ue_A = ue_B$ iff $cl(H(A)) = cl(H(B))$, where $cl(C)$ denotes the topological closure of the set C .*

It is a standard result in linear algebra that the closure of a convex set is itself convex (see, e.g., [19, §8.4]). For this reason the set $cl(H(A))$ is called the *closed convex hull* of A . In what follows we denote with \bar{H} the operation $A \mapsto cl(H(A))$.

The result of Theorem 1 can be used to prove the following alternative characterization of UE-bisimulation (cf. Definition 7).

Theorem 2. *Given a PNTS (X, α) , an equivalence relation $E \subseteq X \times X$ is a UE-bisimulation iff for all $(x, y) \in E$, it holds that:*

- if $x \rightarrow_{CC} \mu$ then there exists ν such that $y \rightarrow_{CC} \nu$ and $(\mu, \nu) \in \hat{E}$, and
- if $y \rightarrow_{CC} \nu$ then there exists μ such that $x \rightarrow_{CC} \mu$ and $(\mu, \nu) \in \hat{E}$,

where $x \rightarrow_{CC} \mu$ if and only if $\mu \in \overline{H}(\alpha(x))$.

Thus UE-bisimilarity can be obtained by replacing \rightarrow_C with \rightarrow_{CC} in Definition 7 of convex bisimilarity. As a consequence UE-bisimilarity can be strictly coarser than convex bisimilarity. The following proposition, however, reveals that the two notions coincide for a wide class of PNTS's.

Proposition 1. *Let (X, α) be a PNTS such that $\alpha(x)$ is closed for all $x \in X$. Then $E \subseteq X \times X$ is a convex bisimulation if and only if it is a UE-bisimulation.*

Restricting to PNTS's of this kind can hardly be seen as a limitation in concrete applications. Every finite set is closed, thus every PNTS representable as a finite graph satisfies Convention 1 and the closedness condition of Proposition 1. Furthermore, the restriction to closed sets seems natural as agrees with the well established *motto* “observable properties are open sets” suggesting that only those sets A and B which can be separated by open sets should be distinguished.

Remark 5. Among the possible ways of relaxing the constraint of Convention 1, a very general (and mathematically convenient) approach consists in modeling PNTS's as \mathcal{PD} -coalgebras in the category of compact Hausdorff topological spaces. Here \mathcal{D} maps the space X to the space of probability measures on X , endowed with the weak*-topology, and \mathcal{P} maps X to the space of its closed (hence compact⁵) subsets, endowed with the Vietoris topology [16]. Experiments on X are now modeled by the Banach space $C(X)$ of continuous functions $X \rightarrow \mathbb{R}$ [19]. Due to the lack of space, we just mention in this remark that it possible to define the functor $\mathcal{P}_{cc}\mathcal{D}$ mapping X to its set of convex closed subsets and show that UE-bisimilarity coincides with the notion of cocongruence of $\mathcal{P}_{cc}\mathcal{D}$ -coalgebras. All the results presented in this paper generalize to this setting at the cost of mathematical complications needed to deal with infinite dimensional spaces.

As discussed in Section 2, the notion of probabilistic scheduler gives good reasons for considering two convex bisimilar states as behaviorally equivalent. On the other hand UE-bisimilarity provides witnesses (experiments) $f : X \rightarrow \mathbb{R}$ which can be used to distinguish states that are not UE-bisimilar (cf. example of Figure 2). Thus the two *a posteriori* equivalent (under the mild closedness assumption) viewpoints complement each other in a nice way.

Remark 6. In the significantly different setting of two-player stochastic concurrent games, a behavioral equivalence called *game bisimilarity* is discovered in [9] as the kernel of a behavioral metric d induced by a quantitative $[0, 1]$ -valued fixed-point logic $\text{qL}\mu$. Up to the necessary modifications, game bisimilarity can be shown to coincide with UE-bisimilarity. The authors of [9] argue in favor of game bisimilarity on the basis of the naturalness of the logic $\text{qL}\mu$ (which is a

⁵ A closed subset of compact set is itself compact [16]. From the modeling point of view, this provides a natural topological generalization of finite-branchingness.

weak logic not capable of encoding PCTL). Our explanation in terms of the simple metaphor of \mathbb{R} -valued experiments is perhaps useful in clarifying and further motivating this notion using logic-free arguments. No connection between game bisimilarity and convexity is discussed in [9]. See also Section 4.1 below.

4 Real Valued Modal Logics and Łukasiewicz μ -Calculus

The result of Theorem 1 states that the closed convex hull operation $A \mapsto \overline{H}(A)$ and the upper expectation $A \mapsto ue_A$ are essentially the same operation.

Proposition 2 ([14]). *Let X be a finite set and $A \in \mathcal{PD}(X)$. Then $\overline{H}(A) = \{\mu \in \mathcal{D}(X) \mid \forall f: X \rightarrow \mathbb{R}. (E_\mu(f) \leq ue_A(f))\}$.*

Thus from the functional ue_A it is possible to construct $\overline{H}(A)$ and, by Theorem 1, from $\overline{H}(A)$ one obtains $ue_A = ue_{\overline{H}(A)}$. Hence, the transition map α of a PNTS (X, α) , with $\alpha(x)$ convex closed for all $x \in X$ (i.e., a $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra, cf. remarks 3 and 5 and Theorem 2), can be seen both as a function $(x \mapsto \alpha(x))$ of type $X \rightarrow \mathcal{P}_{cc}\mathcal{D}(X)$ and as a function $(x \mapsto ue_{\alpha(x)})$ of type $X \rightarrow ((X \rightarrow \mathbb{R}) \rightarrow \mathbb{R})$. Equivalently (by currying) the transition map α can alternatively be seen as the function transformer $\diamond_\alpha : (X \rightarrow \mathbb{R}) \rightarrow (X \rightarrow \mathbb{R})$ defined as:

$$(\diamond_\alpha f)(x) = ue_{\alpha(x)}(f) \stackrel{\text{Def 10}}{=} \bigsqcup_{x \rightarrow \mu} E_\mu(f) \tag{1}$$

It is remarkable here that the function transformer \diamond_α happens to coincide with the interpretation of the *diamond modality* in all \mathbb{R} -valued modal logics for PNTS's in the literature [15,26,8,23,12]. The semantics of a formula ϕ of these logics, interpreted on a PNTS (X, α) , is a function $\llbracket \phi \rrbracket : X \rightarrow \mathbb{R}$ and, in particular, $\llbracket \diamond \phi \rrbracket = \diamond_\alpha(\llbracket \phi \rrbracket)$. While it is obvious that the PNTS (X, α) induces \diamond_α (just as in the definition), it is far from clear that from \diamond_α one can reconstruct the $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra (X, α) . The fact that the core of a \mathbb{R} -valued logic (i.e., the interpretation of the basic modality \diamond) automatically arises from the very elementary observation (motivating UE-bisimilarity) that \mathbb{R} -valued experiments on PNTS's are useful and, due to their greater observational power (cf. example of Figure 2) should replace ordinary Boolean predicates, sheds light on the mathematical nature of the quantitative approach to logics for PNTS's.

Following our discussions, *formulas* can then be thought of as *experiments* and the value $\llbracket \diamond \phi \rrbracket(x)$ as the maximal expected value of experiment ϕ performed after “pushing the button” (in Milner’s terminology) at state x . E.g., the experiment distinguishing the states x and y of Figure 2 corresponds to the formula $\diamond \phi$, for some ϕ crafted in such a way that $\llbracket \phi \rrbracket(x_1) = 0.6$, $\llbracket \phi \rrbracket(x_2) = 0$ and $\llbracket \phi \rrbracket(x_3) = 0.5$.

The many concrete \mathbb{R} -valued modal logics in the literature are obtained by considering other connectives (which can then be thought of as ways of compositionally constructing complex experiments from simpler ones) to be used in combination with \diamond . For example the constant $\underline{1}$ ($\llbracket \underline{1} \rrbracket(x) = 1$) and the connective \sqcup ($\llbracket \phi \sqcup \psi \rrbracket(x) = \max\{\llbracket \phi \rrbracket(x), \llbracket \psi \rrbracket(x)\}$) are considered in all the logics we

are aware of. Different choices of connectives have distinct advantages over each other in terms of, e.g., model checking complexity, expressivity, game semantics, compositional reasoning methods, *etc.* In this paper we focus on the recently introduced quantitative fixed-point logic of [24,22], called Łukasiewicz μ -calculus ($L\mu$), because (unlike the other quantitative logics cited above) it is sufficiently expressive to encode other popular temporal logics for PNTS, such as PCTL. Here we limit ourselves to the basic definition of $L\mu$ and we refer to [24,25,22] for motivational discussions and theoretical results about $L\mu$ including: model checking algorithms, game semantics and proof systems for verification.

Definition 12 (Syntax). *Formulas are generated by the following grammar: $\phi ::= \diamond\phi \mid v \mid \perp \mid \lambda\phi \mid \neg\phi \mid \phi \sqcup \phi \mid \phi \oplus \phi \mid \mu v.\phi$, where $\lambda \in [0, 1] \cap \mathbb{Q}$, v ranges over a countable set \mathbf{Var} of variables and (as usual in fixed point logics) bound variables must occur under the scope of an even number of negations.*

Definition 13 (Semantics). *Given a PNTS (X, α) and an interpretation $\rho : \mathbf{Var} \rightarrow (X \rightarrow [0, 1])$ of the variables, the semantics of ϕ is defined as the map $\llbracket \phi \rrbracket_\rho : X \rightarrow [0, 1]$ defined by structural induction on formulas as:*

- $\llbracket \diamond\phi \rrbracket_\rho = \diamond_\alpha(\llbracket \phi \rrbracket_\rho)$ • $\llbracket v \rrbracket_\rho = \rho(v)$ • $\llbracket \phi \sqcup \psi \rrbracket_\rho(x) = \max\{\llbracket \phi \rrbracket_\rho(x), \llbracket \psi \rrbracket_\rho(x)\}$
- $\llbracket \lambda\phi \rrbracket_\rho(x) = \lambda \cdot \llbracket \phi \rrbracket_\rho(x)$ • $\llbracket \perp \rrbracket_\rho(x) = 1$ • $\llbracket \phi \oplus \psi \rrbracket_\rho(x) = \min\{1, \llbracket \phi \rrbracket_\rho(x) + \llbracket \psi \rrbracket_\rho(x)\}$
- $\llbracket \neg\phi \rrbracket_\rho(x) = 1 - \llbracket \phi \rrbracket_\rho(x)$ • $\llbracket \mu v.\phi \rrbracket_\rho = \text{lfp}(f \mapsto \llbracket \phi \rrbracket_{\rho[f/v]})$

where lfp denotes the least fixed point operator and $\rho[f/v]$ denotes the update of the interpretation ρ at variable v defined as expected [24].

The operations of $L\mu$, except the modality $\diamond(-)$ and the fixed-point operator $\mu v.(-)$, are the operations of Łukasiewicz (fuzzy) logic, the logic of MV-algebras.

The logic $L\mu$, as all other \mathbb{R} -valued logics for PNTS's we are aware of, is *sound* (or *adequate*) with respect to UE-bisimilarity.

Theorem 3 (Soundness). *Let (X, α) be a PNTS and $E \subseteq X \times X$ a UE-bisimulation. For all $L\mu$ formulas ϕ and $(x, y) \in E$ it holds that $\llbracket \phi \rrbracket(x) = \llbracket \phi \rrbracket(y)$.*

Unlike most other logics, however, $L\mu$ enjoys the following strong property which generalizes to the setting described in Remark 5.

Theorem 4 (Denseness of $L\mu$). *Let (X, α) be a PNTS and $E \subseteq X \times X$ a UE-bisimulation. The set of functions $\{\llbracket \phi \rrbracket \mid \phi \text{ is a closed } L\mu \text{ formula}\}$ is dense (w.r.t the sup-norm on $X \rightarrow \mathbb{R}$) in the set of functions $X \rightarrow [0, 1]$ with are E -invariant (cf. Definition 11). The same result holds even if the fixed-point free fragment of $L\mu$ is considered.*

This means that for every $\epsilon > 0$ and every experiment $f : X \rightarrow [0, 1]$ which cannot distinguish between UE-bisimilar states, there exists a closed (fixed-point free) $L\mu$ formula ϕ such that for all $x \in X$, $|f(x) - \llbracket \phi \rrbracket(x)| < \epsilon$. Thus, up-to approximation to an arbitrary degree of precision, the formulas of $L\mu$ syntactically capture the set of experiments on PNTS's that respect UE-bisimilarity. The following completeness (or *expressivity*) result is a simple consequence of Theorem 4.

Proposition 3 (Completeness). *Let (X, α) be a PNTS and $x, y \in X$ be two states which are not UE-bisimilar. Then there exists some (fixed-point free) $L\mu$ formula ϕ such that $\llbracket \phi \rrbracket(x) \neq \llbracket \phi \rrbracket(y)$.*

4.1 Behavioral Metrics

Behavioral metrics for probabilistic systems, first investigated by Panangaden in the context of Markov processes [27], are based on the intuition that small changes in the probabilities of the system corresponds, roughly speaking, to small changes in behavior. Thus one might wish to relate behavior between states not in terms of equivalence relations but rather in terms of metrics⁶ $d: X \times X \rightarrow [0, 1]$ capturing *how much* two states behave differently. This approach generalizes that based on ordinary equivalence relations, as one can always consider the kernel $\ker(d) = \{(x, y) \mid d(x, y) = 0\}$ of the metric, and proved to be useful in developing a general theory of *approximation* for probabilistic systems [27]. In the context of PNTS's a robust notion of behavioral metric is given by the so-called Hausdorff metric which is based on the following idea. For a given PNTS (X, α) , and for any⁷ metric m on $\mathcal{D}(X)$, one can consider the Hausdorff metric d_H^m on the space of (compact) closed subsets of $\mathcal{D}(X)$. Then it is natural to define a metric on states as $d(x, y) = d_H^m(\alpha(x), \alpha(y))$.

An elegant characterization of the Kantorovich-Hasudorff metric d_H^K (where K is the Kantorovich metric on $\mathcal{D}(X)$ [16,27]) using the logic $\text{qL}\mu$ of [9] (cf., Remark 6) can be obtained as $d_H^K = \sup \{ |\llbracket \phi \rrbracket(x) - \llbracket \phi \rrbracket(y)|, \phi \in \text{qL}\mu \}$. This relies on the connectives of $\text{qL}\mu$ carefully chosen so that (following earlier ideas of Panangaden [27]) denotations of formulas are Lipschitz (not expansive) functions. This, however, implies that $\text{qL}\mu$ does not satisfy the denseness property of $\text{L}\mu$ and, as a consequence, cannot express PCTL properties. Since the logic $\text{qL}\mu$ is readily a fragment of $\text{L}\mu$, the Łukasiewicz μ -calculus can be used to characterize the metric d_H^K . However it is immediate to observe that, because of the denseness and completeness properties of $\text{L}\mu$, the distance logically defined as above with ϕ ranging over all $\text{L}\mu$ formulas is necessarily trivial (i.e., $d(x, y) = 0$ if x and y are UE-bisimilar and 1 otherwise). The following result shows how the TV-Hausdorff metric d_H^{TV} (where TV is the Total Variation metric on $\mathcal{D}(X)$ [16]) admits an elegant characterization in terms of Diamond-guarded $\text{L}\mu$ formulas.

Theorem 5. *Let (X, α) be a PNTS quotiented by UE-bisimilarity (cf. Remark 6) and assume without loss of generality (cf. Theorem 1) that $\alpha(x)$ is convex closed for every $x \in X$. Define $d_L(x, y) = \sup \{ |\llbracket \diamond \phi \rrbracket(x) - \llbracket \diamond \phi \rrbracket(y)|, \phi \in \text{L}\mu \}$. Then $d_L(x, y) = d_H^{TV}(\alpha(x), \alpha(y))$.*

Proof. The proof exploits a result about convex closed sets from functional analysis [19, §8.4] and the denseness property of $\text{L}\mu$ (Theorem 4) in the sup-norm.

The definition of a behavioral metric for PNTS's based on the TV-metric on $\mathcal{D}(X)$ appears to be novel and seems worth of further investigation. Indeed the

⁶ Technically, d is a pseudo-metric as states $x \neq y$ with $d(x, y) = 0$ are admitted. The function d becomes an authentic metric on the state space quotiented by $\ker(d)$.

⁷ Many natural metrics on $\mathcal{D}(X)$ exist. E.g., any norm on \mathbb{R}^n (cf. Convention 1) induces a metric on $\mathcal{D}(X)$. Since \mathbb{R}^n is finite dimensional, they all induce the same topology. The sup-norm on \mathbb{R}^n (cf. Theorem 4) induces the *total variation* metric $T(\mu, \nu) = \max_x \{ |\mu(x) - \nu(x)| \}$ [16,19].

TV-metric is well known in probability theory and has important applications in statistics. Furthermore, it is natural to interpret $d_L(x, y)$ as a value related to the probability of distinguishing between x and y in a *one-shot* experiment ϕ , modeled by the formula $\diamond\phi$ (“push the button” once and perform experiment ϕ). We plan to investigate this viewpoint, and the possible relations with the (information-theoretic) one-shot attack models of [5], in future research.

5 Algebraic Aspects of UE-Bisimilarity

As discussed in Section 4, the transition map $\alpha : X \rightarrow \mathcal{P}_{cc}\mathcal{D}(X)$ of a $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra (X, α) can be equivalently presented as the function transformer $\diamond_\alpha : (X \rightarrow \mathbb{R}) \rightarrow (X \rightarrow \mathbb{R})$ defined as in Equation 1. Clearly, not all transformers F of this type can be representations of $\mathcal{P}_{cc}\mathcal{D}$ -coalgebras (e.g., $F(x \mapsto 0) = x \mapsto 1$). The following theorem, based on a type of Riesz representation theorem for convex closed sets from functional analysis [14], provides a precise characterization of those function transformers F that arise from $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra.

Theorem 6. *Let (X, α) be a $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra. Denote with \sqsubseteq and $+$ the pointwise order and sum of the vector space $X \rightarrow \mathbb{R}$, respectively, and with $\lambda(_)$ the operation of multiplication by scalars. The function transformer \diamond_α satisfies the following properties: • (monotone) if $f \sqsubseteq g$ then $\diamond_\alpha f \sqsubseteq \diamond_\alpha g$, • ($\diamond_\alpha(\mathbf{1})$ is Boolean) $\diamond_\alpha(x \mapsto 1) \in X \rightarrow \{0, 1\}$, • (sublinear) $\diamond_\alpha(f + g) \sqsubseteq \diamond_\alpha(f) + \diamond_\alpha(g)$, • (positive affine homogenous) $\diamond_\alpha(\lambda_1 f + \lambda_2 \mathbf{1}) = \lambda_1 \diamond_\alpha(f) + \lambda_2 \diamond_\alpha(\mathbf{1})$, for all $\lambda_1 \geq 0$ and $\lambda_2 \in \mathbb{R}$. Furthermore, every function transformer $F : (X \rightarrow \mathbb{R}) \rightarrow (X \rightarrow \mathbb{R})$ with these properties arise as $F = \diamond_\alpha$ from a unique $\mathcal{P}_{cc}\mathcal{D}$ -coalgebra (X, α) .*

The theorem can be generalized to the setting described in Remark 5. In a follow-up of this paper it will be shown how, on the basis of this result, it is possible to develop an algebraic account of $\mathcal{P}_{cc}\mathcal{D}$ -coalgebras in the form of a correspondence between $\mathcal{P}_{cc}\mathcal{D}$ -coalgebras and certain types of algebras (with a rich vector space structure resembling that of the function space $X \rightarrow \mathbb{R}$). We expect this result will be of help in designing *compositional* verification methods for PNTS’s based on *equational reasoning* and *axiomatizations* of \mathbb{R} -valued logics for PNTS’s.

We conclude this section by stating a result of central importance for the practical applicability of UE-bisimilarity in programming languages which can be proved by means of standard process-algebra techniques (see, e.g., [7, §3]).

Theorem 7. *UE-bisimilarity is a congruence relation for all process algebras specified by the probabilistic-nondeterministic PGSOS format of [2].*

Acknowledgements. The author would like to thank Alex Simpson, Alexandra Silva, Jan Rutten, Marcello Bonsangue, Helle Hvid Hansen, Henning Basold and the anonymous reviewers for their helpful comments and suggestions. The support of Advanced Grant ECSYM of the ERC is acknowledged with gratitude.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
2. Bartels, F.: On Generalised Coinduction and Probabilistic Specification Formats: Distributive Laws in Coalgebraic Modelling. PhD thesis, CWI (2004)
3. Bernardo, M., De Nicola, R., Loretto, M.: Revisiting trace and testing equivalences for nondeterministic and probabilistic processes. In: Birkedal, L. (ed.) FOSSACS 2012. LNCS, vol. 7213, pp. 195–209. Springer, Heidelberg (2012)
4. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
5. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the Bayes risk in information-hiding protocols. *Journal of Computer Security* 16(5) (2008)
6. Crafa, S., Ranzato, F.: A spectrum of behavioral relations over LTSs on probability distributions. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 124–139. Springer, Heidelberg (2011)
7. D’Argenio, P.R., Gebler, D., Lee, M.D.: Axiomatizing bisimulation equivalences and metrics from probabilistic SOS rules. In: Muscholl, A. (ed.) FOSSACS 2014. LNCS, vol. 8412, pp. 284–298. Springer, Heidelberg (2014)
8. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. *Journal of Computer and System Sciences* 68, 374–397 (2004)
9. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game refinement relations and metrics. *Logical Methods in Computer Science* 4 (2008)
10. Deng, Y., van Glabbeek, R.: Characterising probabilistic processes logically. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 278–293. Springer, Heidelberg (2010)
11. Deng, Y., van Glabbeek, R., Hennessy, M., Morgan, C.: Testing finitary probabilistic processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 274–288. Springer, Heidelberg (2009)
12. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Weak bisimulation is sound and complete for PCTL. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 355–370. Springer, Heidelberg (2002)
13. Goubault-Larrecq, J.: Prevision domains and convex powercones. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 318–333. Springer, Heidelberg (2008)
14. Huber, P.: Robust Statistics. Wiley (1991)
15. Huth, M., Kwiatkowska, M.: Quantitative analysis and model checking. In: Proc. of LICS (1997)
16. Kechris, A.S.: Classical Descriptive Set Theory. Springer (1994)
17. Kozen, D.: Results on the propositional mu-calculus. *Theoretical Computer Science*, 333–354 (1983)
18. Kurz, A.: Logics for Coalgebras and Applications to Computer Science. PhD thesis, Ludwig Maximilian University of Munich (2000)
19. Lax, P.: Functional Analysis. Wiley Interscience (2002)
20. Lynch, N., Segala, R.: Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing* 2, 250–273 (1995)
21. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
22. Mio, M.: Game Semantics for Probabilistic μ -Calculi. PhD thesis, School of Informatics, University of Edinburgh (2012)

23. Mio, M.: Probabilistic Modal μ -Calculus with Independent product. *Logical Methods in Computer Science* 8(4) (2012)
24. Mio, M., Simpson, A.: Łukasiewicz mu-calculus. In: *Proc. of Workshop on Fixed Points in Computer Science. EPTCS*, vol. 126 (2013)
25. Mio, M., Simpson, A.: A proof system for compositional verification of probabilistic concurrent processes. In: Pfenning, F. (ed.) *FOSSACS 2013. LNCS*, vol. 7794, pp. 161–176. Springer, Heidelberg (2013)
26. Morgan, C., McIver, A.: A probabilistic temporal calculus based on expectations. In: *Proc. of Formal Methods Pacific* (1997)
27. Panangaden, P.: *Labelled Markov processes*. Imperial College Press (2009)
28. Sangiorgi, D., Rutten, J.: *Advanced topics in bisimulation and coinduction*. Cambridge University Press (2011)
29. Segala, R.: *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis. MIT (1995)
30. Sokolova, A.: *Probabilistic Systems Coalgebraically: A survey*. *Theoretical Computer Science* 412(38) (2011)
31. Song, L., Zhang, L., Godskesen, J.C.: Bisimulations Meet PCTL Equivalences for Probabilistic Automata. In: Katoen, J.-P., König, B. (eds.) *CONCUR 2011. LNCS*, vol. 6901, pp. 108–123. Springer, Heidelberg (2011)
32. Tix, R., Keimel, K., Plotkin, G.D.: Semantic domains for combining probability and non-determinism. *Electronic Notes in Theoretical Computer Science* (2005)

Interacting Bialgebras Are Frobenius

Filippo Bonchi¹, Paweł Sobociński², and Fabio Zanasi¹

¹ ENS de Lyon, Université de Lyon, CNRS, INRIA, France

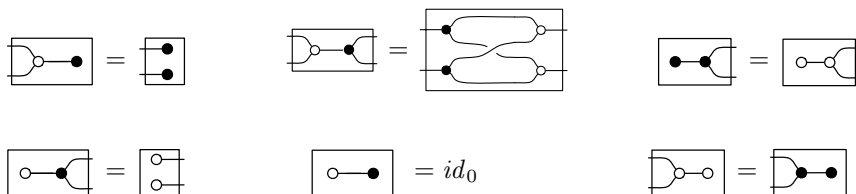
² University of Southampton, UK

Abstract. Bialgebras and Frobenius algebras are different ways in which monoids and comonoids interact as part of the same theory. Such theories feature in many fields: e.g. quantum computing, compositional semantics of concurrency, network algebra and component-based programming.

In this paper we study an important sub-theory of Coecke and Duncan’s ZX-calculus, related to strongly-complementary observables, where two Frobenius algebras interact. We characterize its free model as a category of \mathbb{Z}_2 -vector subspaces. Moreover, we use the framework of PROPs to exhibit the modular structure of its algebra via a universal construction involving span and cospan categories of \mathbb{Z}_2 -matrices and distributive laws between PROPs. Our approach demonstrates that the Frobenius structures result from the interaction of bialgebras.

1 Introduction

We report on a surprising meeting point between two separate threads of research. First, Coecke and Duncan [9] introduced the ZX-calculus as a graphical formalism for multi-qubit systems, featuring two interacting separable Frobenius algebras, which we distinguish here graphically via white and black colouring. The following equations capture the interaction for an important fragment of the calculus related to strongly complementary observables [10]:



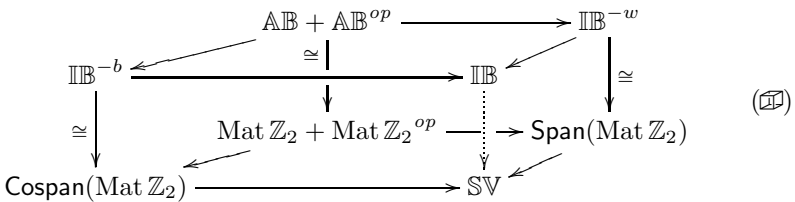
The aforementioned and related works (see e.g. [11]) emphasise the interaction of two different (here, white and black) Frobenius structures. As we will explain, from an algebraic point of view, it is natural to consider this system as two (anti-separable) *bialgebras* interacting via two distributive laws of PROPs. We will show that the individual Frobenius structures arise as a *result* of these interactions. Consequently, we call the theory above *interacting bialgebras*, and the corresponding (free) PROP \mathbb{IB} .

Second, following the work of Katis, Sabadini, Walters and others on the $\text{Span}(\mathbf{Graph})$ algebra [13] of transition systems, the second author introduced the calculus of Petri nets with boundaries [21] and commenced the study of the resulting structures in [22]. That calculus and extensions in [5, 6] are based on the algebra of stateless connectors [4] of Bruni, Lanese and Montanari, also generated by two monoid-comonoid structures—which again, for sake of uniformity we will refer to as black and white.

Intuitively, in [4] a connector $n \rightarrow m$ has n ports on the left boundary and m ports on the right boundary. A black connector forces synchronization on all its ports, while a white one allows only two ports on opposite boundaries to synchronize. The semantics of connectors $n \rightarrow m$ are relations $\{0, 1\}^n \rightarrow \{0, 1\}^m$. For example, the black multiplication $2 \rightarrow 1$ is the relation $\{(00, 0), (11, 1)\}$ while the white multiplication is the relation $\{(00, 0), (01, 1), (10, 1)\}$. The black structure (the semantics of comultiplication is the opposite relation) is a Frobenius algebra. The white structure is not Frobenius, but it becomes so if one adds the behaviour $(11, 0)$ to the semantics of the white multiplication, making it the graph of addition¹ in \mathbb{Z}_2 . The resulting theory satisfies the equations of \mathbb{IB} .

The meeting point of the two, seemingly disparate, threads is thus the PROP \mathbb{IB} . Before accounting for other related work, we outline our contributions.

- We characterise \mathbb{IB} as the PROP SV of \mathbb{Z}_2 -sub-vector spaces: the arrows $n \rightarrow m$ are sub-vector spaces of $\mathbb{Z}_2^n \times \mathbb{Z}_2^m$, with relational composition.
- We use Lack’s framework of distributive laws on PROPs [15] to exhibit the modularity of this theory. The starting point is Lafont’s observation [16, Theorem 5] that the theory of anti-separable bialgebras \mathbb{AB} is precisely the PROP $\text{Mat } \mathbb{Z}_2$ of \mathbb{Z}_2 -matrices. $\text{Mat } \mathbb{Z}_2$ can be composed with its dual $\text{Mat } \mathbb{Z}_2^{op}$ via a distributive law given by pullback: the result of this composition is $\text{Span}(\text{Mat } \mathbb{Z}_2)$, the PROP of spans over $\text{Mat } \mathbb{Z}_2$. Dually, $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$ arises from the distributive law of $\text{Mat } \mathbb{Z}_2^{op}$ over $\text{Mat } \mathbb{Z}_2$ given by pushout. The theories of $\text{Span}(\text{Mat } \mathbb{Z}_2)$ and $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$ are actually the same “up-to-exchanging the colours”: they are the theory of \mathbb{IB} , but *without* the separability equation on precisely one of the white or black structures. We call them, respectively, \mathbb{IB}^{-w} and \mathbb{IB}^{-b} . We prove that the top and bottom faces in the cube below are pushout diagrams in the category of PROPs: the isomorphism between \mathbb{IB} and SV then follows from the universal property.



¹ This works if one takes the graph of addition in any abelian group, which was pointed out to the second author by RFC Walters.

The mapping $\mathbb{I}\mathbb{B} \rightarrow \mathbb{S}\mathbb{V}$ gives a *semantics* for $\mathbb{I}\mathbb{B}$: it can be presented in inductive form, yielding a simple technique for checking term equality in $\mathbb{I}\mathbb{B}$.

From a mathematical point of view, the results in this paper are a continuation of the programme initiated by Lack in [15]. In particular, our focus is on systematically extracting from distributive laws (a) complete axiomatisations and (b) factorisation systems for theories. Recent work on capturing algebraic theories using similar techniques includes [12] and [22].

Frobenius algebras [8, 14] have received much attention in topology, physics, algebra and computer science, partly because of the close correspondence with 2D TQFTs. The algebras we consider are the result of the research initiated by Abramsky and Coecke [1] on applying graphical techniques associated with algebras of monoidal categories [20] to model and reason about quantum protocols.

Related monoid-comonoid structures have been studied by computer scientists: amongst several the connectors in network algebra [23] and the wiring operations of REO [2]. Another closely related thread is Lafont’s work on the algebraic theory of Boolean circuits [16], following the ideas of Burroni [7].

Structure of the paper. In §2 we recall the background on PROPs. In §3 we introduce the PROP $\mathbb{I}\mathbb{B}$ and consider some of its properties. In §4 we recall the theory of anti-separable bialgebras and the characterisation of its free model as $\text{Mat } \mathbb{Z}_2$. In §5 we give the details of the two distributive laws that yield $\text{Span}(\text{Mat } \mathbb{Z}_2)$ and $\text{Cosp}(\text{Mat } \mathbb{Z}_2)$ and their elementary presentations as the free PROPs $\mathbb{I}\mathbb{B}^{-w}$ and $\mathbb{I}\mathbb{B}^{-b}$. In §6 we collect our results to construct the cube (\square).

Notation. Composition of arrows $f: a \rightarrow b, g: b \rightarrow c$ is denoted by $f; g: a \rightarrow c$. $\mathbb{C}[a, b]$ is the set of arrows from a to b in a small category \mathbb{C} and $f^* \in \mathbb{C}^{op}[b, a]$ is the contravariant counterpart of $f \in \mathbb{C}[a, b]$. Given $\mathcal{F}: \mathbb{C}_1 \rightarrow \mathbb{C}_2$, we denote with $\mathcal{F}^{op}: \mathbb{C}_1^{op} \rightarrow \mathbb{C}_2^{op}$ the functor defined by $(a \xrightarrow{f} b) \mapsto (a \xrightarrow{\mathcal{F}(f^*)^*} b)$.

2 Background

In this section we recall PROPs and their composition.

2.1 PROPs and Symmetric Monoidal Theories

Let \mathbb{P} be the skeletal symmetric strict monoidal category of finite sets and bijections. It is harmless to take the naturals $\mathbb{N} = \{0, 1, 2, \dots\}$ as the objects, where $n \in \mathbb{N}$ stands for the finite set $\{0, 1, n - 1\}$. The tensor product on objects is $n + m$. On arrows, given $f: n \rightarrow n$ and $g: m \rightarrow m, f \otimes g = f + g: n + m \rightarrow n + m$ where $+$ is ordinal sum.

Our exposition is founded on symmetric strict monoidal categories called PROPs (**product** and **permutation** categories [15, 17]). They have \mathbb{N} as the set of objects and the tensor product on objects is addition. Any PROP \mathbb{T} contains certain arrows called permutations, which yield the symmetric monoidal

structure and satisfy the same equations as they do in \mathbb{P} —i.e. there is a identity-on-objects symmetric strict (ISS) monoidal functor from \mathbb{P} to \mathbb{T} . \mathbb{P} is actually the initial object in **PROP**, the category of PROPs and their *homomorphisms*: ISS monoidal functors that are homomorphic w.r.t. the permutations. In fact, **PROP** is the slice category \mathbb{P}/\mathbf{PRO} where **PRO** is the category of symmetric strict monoidal categories that have \mathbb{N} as set of objects and ISS functors. The fact that **PROP** is a slice category is vital: e.g. when we calculate the coproduct of two PROPs we must equate the images of the permutations via the injections (coproducts in a slice category are pushouts in the underlying category).



PROPs can encode (one-sorted) symmetric monoidal theories, that are equational theories at the level of abstraction of symmetric monoidal categories. A *symmetric monoidal theory* (SMT) is a pair (Σ, E) where Σ is a signature with elements $o: n \rightarrow m$. Here o is an operation symbol with *arity* n and *coarity* m . The Σ -terms are built by composing operations in Σ , subject to laws of symmetric monoidal categories. The set E consists of equations between Σ -terms.

The *free PROP* $\mathbb{T}_{(\Sigma, E)}$ on the theory (Σ, E) is defined by letting $\mathbb{T}_{(\Sigma, E)}[n, m]$ be the set of Σ -terms with arity n and coarity m quotiented by E . When Σ is clear from the context, we will usually refer to terms of a SMT as *circuits*.

As PROPs describe equational theories, they come equipped with a notion of model: given a PROP \mathbb{T} and a symmetric monoidal category \mathbb{V} , a \mathbb{T} -algebra in \mathbb{V} is any symmetric monoidal functor $\mathcal{A}: \mathbb{T} \rightarrow \mathbb{V}$. On objects, \mathcal{A} is determined by the assignment $\mathcal{A}(1)$, since $\mathcal{A}(n) \cong \mathcal{A}(1)^{\otimes n}$ for any $n \in \mathbb{N}$. The intuition is that $\mathcal{A}(1)$ is the support carrying the structure specified by \mathbb{T} . As expected, if the PROP \mathbb{T} is free on a SMT (Σ, E) , then its algebras have a universal characterization in terms of the models of (Σ, E) [12, 18].

Next we recall two important examples of SMTs: commutative monoids, commutative comonoids and the corresponding free PROPs.

The theory (Σ_M, E_M) of commutative monoids has two operation symbols in Σ_M - multiplication and unit - for which we adopt the graphical notation on the right.

The left diagram represents the multiplication operation $m: 2 \rightarrow 1$: the two ports on the left boundary of the box represent the arity of m , whereas the single port on the right boundary encodes the coarity of m . Similarly, the right diagram depicts the unit operation $u: 0 \rightarrow 1$. Σ_M -terms are built out of those two components, plus the permutation () and identity () circuits, by sequential ($;$) and parallel (\otimes) composition. The set E_M expresses the following equations, stating associativity (M1), commutativity (M2) and identity (M3).

$$\begin{array}{c} \text{Diagram 1} \end{array} = \begin{array}{c} \text{Diagram 2} \end{array} \quad (\text{M1}) \quad \begin{array}{c} \text{Diagram 3} \end{array} = \begin{array}{c} \text{Diagram 4} \end{array} \quad (\text{M2}) \quad \begin{array}{c} \text{Diagram 5} \end{array} = \begin{array}{c} \text{Diagram 6} \end{array} \quad (\text{M3})$$

The free PROP on (Σ_M, E_M) is isomorphic to the skeletal symmetric strict monoidal category \mathbb{F} of finite sets and functions. Indeed, the graph of a function $f: n \rightarrow m$ can be represented as a Σ_M -term: the equations (M1)-(M3) guarantee that this is a bijective representation. Consequently, an \mathbb{F} -algebra $\mathcal{A}: \mathbb{F} \rightarrow \mathbb{V}$ is precisely a commutative monoid in \mathbb{V} with carrier $\mathcal{A}(1)$.

\mathbb{F}^{op} is also a PROP, which is free for the theory (Σ_C, E_C) of commutative comonoids. As \mathbb{F}^{op} is the opposite of \mathbb{F} , the operations in Σ_C (called comultiplication and counit, on the left) and the equations in E_C are those of E_M “rotated by 180° ”.

2.2 Composing PROPs

Given SMTs (Σ, E) and (Σ', E') , one can define their *sum* as the theory $(\Sigma \uplus \Sigma', E \uplus E')$. Usually one quotients the sum by new equations, describing the way in which the operations in Σ and Σ' interact. Both our leading examples of this construction are quotients of the sum of the theories of monoid and comonoids:

- the theory of (commutative/cocommutative) *bialgebras* is given as $(\Sigma_M \uplus \Sigma_C, E_M \uplus E_C \uplus B)$, where B consists of the following equations.

$$\begin{array}{ccc}
 \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} & \text{(B1)} & \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} & \text{(B3)}
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} & \text{(B2)} & \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = id_0 & \text{(B4)}
 \end{array}$$

- The theory of *Frobenius algebras* is given as $(\Sigma_M \uplus \Sigma_C, E_M \uplus E_C \uplus F)$, where F consists of the following two equations.

$$\begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \quad \text{(Frob)}$$

(Frob) states that circuits are invariant with respect to any topological deformation of their internal structure, provided that the link configuration between the ports is preserved. The theory of *separable* Frobenius algebras (SFAs) is given by adding to F the following equation.

$$\begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \begin{array}{|c|} \hline \bullet \\ \hline \end{array} \quad \text{(Sep)}$$

Just as SFAs and bialgebras express different ways of combining a monoid and a comonoid, their free PROPs can be equivalently described as different ways of “composing” the PROPs \mathbb{F} and \mathbb{F}^{op} . As we will see, this composition exactly amounts to the sum of the two SMTs quotiented by new equations.

To make this precise, we recall from [15] how PROP composition is defined in terms of distributive laws between monads. As shown in [24], the whole theory of monads can be developed in an arbitrary bicategory. Of particular interest are monads in the bicategory $\mathbf{Span}(\mathbf{Set})$, as they exactly correspond to small categories. A distributive law between two such monads can be seen as a way of forming the composite of the associated categories (with the same objects) [19].

In an analogous way, a PROP can be represented as a monad in a certain bicategory [15] and any two PROPs \mathbb{T}_1 and \mathbb{T}_2 can be composed via a distributive law $\lambda: \mathbb{T}_2 ; \mathbb{T}_1 \rightarrow \mathbb{T}_1 ; \mathbb{T}_2$ between the associated monads, provided that λ “respects” the monoidal structure [15].

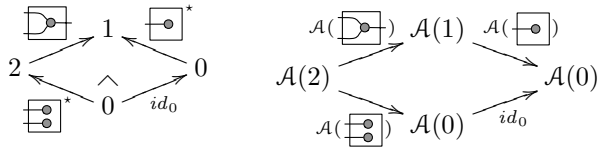
Remark 1. The monad $\mathbb{T}_1 ; \mathbb{T}_2$ yields a PROP with the following properties [15]:

- (†) any arrow $f \in \mathbb{T}_1 ; \mathbb{T}_2[n, m]$ can be factorised into $f' \in \mathbb{T}_1[n, z]$ and $f'' \in \mathbb{T}_2[z, m]$, for some $z \in \mathbb{N}$;
- (‡) a $\mathbb{T}_1 ; \mathbb{T}_2$ -algebra $\mathcal{A}: \mathbb{T}_1 ; \mathbb{T}_2 \rightarrow \mathbb{V}$ gives $\mathcal{A}(1)$ the structure of a \mathbb{T}_1 -algebra and a \mathbb{T}_2 -algebra, subject to the equations induced by the distributive law.

We provide an example of this construction and refer to [15] for further details.

Example 1. Let us consider what it means to define the composite PROP $\mathbb{F}^{op} ; \mathbb{F}$ via a distributive law $\lambda: \mathbb{F} ; \mathbb{F}^{op} \rightarrow \mathbb{F}^{op} ; \mathbb{F}$. By its type, λ should map a pair of arrows $f \in \mathbb{F}[n, z]$, $g \in \mathbb{F}^{op}[z, m]$ into a pair $g' \in \mathbb{F}^{op}[n, z]$, $f' \in \mathbb{F}[z, m]$. This amounts to saying that λ maps *cospans* $n \xrightarrow{f} z \xleftarrow{g^*} m$ into *spans* $n \xleftarrow{g'^*} z \xrightarrow{f'} m$ in \mathbb{F} : a canonical way to define such a mapping is by forming the pullback of the given cospan. This indeed makes λ satisfy the equations of distributive laws [15]. The resulting PROP $\mathbb{F}^{op} ; \mathbb{F}$ is the category of spans on \mathbb{F} , obtained by identifying the isomorphic 1-cells of the bicategory $\mathbf{Span}(\mathbb{F})$ and forgetting the 2-cells. With a slight abuse of notation, we call this category $\mathbf{Span}(\mathbb{F})$.

The SMT of $\mathbf{Span}(\mathbb{F})$ is the sum of the theories of the composed categories \mathbb{F} and \mathbb{F}^{op} , quotiented by the equations induced by the distributive law. Those equations can be obtained by interpreting the pullbacks defining λ in a generic algebra $\mathcal{A}: \mathbf{Span}(\mathbb{F}) \rightarrow \mathbb{V}$. In this case, it suffices to consider four pullbacks [15]. One of them is depicted on the left, and its image in \mathbb{V} is depicted on the right.



Since $\boxed{\circ}$ and $\boxed{\circ}^*$ originally belong to the \mathbb{F}^{op} -algebra structure, what is interpreted is their contravariant counterpart. Commutativity of the right-hand diagram is implied by $\mathbf{Span}(\mathbb{F})$ being a composite PROP [15] and it yields the equation (B1). The remaining three pullbacks to be considered yield (B2), (B3) and (B4). Therefore imposing the equations induced by λ correspond precisely to quotienting the monoidal and comonoidal structure of $\mathcal{A}(1)$ by the bialgebra equations. It follows that $\mathbf{Span}(\mathbb{F})$ is the free PROP on the theory of bialgebras.

We now focus on the dual situation: one can define the PROP $\mathbb{F} ; \mathbb{F}^{op}$ via a distributive law $\lambda': \mathbb{F}^{op} ; \mathbb{F} \rightarrow \mathbb{F} ; \mathbb{F}^{op}$ that forms the pushout of a given span. It follows that $\mathbb{F} ; \mathbb{F}^{op}$ is the category $\mathbf{Cospans}(\mathbb{F})$, obtained from the corresponding bicategory of cospans, analogously to the case of $\mathbb{F}^{op} ; \mathbb{F}$ and $\mathbf{Span}(\mathbb{F})$. One obtains the equations given by λ' by interpreting pushout diagrams, analogously to what we showed for λ . Those correspond to (Frob) and (Sep) [15], meaning that $\mathbf{Cospans}(\mathbb{F})$ is the free PROP on the theory of SFAs.

3 Interacting Bialgebras

In this section we present a fragment of the ZX-calculus [9] that we call \mathbb{IB} . We define it as the free PROP on the SMT of *interacting bialgebras* (below) and we state that it is isomorphic to the PROP \mathbb{SV} of \mathbb{Z}_2 vector subspaces. The remainder of the paper is a modular proof of this fact.

Definition 1. *The SMT of interacting bialgebras $(\Sigma_{\mathbb{IB}}, E_{\mathbb{IB}})$ consists of a signature $\Sigma_{\mathbb{IB}}$ with two copies each of the theory of monoids and of comonoids. In order to distinguish them, we colour one monoid/comonoid white, the other black. We will informally refer to them as the white and the black structures.*



The set $E_{\mathbb{IB}}$ of equations consists of:

- the equations making both the white and the black structures SFAs;
- bialgebra equations for the white monoid and the black comonoid;

(Q1)
(Q3)

(Q2)
(Q4)

- the following two equations, expressing the equivalence between the white and the black (self-dual) compact closed structure.

(Q5)
(Q6)

Remark 2. The given axiomatization enjoys the following properties.

- (a) “Rotating any equation by 180°” is sound.
- (b) All equations (and thus all derived laws) are completely symmetric up-to swapping of white and black structures.
- (c) \mathbb{IB} satisfies the following “anti-separability” law expressing the fact that the white and the black structure cancel each other.

(ASep)

- (d) \mathbb{IB} satisfies the “quasi-Frobenius” law below relating the black and white structures. This, together with the Frobenius black and white structures, amounts to saying that “only the topology matters”.

(QFrob)

(e) \mathbb{IB} has all the “zero laws”, expressing that the only circuit with no ports is id_0 : they are (Q4), (Q4) “rotated by 180°” — cf. (a) — and the following.

$$\boxed{\circ \text{---} \circ} = id_0 \quad (\text{Zero}_w) \qquad \boxed{\bullet \text{---} \bullet} = id_0 \quad (\text{Zero}_b)$$

Definition 2. Let \mathbb{SV} be the following PROP:

- arrows $n \rightarrow m$ are subspaces of $\mathbb{Z}_2^n \times \mathbb{Z}_2^m$ (considered as a \mathbb{Z}_2 -vector space).
- The composition \circ is relational: for subspaces $G = \{(u, v) \mid u \in \mathbb{Z}_2^n, v \in \mathbb{Z}_2^z\}$ and $H = \{(v, w) \mid v \in \mathbb{Z}_2^z, w \in \mathbb{Z}_2^m\}$, their composition is the subspace $\{(u, w) \mid \exists v. (u, v) \in G \wedge (v, w) \in H\}$.
- The tensor product \otimes on arrows is given by direct sum of spaces.
- The permutations $n \rightarrow n$ are induced by bijections of finite sets: to $\rho : n \rightarrow n$ we associate the subspace generated by $\{(1_i, 1_{\rho i})\}_{i < n}$ where 1_k stands for the binary n -vector with 1 at the k +1th coordinate and 0s elsewhere. For instance the twist $2 \rightarrow 2$ is the subspace generated by $\{(\binom{1}{0}, \binom{0}{1}), (\binom{0}{1}, \binom{1}{0})\}$.

We now introduce a semantics homomorphism $\mathcal{S}_{\mathbb{IB}} : \mathbb{IB} \rightarrow \mathbb{SV}$ that we will later prove to be an iso. Even if $\mathcal{S}_{\mathbb{IB}}$ is not necessary for proving $\mathbb{IB} \cong \mathbb{SV}$, we present it as a valuable tool to reason about the equivalence of circuits in \mathbb{IB} .

Definition 3. Let $[v_1, \dots, v_n]$ denote the space generated by the vectors $v_1 \dots v_n$. The homomorphism $\mathcal{S}_{\mathbb{IB}} : \mathbb{IB} \rightarrow \mathbb{SV}$ is inductively defined. For the monoids:

$$\begin{array}{ll} \boxed{\circ \text{---} \bullet} \mapsto [(\binom{1}{1}, (1))] & \boxed{\bullet \text{---} \circ} \mapsto [(\binom{0}{1}, (1)), (\binom{1}{0}, (1))] \\ \boxed{\bullet} \mapsto [(), (1)] & \boxed{\circ} \mapsto [(), (0)] \end{array}$$

For the comonoids: take the reverse relations of the ones above; for composite circuits: $s \otimes t \mapsto \mathcal{S}_{\mathbb{IB}}(s) \otimes \mathcal{S}_{\mathbb{IB}}(t)$ and $s ; t \mapsto \mathcal{S}_{\mathbb{IB}}(s) ; \mathcal{S}_{\mathbb{IB}}(t)$.

The homomorphism is well-defined since all the equations of \mathbb{IB} are sound w.r.t. $\mathcal{S}_{\mathbb{IB}}$, namely if $s = t$ then $\mathcal{S}_{\mathbb{IB}}(s) = \mathcal{S}_{\mathbb{IB}}(t)$. The following theorem guarantees that the axiomatization is also complete.

Theorem 1. $\mathcal{S}_{\mathbb{IB}} : \mathbb{IB} \rightarrow \mathbb{SV}$ is an isomorphism of PROPs.

Remark 3. The asymmetry between the black and the white structure in Definition 3 is forced on us because $\mathcal{S}_{\mathbb{IB}}$ will be uniquely determined by the universal property of pushouts in **PROP**. Yet, strikingly, the axioms of \mathbb{IB} describes two algebraic structures—the white and the black—in a completely symmetric way.

In the sequel, we are going to prove Theorem 1 by exploiting PROP composition, as described in Section 2.2. While a more direct proof might be given, our argument reveals the modular structures underlying \mathbb{IB} and \mathbb{SV} .

4 Bialgebras and Vector Spaces

In this section we lay the foundations for our approach, by considering the SMT $\{\Sigma_{\mathbb{A}\mathbb{B}}, E_{\mathbb{A}\mathbb{B}}\}$ of *anti-separable* bialgebras. The set $\Sigma_{\mathbb{A}\mathbb{B}}$ consists of operations $\boxed{\bullet \rightarrow \bullet}$, $\boxed{\bullet \leftarrow \bullet}$, $\boxed{\circ \rightarrow \circ}$ and $\boxed{\circ \leftarrow \circ}$. The set $E_{\mathbb{A}\mathbb{B}}$ contains the equations making the black structure a commutative comonoid, the white structure a commutative monoid, bialgebra equations (Q1)-(Q4) and (ASep). In short, an anti-separable bialgebra is just a bialgebra quotiented by (ASep)². We call its free PROP $\mathbb{A}\mathbb{B}$.

By virtue of Remark 2.(b)-(c), $\mathbb{I}\mathbb{B}$ contains both a copy of $\mathbb{A}\mathbb{B}$ and of $\mathbb{A}\mathbb{B}^{op}$. These describe the interaction between the black and white structures of $\mathbb{I}\mathbb{B}$.

Remark 4. As the free PROP for bialgebras is the composite $\text{Span}(F) = \mathbb{F}^{op} ; \mathbb{F}$ (cf. Example 1), $\mathbb{A}\mathbb{B}$ enjoys the decomposition of Remark 1.(†): any circuit $t \in \mathbb{A}\mathbb{B}[n, m]$ can be factorised as $s ; s' \in \mathbb{A}\mathbb{B}[n, m]$, where $s \in \mathbb{F}^{op}[n, z]$ is part of the black comonoid and $s' \in \mathbb{F}[z, m]$ is part of the white monoid. Moreover, by (ASep), we can assume that any port on the left boundary has at most one connection with any one on the right boundary.

We say that any circuit $s ; s'$ of the above shape is in *matrix form*: indeed, it has an intuitive representation as a matrix, as shown by the following example.

Example 2. The picture on the left shows a circuit $t \in \mathbb{A}\mathbb{B}[3, 4]$ in matrix form and on the right its representation as a 4×3 matrix.



The values in M are calculated as follows. For each boundary of t , suppose a top-down enumeration of its ports. Then $M[i, j]$ is 1 if, reading the circuit from the left to the right, one finds a path connecting the j^{th} port on the left boundary to the i^{th} port on the right, and 0 otherwise.

We now make the matrix semantics of $\mathbb{A}\mathbb{B}$ formal. Let $\text{Mat } \mathbb{Z}_2$ be the PROP with arrows $n \rightarrow m$ being $m \times n$ \mathbb{Z}_2 -matrices, where $;$ is matrix multiplication and \otimes is defined in the obvious way. The permutations are the rearrangements of the rows of the identity matrix. Clearly, $\text{Mat } \mathbb{Z}_2$ is equivalent to the symmetric monoidal category of finite-dimensional \mathbb{Z}_2 -vector spaces and linear maps.

Definition 4. *The homomorphism $\mathcal{S}_{\mathbb{A}\mathbb{B}}: \mathbb{A}\mathbb{B} \rightarrow \text{Mat } \mathbb{Z}_2$ is defined inductively by*

$$\boxed{\circ \rightarrow \bullet} \mapsto ! \quad \boxed{\bullet \leftarrow \bullet} \mapsto \mathfrak{i} \quad \boxed{\circ \rightarrow \circ} \mapsto (1 \ 1) \quad \boxed{\bullet \leftarrow \bullet} \mapsto \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



$$s \otimes t \mapsto \mathcal{S}_{\mathbb{A}\mathbb{B}}(s) \otimes \mathcal{S}_{\mathbb{A}\mathbb{B}}(t) \quad s ; t \mapsto \mathcal{S}_{\mathbb{A}\mathbb{B}}(s) ; \mathcal{S}_{\mathbb{A}\mathbb{B}}(t)$$

where $! : 0 \rightarrow 1$ and $\mathfrak{i} : 1 \rightarrow 0$ are the arrows given by initiality and finality of 0. It can be checked that $\mathcal{S}_{\mathbb{A}\mathbb{B}}$ is well defined, as it respects the equations of $\mathbb{A}\mathbb{B}$.

² We can consider this as Hopf algebra with a trivial antipode.

Theorem 2 ([16]). $S_{\mathbb{A}\mathbb{B}} : \mathbb{A}\mathbb{B} \rightarrow \text{Mat } \mathbb{Z}_2$ is an isomorphism of PROPs.

Proof. Any circuit t is equivalent to one t' in matrix form (cf. Remark 4), whose matrix $S_{\mathbb{A}\mathbb{B}}(t')$ can be computed as in Example 2. In fact the encoding of Example 2 is a 1-1 correspondence between matrices and circuits. Then $S_{\mathbb{A}\mathbb{B}}$ is full and faithful and (as $\mathbb{A}\mathbb{B}$ and $\text{Mat } \mathbb{Z}_2$ have the same objects) thus an isomorphism. \square

Remark 5. Observe that $S_{\mathbb{A}\mathbb{B}}$ maps the circuits  and  to the same matrix, meaning that equation (ASep) is necessary to establish Theorem 2. On the other hand, the theory of bialgebras with (Sep) in place of (ASep) would yield as free PROP the one of finite sets and relations [3]. The intuition is that in the realm of \mathbb{Z}_2 -vector spaces a sum $v + v$ of a vector with itself is equal to 0, whereas for matrices representing relations $+$ is idempotent, i.e., $v + v = v$.

5 Composing Bialgebras

The PROPs $\mathbb{A}\mathbb{B}$ and $\mathbb{A}\mathbb{B}^{op}$ only describe the interaction between the white and black structures in $\mathbb{I}\mathbb{B}$. We now study their composition, so that the interaction between the two white and the two black structures may also be observed.

5.1 Cospans



First we obtain the PROP $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$ via a distributive law $\lambda_{po} : \text{Mat } \mathbb{Z}_2^{op} ; \text{Mat } \mathbb{Z}_2 \rightarrow \text{Mat } \mathbb{Z}_2 ; \text{Mat } \mathbb{Z}_2^{op}$ that maps a span in $\text{Mat } \mathbb{Z}_2$ into its pushout (cf. Example 1). The conclusion of Theorem 2 and the factorisation of circuits in $\mathbb{A}\mathbb{B}$ (Remark 4) allow us to understand λ_{po} as transforming circuits:

$$\begin{array}{ccc}
 \overbrace{\begin{array}{|c|c|} \hline \text{White} & \text{Black} \\ \hline \text{Comonoid} & \text{Monoid} \\ \hline \end{array}}^{\mathbb{A}\mathbb{B}^{op}} & \overbrace{\begin{array}{|c|c|} \hline \text{Black} & \text{White} \\ \hline \text{Comonoid} & \text{Monoid} \\ \hline \end{array}}^{\mathbb{A}\mathbb{B}} & \xrightarrow{\lambda_{po}} & \overbrace{\begin{array}{|c|c|} \hline \text{Black} & \text{White} \\ \hline \text{Comonoid} & \text{Monoid} \\ \hline \end{array}}^{\mathbb{A}\mathbb{B}} & \overbrace{\begin{array}{|c|c|} \hline \text{White} & \text{Black} \\ \hline \text{Comonoid} & \text{Monoid} \\ \hline \end{array}}^{\mathbb{A}\mathbb{B}^{op}} & (1)
 \end{array}$$

By Remark 1.(‡), a $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$ -algebra will consist of an $\mathbb{A}\mathbb{B}$ -algebra, an $\mathbb{A}\mathbb{B}^{op}$ -algebra and equations between them, given by pushouts of spans in $\mathbb{A}\mathbb{B}$. A free characterization of $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$ can be then given by calculating (in $\text{Mat } \mathbb{Z}_2$) those pushouts. Analogously to the case of $\text{Cospan}(\mathbb{F})$, it suffices to consider merely the few cases that we list below.

$$\begin{array}{cccc}
 \begin{array}{ccc} \bullet^* & 1 & \bullet \\ \swarrow & & \searrow \\ 0 & & 2 \\ \searrow & \wedge & \swarrow \\ \square & 1 & \square^* \end{array} & \begin{array}{ccc} \bullet^* & 1 & \bullet \\ \swarrow & & \searrow \\ 2 & & 0 \\ \searrow & \wedge & \swarrow \\ \square & 1 & \square^* \end{array} & \begin{array}{ccc} \bullet^* & 1 & \bullet \\ \swarrow & & \searrow \\ 2 & & 2 \\ \searrow & \wedge & \swarrow \\ \square & 3 & \square^* \end{array} & \begin{array}{ccc} \bullet^* & 1 & \bullet \\ \swarrow & & \searrow \\ 2 & & 2 \\ \searrow & \wedge & \swarrow \\ \square & 3 & \square^* \end{array} \\
 \begin{array}{ccc} \square^* & 3 & \square \\ \swarrow & & \searrow \\ 2 & & 2 \\ \searrow & \wedge & \swarrow \\ \square & 1 & \square^* \end{array} & \begin{array}{ccc} \square^* & 3 & \square \\ \swarrow & & \searrow \\ 2 & & 2 \\ \searrow & \wedge & \swarrow \\ \square & 1 & \square^* \end{array} & \begin{array}{ccc} \square^* & 2 & \square \\ \swarrow & & \searrow \\ 1 & & 1 \\ \searrow & \wedge & \swarrow \\ \square & 1 & \square^* \end{array} & (2)
 \end{array}$$

The diagrams of the first row yield (Q5) and (Q6) and Frobenius equations for the black structure. The second row implies that the white structure is a SFA.

Therefore, the interaction between $\mathbb{A}\mathbb{B}$ and $\mathbb{A}\mathbb{B}^{op}$ encoded by $\lambda_{p\circ}$ has the effect of adding Frobenius structure to $\mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2)$. In fact, all equations of the theory of interacting bialgebras are covered, with the notable exception of the equation (Sep) for the black structure, which we denote with (Sep_b) . Indeed, the two sides of (Sep_b) ,  and , denote different cospans in $\mathbf{Mat}\ \mathbb{Z}_2$:

$$1 \xrightarrow{\begin{pmatrix} 1 \\ 1 \end{pmatrix}} 2 \xleftarrow{\begin{pmatrix} 1 \\ 1 \end{pmatrix}^*} 1 \quad \neq \quad 1 \xrightarrow{(1)} 1 \xleftarrow{(1)^*} 1.$$


We call $\mathbb{I}\mathbb{B}^{-b}$ the free PROP for the theory of interacting bialgebras minus the equation (Sep_b) . Of the properties in Remark 2, (a), (c) and (d) also hold for $\mathbb{I}\mathbb{B}^{-b}$, whereas (b) does not hold because (Sep_b) is missing. Concerning property (e), (Zero_w) does not hold in $\mathbb{I}\mathbb{B}^{-b}$, as its proof requires (Sep_b) . Symmetrically, (Zero_b) is derivable, as $\mathbb{I}\mathbb{B}^{-b}$ has the white separability equation (Sep_w) .

Theorem 3. $\mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2) \cong \mathbb{I}\mathbb{B}^{-b}$.

As a result, $\mathbb{I}\mathbb{B}^{-b}$ enjoys the properties of composite PROPs. In particular, by Remark 1.(†) we have the following factorisation, where $\tau_1: \mathbb{A}\mathbb{B} \rightarrow \mathbb{I}\mathbb{B}^{-b}$ and $\tau_2: \mathbb{A}\mathbb{B}^{op} \rightarrow \mathbb{I}\mathbb{B}^{-b}$ denote the obvious inclusion maps.

Corollary 1 (Factorisation). *For every circuit $t \in \mathbb{I}\mathbb{B}^{-b}[n, m]$, there exist $z \in \mathbb{N}$, $t_1 \in \mathbb{A}\mathbb{B}[n, z]$ and $t_2 \in \mathbb{A}\mathbb{B}^{op}[z, m]$ such that $t = \tau_1(t_1) ; \tau_2(t_2)$.*

The decomposition of Corollary 1 is the one given in the right-hand side of (1).

Remark 6. The distributive laws for spans and cospans of finite sets [15] determine factorisation systems unique up-to “internal” permutation: i.e. if t factorises as $t_1 ; t_2$ and $t'_1 ; t'_2$ then there exists a permutation p such that $t_1 = t'_1 ; p$ and $p ; t_2 = t'_2$. The factorisation system of Corollary 1 is strictly weaker, being up-to “internal” isomorphism in $\mathbf{Mat}\ \mathbb{Z}_2$. These are all the invertible \mathbb{Z}_2 -matrices, not merely the permutations in $\mathbf{Mat}\ \mathbb{Z}_2$. For instance, the two rightmost diagrams in the first row of (2) give different (but isomorphic) decompositions of .

In order to make the isomorphism between $\mathbb{I}\mathbb{B}^{-b}$ and $\mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2)$ explicit, we define a semantics homomorphism $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}: \mathbb{I}\mathbb{B}^{-b} \rightarrow \mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2)$ extending that of $\mathbb{A}\mathbb{B}$ on $\mathbf{Mat}\ \mathbb{Z}_2$. It is defined inductively on circuits t in $\mathbb{I}\mathbb{B}^{-b}$ as follows³:

$$t \mapsto \begin{cases} \kappa_1(\mathcal{S}_{\mathbb{A}\mathbb{B}}(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}} \\ \kappa_2(\mathcal{S}_{\mathbb{A}\mathbb{B}^{op}}(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}^{op}} \\ \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}(t_1) ; \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}(t_2) & \text{if } t = t_1 ; t_2 \\ \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}(t_1) \otimes \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}(t_2) & \text{if } t = t_1 \otimes t_2 \end{cases}$$

where $\kappa_1: \mathbf{Mat}\ \mathbb{Z}_2 \rightarrow \mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2)$ and $\kappa_2: \mathbf{Mat}\ \mathbb{Z}_2^{op} \rightarrow \mathbf{Cospans}(\mathbf{Mat}\ \mathbb{Z}_2)$ are the canonical injections mapping $f \in \mathbf{Mat}\ \mathbb{Z}_2[n, m]$ and $g \in \mathbf{Mat}\ \mathbb{Z}_2^{op}[n, m]$ in

³ For the base cases, recall that the signature $\Sigma_{\mathbb{I}\mathbb{B}^{-b}}$ of $\mathbb{I}\mathbb{B}^{-b}$ is that of $\Sigma_{\mathbb{A}\mathbb{B}} \uplus \Sigma_{\mathbb{A}\mathbb{B}^{op}}$.

$n \xrightarrow{f} m \xleftarrow{id} m$ and $n \xrightarrow{id} n \xleftarrow{g^*} m$, respectively. The semantics is well-defined as all the equations of $\mathbb{I}\mathbb{B}^{-b}$ are sound w.r.t. $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}$.

Lemma 1. $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}: \mathbb{I}\mathbb{B}^{-b} \rightarrow \text{Cospan}(\text{Mat } \mathbb{Z}_2)$ is an isomorphism of PROPs.

Proof. By Corollary 1, any circuit of $\mathbb{I}\mathbb{B}^{-b}$ factorises as a cospan $n \xrightarrow{t_1} z \xleftarrow{t_2^*} m$ in $\mathbb{A}\mathbb{B}$. The statement then follows by Theorem 2.

5.2 Spans

Dually, a distributive law $\lambda_{pb}: \text{Mat } \mathbb{Z}_2 ; \text{Mat } \mathbb{Z}_2^{op} \rightarrow \text{Mat } \mathbb{Z}_2^{op} ; \text{Mat } \mathbb{Z}_2$ given by pullback yields a composite PROP $\text{Span}(\text{Mat } \mathbb{Z}_2) = \text{Mat } \mathbb{Z}_2^{op} ; \text{Mat } \mathbb{Z}_2$. The algebraic characterization of $\text{Span}(\text{Mat } \mathbb{Z}_2)$ follows the same steps as the one of $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$, albeit with the white and black structures swapped.

More formally, let $\mathbb{I}\mathbb{B}^{-w}$ be the free PROP on the theory of interacting bialgebras without the *white* separability equation (Sep_w). We define a semantics homomorphism $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}: \mathbb{I}\mathbb{B}^{-w} \rightarrow \text{Span}(\text{Mat } \mathbb{Z}_2)$ by induction on circuits t of $\mathbb{I}\mathbb{B}^{-w}$:

$$t \mapsto \begin{cases} \iota_1(\mathcal{S}_{\mathbb{A}\mathbb{B}}(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}} \\ \iota_2(\mathcal{S}_{\mathbb{A}\mathbb{B}}^{op}(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}}^{op} \\ \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}(t_1) ; \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}(t_2) & \text{if } t = t_1 ; t_2 \\ \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}(t_1) \otimes \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}(t_2) & \text{if } t = t_1 \otimes t_2 \end{cases}$$

where $\iota_1: \text{Mat } \mathbb{Z}_2 \rightarrow \text{Span}(\text{Mat } \mathbb{Z}_2)$ and $\iota_2: \text{Mat } \mathbb{Z}_2^{op} \rightarrow \text{Span}(\text{Mat } \mathbb{Z}_2)$ are the canonical injections mapping $f \in \text{Mat } \mathbb{Z}_2[n, m]$ and $g \in \text{Mat } \mathbb{Z}_2^{op}[n, m]$ in $n \xleftarrow{id} n \xrightarrow{f} m$ and $n \xleftarrow{g^*} m \xrightarrow{id} m$, respectively.

Lemma 2. $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}: \mathbb{I}\mathbb{B}^{-w} \rightarrow \text{Span}(\text{Mat } \mathbb{Z}_2)$ is an isomorphism of PROPs.

Proof. The proof relies on the transpose homomorphism $\xi: \text{Mat } \mathbb{Z}_2 \rightarrow \text{Mat } \mathbb{Z}_2^{op}$ mapping matrices to their transposes. This can be equivalently defined for the circuits in $\mathbb{A}\mathbb{B}$: taking the transpose of a circuit means to take its *photographic negative*, that is swapping of black and white structures. We call this homomorphism $\nu: \mathbb{A}\mathbb{B} \rightarrow \mathbb{A}\mathbb{B}^{op}$. Both ξ and ν are full and faithful and they can be extended to full and faithful homomorphisms $\xi': \text{Cospan}(\text{Mat } \mathbb{Z}_2) \rightarrow \text{Span}(\text{Mat } \mathbb{Z}_2)$ and $\nu': \mathbb{I}\mathbb{B}^{-w} \rightarrow \mathbb{I}\mathbb{B}^{-b}$. By a simple inductive argument, it holds that $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}} = \nu'$; $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}$; ξ' and therefore $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}$ is full and faithful. Since $\mathbb{I}\mathbb{B}^{-w}$ and $\text{Span}(\text{Mat } \mathbb{Z}_2)$ have the same objects, $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}$ is thus an isomorphism of PROPs. \square

As evident from the above, $\mathbb{I}\mathbb{B}^{-w} \cong \mathbb{I}\mathbb{B}^{-b}$ and $\text{Cospan}(\text{Mat } \mathbb{Z}_2) \cong \text{Span}(\text{Mat } \mathbb{Z}_2)$ (by self-duality of $\text{Mat } \mathbb{Z}_2$). This observation gives a straightforward proof that $\mathbb{I}\mathbb{B}^{-w} \cong \text{Span}(\text{Mat } \mathbb{Z}_2)$. However, our explicit characterization via $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}$ is instrumental in the construction of the next section.

6 The Cube

We now have all the ingredients in order to construct the diagram (\square) discussed in the Introduction and to prove Theorem 1.

The backward faces. By definitions of $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}}$ and $\mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}}$, the following diagram commutes, where $\sigma_1: \mathbb{A}\mathbb{B} \rightarrow \mathbb{I}\mathbb{B}^{-w}$ and $\sigma_2: \mathbb{A}\mathbb{B}^{op} \rightarrow \mathbb{I}\mathbb{B}^{-w}$ are inclusions.

$$\begin{array}{ccc}
 \mathbb{I}\mathbb{B}^{-b} & \xleftarrow{[\tau_1, \tau_2]} \mathbb{A}\mathbb{B} + \mathbb{A}\mathbb{B}^{op} \xrightarrow{[\sigma_1, \sigma_2]} & \mathbb{I}\mathbb{B}^{-w} \\
 \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}} \downarrow & \mathcal{S}_{\mathbb{A}\mathbb{B}} + \mathcal{S}_{\mathbb{A}\mathbb{B}^{op}} \downarrow & \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}} \downarrow \\
 \text{Cospan}(\text{Mat } \mathbb{Z}_2) & \xleftarrow{[\kappa_1, \kappa_2]} \text{Mat } \mathbb{Z}_2 + \text{Mat } \mathbb{Z}_2^{op} \xrightarrow{[\iota_1, \iota_2]} & \text{Span}(\text{Mat } \mathbb{Z}_2)
 \end{array} \quad (\text{Back})$$

The bottom face. Given a span $n \xleftarrow{f} z \xrightarrow{g} m$ and a cospan $n \xrightarrow{p} z \xleftarrow{q} m$, we define

$$\varphi(f, g) = \{(u, v) \mid \exists x \in \mathbb{Z}_2^z. fx = u, gx = v\} \quad \psi(p, q) = \{(u, v) \mid pu = qv\}.$$

It is easy to show that φ and ψ are homomorphisms and that the diagram

$$\begin{array}{ccc}
 \text{Mat } \mathbb{Z}_2 + \text{Mat } \mathbb{Z}_2^{op} & \xrightarrow{[\iota_1, \iota_2]} & \text{Span}(\text{Mat } \mathbb{Z}_2) \\
 \downarrow [\kappa_1, \kappa_2] & & \downarrow \varphi \\
 \text{Cospan}(\text{Mat } \mathbb{Z}_2) & \xrightarrow{\psi} & \mathbb{S}\mathbb{V}
 \end{array} \quad (\text{Bottom})$$

commutes. It is straightforward to verify that it is a pushout in **PROP**.

The top face. Take $\text{Sep}_w: \mathbb{I}\mathbb{B}^{-w} \rightarrow \mathbb{I}\mathbb{B}$ and $\text{Sep}_b: \mathbb{I}\mathbb{B}^{-b} \rightarrow \mathbb{I}\mathbb{B}$ to be the homomorphisms quotienting the arrows in $\mathbb{I}\mathbb{B}^{-w}$ and $\mathbb{I}\mathbb{B}^{-b}$ w.r.t. the equations (Sep_w) and (Sep_b) , respectively. It is immediate to see that the following diagram commutes.

$$\begin{array}{ccc}
 \mathbb{A}\mathbb{B} + \mathbb{A}\mathbb{B}^{op} & \xrightarrow{[\sigma_1, \sigma_2]} & \mathbb{I}\mathbb{B}^{-w} \\
 \downarrow [\tau_1, \tau_2] & & \downarrow \text{Sep}_w \\
 \mathbb{I}\mathbb{B}^{-b} & \xrightarrow{\text{Sep}_b} & \mathbb{I}\mathbb{B}
 \end{array} \quad (\text{Top})$$

To see that (Top) is a pushout, take any $\alpha: \mathbb{I}\mathbb{B}^{-w} \rightarrow \mathbb{C} \leftarrow \mathbb{I}\mathbb{B}^{-b}: \beta$ such that $[\sigma_1, \sigma_2]; \alpha = [\tau_1, \tau_2]; \beta$. The mediating homomorphism $\chi: \mathbb{I}\mathbb{B} \rightarrow \mathbb{C}$ is defined inductively on circuits t in $\mathbb{I}\mathbb{B}$ as follows:

$$t \mapsto \begin{cases} \alpha(\sigma_1(t)) = \beta(\tau_1(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}} \\ \alpha(\sigma_2(t)) = \beta(\tau_2(t)) & \text{if } t \in \Sigma_{\mathbb{A}\mathbb{B}^{op}} \\ \chi(t_1); \chi(t_2) & \text{if } t = t_1; t_2 \\ \chi(t_1) \otimes \chi(t_2) & \text{if } t = t_1 \otimes t_2 \end{cases} \quad (3)$$

This is well-defined as all equations of $\mathbb{I}\mathbb{B}$ hold in either $\mathbb{I}\mathbb{B}^{-w}$ or in $\mathbb{I}\mathbb{B}^{-b}$.

The front faces. By commutativity of (Back) and (Bottom), the universal property of (Top) induces an homomorphism making the following diagram commute.

$$\begin{array}{ccccc}
 \mathbb{I}\mathbb{B}^{-b} & \xrightarrow{Sep_b} & \mathbb{I}\mathbb{B} & \xleftarrow{Sep_w} & \mathbb{I}\mathbb{B}^{-w} \\
 \mathcal{S}_{\mathbb{I}\mathbb{B}^{-b}} \downarrow & & \vdots \downarrow & & \mathcal{S}_{\mathbb{I}\mathbb{B}^{-w}} \downarrow \\
 \text{Cospan}(\text{Mat } \mathbb{Z}_2) & \xrightarrow{\psi} & \mathbb{S}\mathbb{V} & \xleftarrow{\varphi} & \text{Span}(\text{Mat } \mathbb{Z}_2)
 \end{array} \tag{Front}$$

This homomorphism is defined as in (3). By induction, one can show that this is exactly $\mathcal{S}_{\mathbb{I}\mathbb{B}}$ in Definition 3. Fullness and faithfulness follow from fullness and faithfulness of the other semantics homomorphisms and from the fact that (Top) and (Bottom) are pushouts.

7 Conclusions

We have studied the theory of interacting bialgebras $\mathbb{I}\mathbb{B}$ which is relevant for both categorical quantum computing [9–11] and compositional models of concurrent systems [4, 5, 21]. We have shown that the PROP $\mathbb{S}\mathbb{V}$ of \mathbb{Z}_2 sub-vector spaces freely characterizes $\mathbb{I}\mathbb{B}$ and provided an inductive semantics which is useful for reasoning about equality of circuits in $\mathbb{I}\mathbb{B}$.

Most importantly, we have exhibited the modular structure of $\mathbb{I}\mathbb{B}$. The theory of antiseparable bialgebras $\mathbb{A}\mathbb{B}$ —freely characterized by $\text{Mat } \mathbb{Z}_2$, the PROP of \mathbb{Z}_2 -matrices—can be composed with its dual $\mathbb{A}\mathbb{B}^{op}$ in two different ways, resulting in two different, albeit isomorphic theories: $\mathbb{I}\mathbb{B}^{-w}$ and $\mathbb{I}\mathbb{B}^{-b}$. These have the same equations as $\mathbb{I}\mathbb{B}$ but without the white and the black separability axioms, respectively. The former is freely characterized by $\text{Span}(\text{Mat } \mathbb{Z}_2)$ and the latter by $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$. Finally, by gluing $\mathbb{I}\mathbb{B}^{-b}$ and $\mathbb{I}\mathbb{B}^{-w}$ we obtain $\mathbb{I}\mathbb{B}$ and, by gluing $\text{Span}(\text{Mat } \mathbb{Z}_2)$ and $\text{Cospan}(\text{Mat } \mathbb{Z}_2)$, we arrive at $\mathbb{S}\mathbb{V}$.

In fact, a similar story can be told in the simpler setting of the theory of monoids and \mathbb{F} (the PROP of functions) in place of $\mathbb{A}\mathbb{B}$ and $\text{Mat } \mathbb{Z}_2$. Following essentially the same script, one obtains in place of $\mathbb{I}\mathbb{B}^{-w}$ and $\mathbb{I}\mathbb{B}^{-b}$ the theory of bialgebras and the theory of SFAs, as shown in [15]. Instead of $\mathbb{S}\mathbb{V}$, one gets the PROP of equivalence relations over finite sets and, in place of $\mathbb{I}\mathbb{B}$, the gluing of the theories of bialgebras and SFAs which, as shown in [3], can be presented by the equations (Frob), (Sep) and (B4).

It is thus natural to ask whether this general pattern reoccurs in other settings. For example, we are interested in *sets and relations with contention* which, as shown in [22], are structures underlying the compositional semantics of C/E Petri nets. We are confident that, following the work of Lafont [16], our results can be generalized to vector spaces over arbitrary fields. Following in this direction, one could take aim at the ZX-calculus in its entirety.

Acknowledgment. The first and third author acknowledge support by project ANR 12IS02001 PACE.

References

1. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: LiCS 2004. IEEE Press (2004)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. *Math. Struct. Comp. Sci.* 14(3), 1–38 (2004)
3. Bruni, R., Gadducci, F.: Some algebraic laws for spans (and their connections with multi-relations). In: RelMiS 2001. Elsevier (2001)
4. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. *Theor. Comput. Sci.* 366, 98–120 (2006)
5. Bruni, R., Melgratti, H., Montanari, U.: A connector algebra for P/T nets interactions. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 312–326. Springer, Heidelberg (2011)
6. Bruni, R., Melgratti, H.C., Montanari, U., Sobociński, P.: Connector algebras for C/E and P/T nets' interactions. *Log. Meth. Comput. Sci.* (2013) (to appear)
7. Burroni, A.: Higher dimensional word problems with applications to equational logic. *Theor. Comput. Sci.* 115, 43–62 (1993)
8. Carboni, A., Walters, R.F.C.: Cartesian bicategories I. *J. Pure Appl. Algebra* 49, 11–32 (1987)
9. Coecke, B., Duncan, R.: Interacting quantum observables. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 298–310. Springer, Heidelberg (2008)
10. Coecke, B., Duncan, R., Kissinger, A., Wang, Q.: Strong complementarity and non-locality in categorical quantum mechanics. In: LiCS 2012, pp. 245–254 (2012)
11. Coecke, B., Kissinger, A.: Interacting Frobenius algebras and the structure of multipartite entanglement. Technical Report PGR-RR-09-12, Oxford (2009)
12. Fiore, M., Devesas Campos, M.: The algebra of directed acyclic graphs. In: Coecke, B., Ong, L., Panangaden, P. (eds.) Computation, Logic, Games and Quantum Foundations. LNCS, vol. 7860, pp. 37–51. Springer, Heidelberg (2013)
13. Katis, P., Sabadini, N., Walters, R.F.C.: Span(Graph): A Categorical algebra of transition systems. In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 307–321. Springer, Heidelberg (1997)
14. Kock, J.: Frobenius algebras and 2D topological quantum field theories. CUP (2003)
15. Lack, S.: Composing PROPs. *Theor. App. Categories* 13(9), 147–163 (2004)
16. Lafont, Y.: Towards an algebraic theory of boolean circuits. *J. Pure Appl. Alg.* 184, 257–310 (2003)
17. Mac Lane, S.: Categorical algebra. *Bull. Amer. Math. Soc.* 71, 40–106 (1965)
18. Mac Lane, S.: Categories for the Working Mathematician. Springer (1998)
19. Rosebrugh, R., Wood, R.J.: The formal theory of monads II. *J. Pure Appl. Algebra* 175(1), 327–353 (2002)
20. Selinger, P.: A survey of graphical languages for monoidal categories. arXiv:0908.3347v1 [math.CT] (2009)
21. Sobociński, P.: Representations of Petri net interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 554–568. Springer, Heidelberg (2010)
22. Sobociński, P.: Nets, relations and linking diagrams. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 282–298. Springer, Heidelberg (2013)
23. Stefanescu, G.: Network Algebra. Springer (2000)
24. Street, R.: The formal theory of monads. *J. Pure Appl. Algebra* 2(1), 243–265 (2002)

Generalized Eilenberg Theorem I: Local Varieties of Languages

Jiří Adámek¹, Stefan Milius², Robert S.R. Myers¹, and Henning Urbat¹

¹ Institut für Theoretische Informatik

Technische Universität Braunschweig, Germany

² Lehrstuhl für Theoretische Informatik

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Dedicated to Manuela Sobral

Abstract. We investigate the duality between algebraic and coalgebraic recognition of languages to derive a generalization of the local version of Eilenberg’s theorem. This theorem states that the lattice of all boolean algebras of regular languages over an alphabet Σ closed under derivatives is isomorphic to the lattice of all pseudovarieties of Σ -generated monoids. By applying our method to different categories, we obtain three related results: one, due to Gehrke, Grigorieff and Pin, weakens boolean algebras to distributive lattices, one due to Polák weakens them to join-semilattices, and the last one considers vector spaces over \mathbb{Z}_2 .

1 Introduction

Regular languages are precisely the behaviours of finite automata. A machine-independent characterization of regularity is the starting point of algebraic automata theory (see e.g. [10]): one defines recognition via preimages of monoid morphisms $f : \Sigma^* \rightarrow M$, where M is a finite monoid, and every regular language is recognized in this way by its syntactic monoid. A key result in this field is Eilenberg’s variety theorem, which establishes a lattice isomorphism

varieties of regular languages \cong pseudovarieties of monoids.

Here a *variety of regular languages* is a family of sets $V_\Sigma \subseteq \text{Reg}_\Sigma$, where Σ ranges over all finite alphabets and Reg_Σ are the regular languages over Σ , such that each V_Σ is closed under left and right derivatives¹ and boolean operations (union, intersection and complement), and moreover $\bigcup_\Sigma V_\Sigma$ is closed under preimages of monoid homomorphisms $\Sigma^* \rightarrow \Gamma^*$. And a *pseudovariety of monoids* is a set of finite monoids closed under finite products, submonoids and quotients (homomorphic images).

Recently Gehrke, Grigorieff and Pin [6, 7] proved a “local” version of Eilenberg’s theorem where one works with a *fixed* alphabet Σ : there is a lattice isomorphism between *local varieties of regular languages* (sets of regular languages over Σ closed

¹ Recall that the left and right derivatives of a language $L \subseteq \Sigma^*$ are the languages $w^{-1}L = \{u : wu \in L\}$ and $Lw^{-1} = \{u : uw \in L\}$ for $w \in \Sigma^*$, respectively.

under boolean operations and derivatives) and *local pseudovarieties of monoids* (sets of Σ -generated finite monoids closed under quotients and subdirect products). At the heart of this result lies the use of Stone duality to relate the boolean algebra Reg_Σ , equipped with left and right derivatives, to the free Σ -generated profinite monoid.

In this paper we generalize the local Eilenberg theorem to the level of an abstract duality of categories. Our approach is based on the observation that deterministic automata are coalgebras for the functor $T_\Sigma X = 2 \times X^\Sigma$ on sets, and that Reg_Σ can be captured categorically as the *rational fixpoint* ϱT_Σ of T_Σ , i.e., the terminal locally finite T_Σ -coalgebra [9]. The rational fixpoint ϱT exists more generally for every finitary endofunctor T on a locally finitely presentable category \mathcal{C} [1]. In this paper we work with such a category \mathcal{C} and its dual $\hat{\mathcal{D}} \cong \mathcal{C}^{op}$. The functor $T_\Sigma X = 2 \times X^\Sigma$ on \mathcal{C} (where 2 is a fixed \mathcal{C} -object) has the dual endofunctor $\hat{L}_\Sigma X = \mathbb{1} + \coprod_\Sigma X$ on $\hat{\mathcal{D}}$ (where $\mathbb{1}$ is dual to 2), so that T_Σ -coalgebras correspond to \hat{L}_Σ -algebras. This already gives an equivalent description of (possibly infinite) automata as algebras. However, we are mainly interested in *finite* automata, and so we will work with another category \mathcal{D} – a “finitary approximation” of $\hat{\mathcal{D}}$ – and an endofunctor L_Σ on \mathcal{D} induced by \hat{L}_Σ . Finite automata are then modeled either as T_Σ -coalgebras or L_Σ -algebras with finitely presentable carrier, shortly *fp-(co)algebras*. As a first approximation to the local Eilenberg theorem, we establish a lattice isomorphism

$$\text{subcoalgebras of } \varrho T_\Sigma \cong \text{ideal completion of the poset of fp-quotient algebras of } \mu L_\Sigma$$

where μL_Σ is L_Σ 's initial algebra. This is “almost” the desired general local Eilenberg theorem. For the classical case one takes Stone duality (\mathcal{C} = boolean algebras, $\hat{\mathcal{D}}$ = Stone spaces). Then \mathcal{D} = sets, ϱT_Σ is the boolean algebra Reg_Σ , $L_\Sigma = 1 + \coprod_\Sigma \text{Id}$ on sets and $\mu L_\Sigma = \Sigma^*$. The above isomorphism states that the boolean subalgebras of Reg_Σ closed under *left* derivatives correspond to sets of finite quotient algebras of Σ^* closed under quotients and subdirect products. What is missing is the closure under *right* derivatives on the coalgebra side, and quotient algebras of Σ^* which are *monoids* on the algebra side.

The final step is to prove that the above isomorphism restricts to one between local varieties of regular languages (= subcoalgebras of ϱT_Σ closed under right derivatives) and local pseudovarieties of monoids. For this purpose we introduce the concept of a bimonoid. If \mathcal{D} is a concrete category with forgetful functor $|\cdot| : \mathcal{D} \rightarrow \text{Set}$, then a *bimonoid* is a \mathcal{D} -object A equipped with a “bilinear” monoid multiplication \circ on $|A|$, which means that the maps $a \circ -$ and $- \circ a$ carry \mathcal{D} -morphisms for all $a \in |A|$. For example, bimonoids in \mathcal{D} = sets, posets, join-semilattices and vector spaces over \mathbb{Z}_2 are monoids, ordered monoids, idempotent semirings and \mathbb{Z}_2 -algebras (in the sense of algebras over a field), respectively. Our General Local Eilenberg Theorem (Theorem 5.19) holds on this level of generality: if \mathcal{C} and \mathcal{D} are concrete categories satisfying some natural properties, there is a lattice isomorphism

$$\text{local varieties of regular languages in } \mathcal{C} \cong \text{local pseudovarieties of bimonoids in } \mathcal{D}.$$

This is the main result of our paper. By instantiating it to Stone duality (\mathcal{C} = boolean algebras, $\hat{\mathcal{D}}$ = Stone spaces, \mathcal{D} = sets) we recover the “classical” local Eilenberg theorem. Priestley duality (\mathcal{C} = distributive lattices, $\hat{\mathcal{D}}$ = Priestley spaces, \mathcal{D} = posets)

gives another result of Gehrke et. al, namely a lattice isomorphism between *local lattice varieties of regular languages* (subsets of Reg_Σ closed under union, intersection and derivatives) and local pseudovarieties of ordered monoids. Finally, by taking $\mathcal{C} =$ join-semilattices and $\mathcal{C} =$ vector spaces over \mathbb{Z}_2 , we obtain two new local Eilenberg theorems. The first one establishes a lattice isomorphism between *local semilattice varieties of regular languages* (subsets of Reg_Σ closed under union and derivatives) and local pseudovarieties of idempotent semirings, and the second one gives an isomorphism between *local linear varieties of regular languages* (subsets of Reg_Σ closed under symmetric difference and derivatives) and local pseudovarieties of \mathbb{Z}_2 -algebras.

Related work. Our paper is inspired by the work of Gehrke, Grigorieff and Pin [6] who showed that the algebraic operation of the free profinite monoid on Σ dualizes to the derivative operations on the boolean algebra of regular languages (and similarly for the free ordered profinite monoid on Σ). Previously, the duality between the boolean algebra of regular languages and the Stone space of profinite words appeared (implicitly) in work by Almeida [3] and was formulated by Pippenger [11] in terms of Stone duality.

A categorical approach to the duality theory of regular languages has been suggested by Rhodes and Steinberg [14]. They introduce the notion of a boolean bialgebra, which is conceptually rather different from our bimonoids, and prove the equivalence of bialgebras and profinite semigroups. The precise connection to their work is yet to be understood.

Another related work is Polák [12]. He considered what we treat as the example of join-semilattices and obtained a (non-local) Eilenberg type theorem in this case. To the best of our knowledge the local version we prove does not follow from the global version, and so we believe that our result is new.

The origin of all the above work is, of course, Eilenberg's theorem [4]. Later Reiterman [13] proved another characterization of pseudovarieties of monoids in the spirit of Birkhoff's classical variety theorem. Reiterman's theorem states that any pseudovariety of monoids can be characterized by profinite equations (i.e., pairs of elements of a free profinite monoid). We do not treat profinite equations in the present paper.

2 The Rational Fixpoint

The aim of this section is to recall the rational fixpoint of a functor, which provides a coalgebraic view of the set of regular languages. As a prerequisite, we need a categorical notion of "finite automaton", and so we will work with categories where "finite" objects exist and are well-behaved – viz. *locally finitely presentable* categories [2].

Definition 2.1. (a) An object X of a category \mathcal{C} is *finitely presentable* if the hom-functor $\mathcal{C}(X, -) : \mathcal{C} \rightarrow \text{Set}$ is finitary (i.e., preserves filtered colimits). Let \mathcal{C}_{fp} denote the full subcategory of all finitely presentable objects of \mathcal{C} .

(b) \mathcal{C} is *locally finitely presentable* if it is cocomplete, \mathcal{C}_{fp} is small up to isomorphism and every object of \mathcal{C} is a filtered colimit of finitely presentable objects.

(c) \mathcal{C} is *locally finitely super-presentable* if it is locally finitely presentable and \mathcal{C}_{fp} is closed under finite products, subobjects (= monos) and quotients (= epis).

Example 2.2. The categories in the table below are locally finitely super-presentable. In each case, the finitely presentable objects are precisely the finite ones.

\mathcal{C}	objects	morphisms
Set	sets	functions
BA	boolean algebras	boolean morphisms
DL ₀₁	distributive lattices with 0 and 1	lattice morphisms preserving 0 and 1
JSL ₀	join-semilattices with 0	semilattice morphisms preserving 0
Vect \mathbb{Z}_2	vector spaces over the field \mathbb{Z}_2	linear maps
Pos	partially ordered sets	monotone functions

In contrast to DL₀₁, the category of lattices is not locally finitely super-presentable: a finitely generated lattice can have sublattices that are not finitely generated.

Definition 2.3. An endofunctor $T : \mathcal{C} \rightarrow \mathcal{C}$ is *strongly finitary* if it is finitary and preserves finitely presentable objects, i.e., $T[\mathcal{C}_{fp}] \subseteq \mathcal{C}_{fp}$.

Example 2.4. (a) If \mathcal{C} is locally finitely super-presentable, then the functor

$$T_\Sigma = 2 \times \text{Id}^\Sigma = 2 \times \text{Id} \times \text{Id} \times \dots \times \text{Id}$$

where Σ is a finite alphabet and 2 is a finitely presentable object of \mathcal{C} is strongly finitary. T_Σ -coalgebras are deterministic automata, see e.g. [15]. Indeed, by the universal property of the product, to give a morphism $Q \rightarrow T_\Sigma Q = 2 \times Q^\Sigma$ means precisely to give an object Q (of states), morphisms $\delta_a : Q \rightarrow Q$ for every $a \in \Sigma$ (representing a -transitions) and a morphism $f : Q \rightarrow 2$ (representing final states). The usual concept of a deterministic automaton (without initial states) is captured as a coalgebra for T_Σ where $\mathcal{C} = \text{Set}$ and $2 = \{0, 1\}$. An important example of a T_Σ -coalgebra is the automaton Reg_Σ of regular languages. Its states are the regular languages over Σ , its transitions are

$$\delta_a(L) = a^{-1}L \quad \text{for all } L \in \text{Reg}_\Sigma \text{ and } a \in \Sigma,$$

and the final states are precisely the languages containing the empty word ε .

- (b) Analogously, consider T_Σ as an endofunctor of $\mathcal{C} = \text{BA}$ with $2 = \{0, 1\}$ (the two-element boolean algebra). A coalgebra for T_Σ is a deterministic automaton with a boolean algebra structure on the state set Q . Moreover, the transition maps $\delta_a : Q \rightarrow Q$ are boolean homomorphisms, and the final states (given by the inverse image of 1 under $f : Q \rightarrow 2$) form an ultrafilter. The above automaton Reg_Σ is also a T_Σ -coalgebra in BA: the set of regular languages is a boolean algebra w.r.t. the usual set-theoretic operations, left derivatives preserve these operations, and the languages containing ε form a principal ultrafilter.
- (c) Mealy automata with output object 2 are coalgebras for the strongly finitary functor $T = (2 \times \text{Id})^\Sigma$.
- (d) Nondeterministic automata in $\mathcal{C} = \text{Set}$ are coalgebras for the strongly finitary functor $TQ = 2 \times (\mathcal{P}_f Q)^\Sigma$ where \mathcal{P}_f is the finite powerset functor and $2 = \{0, 1\}$.

Notation 2.5. $\text{Coalg } T$ denotes the category of all T -coalgebras and their homomorphisms, and $\text{Coalg}_{fp} T$ denotes the full subcategory of all *fp-coalgebras*, i.e., coalgebras $Q \rightarrow TQ$ with finitely presentable carrier Q .

Remark 2.6. If \mathcal{C} is locally finitely presentable and $T : \mathcal{C} \rightarrow \mathcal{C}$ is finitary, let

$$r : \varrho T \rightarrow T(\varrho T)$$

be the filtered colimit of all fp-coalgebras, i.e., the colimit of the diagram $\text{Coalg}_{fp} T \hookrightarrow \text{Coalg} T$. As shown in [1], ϱT is a fixpoint of T , i.e. r is an isomorphism.

Definition 2.7. ϱT is called the *rational fixpoint* of T .

Example 2.8. The rational fixpoint of $T_\Sigma : \text{Set} \rightarrow \text{Set}$ is the automaton $\varrho T_\Sigma = \text{Reg}_\Sigma$ of all regular languages over Σ , see Example 2.4(a). Analogously, the functor $T_\Sigma : \text{BA} \rightarrow \text{BA}$ has the rational fixpoint $\varrho T_\Sigma = \text{Reg}_\Sigma$.

Definition 2.9 (see [9]). A coalgebra is called *locally finitely presentable* if it is a filtered colimit of fp-coalgebras. $\text{Coalg}_{lfp} T$ denotes the full subcategory of $\text{Coalg} T$ of all locally finitely presentable coalgebras. Hence $\text{Coalg}_{fp} T \subseteq \text{Coalg}_{lfp} T \subseteq \text{Coalg} T$.

Example 2.10. A Σ -automaton in Set is locally finitely presentable iff, for every state q , the set of all states reachable from q is finite.

Remark 2.11. (a) Recall the *free completion* $\mathcal{A} \hookrightarrow \text{Ind } \mathcal{A}$ of a small category \mathcal{A} under filtered colimits: it is characterized up to equivalence by the property that $\text{Ind } \mathcal{A}$ has filtered colimits and every functor $F : \mathcal{A} \rightarrow \mathcal{B}$ into a category \mathcal{B} with filtered colimits has an essentially unique finitary extension $\overline{F} : \text{Ind } \mathcal{A} \rightarrow \mathcal{B}$. If \mathcal{A} has finite colimits then $\text{Ind } \mathcal{A}$ is locally finitely presentable and $(\text{Ind } \mathcal{A})_{fp} \cong \mathcal{A}$. Conversely, every locally finitely presentable category \mathcal{C} arises in this way: $\mathcal{C} \cong \text{Ind}(\mathcal{C}_{fp})$.

(b) If \mathcal{A} is a join-semilattice then $\text{Ind } \mathcal{A}$ is its ideal completion, see Remark 4.3.

Theorem 2.12. Let $T : \mathcal{C} \rightarrow \mathcal{C}$ be a finitary endofunctor of a locally finitely presentable category \mathcal{C} .

(a) ϱT is the terminal object of $\text{Coalg}_{lfp} T$, i.e., the terminal locally finitely presentable T -coalgebra [9].

(b) $\text{Coalg}_{lfp} T$ is the Ind -completion of $\text{Coalg}_{fp} T$.

3 The Dual of the Rational Fixpoint

At the heart of our main results lies the investigation of a duality for our categories of interest (e.g. Stone duality for BA and Priestley duality for DL_{01}) and the induced algebra-coalgebra duality.

Assumptions 3.1. Throughout the rest of the paper we work with

(a) a locally finitely super-presentable category \mathcal{C} ,

(b) a dual category $\hat{\mathcal{D}}$ with an equivalence functor $P : \hat{\mathcal{D}} \xrightarrow{\cong} \mathcal{C}^{op}$, such that the category

$$\mathcal{D} = \text{Ind}(\mathcal{C}_{fp}^{op})$$

is locally finitely super-presentable, and

(c) a strongly finitary functor $T : \mathcal{C} \rightarrow \mathcal{C}$ preserving monomorphisms.

Example 3.2. In our applications we will work with the automata functor $T = T_\Sigma : \mathcal{C} \rightarrow \mathcal{C}$ from Example 2.4 and with the following categories:

\mathcal{C}	$\hat{\mathcal{D}}$	\mathcal{D}
BA	Stone	Set
DL ₀₁	Priest	Pos
JSL ₀	JSL ₀ in Stone	JSL ₀
Vect \mathbb{Z}_2	Vect \mathbb{Z}_2 in Stone	Vect \mathbb{Z}_2

(a) For the category $\mathcal{C} = \text{BA}$ we have the classical Stone duality: $\hat{\mathcal{D}}$ is the category Stone of Stone spaces (i.e., compact Hausdorff spaces with a base of clopen sets) and continuous maps. The equivalence functor $P : \text{Stone} \rightarrow \text{BA}^{op}$ assigns to each Stone space the boolean algebra of clopen sets, and its associated equivalence $P^{-1} : \text{BA}^{op} \rightarrow \text{Stone}$ assigns to each boolean algebra the Stone space of all ultrafilters. Since Stone duality restricts to a dual equivalence $\text{BA}_{fp}^{op} \cong \text{Set}_{fp}$, we have

$$\mathcal{D} = \text{Ind}(\text{BA}_{fp}^{op}) \cong \text{Ind}(\text{Set}_{fp}) \cong \text{Set}.$$

(b) For the category $\mathcal{C} = \text{DL}_{01}$ we have the classical Priestley duality: $\hat{\mathcal{D}}$ is the category Priest of Priestley spaces (i.e., ordered Stone spaces such that given $x \not\leq y$ there is a clopen set containing x but not y) and continuous monotone maps. The equivalence functor $P : \text{Priest} \rightarrow \text{DL}_{01}^{op}$ assigns to each Priestley space the lattice of all clopen upsets, and its associated equivalence $P^{-1} : \text{DL}_{01}^{op} \rightarrow \text{Priest}$ assigns to each distributive lattice the Priestley space of all prime filters. Since Priestley duality restricts to a dual equivalence $(\text{DL}_{01})_{fp}^{op} \cong \text{Pos}_{fp}$, we have

$$\mathcal{D} = \text{Ind}((\text{DL}_{01})_{fp}^{op}) \cong \text{Ind}(\text{Pos}_{fp}) \cong \text{Pos}.$$

(c) For $\mathcal{C} = \text{JSL}_0$ the dual category $\hat{\mathcal{D}}$ is the category of join-semilattices in Stone, see [8]. Using the self-duality $(\text{JSL}_0)_{fp}^{op} \cong (\text{JSL}_0)_{fp}$ we obtain

$$\mathcal{D} = \text{Ind}((\text{JSL}_0)_{fp}^{op}) \cong \text{Ind}((\text{JSL}_0)_{fp}) \cong \text{JSL}_0.$$

(d) For $\mathcal{C} = \text{Vect } \mathbb{Z}_2$ the dual category $\hat{\mathcal{D}}$ is the category of \mathbb{Z}_2 -vector spaces in Stone, see [8]. The self-duality $(\text{Vect } \mathbb{Z}_2)_{fp}^{op} \cong (\text{Vect } \mathbb{Z}_2)_{fp}$ yields

$$\mathcal{D} = \text{Ind}((\text{Vect } \mathbb{Z}_2)_{fp}^{op}) \cong \text{Ind}((\text{Vect } \mathbb{Z}_2)_{fp}) \cong \text{Vect } \mathbb{Z}_2.$$

Remark 3.3. (a) Dually to Definition 2.1, an object X of $\hat{\mathcal{D}}$ is called *cofinitely presentable* if the hom-functor $\hat{\mathcal{D}}(-, X) : \hat{\mathcal{D}}^{op} \rightarrow \text{Set}$ preserves filtered colimits. The full subcategory of all cofinitely presentable objects is denoted by $\hat{\mathcal{D}}_{cfp}$. Since $\mathcal{C}_{fp}^{op} \cong \hat{\mathcal{D}}_{cfp}$ we have $\mathcal{D} = \text{Ind}(\mathcal{C}_{fp}^{op}) \cong \text{Ind}(\hat{\mathcal{D}}_{cfp})$.

(b) The dual of Ind is denoted by Pro : if \mathcal{A} is a small category, then $\text{Pro } \mathcal{A}$ is its free completion under cofiltered limits. By duality, $\text{Pro } \mathcal{A} \cong (\text{Ind } \mathcal{A}^{op})^{op}$.

Example 3.4. (a) For the category Set_{fp} of finite sets, we have $\text{Pro}(\text{Set}_{fp}) \cong \text{Stone}$. Indeed, Stone duality restricts to a duality between Set_{fp} (= finite Stone spaces) and BA_{fp} , so

$$\text{Pro}(\text{Set}_{fp}) \cong \text{Pro}(\text{BA}_{fp}^{op}) \cong (\text{Ind}(\text{BA}_{fp}))^{op} \cong \text{BA}^{op} \cong \text{Stone}.$$

(b) Analogously $\text{Pro}(\text{Pos}_{fp}) \cong \text{Priest}$.

Definition 3.5. We denote by $\hat{L} : \hat{\mathcal{D}} \rightarrow \hat{\mathcal{D}}$ the dual of the functor $T : \mathcal{C} \rightarrow \mathcal{C}$, i.e., the essentially unique functor with $P\hat{L} = T^{op}P$.

Remark 3.6. The categories $\text{Alg } \hat{L}$ and $\text{Coalg } T$ are dually equivalent. Indeed, the equivalence functor $P : \hat{\mathcal{D}} \rightarrow \mathcal{C}^{op}$ induces an equivalence functor

$$\bar{P} : \text{Alg } \hat{L} \rightarrow (\text{Coalg } T)^{op}, \quad (\hat{L}Z \xrightarrow{z} Z) \mapsto (PZ \xrightarrow{Pz} P\hat{L}Z = TPZ).$$

Example 3.7. The dual of $T_\Sigma X = 2 \times X^\Sigma = 2 \times \prod_\Sigma X : \mathcal{C} \rightarrow \mathcal{C}$, see Example 2.4, is the endofunctor of $\hat{\mathcal{D}}$

$$\hat{L}_\Sigma Z = \mathbb{1} + \coprod_\Sigma Z$$

where $\mathbb{1} = P^{-1}2$. In $\hat{\mathcal{D}} = \text{Stone}$ the object $\mathbb{1}$ is the one-element space. Hence, by the universal property of the coproduct, an \hat{L}_Σ -algebra $\hat{L}_\Sigma Z = \mathbb{1} + \coprod_\Sigma Z \rightarrow Z$ is a deterministic Σ -automaton (without final states) in Stone, given by a Stone space Z of states, continuous transition functions $\delta_a : Z \rightarrow Z$ for $a \in \Sigma$, and an initial state $\mathbb{1} \rightarrow Z$. Analogously for the other dualities of Example 3.2.

Remark 3.8. By the dual of Assumption 3.1(c), the functor \hat{L} is *strongly cofinitary*, i.e., it preserves cofiltered limits and cofinitely presentable objects. In particular, \hat{L} restricts to a functor

$$\hat{L}_{cfp} : \hat{\mathcal{D}}_{cfp} \rightarrow \hat{\mathcal{D}}_{cfp}.$$

Definition 3.9. The essentially unique finitary extension of the functor

$$\hat{\mathcal{D}}_{cfp} \xrightarrow{\hat{L}_{cfp}} \hat{\mathcal{D}}_{cfp} \hookrightarrow \text{Ind}(\hat{\mathcal{D}}_{cfp}) = \mathcal{D}$$

is denoted by $L : \mathcal{D} \rightarrow \mathcal{D}$. It takes a formal filtered colimit to the actual colimit in \mathcal{D} .

Example 3.10. For $\hat{L}_\Sigma Z = \mathbb{1} + \coprod_\Sigma Z$ on $\hat{\mathcal{D}}$ (see Example 3.7) we get the endofunctor of \mathcal{D}

$$L_\Sigma Z = \mathbb{1} + \coprod_\Sigma Z.$$

Here $\mathbb{1} = P^{-1}2 \in \hat{\mathcal{D}}_{cfp}$ is an object of $\mathcal{D} = \text{Ind}(\hat{\mathcal{D}}_{cfp})$. For $\hat{\mathcal{D}} = \text{Stone}$ we have $\mathcal{D} = \text{Set}$, so L_Σ -algebras are the classical deterministic automata without final states. Analogously for the other dualities of Example 3.2.

Notation 3.11. The category of all \hat{L} -algebras with a cofinitely presentable carrier (shortly *cfp-algebras*) is denoted by $\text{Alg}_{cfp} \hat{L}$.

- Example 3.12.** (a) If $\hat{\mathcal{D}} = \text{Stone}$, we have $\hat{\mathcal{D}}_{cfp} \cong \text{BA}_{fp}^{op} \cong \text{Set}_{fp}$, so cfp-algebras for \hat{L}_Σ are the classical deterministic finite automata without final states.
 (b) If $\hat{\mathcal{D}} = \text{Priest}$, since $\hat{\mathcal{D}}_{cfp} = \text{DL}_{01,fp}^{op} \cong \text{Pos}_{fp}$, cfp-algebras for \hat{L}_Σ are precisely the deterministic finite *ordered* automata without final states.

Definition 3.13. An \hat{L} -algebra is called *locally cofinitely presentable* if it is a cofiltered limit of cfp-algebras.

Remark 3.14. The category of all locally cofinitely presentable algebras is equivalent to $\text{Pro}(\text{Alg}_{cfp} \hat{L})$. This is the dual of Theorem 2.12. The initial object $\tau \hat{L}$ is what one can call the *dual of the rational fixpoint*. By the dual of Remark 2.6, one can construct $\tau \hat{L}$ as the limit of all cfp-algebras in $\text{Alg} \hat{L}$, and $\tau \hat{L}$ is a fixpoint of \hat{L} .

- Example 3.15.** (a) For $\mathcal{C} = \text{BA}$ and $\hat{\mathcal{D}} = \text{Stone}$, we have $\tau \hat{L}_\Sigma =$ ultrafilters of regular languages.
 (b) Analogously, for $\mathcal{C} = \text{DL}_{01}$ and $\hat{\mathcal{D}} = \text{Priest}$, we have $\tau \hat{L}_\Sigma =$ prime filters of regular languages.

Definition 3.16. We denote by $F : \mathcal{D} \rightarrow \hat{\mathcal{D}}$ the unique finitary functor for which

$$\begin{array}{ccc} & \hat{\mathcal{D}}_{cfp} \mathcal{C} & \\ \swarrow & & \searrow \\ \mathcal{D} = \text{Ind}(\hat{\mathcal{D}}_{cfp}) & \xrightarrow{F} & \text{Pro}(\hat{\mathcal{D}}_{cfp}) = \hat{\mathcal{D}} \end{array}$$

commutes, and by $U : \hat{\mathcal{D}} \rightarrow \mathcal{D}$ the unique cofinitary functor for which

$$\begin{array}{ccc} & \hat{\mathcal{D}}_{cfp} \mathcal{C} & \\ \swarrow & & \searrow \\ \hat{\mathcal{D}} = \text{Pro}(\hat{\mathcal{D}}_{cfp}) & \xrightarrow{U} & \text{Ind}(\hat{\mathcal{D}}_{cfp}) = \mathcal{D} \end{array}$$

commutes.

Lemma 3.17. *The functors F and U are well-defined and F is a left adjoint to U .*

- Example 3.18.** 1. If $\mathcal{C} = \text{BA}$ then $\hat{\mathcal{D}} = \text{Stone}$ and $\mathcal{D} = \text{Set}$. Then $F : \text{Set} \rightarrow \text{Stone}$ is the Stone-Ćech compactification and $U : \text{Stone} \rightarrow \text{Set}$ is the forgetful functor.
 2. If $\mathcal{C} = \text{DL}_{01}$ then $\hat{\mathcal{D}} = \text{Priest}$ and $\mathcal{D} = \text{Pos}$. Then $F : \text{Pos} \rightarrow \text{Priest}$ constructs the free Priestley space on a poset and $U : \text{Priest} \rightarrow \text{Pos}$ is the forgetful functor.

Notation 3.19. $\text{Alg}_{fp} L$ is the full subcategory of $\text{Alg} L$ of L -algebras with finitely presentable carrier, shortly *fp-algebras*. Note that $\text{Alg}_{fp} L \cong \text{Alg}_{cfp} \hat{L}$ because $\hat{\mathcal{D}}_{cfp} \cong \mathcal{D}_{fp}$.

Definition 3.20. $\hat{U} : \text{Pro}(\text{Alg}_{cfp} \hat{L}) \rightarrow \text{Alg} L$ is the unique cofinitary functor that makes the triangle below commute:

$$\begin{array}{ccc} & \text{Alg}_{cfp} \hat{L} \cong \text{Alg}_{fp} L & \\ \swarrow & & \searrow \\ \text{Pro}(\text{Alg}_{cfp} \hat{L}) & \xrightarrow{\hat{U}} & \text{Alg} L \end{array}$$

Example 3.21. For $T_\Sigma = 2 \times \text{Id}^\Sigma : \text{BA} \rightarrow \text{BA}$ we have $\hat{L}_\Sigma = \mathbb{1} + \coprod_\Sigma \text{Id} : \text{Stone} \rightarrow \text{Stone}$ and $L_\Sigma = \mathbb{1} + \coprod_\Sigma \text{Id} : \text{Set} \rightarrow \text{Set}$. The objects of $\text{Pro}(\text{Alg}_{c\text{fp}} \hat{L}_\Sigma)$ are the locally cofinitely presentable \hat{L}_Σ -algebras, and the functor $\hat{U} : \text{Pro}(\text{Alg}_{c\text{fp}} \hat{L}_\Sigma) \rightarrow \text{Alg } L_\Sigma$ simply forgets the topology on the carrier of an L_Σ -algebra.

Proposition 3.22. \hat{U} is a right adjoint.

Remark 3.23. It follows that the left adjoint \hat{F} of \hat{U} maps the initial L -algebra to the initial locally cofinitely presentable \hat{L} -algebra: $\hat{F}(\mu L) = \tau \hat{L}$. One can prove that \hat{F} assigns to every L -algebra $\alpha : LA \rightarrow A$ the limit of the diagram of all its quotients in $\text{Alg}_{\text{fp}} L = \text{Alg}_{c\text{fp}} \hat{L}$. Thus, we see that $\tau \hat{L}$ can be constructed as the limit (taken in $\text{Alg } \hat{L}$) of all finite quotient L -algebras of μL . This construction generalizes a similar one given by Gehrke [5]. See also Section 5.1.

4 Algebraic and Coalgebraic Recognition

We are ready to present our first take on the duality of algebraic and coalgebraic recognition and Eilenberg’s theorem (see Proposition 4.2 and Theorem 4.4 below). At this stage our results are about subcoalgebras of the rational fixpoint ϱT and quotients of the initial L -algebra μL , and we obtain uniform proofs at the level of generality of the previous section. Recall that we have the following dualities:

Category	Equivalently	Dual category	Equivalently
\mathcal{C}		$\hat{\mathcal{D}}$	
$\text{Coalg } T$		$\text{Alg } \hat{L}$	
$\text{Coalg}_{\text{fp}} T$		$\text{Alg}_{c\text{fp}} \hat{L}$	$\text{Alg}_{\text{fp}} L$
$\text{Coalg}_{\text{lfp}} T$	$\text{Ind}(\text{Coalg}_{\text{fp}} T)$	$\text{Pro}(\text{Alg}_{c\text{fp}} T)$	

- Definition 4.1.** (a) By a *subcoalgebra* of a T -coalgebra (C, γ) is meant one represented by a homomorphism $m : (C', \gamma') \rightarrow (C, \gamma)$ with m monic in \mathcal{C} . Subcoalgebras are ordered as usual: $m \leq \bar{m}$ iff m factorizes through \bar{m} in $\text{Coalg } T$. We denote by $\text{Sub}(\varrho T)$ the poset of all subcoalgebras of ϱT , and by $\text{Sub}_{\text{fp}}(\varrho T)$ the subposet of all *fp-subcoalgebras* of ϱT , i.e., those with finitely presentable carrier in \mathcal{C} .
- (b) Likewise, a *quotient algebra* of an L -algebra (A, α) is one represented by an epicarried homomorphism $e : (A, \alpha) \rightarrow (A', \alpha')$. Again the ordering is $e \leq \bar{e}$ iff e factorizes through \bar{e} in $\text{Alg } L$ (so $\text{id}_{(A, \alpha)}$ is the largest quotient). We denote by $\text{Quo}(\mu L)$ the poset of all quotient algebras of μL , and by $\text{Quo}_{\text{fp}}(\mu L)$ the subposet of all *fp-quotient algebras*, i.e., those with finitely presentable carrier in \mathcal{D} .

Proposition 4.2. The posets $\text{Sub}_{\text{fp}}(\varrho T)$ and $\text{Quo}_{\text{fp}}(\mu L)$ are isomorphic.

Proof (Sketch). The inverse $\bar{P}^{-1} : (\text{Coalg } T)^{\text{op}} \rightarrow \text{Alg } \hat{L}$ of \bar{P} in Remark 3.6 assigns to each T -coalgebra $C \xrightarrow{\gamma} TC$ the \hat{L} -algebra $\hat{L}(P^{-1}C) = P^{-1}(TC) \xrightarrow{P^{-1}\gamma} P^{-1}C$. If (C, γ) is an fp-coalgebra, the L -algebra $\bar{P}^{-1}(C, \gamma)$ has a cofinitely presentable carrier in $\hat{\mathcal{D}}$ and (since $\hat{\mathcal{D}}_{c\text{fp}} = \mathcal{D}_{\text{fp}}$) can be viewed as an fp-algebra for L . We denote by

$$(C, \gamma) \xrightarrow{m} \varrho T \quad \text{and} \quad \mu L \xrightarrow{e} \bar{P}^{-1}(C, \gamma)$$

the unique homomorphisms, and prove that

$$m \text{ is monic (in } \mathcal{C}) \text{ iff } e \text{ is epic (in } \mathcal{D}),$$

using the (epi, strong mono)- and (strong epi, mono)-factorization systems of \mathcal{C} and \mathcal{D} , respectively. Therefore $m \mapsto e$ is an isomorphism $\text{Sub}_{fp}(\varrho T) \cong \text{Quo}_{fp}(\mu L)$.

Remark 4.3. Recall that the *ideal completion* $\text{Ideal}(A)$ of a join-semilattice A is the complete lattice of all ideals (= join-closed downsets) of A ordered by inclusion. Up to isomorphism $\text{Ideal}(A)$ is characterized as a complete lattice containing A such that:

- (1) every element of $\text{Ideal}(A)$ is a directed join of elements of A , and
- (2) the elements of A are compact in $\text{Ideal}(A)$: if $x \in A$ lies under a directed join of elements $y_i \in \text{Ideal}(A)$, then $x \leq y_i$ for some i .

Theorem 4.4. *If T preserves preimages, then $\text{Sub}(\varrho T) \cong \text{Ideal}(\text{Quo}_{fp}(\mu L))$.*

Proof (Sketch). Since $\text{Sub}_{fp}(\varrho T) \cong \text{Quo}_{fp}(\mu L)$ by Proposition 4.2, it suffices to prove that $\text{Sub}(\varrho T)$ is the ideal completion of $\text{Sub}_{fp}(\mu L)$. Firstly $\text{Sub}(\varrho T)$ forms a complete lattice because $\text{Coalg } T$ is cocomplete and has a factorization system carried by strong epis and monos in \mathcal{C} . Now one proves that $\text{Sub}(\varrho T) \cong \text{Ideal}(\text{Sub}_{fp}(\varrho T))$ by establishing the properties (1) and (2) of Remark 4.3.

5 Local Eilenberg Theorem

The aim of this section is to prove our main result: a general local Eilenberg Theorem for deterministic automata, i.e., coalgebras for the functor $T_\Sigma = 2 \times \text{Id}^\Sigma : \mathcal{C} \rightarrow \mathcal{C}$. Here 2 is a fixed object of \mathcal{C}_{fp} , and we write $\mathbb{1} = P^{-1}2$ for the corresponding $\hat{\mathcal{D}}$ -object. Note that $\mathbb{1}$ lies in $\mathcal{D}_{c_{fp}}$ and thus is also an object of $\mathcal{D} = \text{Ind}(\hat{\mathcal{D}}_{c_{fp}})$. T_Σ -coalgebras $Q \rightarrow 2 \times Q^\Sigma$ and L_Σ -algebras $\mathbb{1} + \coprod_\Sigma A \rightarrow A$ are represented as triples

$$Q = (Q, \delta_a : Q \rightarrow Q, f : Q \rightarrow 2) \quad \text{and} \quad A = (A, \delta_a : A \rightarrow A, i : \mathbb{1} \rightarrow A).$$

Assumptions 5.1. We continue to work under the Assumptions 3.1 and make the following additional assumptions on \mathcal{C} and \mathcal{D} :

- (a) \mathcal{C} and \mathcal{D} are concrete categories, i.e., forgetful functors to Set are given (notation: $A \mapsto |A|$ for objects and $f \mapsto f$ for morphisms). We assume that these forgetful functors are strongly finitary right adjoints, and that \mathcal{D} 's forgetful functor preserves epimorphisms.
- (b) An object $2 \in \mathcal{C}_{fp}$ is selected with underlying set $|2| = \{0, 1\}$, and the corresponding object $\mathbb{1} = P^{-1}2$ in \mathcal{D} is free on one generator: $\mathbb{1} = \Psi \mathbb{1}$ for the left adjoint $\Psi : \text{Set} \rightarrow \mathcal{D}$ of the forgetful functor.
- (c) Every object C of \mathcal{C} has, for a given subset $m : M \rightarrow |C|$, at most one subobject carried by m .
- (d) \mathcal{D} has hom-objects, i.e., for every pair of objects A and B the power $B^{|A|}$ has a subobject $[A, B] \rightarrow B^{|A|}$ carried by the set $\mathcal{D}(A, B)$ of all morphisms $A \rightarrow B$.

Example 5.2. All the categories in Example 3.2 meet these assumptions. In BA, JSL_0 and DL_{01} we choose 2 to be the chain $0 < 1$, and in $\text{Vect } \mathbb{Z}_2$ we choose $2 = \mathbb{Z}_2$.

Remark 5.3. Since the forgetful functors are strongly finitary, the finitely presentable objects of \mathcal{C} and \mathcal{D} are carried by finite sets. Hence we will talk about *finite* objects rather than finitely presentable ones.

Proposition 5.4. *The rational fixpoint ϱT_Σ is carried by the automaton Reg_Σ of Example 2.8. Consequently, a subcoalgebra of ϱT_Σ is a set of regular languages closed under left derivatives and carrying a subobject of ϱT_Σ in \mathcal{C} .*

What about closure under *right* derivatives? Given a T_Σ -coalgebra $Q = (Q, \delta_a, f)$ and $w \in \Sigma^*$ we consider the coalgebra

$$Q_w = (Q, \delta_a, f \cdot \delta_w)$$

where, as usual, $\delta_w = \delta_{a_n} \cdots \delta_{a_1}$ for $w = a_1 \dots a_n$. Closure under right derivatives can be characterized coalgebraically as follows:

Proposition 5.5

A subcoalgebra Q of ϱT_Σ is closed under right derivatives (i.e., $L \in |Q|$ implies $Lw^{-1} \in |Q|$ for each $w \in \Sigma^$) iff there exists a coalgebra morphism from Q_w to Q for each $w \in \Sigma^*$.*

Remark 5.6. Analogously, for an L_Σ -algebra $A = (A, \delta_a, i)$ and $w \in \Sigma^*$ we define

$$A_w = (A, \delta_a, \delta_w \cdot i).$$

Now let A be a finite quotient algebra μL_Σ . It corresponds to a finite right-derivative closed subcoalgebra of ϱT_Σ under the isomorphism of Proposition 4.2 iff an L_Σ -algebra morphism from A to A_w exists for every $w \in \Sigma^*$. Indeed, a coalgebra morphism $Q_w \rightarrow Q$ corresponds to an algebra morphism $A \rightarrow A_{w^r}$, where $A = \overline{P}^{-1}Q$ (see Remark 3.6) and w^r is the reversed word of w . Fortunately a better characterization is possible, using the concept of a bimonoid.

Definition 5.7. *A bimonoid in \mathcal{D} is a triple (A, \circ, i) where (i) A is a \mathcal{D} -object, (ii) $(|A|, \circ, i)$ is a monoid in Set and (iii) for all $a \in |A|$, the translations $a \circ -$ and $- \circ a$ carry endomorphisms of A . It is called *finite* if $A \in \mathcal{D}_{fp}$. A bimonoid morphism $h : (A, \circ, i) \rightarrow (A', \circ', i')$ is a \mathcal{D} -morphism $h : A \rightarrow A'$ that is also a monoid morphism.*

Example 5.8. Bimonoids in $\mathcal{D} = \text{Set}, \text{Pos}, \text{JSL}_0$ and $\text{Vect } \mathbb{Z}_2$ correspond to monoids, ordered monoids, idempotent semirings and \mathbb{Z}_2 -algebras, respectively.

Construction 5.9. We define a monoid multiplication \bullet on the free object $\Psi \Sigma^*$. For all $w \in \Sigma^*$, let $r_w : \Psi \Sigma^* \rightarrow \Psi \Sigma^*$ be the unique \mathcal{D} -morphism extending the map $- \cdot w$ on Σ^* . Let $\overline{r} : \Psi \Sigma^* \rightarrow [\Psi \Sigma^*, \Psi \Sigma^*]$ (see Assumptions 5.1(d)) be the unique \mathcal{D} -morphism extending the map $r : \Sigma^* \rightarrow \mathcal{D}(\Psi \Sigma^*, \Psi \Sigma^*)$, $w \mapsto r_w$.

$$\begin{array}{ccc}
 \Sigma^* & \xrightarrow{- \cdot w} & \Sigma^* \\
 \eta \downarrow & & \downarrow \eta \\
 |\Psi \Sigma^*| & \xrightarrow{r_w} & |\Psi \Sigma^*|
 \end{array}
 \qquad
 \begin{array}{ccc}
 & & \Sigma^* \\
 & \nearrow \eta & \downarrow r \\
 |\Psi \Sigma^*| & \xrightarrow{\overline{r}} & [|\Psi \Sigma^*, \Psi \Sigma^*|]
 \end{array}$$

Then define the multiplication \bullet by

$$x \bullet y := [\overline{f}(y)](x) \quad \text{for all } x, y \in |\Psi \Sigma^*|.$$

Lemma 5.10. $(\Psi \Sigma^*, \bullet, \eta \varepsilon)$ is a bimonoid, in fact the free bimonoid on Σ : for any bimonoid (A, \circ, \hat{i}) and any function $f : \Sigma \rightarrow |A|$, there is a unique extension to a bimonoid morphism $\overline{f} : \Psi \Sigma^* \rightarrow A$.

$$\begin{array}{ccc} \Sigma^* & \xrightarrow{\eta} & |\Psi \Sigma^*| \\ \uparrow & & \downarrow \overline{f} \\ \Sigma & \xrightarrow{f} & |A| \end{array}$$

- Example 5.11.** (a) For $\mathcal{D} = \text{Set}$ and Pos we have $\Psi \Sigma^* = \Sigma^*$ (discretely ordered in the case $\mathcal{D} = \text{Pos}$) with monoid multiplication = concatenation of words.
 (b) For $\mathcal{D} = \text{JSL}_0$ we have $\Psi \Sigma^* = \mathcal{P}_f \Sigma^*$ (finite languages over Σ) with join = union and monoid multiplication = concatenation of languages.
 (c) For $\mathcal{D} = \text{Vect } \mathbb{Z}_2$ we have $\Psi \Sigma^* = \mathcal{P}_f \Sigma^*$ with addition = symmetric difference and monoid multiplication = \mathbb{Z}_2 -weighted concatenation of languages, i.e., $L \otimes L'$ consists of all words w having an odd number of factorizations $w = uu'$ with $u \in L$ and $u' \in L'$.

This motivates the following definition:

- Definition 5.12.** (a) A Σ -generated bimonoid is a quotient bimonoid of $\Psi \Sigma^*$, represented by a bimonoid morphism $e : \Psi \Sigma^* \twoheadrightarrow A$ with e epic in \mathcal{D} .
 (b) We denote by $\Sigma\text{-Bim}_{fp}(\mathcal{D})$ the poset of all Σ -generated finite bimonoids under the usual quotient ordering.

Remark 5.13. $\Sigma\text{-Bim}_{fp}(\mathcal{D})$ is a join-semilattice. Indeed, it is easy to see that the category of finite bimonoids has finite limits, computed on the level of \mathcal{D} , and also inherits the (strong epi, mono)-factorization system from \mathcal{D} . Hence the join of two Σ -generated bimonoids $e : \Psi \Sigma^* \twoheadrightarrow A$ and $e' : \Psi \Sigma^* \twoheadrightarrow A'$ in $\Sigma\text{-Bim}_{fp}(\mathcal{D})$ is their *subdirect product*, obtained by factorizing the product map $\langle e, e' \rangle : \Psi \Sigma^* \rightarrow A \times A'$.

Remark 5.14. Every Σ -generated bimonoid $e : \Psi \Sigma^* \twoheadrightarrow (A, \circ, \hat{i})$ induces an L_Σ -algebra $\widetilde{A} = (A, \delta_a, \hat{i})$ where $\delta_a : A \rightarrow A$ is the \mathcal{D} -morphism with $\delta_a(x) = x \circ e(\eta a)$ for all $x \in |A|$, and $\hat{i} : \mathbb{1} \rightarrow A$ is the free extension of $i : \mathbb{1} \rightarrow |A|$. One can show that

$$\widetilde{\Psi \Sigma^*} = \mu L_\Sigma.$$

Proposition 5.15. An finite quotient algebra A of μL_Σ is induced by a Σ -generated bimonoid iff L_Σ -algebra morphisms from A to A_w exist for all $w \in \Sigma^*$.

Proof (Sketch). Every $e : \Psi \Sigma^* \twoheadrightarrow A$ in $\Sigma\text{-Bim}_{fp}(\mathcal{D})$ yields a quotient algebra $e : \mu L_\Sigma \twoheadrightarrow \widetilde{A}$ of μL_Σ . For each $w \in \Sigma^*$, the desired L_Σ -algebra morphism $\widetilde{A} \rightarrow \widetilde{A}_w$ is the \mathcal{D} -morphism carried by $e(\eta w) \bullet -$. Conversely, let $e : \mu L_\Sigma \twoheadrightarrow (A, \delta_a, \hat{i})$ be any quotient algebra of μL_Σ such that L_Σ -algebra morphisms $A \rightarrow A_w$ exist. Define a

monoid multiplication \circ on $|A|$ as follows: given $x, y \in |A|$, choose $x' \in |\Psi\Sigma^*|$ and $y' \in |\Psi\Sigma^*|$ with $ex' = x$ and $ey' = y$ (using that e is surjective by Assumptions 5.1(a)), and put

$$x \circ y := e(x' \bullet y').$$

One then proves that (A, \circ, i) is a well-defined bimonoid whose induced L_Σ -algebra is precisely (A, δ_a, i) .

Definition 5.16. *By a local variety of regular languages in \mathcal{C} is meant a subcoalgebra of ϱT_Σ closed under right derivatives.*

Example 5.17. Local varieties of regular languages in $\mathcal{C} = \text{BA}, \text{DL}_{01}, \text{JSL}_0$ and $\text{Vect } \mathbb{Z}_2$ are called *local varieties, local lattice varieties, local semilattice varieties* and *local linear varieties of regular languages*, respectively; see Introduction.

Definition 5.18. *By a local pseudovariety of bimonoids in \mathcal{D} is meant a set of finite Σ -generated bimonoids in \mathcal{D} closed under subdirect products and quotients, i.e., an ideal in the join-semilattice $\Sigma\text{-Bim}_{fp}(\mathcal{D})$.*

Theorem 5.19 (General Local Eilenberg Theorem). *The lattice of local varieties of regular languages in \mathcal{C} is isomorphic to the lattice of local pseudovarieties of bimonoids in \mathcal{D} .*

Proof (Sketch). Let $\text{Sub}_{fp}^r(\varrho T_\Sigma)$ denote the poset of all finite local varieties of regular languages in \mathcal{C} , i.e., of all finite subcoalgebras of ϱT_Σ closed under right derivatives. From Remark 5.6 and Proposition 5.15 we get a join-semilattice isomorphism

$$\text{Sub}_{fp}^r(\varrho T_\Sigma) \cong \Sigma\text{-Bim}_{fp}(\mathcal{D}).$$

Taking ideal completions on both sides yields a complete lattice isomorphism

$$\text{Ideal}(\text{Sub}_{fp}^r(\varrho T_\Sigma)) \cong \text{Ideal}(\Sigma\text{-Bim}_{fp}(\mathcal{D})).$$

One then proves that $\text{Ideal}(\text{Sub}_{fp}^r(\varrho T_\Sigma))$ is isomorphic to the lattice of local varieties of regular languages in \mathcal{C} . Moreover, by definition $\text{Ideal}(\Sigma\text{-Bim}_{fp}(\mathcal{D}))$ is precisely the lattice of local pseudovarieties of bimonoids in \mathcal{D} .

Corollary 5.20. *By instantiating Theorem 5.19 to the categories of Example 3.2 we obtain the following lattice isomorphisms:*

\mathcal{C}	\mathcal{D}	local varieties of regular languages \cong local pseudovarieties of ...
BA	Set	local varieties \cong monoids
DL ₀₁	Pos	local lattice varieties \cong ordered monoids
JSL ₀	JSL ₀	local semilattice varieties \cong idempotent semirings
Vect \mathbb{Z}_2	Vect \mathbb{Z}_2	local linear varieties \cong \mathbb{Z}_2 -algebras

5.1 Profinite Monoids

As a consequence of Theorem 5.19 we obtain a generalization of the result of Gehrke, Grigorieff and Pin [6, 7] that Reg_Σ endowed with boolean operations and derivatives is dual to the free profinite monoid on Σ . To see this one observes first that the finite local varieties of languages form a cofinal subposet of $\text{Sub}_{fp}(\varrho T_\Sigma)$ – in other words, every finite subcoalgebra of ϱT_Σ is contained in a finite local variety. Therefore the finite Σ -generated bimonoids form a cofinal subposet of $\text{Quo}_{fp}(\mu L_\Sigma)$. Thus, the corresponding diagrams have the same limit in $\text{Alg } \hat{L}_\Sigma$. Since the limit of all fp-quotients of μL_Σ is the initial L_Σ -algebra μL_Σ , we see that $\tau \hat{L}_\Sigma$ is also the limit of the directed diagram of all finite Σ -generated bimonoids. Hence, $\tau \hat{L}_\Sigma$ is a bimonoid and it is then easy to see that it is the free profinite bimonoid on Σ , where bimonoid now means bimonoid in $\hat{\mathcal{D}}$ w.r.t. $|U| = |-| \circ U : \hat{\mathcal{D}} \rightarrow \text{Set}$ and “profinite” refers to the category $\text{Pro}(\hat{\mathcal{D}}_{c,fp}) = \hat{\mathcal{D}}$; in fact, (the carrier of) $\tau \hat{L}_\Sigma$ is $F(\Psi(\Sigma^*))$, where $F \cdot \Psi$ is the left adjoint of $|U|$.

Theorem 5.21. *$\tau \hat{L}_\Sigma$ is the free profinite bimonoid on Σ , and this structure is dual to the structure on ϱT_Σ given by its T_Σ -coalgebra structure and right derivatives.*

6 Conclusions and Future Work

Inspired by recent work of Gehrke, Grigorieff and Pin [6, 7] we have proved a generalized local Eilenberg theorem, parametric in a pair of dual categories \mathcal{C} and $\hat{\mathcal{D}}$ and a type of coalgebras $T : \mathcal{C} \rightarrow \mathcal{C}$. By instantiating our framework to deterministic automata, i.e., the functor $T_\Sigma = 2 \times \text{Id}^\Sigma$ on $\mathcal{C} = \text{BA}, \text{DL}_{01}, \text{JSL}_0$ and $\text{Vect } \mathbb{Z}_2$, we derived the local Eilenberg theorems for (ordered) monoids as in [6], as well as two new local Eilenberg theorems for idempotent semirings and \mathbb{Z}_2 -algebras.

There remain a number of open points for further work. Firstly, our general approach should be extended to the ordinary (non-local) version of Eilenberg’s theorem. Secondly, for different functors T on the categories we have considered our approach should provide the means to relate varieties of rational behaviours of T with varieties of appropriate algebras. In this way, we hope to obtain Eilenberg type theorems for systems such as Mealy and Moore automata, but also weighted or probabilistic automata – ideally, such results would be proved uniformly for a certain class of functors.

Another very interesting aspect we have not treated in this paper are profinite equations and syntactic presentations of varieties (of bimonoids or regular languages, resp.) as in the work of Gehrke, Grigorieff and Pin [6]. An important role in studying profinite equations will be played by the \hat{L} -algebra $\tau \hat{L}$, the dual of the rational fixpoint, that we identified as the free profinite bimonoid. A profinite equation is then a pair of elements of $\tau \hat{L}$. We intend to investigate this in future work.

References

- [1] Adámek, J., Milius, S., Velebil, J.: Iterative algebras at work. *Math. Structures Comput. Sci.* 16(6), 1085–1131 (2006)
- [2] Adámek, J., Rosický, J.: *Locally presentable and accessible categories.* Cambridge University Press (1994)

- [3] Almeida, J.: Finite semigroups and universal algebra. World Scientific Publishing, River Edge (1994)
- [4] Eilenberg, S.: Automata, languages and machines, vol. B. Academic Press, Harcourt Brace Jovanovich Publishers, New York (1976)
- [5] Gehrke, M.: Stone duality, topological algebra and recognition (2013), <http://hal.archives-ouvertes.fr/hal-00859717>
- [6] Gehrke, M., Grigorieff, S., Pin, J.-É.: Duality and equational theory of regular languages. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 246–257. Springer, Heidelberg (2008)
- [7] Gehrke, M., Grigorieff, S., Pin, J.-É.: A topological approach to recognition. In: Abramsky, S., Gavioille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 151–162. Springer, Heidelberg (2010)
- [8] Johnstone, P.T.: Stone Spaces, Cambridge studies in advanced mathematics, vol. 3. Cambridge University Press (1982)
- [9] 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)
- [10] Pin, J.-É.: Mathematical foundations of automata theory (2013), <http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf>
- [11] Pippenger, N.: Regular languages and Stone duality. *Theory Comput. Syst.* 30(2), 121–134 (1997)
- [12] Polák, L.: Syntactic semiring of a language. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 611–620. Springer, Heidelberg (2001)
- [13] Reiterman, J.: The Birkhoff theorem for finite algebras. *Algebra Universalis* 14(1), 1–10 (1982)
- [14] Rhodes, J., Steinberg, B.: The Q-theory of Finite Semigroups, 1st edn. Springer Publishing Company, Incorporated (2008)
- [15] Rutten, J.J.M.M.: Universal coalgebra: A theory of systems. *Theor. Comput. Sci.* 249(1), 3–80 (2000)

Combining Bialgebraic Semantics and Equations

Jurriaan Rot¹ and Marcello Bonsangue²

¹ LIACS — Leiden University, Niels Bohrweg 1, Leiden, The Netherlands

² Centrum Wiskunde en Informatica, Science Park 123, Amsterdam, The Netherlands
{j.c.rot,m.m.bonsangue}@liacs.leidenuniv.nl

Abstract. It was observed by Turi and Plotkin that structural operational semantics can be studied at the level of universal coalgebra, providing specification formats for well-behaved operations on many different types of systems. We extend this framework with non-structural assignment rules which can express, for example, the syntactic format for structural congruences proposed by Mousavi and Reniers. Our main result is that the operational model of such an extended specification is well-behaved, in the sense that bisimilarity is a congruence and that bisimulation-up-to techniques are sound.

1 Introduction

Structural operational semantics (SOS) is a framework for defining the semantics of programming languages and calculi in terms of transition system specifications [1]. By imposing syntactic restrictions, one can prove well-behavedness properties of transition systems at the meta-level of their specification. For instance, any specification in the GSOS format [4] has a unique operational model, on which bisimilarity is a congruence.

Traditionally, research in SOS has focused on labelled transition systems as the fundamental model of behaviour. Turi and Plotkin [27] introduced the *bialgebraic* approach to structural operational semantics, where in particular GSOS can be studied at the level of *universal coalgebra* [22]. The theory of coalgebras provides a mathematical framework for the uniform study of many types of state-based systems, including labelled transition systems but also, e.g., (non)-deterministic automata, stream systems and various types of probabilistic and weighted automata [23,11,3]. In the coalgebraic framework, there is a canonical notion of bisimilarity, which instantiates to the classical definition of (strong) bisimilarity in the case of labelled transition systems. It is shown in [27] that GSOS specifications can be generalised by certain natural transformations, which are called *abstract GSOS specifications*, and that these correspond to the categorical notion of *distributive laws*. This provides enough structure to prove at this general level that bisimilarity is a congruence. By instantiating the theory to concrete instances, one can then obtain congruence formats for systems such as probabilistic automata, weighted transition systems, streams, etc. — see [12] for an overview. Another advantage of abstract GSOS is that bisimulation up to

context is “compatible” [21,20], providing a sound enhancement of the bisimulation proof method which can be combined with other compatible enhancements such as bisimulation up to bisimilarity [24,19].

In this paper we add rules such as in (1) to this framework. The rule in (1) properly defines the replication operator in

CCS¹: intuitively $!x$ represents $x \mid x \mid x \mid \dots$,

$$\frac{!x \mid x \xrightarrow{a} t}{!x \xrightarrow{a} t} \quad (1)$$

i.e., the infinite parallel composition of x with

itself. In fact, the above rule can be seen as assigning the behaviour of the term $!x \mid x$ to the simpler term $!x$, therefore we call it an *assignment rule*. Being inherently non-structural, such an assignment rule cannot directly be embedded in the bialgebraic framework of Turi and Plotkin, where the behaviour of terms is computed inductively. In this paper we show how to interpret assignment rules together with abstract GSOS specifications. As it turns out, this requires the assumption that the functor which represents the type of coalgebra is *ordered* as a complete lattice; for example, in the case of labelled transition systems this order is simply inclusion of sets of pairs (a, x) of a label a and a state x . The operational model on closed terms then is the *least* model such that every transition can either be derived from a rule in the specification, or there is a rule assigning to an operator σ the behaviour of a term t in the model. To ensure the existence of such least models, we disallow negative premises by using *monotone* abstract GSOS specifications, a generalisation of the *positive GSOS format* for transition systems [8]. Positive GSOS can be seen as the greatest common divisor of GSOS and the tyft/tyxt format [2].

Our main result is that the interpretation of a monotone abstract GSOS specification together with a set of assignment rules is itself the operational model of another (typically larger) abstract GSOS specification. Like the interpretation of a GSOS specification with assignment rules, we construct this latter specification by fixpoint induction. As a direct consequence of this alternative representation of the interpretation, we obtain that bisimilarity is a congruence and that bisimulation up to context is sound and even compatible — properties that do not follow from bisimilarity being a congruence [19]. As an example application, we obtain the compatibility of bisimulation-up-to techniques for CCS with replication, which so far had to be shown with an ad-hoc argument [19].

A further contribution of this paper consists in combining *structural congruences* [16,17] with the bialgebraic framework using assignment rules. Structural congruences were introduced in the operational semantics of the π -calculus in [16]. The basic idea is that SOS specifications are extended with *equations* on terms, which are then linked by a special deduction rule. This rule essentially states that if two processes are equated by the congruence generated by the set of equations, then they can perform the same transitions. Prototypical examples are the specification of the parallel operator by combining a single rule with commutativity, and the specification of the replication operator by an equation, both shown below:

¹ The simpler rule $\frac{x \rightarrow x'}{!x \rightarrow !x \mid x'}$ is problematic in the presence of the sum operator [19,26].

$$\frac{x \xrightarrow{a} x'}{x \mid y \xrightarrow{a} x' \mid y} \quad x \mid y = y \mid x \quad !x = !x \mid x \quad (2)$$

In [17] Mousavi and Reniers show how to interpret SOS rules with structural congruences in various equivalent ways. They exhibit very simple examples of equations and SOS rules for which bisimilarity is not a congruence, even when the SOS rules are in the tyft (or the GSOS) format. As a solution to this problem they introduce a restricted format for equations, called *cfsc*, for which bisimilarity is a congruence when combined with tyft specifications.

In the present paper we show how to interpret structural congruences at the general level of coalgebras, in terms of an operational model on closed terms. We prove that when the equations are in the *cfsc* format then they can be encoded by assignment rules, in such a way that their respective interpretations coincide up to bisimilarity. Consequently, not only is bisimilarity a congruence for monotone abstract GSOS combined with *cfsc* equations, but also bisimulation up to context and bisimilarity is compatible.

Outline. In Section 2 we recall some preliminaries on (co)algebras and abstract GSOS. In Section 3, assignment rules and their interpretation are introduced. We show in Section 4 that this interpretation can be obtained as the operational model of another abstract GSOS specification. Section 5 contains the integration of structural congruence with the bialgebraic framework. In Section 6 we discuss related work, and in Section 7 we conclude with some directions for future work.

To fully understand the technical development in this paper, familiarity with basic notions in category theory, bialgebraic semantics and order theory is useful. However, many of the main results and definitions are illustrated with concrete examples, in particular on the familiar case of transition systems.

2 Coalgebras, Signatures and Bialgebraic Semantics

By **Set** we denote the category of sets and total functions. We write *Id* for the identity functor on **Set**.

Coalgebras. For an extensive treatment with many examples we refer to [22]. An (*F*-)coalgebra for a functor $F: \mathbf{Set} \rightarrow \mathbf{Set}$ consists of a set of states C and a map $\alpha: C \rightarrow FC$. Let (C, α) and (D, β) be two coalgebras. A function $f: C \rightarrow D$ is an (*F*-coalgebra) homomorphism if $Ff \circ \alpha = \beta \circ f$. A relation $R \subseteq C \times D$ is an (*F*-)bisimulation if R can be equipped with a transition structure $\gamma: R \rightarrow FR$ such that the two projection functions $\pi_1: R \rightarrow C$ and $\pi_2: R \rightarrow D$ are homomorphisms. The largest bisimulation between two systems α and β is called *bisimilarity*. Two *F*-coalgebras $\alpha, \beta: X \rightarrow FX$ (on a common carrier X) are equal up to bisimilarity if the diagonal on X progresses [21] to \sim .

Example 1. Labelled transition systems (LTSs) over a set of labels A are coalgebras for the functor $FX = (\mathcal{P}X)^A$. For an LTS $\alpha: X \rightarrow (\mathcal{P}X)^A$ we write $x \xrightarrow{a} x'$ iff $x' \in \alpha(x)(a)$. Intuitively, for a state $x \in X$, $\alpha(x)(a)$ contains all the

outgoing transitions from x labelled by a . Coalgebraic bisimulation instantiates to the classical definition by Milner and Park: a relation $R \subseteq X \times X$ is called a *bisimulation* provided that for all $(x, y) \in R$, if $x \xrightarrow{a} x'$ then there exists a state y' such that $y \xrightarrow{a} y'$ and $(x', y') \in R$, and vice versa.

Image-finite labelled transition systems are coalgebras for the functor $FX = (\mathcal{P}_\omega X)^A$, where $\mathcal{P}_\omega X$ is the set of all *finite* subsets of X .

Coalgebras for the functor $FX = \mathbb{R} \times X$, where \mathbb{R} is the set of real numbers, are called *stream systems* (over the reals). In a stream system, for every state we can observe an output in \mathbb{R} and a next state.

Signatures. A *signature* Σ is a (possibly infinite) collection of operator names $\sigma \in \Sigma$ with (finite) arities $|\sigma| \in \mathbb{N}$. Equivalently, it is a polynomial functor as in (3). In the sequel we write $\sigma(x_1, \dots, x_n)$ instead of $(\sigma, (x_1, \dots, x_n))$ for elements of ΣX . The functor ΣX acts on a map $f: X \rightarrow Y$ as follows: $(\Sigma f)(\sigma(x_1, \dots, x_n)) = \sigma(f(x_1), \dots, f(x_n))$. Above and in the sequel we abuse notation and use Σ to represent signatures as well as their associated functors.

$$\Sigma X = \coprod_{\sigma \in \Sigma} X^{|\sigma|} \tag{3}$$

A Σ -*algebra* consists of a set A and a function $\alpha: \Sigma A \rightarrow A$. This coincides with the standard notion of an algebra for the signature Σ . For a set of variables X and a signature Σ we denote by TX the set of *terms*, as defined by the grammar $t ::= \sigma(t_1, \dots, t_n) \mid x$ where σ ranges over Σ , n is the arity of σ and x ranges over X . The special case $T\emptyset$ is the set of *closed terms*. Every set TX can be turned into the (free) Σ -algebra $\nu_X: \Sigma TX \rightarrow TX$ by defining $\nu_X(\sigma(t_1, \dots, t_n)) = \sigma(t_1, \dots, t_n)$. Note that ν_\emptyset is an isomorphism, since it is the initial Σ -algebra.

We note that T is the *free monad* for the signature Σ , without going into details. Of importance to our purposes is that it comes equipped with natural transformations $\eta: \text{Id} \Rightarrow T$ and $\mu: TT \Rightarrow T$; for a set (of variables) X , η_X is the injection of variables into terms, and μ_X turns a term over terms into a single term in the expected manner. In the sequel, whenever the type can be deduced from the context, we omit subscripts from natural transformations to avoid notational clutter.

Bialgebraic operational semantics. See [12] for an overview of this topic. In the remainder of this paper, we assume some fixed signature Σ with associated term monad T , and a **Set** endofunctor F representing the type of behaviour. An (*abstract GSOS*) *specification* is a natural transformation of the form

$$\rho: \Sigma(F \times \text{Id}) \Rightarrow FT.$$

As first observed by Turi and Plotkin [27], if F is the functor $(\mathcal{P}_\omega -)^A$ of image-finite labelled transition systems then specifications of the above type can be induced by specifications in the well-known GSOS format introduced in [4]. A GSOS rule for an operator $\sigma \in \Sigma$ of arity n is of the form

$$\frac{\{x_{i_j} \xrightarrow{a_j} y_j\}_{j=1..m} \quad \{x_{i_k} \xrightarrow{b_k} \cdot\}_{k=1..l}}{\sigma(x_1, \dots, x_n) \xrightarrow{c} t} \tag{4}$$

where m is the number of positive premises, l is the number of negative premises, and $a_1, \dots, a_m, b_1, \dots, b_l, c \in A$ are labels. The variables $x_1, \dots, x_n, y_1, \dots, y_m$ are pairwise distinct, and t is a term over these variables.

If we instantiate F to the functor $\mathbb{R} \times \mathbf{Id}$ of stream systems over the reals, specifications correspond to the format of *behavioural differential equations* [23] presented in [13]. By instantiating abstract GSOS specifications to other functors one can obtain formats for many types of systems, including, e.g., syntactic formats for probabilistic and weighted transition systems [3,11].

Each specification $\rho: \Sigma(F \times \mathbf{Id}) \Rightarrow FT$ induces a unique *operational model* $f: T\emptyset \rightarrow FT\emptyset$, also called ρ -*model* (on the initial algebra), with the following property:

$$f \circ \nu = F\mu \circ \rho \circ \Sigma\langle f, \text{id} \rangle. \tag{5}$$

By this equation, to compute the behaviour of a term $\sigma(t_1, \dots, t_n)$ in f , we may compute the behaviour of its subterms t_1, \dots, t_n and then instantiate a rule from ρ . For labelled transition systems, f is precisely the unique *supported model* corresponding to a GSOS specification ρ : every transition in f is derived from rules in the specification ρ and each derivable transition occurs in f (see [1,4]).

An important property of GSOS is that bisimilarity is a congruence on the operational model corresponding to a specification [4]. Turi and Plotkin [27] proved at the general level of abstract GSOS specifications that coalgebraic bisimilarity on the operational model is a congruence, extending the result of [4] to calculi for many other types of systems. Furthermore, these specifications guarantee the (strictly stronger property of) compatibility of *bisimulation-up-to context* on the operational model [20]. This yields an enhanced proof technique for bisimilarity in which one can use the syntactic structure of the terms to relate their successors.

3 Adding Assignment Rules

In this section we consider the interpretation of abstract GSOS specifications (without negative premises) together with *assignment rules* of the form

$$\sigma(x_1, \dots, x_n) := t \tag{6}$$

where t is a term over the variables x_1, \dots, x_n . These will be interpreted as a kind of rewriting rules: the behaviour of t induces behaviour of $\sigma(x_1, \dots, x_n)$. An example is the replication operator given in equation (1) of the introduction; this can be given by $!x := !x \mid x$. Notice that the above rules, which we will call *assignment rules*, do not fit directly into the bialgebraic framework, since they are inherently non-structural. That is, they ruin the property of GSOS specifications that the behaviour of terms in the operational model can be computed from the behaviour of their subterms.

In the case of labelled transition systems, given a GSOS specification and a set of rules of the above form, the desired interpretation is informally as follows: every transition from a term $\sigma(t_1, \dots, t_n)$ should either be derived from the

transitions of t_1, \dots, t_n and a rule in the specification, or from an assignment rule which has σ on the left-hand side. This suggests a natural extension of the fixpoint equation of the supported model (equation (5)) to incorporate the assignment rules. However, because of assignment rules there is not necessarily a *unique* supported model anymore, since now there may be infinite inferences. For example, the rule $\sigma(x) := \sigma(x)$ does not have a unique solution. In order to rule this out, one is interested in the *least* transition system on closed terms which satisfies the extended fixpoint equation. Such a least model does not exist in general because of negative premises, so we will use a formalization of the notion of *positive* GSOS at the level of abstract specifications, and restrict to such specifications in the remainder of this paper.

To interpret specifications which involve assignment rules at the general level of a functor F one needs a notion of order on F . In the case of labelled transition systems this order is clear and often left implicit: in that case $FX = (\mathcal{P}X)^A$, and it is simply the (pointwise) subset order. To allow the desired generalisation, we assume that our behaviour functor F is *ordered* [9], that is, it factors through CJSL, which is the category of complete (join semi-)lattices and join-preserving functions. Thus we assume a functor $\hat{F}: \mathbf{Set} \rightarrow \mathbf{CJSL}$ such that $U \circ \hat{F} = F$, where $U: \mathbf{CJSL} \rightarrow \mathbf{Set}$ is the forgetful functor that takes a complete lattice to its underlying set. Thus for every set X we have arbitrary joins in FX ; in the sequel we denote the join of a set $S \subseteq FX$ by $\bigvee S$, and we write \perp for $\bigvee \emptyset$ and $x \leq y$ if $x \vee y = y$, for $x, y \in FX$.

Example 2. As mentioned above, the functor $(\mathcal{P}-)^A$ of labelled transition systems has a natural complete lattice structure.

Any functor $F: \mathbf{Set} \rightarrow \mathbf{Set}$ can be extended to an ordered functor F' by taking $F'X = FX + 2$ where $2 = \{\perp, \top\}$: we then define, for $x, y \in FX$, $x \leq y$ iff $x = y$ and use \perp and \top as the least and greatest element respectively.

The functor for (possibly infinitely branching) weighted transition systems [11] over a complete lattice, is ordered. For example, one can take as weights the set $\mathbb{R} \cup \{\infty, -\infty\}$, i.e., the reals extended with top and bottom elements.

For any function $f: X \rightarrow Y$ we have $Ff = U \circ \hat{F}(f)$ and thus Ff is a join-preserving map: $f[\bigvee S] = \bigvee f[S]$. Consequently, Ff is also monotone, i.e., $x \leq y$ implies $f(x) \leq f(y)$. Given arbitrary sets X and Y , the complete lattice on FY lifts pointwise to a complete lattice on functions of type $X \rightarrow FY$, i.e., for a collection $\{f_i\}_{i \in I}$ of functions of the form $f_i: X \rightarrow FY$ we define $(\bigvee \{f_i\}_{i \in I})(x) = \bigvee_{i \in I}(f_i(x))$. This induces in particular a complete lattice on the set of all coalgebras on closed terms, which we denote by

$$\mathbb{M} = \{f \mid f: T\emptyset \rightarrow FT\emptyset\}.$$

The order on F lifts to an order on $F \times \mathbf{Id}$ by defining $(b_1, x_1) \leq (b_2, x_2)$ iff $b_1 \leq b_2$ and $x_1 = x_2$ for $(b_1, x_1), (b_2, x_2) \in FX \times X$. Moreover, the order lifts component-wise to ΣFX (and also to $\Sigma(FX \times X)$) for any set X , by defining, for any $\sigma, \tau \in \Sigma$ of arity n and m respectively, $\sigma(k_1, \dots, k_n) \leq \tau(l_1, \dots, l_m)$ iff $\sigma = \tau$ (so also $n = m$) and $k_i \leq l_i$ for all $i \leq n$.

Definition 3. *Using the above lifting of the order to $\Sigma(F \times \text{Id})$, a specification ρ is said to be monotone if all of its components are.*

Example 4. As stated in [8], for the functor $F = (\mathcal{P}-)^A$ of labelled transition systems, monotone specifications correspond to specifications in (an infinitary version of) the *positive GSOS* format.

Any abstract GSOS specification for a functor F induces a monotone GSOS specification for the discretely ordered functor $F + 2$ (see Example 2).

Assignment rules (6) can be formalised categorically in terms of natural transformations. These are independent of the behaviour functor F .

Definition 5. *An assignment rule is a natural transformation $d: \Sigma \Rightarrow T$.*

For example, the assignment rule for the replication operator is the natural transformation which sends $!x$ to $!x \mid x$ for any x , and is the identity on all other operators in Σ .

Assumption 6. *In the remainder of this paper we assume all our abstract GSOS specifications to be monotone. In particular we fix a monotone GSOS specification ρ and a set Δ of assignment rules.*

Now we have all the necessary tools to define a model on closed terms of an abstract GSOS specification together with a set of assignment rules.

Definition 7. *Let $\psi: \mathbb{M} \rightarrow \mathbb{M}$ be the (unique) function such that*

$$\psi(f) \circ \nu = F\mu \circ \rho \circ \Sigma\langle f, \text{id} \rangle \vee \bigvee_{d \in \Delta} f \circ \mu \circ d.$$

A (ρ, Δ) -model is a coalgebra $f \in \mathbb{M}$ such that $\psi(f) = f$.

Notice that $\nu: \Sigma T\emptyset \rightarrow T\emptyset$ is an isomorphism, so ψ is uniquely defined. As argued above, in general there may be more than one model for a fixed ρ and Δ . We will be interested in the *least* supported model as the correct interpretation. In order to show that this exists we need the following:

Lemma 8. *$\psi: \mathbb{M} \rightarrow \mathbb{M}$ is monotone.*

By the Knaster-Tarski theorem, ψ has a least fixpoint (e.g., [25]).

Definition 9. *The interpretation of ρ and Δ is the least (ρ, Δ) -model.*

Example 10. For a GSOS specification on transition systems together with assignment rules, the interpretation is the least system where $\sigma(t_1, \dots, t_n) \xrightarrow{\alpha} t'$ iff it can be derived from a rule in the specification or there is an assignment of t to σ , and $t \xrightarrow{\alpha} t'$. This is a recursive definition; being the least such transition system has the consequence that every derivation of a transition $t \xrightarrow{\alpha} t'$ is finite.

4 Abstract GSOS Specifications for Assignment Rules

In the previous section we have seen how to interpret an abstract GSOS specification ρ together with a set of assignment rules Δ as a coalgebra on closed terms. In this section we will show that we can alternatively construct this coalgebra as the operational model of another specification (without assignment rules), which is constructed as a least fixpoint of a function on the complete lattice of specifications. The consequence of this alternative representation is well-behavedness properties, in particular bisimilarity being a congruence and the compatibility of bisimulation up to context, on the interpretation of ρ and Δ .

Let \mathbb{S} be the set of all monotone abstract GSOS specifications. We turn \mathbb{S} into a complete lattice by defining the order componentwise, i.e., for any $L \subseteq \mathbb{S}$ and any set X : $(\bigvee L)_X = \bigvee_{\rho \in L} \rho_X$. The join is well-defined:

Lemma 11. *For any $L \subseteq \mathbb{S}$: the family of functions $(\bigvee L)$ as defined above is a monotone specification.*

This provides a way of pointwise combining specifications.

Consider, for some assignment rule $d \in \Delta$ and specification τ , the following natural transformation:

$$\Sigma(F \times \text{Id}) \xrightarrow{d} T(F \times \text{Id}) \xrightarrow{\tau^*} FT \times T \xrightarrow{\pi_1} FT \tag{7}$$

Here, and in the sequel, we use τ^* to denote the inductive extension of τ to terms (e.g., [3]). Informally, the above natural transformation acts as follows. For an operator σ of arity n , given behaviour $k_1, \dots, k_n \in FX \times X$ of its arguments, it first applies the assignment rule d to obtain a term $t(k_1, \dots, k_n)$. Subsequently τ^* is used to compute the behaviour of t given the behaviour k_1, \dots, k_n . In short, the above transformation computes the behaviour of an operator by using rules from τ and a single application of the rule d .

Example 12. Suppose ρ is the specification of CCS, without any rules for the replication operator $!x$. Moreover suppose d is the assignment rule associated to the replication. Then the natural transformation in (7) is a specification which has for the replication operator the rule (for any label a) $\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} !x|x'}$ and is unchanged on all other operators. This rule is deduced from the behaviour of the parallel operator; since a process $!t$ can not make any transitions by the rules in ρ , after the construction in (7), $!t$ can make precisely the transitions that t can.

To obtain the correct specification of the replication operator we will need to apply such a construction recursively, which we will do below. First we define a function φ on \mathbb{S} which uses the above construction to build, from an argument specification τ , the specification containing all rules from our fixed specification ρ and all rules which can be formed as in (7).

Definition 13. *Given our fixed ρ and Δ , the map $\varphi: \mathbb{S} \rightarrow \mathbb{S}$ is defined as*

$$\varphi(\tau) = \rho \vee \bigvee_{d \in \Delta} (\pi_1 \circ \tau^* \circ d).$$

For well-definedness, we need to check that φ preserves monotonicity.

Lemma 14. *The function $\varphi: \mathbb{S} \rightarrow \mathbb{S}$ is monotone. Moreover, if τ is a monotone specification, then $\varphi(\tau)$ is monotone as well.*

As a consequence of φ being monotone, it has a least fixpoint, which we denote by $\text{lfp } \varphi$. Moreover, since φ preserves monotonicity we obtain monotonicity of $\text{lfp } \varphi$ by transfinite induction (the base case and limit steps are rather easy). This proof technique, which we also use several times below, is justified by the fact that the least fixpoint of a monotone function in a complete lattice can be constructed as the supremum of an ascending chain obtained by iterating the function over the ordinals (see, e.g., [25]).

Corollary 15. *$\text{lfp } \varphi$ is monotone.*

Informally, $\text{lfp } \varphi$ is the specification consisting of rules from ρ and Δ . By

$$M: \mathbb{S} \rightarrow \mathbb{M}$$

we denote the function which assigns to a specification its unique operational model (5) (Section 2). We proceed to prove that the operational model of the least fixpoint of φ is precisely the interpretation of ρ and Δ , i.e., that $M(\text{lfp } \varphi) = \text{lfp } \psi$. First, we show that $M(\text{lfp } \varphi)$ is a fixpoint of ψ .

Lemma 16. *$M(\text{lfp } \varphi)$ is a (ρ, Δ) -model.*

We proceed to show that $M(\text{lfp } \varphi) \leq \text{lfp } \psi$. The main step is that any fixpoint of ψ is “closed under ρ ”, i.e., that in such a model, each transition which we can derive by the specification is already there. This result is the contents of Lemma 17 below. For the proof, one shows that $F\mu \circ h \circ \Sigma\langle f, \text{id} \rangle \leq f \circ \nu$ holds for any approximation h of $\text{lfp } \varphi$, by transfinite induction.

Lemma 17. *Let $f \in \mathbb{M}$ be a fixpoint of ψ . Then $F\mu \circ \text{lfp } \varphi \circ \Sigma\langle f, \text{id} \rangle \leq f \circ \nu$.*

This allows to prove our main result.

Theorem 18. *$M(\text{lfp } \varphi) = \text{lfp } \psi$, i.e., the interpretation of ρ and Δ coincides with the operational model of the specification $\text{lfp } \varphi$.*

Proof. By Lemma 16, $M(\text{lfp } \varphi)$ is a fixpoint of ψ . To show it is the least one, let f be any fixpoint of ψ ; we proceed to prove $M(\text{lfp } \varphi) \leq f$ by structural induction on closed terms. Suppose $\sigma \in \Sigma$ is an operator of arity n , and suppose we have $t_1, \dots, t_n \in T\emptyset$ such that $M(\text{lfp } \varphi)(t_i) \leq f(t_i)$ for all i with $1 \leq i \leq n$ (note that this trivially holds in the base case, when $n = 0$). Then $M(\text{lfp } \varphi)(\sigma(t_1, \dots, t_n)) = F\mu \circ \text{lfp } \varphi \circ \Sigma\langle M(\text{lfp } \varphi), \text{id} \rangle(\sigma(t_1, \dots, t_n)) \leq F\mu \circ \text{lfp } \varphi \circ \Sigma\langle f, \text{id} \rangle(\sigma(t_1, \dots, t_n)) \leq f(\sigma(t_1, \dots, t_n))$ where the first inequality holds by assumption and monotonicity of $F\mu$ and $\text{lfp } \varphi$ (Corollary 15) and the second by Lemma 17. \square

As a consequence, the interpretation of ρ and Δ is well-behaved:

Corollary 19. *Bisimilarity is a congruence on the interpretation of ρ and Δ , and bisimulation up to context is compatible.*

Example 20. The operators of CCS can be given by a positive GSOS specification, and equation (1) of the introduction contains a rule for the replication operator. Thus, by the above Corollary, bisimilarity is a congruence on the operational model of CCS with replication, and bisimulation up to context is compatible; this is proved and used in [25], but here we obtain it directly from the format and the above results.

5 Structural Congruences (as Assignment Rules)

The assignment rules considered in the theory of the previous sections copy behaviour from a term to an operator, but this assignment goes one way only. In this section we consider the combination of abstract GSOS specifications with actual *equations*, which are elements of $TV \times TV$ (where V is an arbitrary set of variables). Any set of equations $E \subseteq TV \times TV$ induces a *congruence* \equiv_E :

Definition 21. *Let $E \subseteq TV \times TV$ be a set of equations. The congruence closure \equiv_E of E is the least relation $\equiv \subseteq T\emptyset \times T\emptyset$ satisfying the following rules:*

$$\frac{t E u \quad s: V \rightarrow T\emptyset}{s^\sharp(t) \equiv s^\sharp(u)} \qquad \frac{}{t \equiv t} \qquad \frac{u \equiv t}{t \equiv u} \qquad \frac{t \equiv u \quad u \equiv v}{t \equiv v}$$

$$\frac{t_1 \equiv u_1 \quad \dots \quad t_n \equiv u_n}{\sigma(t_1, \dots, t_n) \equiv \sigma(u_1, \dots, u_n)} \qquad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

where s^\sharp is the extension of s to terms (defined by substitution).

In the context of structural operational semantics, equations are often interpreted by the *structural congruence rule*:

$$\frac{t \equiv_E u \quad u \xrightarrow{a} u' \quad u' \equiv_E v}{t \xrightarrow{a} v} \tag{8}$$

Informally, this rule states that we can deduce transitions modulo the congruence generated by the equations. In fact, removing the part $u' \equiv_E v$ from the premise (and writing u' instead of v in the conclusion) does not affect the behaviour, modulo bisimilarity [17]. See [17] for details on the interpretation of structural congruences in the context of transition systems.

We denote by $(T\emptyset)/\equiv_E$ the set of equivalence classes, and by $q: T\emptyset \rightarrow (T\emptyset)/\equiv_E$ the quotient map of \equiv_E . Thus $q(t) = q(u)$ iff $t \equiv_E u$. Assuming the axiom of choice, we further have $t \equiv_E u$ iff there is a right inverse $r: (T\emptyset)/\equiv_E \rightarrow T\emptyset$ of q such that $q \circ r(t) = u$. This is exploited in the operational interpretation of a specification together with a set of equations.

Definition 22. *Let $\theta: \mathbb{M} \rightarrow \mathbb{M}$ be the (unique) function such that*

$$\theta(f) \circ \nu = F\mu \circ \rho \circ \Sigma\langle f, \text{id} \rangle \vee \bigvee_{r \in R} f \circ r \circ q \circ \nu.$$

where R is the set of right inverses of q . A (ρ, E) -model is a coalgebra $f \in \mathbb{M}$ such that $\theta(f) = f$.

Lemma 23. θ is monotone.

Definition 24. The interpretation of ρ and E is the least (ρ, E) -model.

Example 25. Consider the specification of the parallel operator $x \mid y$ as given in (2) in the introduction, i.e., by a single rule and commutativity. In the interpretation, if $t \xrightarrow{a} t'$ then $t \mid u \xrightarrow{a} t' \mid u$ simply by the SOS rule. But also $u \mid t \xrightarrow{a} t' \mid u$, since $t \mid u \equiv_E u \mid t$. As for the definition of the replication operator by the equation $!x = !x \mid x$, for a term t the interpretation contains the least set of transitions from $!t$ which satisfy the equation, as desired.

On stream systems, abstract GSOS specifications correspond to behavioural differential equations, which are guarded, that is, for each operator (or constant) one defines its initial value concretely. For example, one can define $\text{zip}(x, y) = o(x) : (y, x')$, where $o(x)$ denotes the initial value of x and x' its derivative (its tail); and, e.g., $\text{zeros} = 0 : \text{zeros}$ and $\text{ones} = 1 : \text{ones}$ define the streams consisting only of zeros and ones, respectively. Taking the discrete order on the functor (Example 2) we can now add *equations* to such specifications. For instance, the paper folding sequence can be defined by the equations² $h = \text{zip}(\text{ones}, \text{zeros})$ and $\text{pf} = \text{zip}(h, \text{pf})$. In the interpretation, pf then defines the paper folding sequence.

Unfortunately, bisimilarity is not a congruence when equations are added [17]. For convenience we recall the counterexample on transition systems.

Example 26 ([17]). Consider rules $\overline{p \xrightarrow{a} p}$ and $\overline{q \xrightarrow{a} p}$ and the single equation $p = \sigma(q)$, where p, q are constants, σ is a unary operator and a is an arbitrary label. In the interpretation, p is bisimilar to q , but $\sigma(p)$ is not bisimilar to $\sigma(q)$.

The solution of [17] is to introduce a restricted format of equations, called *cfsc*. It is then shown that for any tyft specification combined with *cfsc* equations, bisimilarity is a congruence.

Definition 27. A set of equations $E \subseteq TV \times TV$ is in *cfsc* format with respect to ρ if every equation is of one of the following forms:

1. A σx -equation: $\sigma_1(x_1, \dots, x_n) = \sigma_2(y_1, \dots, y_n)$, where $\sigma_1, \sigma_2 \in \Sigma$ are of arity n (possibly $\sigma_1 = \sigma_2$), x_1, \dots, x_n are distinct variables and y_1, \dots, y_n is a permutation of x_1, \dots, x_n .
2. A defining equation: $\sigma(x_1, \dots, x_n) = t$ where $\sigma \in \Sigma$ and t is an arbitrary term (which may involve σ again); x_1, \dots, x_n are distinct variables, and all variables that occur in t are in x_1, \dots, x_n . Moreover σ does not appear in any other equation in E , and $\rho_X(\sigma(u_1, \dots, u_n)) = \perp$ for any set X and any $u_1, \dots, u_n \in FX \times X$.

² This example was taken from a presentation by Jörg Endrullis.

A σx -equation allows to assign simple algebraic properties to operators which already have behaviour; the prototypical example here is commutativity, like in the specification of the parallel operator in (2). With a *defining equation*, one can define the behaviour of an operator. Examples are $!x = !x \mid x$ and $\text{pf} = \text{zip}(\text{h}, \text{pf})$ (pf and h are constants). Associativity of \mid is neither a σx -equation nor a defining one; see [17] for a discussion why the *cfsc* format cannot be trivially extended. The *cfsc* format depends on an abstract GSOS specification: operators at the left hand side of a defining equation should not get any behaviour in the specification.

In [17], σx -equations are a bit more liberal in that they do not require the arities of σ and σ' to coincide, and do allow variables which only occur on one side of the equation. But in the interpretation these variables are quantified universally over closed terms; thus, we can encode this using infinitely many equations. We work with the simpler format above for technical convenience.

We proceed to show that the interpretation of an abstract GSOS specification ρ and a set of equations E in *cfsc* equals the operational model of a certain other specification, up to bisimilarity. This is done by encoding equations in this format as assignment rules, and using the theory of the previous section to obtain the desired result.

First, notice that for any σx -equation $\sigma_1(x_1, \dots, x_n) = \sigma_2(y_1, \dots, y_n)$, the variables on one side are a permutation of the variables on the other, and thus it can equivalently be represented as a triple (σ_1, σ_2, p) where $p: \text{ld}^n \rightarrow \text{ld}^n$ is the natural transformation corresponding to the permutation given by the equation. Below, we use $t[x_1, \dots, x_n := t_1, \dots, t_n]$ to denote the simultaneous substitution of variables x_1, \dots, x_n by terms t_1, \dots, t_n in a term t .

Definition 28. *A set of equations E in cfsc defines a set of assignment rules Δ^E as follows:*

1. For every σx -equation (σ_1, σ_2, p) we define d and d' on a component X as

$$d_X(\sigma(u_1, \dots, u_n)) = \begin{cases} \sigma_2(p_X(u_1, \dots, u_n)) & \text{if } \sigma = \sigma_1 \\ \sigma(u_1, \dots, u_n) & \text{otherwise} \end{cases}$$

for all $u_1, \dots, u_n \in X$, and d' is similarly defined using the inverse permutation p^{-1} , and σ_1 and σ_2 swapped.

2. For every defining equation $\sigma_1(x_1, \dots, x_n) = t$ we define a corresponding assignment rule $d_X(\sigma(u_1, \dots, u_n)) = \begin{cases} t[x_1, \dots, x_n := u_1, \dots, u_n] & \text{if } \sigma = \sigma_1 \\ \sigma(u_1, \dots, u_n) & \text{otherwise} \end{cases}$ for any set X and all $u_1, \dots, u_n \in X$.

If $\sigma(x_1, \dots, x_n) = t$ is a defining equation of a set of equations in the *cfsc* format, then the behaviour of $\sigma(x_1, \dots, x_n)$ will be the same as that of t .

Lemma 29. *Let E be a set of equations in cfsc format w.r.t. ρ , and let ψ be the function of Definition 7 for ρ and Δ^E . Then for any defining equation $\sigma(x_1, \dots, x_n) = t$ and any terms $t_1, \dots, t_n \in T\emptyset$: $(\text{lfp } \psi)(\sigma(t_1, \dots, t_n)) = (\text{lfp } \psi)(t[x_1, \dots, x_n := t_1, \dots, t_n])$.*

The following lemma is the main step towards the correctness of the encoding.

Lemma 30. *Let E and ψ be as above. If $t \equiv_E u$ then $Fq \circ (\text{lfp } \psi)(t) = Fq \circ (\text{lfp } \psi)(u)$, where q is the quotient map of \equiv_E .*

This allows to prove that $\text{lfp } \psi$ and $\text{lfp } \theta$ coincide “up to \equiv_E ”.

Lemma 31. *Let ψ and q be as above. Then $Fq \circ (\text{lfp } \theta) = Fq \circ (\text{lfp } \psi)$.*

This implies that $\text{lfp } \theta$ and $\text{lfp } \psi$ are *behaviourally equivalent* up to \equiv_E . It is well-known that behavioural equivalence coincides with bisimilarity whenever the functor F preserves weak pullbacks [22], a mild condition satisfied by most functors used in practice, including, e.g., transition systems and stream systems. Under this assumption one can prove that $\text{lfp } \theta$ is equal to $\text{lfp } \psi$ up to bisimilarity, and by Theorem 18 we then obtain our main result of this section.

Theorem 32. *Suppose E is a set of equations which is in *cfsc* format w.r.t. ρ , and suppose the behaviour functor F preserves weak pullbacks. Then the interpretation of ρ and E equals the operational model of some abstract GSOS specification, up to bisimilarity. Bisimilarity is a congruence, and bisimulation up to context and bisimilarity is compatible.*

6 Related Work

The main work on structural congruences [17] focuses on labelled transition systems, whereas our results apply to the more general notion of coalgebras. As for transition systems, the basic rule format in [17] is *tyft/tyxt*³, which is strictly more general than positive GSOS since it allows lookahead. However, while [17] proves congruence of bisimilarity this does not imply the compatibility (or even soundness) of bisimulation up to context [19], which we obtain in the present work (and is in fact problematic in the presence of lookahead).

In the bialgebraic setting, Klin [10] showed that by moving to CPPO-enriched categories, one can interpret recursive constructs which have a similar form as our assignment rules. Technically our approach, based on ordered functors, is different; it allows us to stay in the familiar category of sets and apply the coalgebraic bisimulation-up-to techniques of [20], which are based in this category. Further, in [10] each operator is either specified by an equation or by operational rules, disallowing a specification such as that of the parallel operator in equation (2). Plotkin proposed to move to CPPO in [18] to model recursion. The recent [15] applies the theory of [10] to add silent transitions to their concept of open GSOS laws. This is then used to show that equations are preserved by conservative extensions.

In [14] various constructions on distributive laws are presented. Their Example 32 discusses the definition of the parallel operator as in (2) above, but a

³ In [17] it is sketched how to extend the results to the *ntyft/ntyxt*, which involves however a complicated integration of the *cfsc* format with the notion of stable model.

general theory for structural congruence is missing. In [5] it is shown how to obtain a distributive law for a monad that is the quotient of another one by imposing extra equations, under the condition that the distributive law respects the equations. However, this condition requires that the equations already hold semantically, which is fundamentally different from the present paper where we define behaviour by imposing equations on an operational specification. Similarly in [6,7] it is shown how to lift calculi with structural axioms to coalgebraic models, but under the assumption, again, that the equations already hold.

7 Conclusions

We extended Turi and Plotkin's bialgebraic approach to operational semantics with non-structural assignment rules and structural congruence, providing a general coalgebraic framework for monotone abstract GSOS with equations. Our main result is that the interpretation of a specification involving assignment rules is well-behaved, in the sense that bisimilarity is a congruence and bisimulation-up-to techniques are sound. This result carries over to specifications with structural congruence in the *cfsc* format proposed in [17].

There are several promising directions for future work. First, one could extend our techniques to allow lookahead in premises by using cofree comonads (e.g., [12]). While in general the combined use of cofree comonads and free monads in specifications is known to be problematic, we expect that these problems do not arise when considering positive (monotone) specifications. In fact, this could form the basis for a bialgebraic account of the *tyft* format. Unfortunately, the compatibility results of bisimulation-up-to do not hold in a setting with lookahead. Second, in the current work we only consider free monads. One can possibly incorporate equations which already hold, by using the theory of [5]. Finally, it is worthwhile to consider categorical generalisations to allow, e.g., to study structural congruences for calculi with names. It could also lead to congruence formats for notions of equivalence other than bisimilarity.

Acknowledgments. We would like to thank Daniel Gebler, Bartek Klin, Mohammad Reza Mousavi and Jan Rutten for helpful suggestions and discussions. Our research has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier number: 612.063.920.

References

1. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Handbook of Process Algebra, pp. 197–292. Elsevier Science (2001)
2. Baeten, J.C.M., Verhoef, C.: A congruence theorem for structured operational semantics with predicates. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 477–492. Springer, Heidelberg (1993)
3. Bartels, F.: On generalised coinduction and probabilistic specification formats. PhD thesis, CWI, Amsterdam (April 2004)

4. Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. *J. ACM* 42(1), 232–268 (1995)
5. Bonsangue, M.M., Hansen, H.H., Kurz, A., Rot, J.: Presenting distributive laws. In: Heckel, R., Milius, S. (eds.) *CALCO 2013*. LNCS, vol. 8089, pp. 95–109. Springer, Heidelberg (2013)
6. Buscemi, M.G., Montanari, U.: A first order coalgebraic model of π -calculus early observational equivalence. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) *CONCUR 2002*. LNCS, vol. 2421, pp. 449–465. Springer, Heidelberg (2002)
7. Corradini, A., Heckel, R., Montanari, U.: Compositional SOS and beyond: a coalgebraic view of open systems. *Theor. Comput. Sci.* 280(1-2), 163–192 (2002)
8. Fiore, M., Staton, S.: Positive structural operational semantics and monotone distributive laws. *CMCS Short Contributions*, 8 (2010)
9. Hughes, J., Jacobs, B.: Simulations in coalgebra. *Theor. Comput. Sci.* 327(1-2), 71–108 (2004)
10. Klin, B.: Adding recursive constructs to bialgebraic semantics. *JLAP* 60-61, 259–286 (2004)
11. Klin, B.: Structural operational semantics for weighted transition systems. In: Palsberg, J. (ed.) *Semantics and Algebraic Specification*. LNCS, vol. 5700, pp. 121–139. Springer, Heidelberg (2009)
12. Klin, B.: Bialgebras for structural operational semantics: An introduction. *TCS* 412(38), 5043–5069 (2011)
13. Kupke, C., Niqui, M., Rutten, J.: Stream differential equations: concrete formats for coinductive definitions. *Tech. Report No. RR-11-10*, Oxford University (2011)
14. Lenisa, M., Power, J., Watanabe, H.: Category theory for operational semantics. *Theor. Comput. Sci.* 327(1-2), 135–154 (2004)
15. Madlener, K., Smetsers, S., van Eekelen, M.: Modular bialgebraic semantics and algebraic laws. In: Du Bois, A.R., Trinder, P. (eds.) *SBLP 2013*. LNCS, vol. 8129, pp. 46–60. Springer, Heidelberg (2013)
16. Milner, R.: Functions as processes. *MSCS* 2(2), 119–141 (1992)
17. Mousavi, M.R., Reniers, M.A.: Congruence for structural congruences. In: Sassone, V. (ed.) *FOSSACS 2005*. LNCS, vol. 3441, pp. 47–62. Springer, Heidelberg (2005)
18. Plotkin, G.D.: Bialgebraic semantics and recursion (extended abstract). *Electr. Notes Theor. Comput. Sci.* 44(1), 285–288 (2001)
19. Pous, D., Sangiorgi, D.: Enhancements of the bisimulation proof method. In: *Advanced Topics in Bisimulation and Coinduction*, pp. 233–289. Cambridge University Press (2012)
20. Rot, J., Bonchi, F., Bonsangue, M.M., Pous, D., Rutten, J.J.M.M., Silva, A.: Enhanced coalgebraic bisimulation, <http://www.liacs.nl/~jrot/up-to.pdf>
21. Rot, J., Bonsangue, M., Rutten, J.: Coalgebraic bisimulation-up-to. In: van Emde Boas, P., Groen, F.C.A., Italiano, G.F., Nawrocki, J., Sack, H. (eds.) *SOFSEM 2013*. LNCS, vol. 7741, pp. 369–381. Springer, Heidelberg (2013)
22. Rutten, J.: Universal coalgebra: a theory of systems. *TCS* 249(1), 3–80 (2000)
23. Rutten, J.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *TCS* 308(1-3), 1–53 (2003)
24. Sangiorgi, D.: On the bisimulation proof method. *MSCS* 8(5), 447–479 (1998)
25. Sangiorgi, D.: *An introduction to Bisimulation and Coinduction*. Cambridge University Press (2012)
26. Sangiorgi, D., Walker, D.: *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press (2001)
27. Turi, D., Plotkin, G.: Towards a mathematical operational semantics. In: *LICS*, pp. 280–291. IEEE Computer Society (1997)

Models of a Non-associative Composition*

Guillaume Munch-Maccagnoni

Univ Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126 CNRS,
PIR2, INRIA Paris-Rocquencourt, F-75205 Paris, France

Abstract. We characterise the polarised evaluation order through a categorical structure where the hypothesis that composition is associative is relaxed. Duploid is the name of the structure, as a reference to Jean-Louis Loday’s duplicit algebras. The main result is a reflection $\mathcal{Adj} \rightarrow \mathcal{Dupl}$ where \mathcal{Dupl} is a category of duploids and duploid functors, and \mathcal{Adj} is the category of adjunctions and pseudo maps of adjunctions. The result suggests that the various biases in denotational semantics: indirect, call-by-value, call-by-name... are ways of hiding the fact that composition is not always associative.

1 Introduction

In a term language where the order of evaluation is determined by the polarity of a type or formula, it is not immediate that composition is associative. The associativity of categorical composition amounts to the following equation on terms:

$$\text{let } y \text{ be } t \text{ in let } x \text{ be } u \text{ in } v \stackrel{?}{=} \text{let } x \text{ be } (\text{let } y \text{ be } t \text{ in } u) \text{ in } v$$

Now, in any setting where t could be of a type that implies a strict evaluation order, and where u could be of a type that implies a delayed evaluation, we can see a difference in spirit between these two terms. Indeed, in the left-hand term, the evaluation of t would happen before the one of v . On the contrary, in the right-hand term, the evaluation of x is delayed since it has the same type as u , so the term would compute v before t .

This phenomenon is observed with polarisation in logic and denotational semantics [7,21,5,16,12]. Polarisation can thus be described (negatively) as rejecting, either directly or indirectly, the hypothesis that composition is *a priori* associative. In this article, we give a positive and direct description of a polarised evaluation order. To this effect we introduce a category-like structure where not all composites associate. *Duploid* is the name of the structure, as a reference to Jean-Louis Loday’s duplicit algebras [15].

The main result relates duploids to adjunctions. To help understand this relation, let us first recall the correspondence between direct models of call by value and indirect models *à la* Moggi.

Direct Models. In a *direct* denotational model, there should be a close match between the given operations in the model and the constructions in the language. Essentially, type and program constructors should respectively correspond to operations on objects

* This is a shortened version of Chapter II from the author’s PhD thesis [20, pp. 86-91,103-152].

Table 1. Comparison of the structures underlying various direct models of computation

Evaluation order	By value	By name	Polarised
Direct model	Thunk	Runnable monad	Duploid
Indirect model	Monad T	Co-monad L	Adjunction $F \dashv G$
Programs	Kleisli maps $P \rightarrow TQ$	Co-Kleisli maps $LN \rightarrow M$	Oblique maps $FP \rightarrow N$ $\simeq P \rightarrow GN$
Syntactic data	Values	Stacks	Both
Completion into	Thunkable expressions	Linear evaluation contexts	Both

and on morphisms in a category. In particular, it should be possible to reason about an instance of the model within the language.¹ An example of direct models for the simply-typed lambda calculus is given by cartesian-closed categories.

In a model such as Moggi's λ_C models [18], or Lafont, Reus and Streicher's models of call by name [10], however, the language is not interpreted directly but through a Kleisli construction for a monad or a co-monad. We have a precise description of the link between direct models and indirect models thanks to Führmann [6]. Categories that model call by value directly are characterised by the presence of a *think*, a formal account of the well-known structure used to implement laziness in call-by-value languages.

The characterisation takes the following form: any direct model arises from the Kleisli construction starting from a λ_C model. However, from the direct model we can only recover a specific λ_C model: its values are made of all the pure expressions. More precisely, the Kleisli construction is a *reflection* that conflates any two values equalised by the monad, and turns into a value any *thinkable* expression. An expression is thinkable if it behaves similarly to a value in a sense determined by the monad.

Selinger [23] proves a similar relationship between direct models of the call-by-name $\lambda\mu$ calculus and Lafont, Reus and Streicher's models [10].

Adjunction-Based Models. This article deals with the underlying algebraic structure in these models: a monad over a category of values for call by value, a co-monad over a category of stacks for call by name. Duploids generalise the underlying structure to an adjunction between a category of values and a category of stacks. (See table 1.)

Relationship with polarities comes from Girard's *polarised translation* of classical logic [7,5,11]. Our duploid construction extends the (skeleton of the) polarised translation to any adjunction. (Notably, we do not need the assumption that there is an involutive negation operation on formulae.)

We know that there is a practical relevance of decomposing monads, when seen as notions of computation, into adjunctions, thanks to Levy [13,14]. Levy's adjunctions subsume models of call by value and call by name. However the model is indirect, and still lacks a corresponding notion of direct model.

¹ Führmann [6], Selinger [22].

Outline. Section 2 introduces pre-duploids as categories where the associativity of composition is deficient. Section 3 defines duploids as pre-duploids with additional structure, and characterises this additional structure. The category *Dupl* of duploids and duploid functors is introduced. Section 4 proves the main result.

Structure Theorem. The main result is a reflection $\boxed{\text{Dupl} \triangleleft \text{Adj}}$, where *Adj* is the category of adjunctions and *pseudo maps of adjunctions*. In other words, the duploid construction extends to a functor $\text{Adj} \rightarrow \text{Dupl}$ that admits a full and faithful right adjoint. In particular, any duploid is obtained from an adjunction, but adjunctions obtained from duploids are peculiar.

As a consequence of the main result, duploids account for a wide range of computational models, as we will see in various examples. It suggests that the various biases in denotational semantics: indirect, call-by-value, call-by-name... are ways of hiding the fact that composition is not always associative.

In addition, the article develops an internal language for duploids. It provides intuitions from programming languages and abstract machines about polarisation.

Characterisation of Duploids. We also characterise the adjunctions obtained from duploids. We show that there is an equivalence of categories $\boxed{\text{Dupl} \simeq \text{Adj}_{\text{eq}}}$, where Adj_{eq} is the full subcategory of adjunctions that satisfies the *equalizing requirement*: the unit and the co-unit of the adjunction are respectively equalisers and co-equalisers.

This means that the duploid operates from the point of view of the model of computation defined by the adjunction: first any two values and any two stacks that are not distinguished by the model of computation are identified; and then the categories of values and stacks are respectively completed with all the expressions that are thinkable, and with all the evaluation contexts that are linear.

2 Pre-duploids

We define pre-duploids, which are category-like structures whose objects have a polarity, and which miss associativity of composition when the middle map has polarity $+$ \rightarrow \ominus .

Definition 1. A pre-duploid \mathcal{D} is given by:

1. A set $|\mathcal{D}|$ of objects together with a polarity mapping $\varpi : |\mathcal{D}| \rightarrow \{+, \ominus\}$.
2. For all $A, B \in |\mathcal{D}|$, a set of morphisms or hom-set $\mathcal{D}(A, B)$.
3. For all morphisms $f \in \mathcal{D}(A, B)$ and $g \in \mathcal{D}(B, C)$, a morphism $g \circ f \in \mathcal{D}(A, C)$, also written as follows depending on the polarity of B :

$$\begin{aligned} g \bullet f &\in \mathcal{D}(A, C) \text{ if } \varpi(B) = +, \\ g \circ f &\in \mathcal{D}(A, C) \text{ if } \varpi(B) = \ominus. \end{aligned}$$

The following associativities must hold for all objects $A, B \in |\mathcal{D}|$; $P, Q \in \varpi^{-1}(\{+\})$ and $N, M \in \varpi^{-1}(\{\ominus\})$:

($\bullet\bullet$) For all $A \xrightarrow{f} P \xrightarrow{g} Q \xrightarrow{h} B$, one has $(h \bullet g) \bullet f = h \bullet (g \bullet f)$;

($\circ\circ$) For all $A \xrightarrow{f} N \xrightarrow{g} M \xrightarrow{h} B$, one has $(h \circ g) \circ f = h \circ (g \circ f)$;

($\bullet\circ$) For all $A \xrightarrow{f} N \xrightarrow{g} P \xrightarrow{h} B$, one has $(h \bullet g) \circ f = h \bullet (g \circ f)$.

4. For all $A \in |\mathcal{D}|$, a morphism $\text{id}_A \in \mathcal{D}(A, A)$ neutral for \circ .

The mapping ϖ defines a partition of $|\mathcal{D}|$ into the *positive* objects P, Q, \dots in $|\mathcal{P}| \stackrel{\text{def}}{=} \varpi^{-1}(\{+\})$ and the *negative* objects N, M, \dots in $|\mathcal{N}| \stackrel{\text{def}}{=} \varpi^{-1}(\{\ominus\})$. This partition defines categories \mathcal{P} (whose composition is given by \bullet) and \mathcal{N} (whose composition is given by \circ) in an obvious way.

2.1 Linear and Thinkable Morphisms

Definition 2. Let \mathcal{D} be a pre-duploid. A morphism f of \mathcal{D} is *linear* if for all g, h one has:

$$f \circ (g \circ h) = (f \circ g) \circ h$$

A morphism f of \mathcal{D} is *thinkable* if for all g, h one has:

$$h \circ (g \circ f) = (h \circ g) \circ f$$

Thus any morphism $f : P \rightarrow A$ is linear, and any morphism $f : A \rightarrow N$ is thinkable. The terminology *thinkable* is borrowed from [24,6]. These notions are closed under composition and identity.

Definition 3. We define sub-categories of \mathcal{D} as follows:

\mathcal{D}_l is the sub-category of linear morphisms of \mathcal{D} .

\mathcal{D}_t ————— thinkable morphisms of \mathcal{D} .

\mathcal{N}_l ————— linear morphisms of \mathcal{N} .

\mathcal{P}_t ————— thinkable morphisms of \mathcal{P} .

Observe that \mathcal{N} and \mathcal{N}_l are respectively the full sub-categories of \mathcal{D}_t and \mathcal{D}_l with negative objects. Symmetrically, \mathcal{P} and \mathcal{P}_t are respectively the full sub-categories of \mathcal{D}_l and \mathcal{D}_t whose objects are positive.

Proposition 4. The hom-sets $\mathcal{D}(A, B)$ of a pre-duploid \mathcal{D} extend to a (pro-)functor $\mathcal{D}(-, =) : \mathcal{D}_l^{\text{op}} \times \mathcal{D}_l \rightarrow \mathbf{Set}$ defined for $f \in \mathcal{D}_l(A, B)$ and $g \in \mathcal{D}_l(C, D)$ with $\mathcal{D}(f, g) : \mathcal{D}(B, C) \rightarrow \mathcal{D}(A, D)$; $\mathcal{D}(f, g)(h) = g \circ h \circ f$.

Proof. Restricting to f thinkable and g linear makes the definition unambiguous. Functoriality follows from $(g_1 \circ g_2) \circ h \circ (f_2 \circ f_1) = g_1 \circ (g_2 \circ h \circ f_2) \circ f_1$, which holds when f_1 and f_2 are thinkable and g_1 and g_2 are linear. ■

2.2 Examples of Pre-duploids

Girard’s Classical Logic. Girard’s correlation spaces are a denotational semantics for classical logic. They do not form a category for lack of associativity of the composition [7,12]. However, they form a pre-duploid.

Blass Games. Blass [3] gives a game model for linear logic that fails to satisfy the associativity of composition. Thanks to Abramsky’s analysis of this issue [1], we know that associativity fails due to composites of the form $N \rightarrow P \rightarrow M \rightarrow Q$. According to Abramsky, “*none of the other 15 polarisations give rise to a similar problem*”. Therefore, Abramsky’s formalisation of Blass games yields a pre-duploid. Thanks to Melliès’s analysis of this so-called “*Blass problem*” [16], we know that the phenomenon is essentially the same as for Girard’s classical logic.

Direct Models of Call by Value. Führmann [6] characterises the Kleisli category of a monad *via* the presence of a structure called *thunk*. In the contexts of models of call by value, the thunk implements laziness. Recall that a *thunk-force category* is a category $(\mathcal{P}, \bullet, \text{id})$ together with a thunk $(L, \varepsilon, \vartheta)$ as defined next.

Definition 5 (Führmann). A thunk on \mathcal{P} is given by a functor $L : \mathcal{P} \rightarrow \mathcal{P}$ together with a natural transformation $\varepsilon : L \rightarrow 1$ and a transformation $\vartheta : 1 \rightarrow L$ such that the transformation $\vartheta_L : L \rightarrow L^2$ is natural; satisfying the equations $\varepsilon \bullet \vartheta = \text{id}$ and $L\varepsilon \bullet \vartheta_L = \text{id}_L$ and $\vartheta_L \bullet \vartheta = L\vartheta \bullet \vartheta$.

A thunk induces a comonad $(L, \varepsilon, \vartheta_L)$.

Observe that in a thunk-force category $(\mathcal{P}, \bullet, \text{id}, L, \vartheta, \varepsilon)$, we can define a composite of $g : P \rightarrow Q$ and $f : LQ \rightarrow R$ with $g \circ f \stackrel{\text{def}}{=} g \bullet Lf \bullet \vartheta_P$. This compositions admits ε_P as a neutral element. This extends to a pre-duploid with compositions \bullet and \circ as follows. The positive objects are the objects of \mathcal{P} . The set of negative objects is given with $|\mathcal{N}| = \uparrow|\mathcal{P}|$ for \uparrow a suitably chosen bijection with domain $|\mathcal{P}|$ (in other words $|\mathcal{N}|$ is a disjoint copy of $|\mathcal{P}|$). Then we take $\mathcal{D}(A, B) \stackrel{\text{def}}{=} \mathcal{P}(\overline{A}, \underline{B})$ where we define $\overline{P} = \underline{P} \stackrel{\text{def}}{=} P$ and $\overline{\uparrow P} \stackrel{\text{def}}{=} LP$ and $\underline{\uparrow P} \stackrel{\text{def}}{=} P$. With this definition, \circ is a map $\mathcal{D}(\uparrow P, B) \times \mathcal{D}(A, \uparrow P) \rightarrow \mathcal{D}(A, B)$ and ε_P is an element of $\mathcal{D}(\uparrow P, \uparrow P)$. It is easy to check that this defines a pre-duploid.

In the context of λ_C models, this pre-duploid formalises how thunks implement laziness in call by value.

Now recall that Führmann calls *thunkable* any morphism $f \in \mathcal{P}(P, Q)$ such that $Lf \bullet \vartheta_P = \vartheta_Q \bullet f$. Not all morphisms of \mathcal{P} are *thunkable* in general because ϑ is not necessarily natural. We can prove the following:

Proposition 6. A morphism $f : P \rightarrow Q$ is *thunkable* in the sense of *thunk-force categories* if and only if it is *thunkable* in the sense of *pre-duploids*.

Thus the transformation ϑ is natural if and only if the pre-duploid is a category (*i.e.* satisfies $\circ \bullet$ -associativity).

Direct Models of Call by Name. The concept dual to Führmann’s thunk is the one of *runnable monad*. A runnable monad on a category \mathcal{C} is given by a functor $T : \mathcal{C} \rightarrow \mathcal{C}$ together with a natural transformation $\eta : 1 \rightarrow T$ and a transformation $\rho : T \rightarrow 1$ such that the transformation $\rho_T : T^2 \rightarrow T$ is natural; satisfying the equations $\rho \circ \eta = \text{id}$; $\rho_T \circ T\eta = \text{id}_T$ and $\rho \circ T\rho = \rho \circ \rho_T$.

Runnable monads implement strictness in call by name. An example of a category with a runnable monad is given by Selinger’s direct models of the call-by-name $\lambda\mu$ calculus [23]. Given a runnable monad, we can define, symmetrically to *thunk-force categories* above, a pre-duploid with a bijective map $\Downarrow : |\mathcal{N}| \rightarrow |\mathcal{P}|$.

2.3 Syntactic Pre-duploid

The syntactic pre-duploid is given by a term syntax. It is made of terms (t) with a polarity which are identified up to β - and η -like equations. These equations are best described with auxiliary syntactic categories for evaluation contexts (e) and abstract machines ($c = \langle t \parallel e \rangle$); a technique that arose in the theory of control operators [4,8,2].

There are four sets of variables written x^+ , α^+ , x^\ominus , α^\ominus to consider, and the following grammar (“...” indicates that we consider an extensible grammar):

$$\begin{array}{l|l|l}
 t_+ ::= V_+ \mid \mu\alpha^+.c \mid \dots & e_+ ::= \alpha^+ \mid \tilde{\mu}x^+.c \mid \dots & \\
 t_\ominus ::= x^\ominus \mid \mu\alpha^\ominus.c \mid \dots & e_\ominus ::= \pi \mid \tilde{\mu}x^\ominus.c \mid \dots & \\
 V_+ ::= x^+ \mid \dots & \pi_\ominus ::= \alpha^\ominus \mid \dots & c ::= \langle t_+ \parallel e_+ \rangle \mid \langle t_\ominus \parallel e_\ominus \rangle \\
 V ::= V_+ \mid t_\ominus & \pi ::= \pi_\ominus \mid e_+ & \text{(c) Commands} \\
 \text{(a) Terms and values} & \text{(b) Contexts and stacks} &
 \end{array}$$

Fig. 1. The syntactic pre-duploid (*the variables that appear before a dot are bound*)

The binders are μ and $\tilde{\mu}$. They allow us to define composition as follows:

$$\boxed{\text{let } x \text{ be } t \text{ in } u \stackrel{\text{def}}{=} \mu\alpha. \langle t \parallel \tilde{\mu}x. \langle u \parallel \alpha \rangle \rangle} \quad (\alpha \notin \text{fv}(t, u))$$

In this macro-definition, the polarities of t and x must be the same, and the polarity of α and of “let x be t in u ” is determined by the one of u .

The contextual equivalence relation \simeq determines the equality of morphisms. It is induced by the following rewrite rules:

$$\begin{array}{ll}
 \langle \mu\alpha.c \parallel \pi \rangle \triangleright c[\pi/\alpha] & t \triangleright \mu\alpha. \langle t \parallel \alpha \rangle \quad (\alpha \notin \text{fv}(t)) \\
 \langle V \parallel \tilde{\mu}x.c \rangle \triangleright c[V/x] & e \triangleright \tilde{\mu}x. \langle x \parallel e \rangle \quad (x \notin \text{fv}(e))
 \end{array}$$

The intuition is that positive terms are called by value while negative terms are called by name. Indeed we have:

$$\langle \text{let } x \text{ be } V \text{ in } u \parallel \pi \rangle \triangleright^* \langle u[V/x] \parallel \pi \rangle$$

but for a positive non-value t_+ instead of V , computation continues with t_+ . This describes a call-by-value reduction. And with a negative non-stack e_\ominus instead of π computation is delayed until a stack (a linear evaluation context) is reached. This describes a call-by-name reduction. In the latter case, “let x be V in u ” and therefore u are negative.

Among other equations, we have for all terms and variables:

$$\text{let } x \text{ be } y \text{ in } t \simeq t[y/x] \tag{1}$$

$$\text{let } x \text{ be } t \text{ in } x \simeq t \tag{2}$$

$$\text{let } y^+ \text{ be } (\text{let } x \text{ be } t \text{ in } u_+) \text{ in } v \simeq \text{let } x \text{ be } t \text{ in let } y^+ \text{ be } u_+ \text{ in } v \tag{3}$$

$$\text{let } y \text{ be } (\text{let } x^\ominus \text{ be } t_\ominus \text{ in } u) \text{ in } v \simeq \text{let } x^\ominus \text{ be } t_\ominus \text{ in let } y \text{ be } u \text{ in } v \tag{4}$$

The syntactic pre-duploid gives rise to a pre-duploid with two objects $+$ and \ominus and with formal objects $(x \mapsto t) : \varepsilon_1 \rightarrow \varepsilon_2$ as morphisms, where ε_1 and ε_2 determine the polarities of x and t (respectively). Equations (1) and (2) mean that modulo α -conversion, variables provide a neutral element for the composition. Equations (3) and (4) correspond respectively to $\bullet\ominus$ - and $\ominus\bullet$ - associativity.

It is not possible to rewrite “let y^\ominus be (let x^+ be t_+ in u_\ominus) in v ” into “let x^+ be t_+ in let y^\ominus be u_\ominus in v ”. In other words, without imposing additional equations, we have in general $h \circ (g \bullet f) \neq (h \circ g) \bullet f$.

3 Duploids

We now enrich pre-duploids with operators of polarity coercion \Downarrow, \Uparrow called *shifts*.¹

Definition 7. A duploid is a pre-duploid \mathcal{D} given with mappings $\Downarrow : |\mathcal{N}| \rightarrow |\mathcal{P}|$ and $\Uparrow : |\mathcal{P}| \rightarrow |\mathcal{N}|$, together with, for all $P \in |\mathcal{P}|$ and $N \in |\mathcal{N}|$, morphisms subject to equations:

$\begin{aligned} \text{delay}_P &: P \rightarrow \Uparrow P \\ \text{force}_P &: \Uparrow P \rightarrow P \\ \text{wrap}_N &: N \rightarrow \Downarrow N \\ \text{unwrap}_N &: \Downarrow N \rightarrow N \end{aligned}$	$\begin{aligned} \text{force}_P \circ (\text{delay}_P \bullet f) &= f \quad (\forall f \in \mathcal{D}(A, P)) \\ (f \circ \text{unwrap}_N) \bullet \text{wrap}_N &= f \quad (\forall f \in \mathcal{D}(N, A)) \\ \text{delay}_P \bullet \text{force}_P &= \text{id}_{\Uparrow P} \\ \text{wrap}_N \circ \text{unwrap}_N &= \text{id}_{\Downarrow N} \end{aligned}$
--	--

Proposition 8. For any N , wrap_N is thinkable. Dually, for any P , force_P is linear.

Proof. For all g, h we have $h \circ (g \bullet \text{wrap}_N) = (h \circ (g \bullet \text{wrap}_N) \circ \text{unwrap}_N) \bullet \text{wrap}_N = (h \circ (g \bullet \text{wrap}_N \circ \text{unwrap}_N)) \bullet \text{wrap}_N = (h \circ g) \bullet \text{wrap}_N$. Hence wrap_N is linear. The other result follows by symmetry. ■

Thus we have the following equivalent definition of a duploid:

Definition 9. A duploid is a pre-duploid \mathcal{D} given with mappings $\Downarrow : |\mathcal{N}| \rightarrow |\mathcal{P}|$ and $\Uparrow : |\mathcal{P}| \rightarrow |\mathcal{N}|$, together with a family of invertible linear maps $\text{force}_P : \Uparrow P \rightarrow P$ and a family of invertible thinkable maps $\text{wrap}_N : N \rightarrow \Downarrow N$.

3.1 Syntactic Duploid

Let us start with the syntax, with which we provide computational intuitions for the shifts. The syntactic duploid extends the syntactic pre-duploid with a type $\Uparrow P$ of suspended strict computations, and a type $\Downarrow N$ of lazy computations encapsulated into a value. Then $\text{delay} \bullet f$ represents the suspended strict computation f and the inverse operation force triggers the evaluation of its argument (this is why it is linear in its negative argument). The morphism $\text{wrap} \circ f$ represents f encapsulated into a value (this is why it is thinkable) and unwrap removes the encapsulation.

¹ Our notation is reminiscent of Mellès [16].

We extend the syntactic pre-duploid as follows:

$$\begin{array}{l|l}
 V_+ ::= \dots | \{t_\ominus\} | \dots & e_+ ::= \dots | \tilde{\mu}\{x^\ominus\}.c | \dots \\
 t_\ominus ::= \dots | \mu\{\alpha^+\}.c | \dots & \pi_\ominus ::= \dots | \{e_+\} | \dots \\
 \text{(a) Terms and values} & \text{(b) Contexts and stacks}
 \end{array}$$

Fig. 2. The syntactic duploid (extending the syntactic pre-duploid)

We also extend the relation \triangleright with the following rules:

$$\boxed{
 \begin{array}{ll}
 \langle \{t_\ominus\} \parallel \tilde{\mu}\{x^\ominus\}.c \rangle \triangleright c[t_\ominus/x^\ominus] & e_+ \triangleright \tilde{\mu}\{x^\ominus\}.\langle \{x^\ominus\} \parallel e_+ \rangle \quad (x^\ominus \notin \text{fv}(e_+)) \\
 \langle \mu\{\alpha^+\}.c \parallel \{e_+\} \rangle \triangleright c[e_+/\alpha^+] & t_\ominus \triangleright \mu\{\alpha^+\}.\langle t_\ominus \parallel \{\alpha^+\} \rangle \quad (\alpha^+ \notin \text{fv}(t_\ominus))
 \end{array}
 }$$

The new constructions add to the syntax of terms the following operations (in addition to values $\{t_\ominus\}$):

$$\begin{aligned}
 \text{let } \{x^\ominus\} \text{ be } t_+ \text{ in } u &\stackrel{\text{def}}{=} \mu\alpha.\langle t_+ \parallel \tilde{\mu}\{x^\ominus\}.\langle u \parallel \alpha \rangle \rangle \\
 \text{delay}(t_+) &\stackrel{\text{def}}{=} \mu\{\alpha^+\}.\langle t_+ \parallel \alpha^+ \rangle \\
 \text{force}(t_\ominus) &\stackrel{\text{def}}{=} \mu\alpha^+.\langle t_\ominus \parallel \{\alpha^+\} \rangle
 \end{aligned}$$

We have in particular:

$$\begin{array}{ll}
 \text{let } \{x^\ominus\} \text{ be } \{t_\ominus\} \text{ in } u \simeq u[t_\ominus/x^\ominus] & \text{let } \{x^\ominus\} \text{ be } t_+ \text{ in } \{x^\ominus\} \simeq t_+ \\
 \text{force}(\text{delay}(t_+)) \simeq t_+ & \text{delay}(\text{force}(t_\ominus)) \simeq t_\ominus
 \end{array}$$

We can show that this extends the syntactic pre-duploid into a duploid (with wrap and unwrap interpreted as $x^\ominus \mapsto \{x^\ominus\}$ and $x^+ \mapsto \text{let } \{y^\ominus\} \text{ be } x^+ \text{ in } y^\ominus$, respectively).

3.2 The Duploid Construction

Let \mathcal{C}_1 and \mathcal{C}_2 be two categories and $F \dashv G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ an adjunction given by natural transformations $\sharp : \mathcal{C}_1(F-, =) \rightarrow \mathcal{C}_2(-, G=)$ and $\flat = \sharp^{-1}$. Note $G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$.

The goal of the duploid construction is to define a notion of morphisms $A \rightarrow B$ for A and B objects of *either* category \mathcal{C}_1 and \mathcal{C}_2 . Let us introduce the convention that objects of \mathcal{C}_1 are negative and written N, M, \dots , while the objects of \mathcal{C}_2 are positive and written P, Q, \dots . Also, we write \bullet the composition in \mathcal{C}_1 and \circ the composition in \mathcal{C}_2 .

We first define *oblique morphisms* $P \rightarrow_{\mathcal{Q}} N$, with $P \in |\mathcal{C}_2|$ and $N \in |\mathcal{C}_1|$, equivalently as maps $P \rightarrow GN$ or $FP \rightarrow N$ (thanks to the isomorphism \sharp). Then we observe that oblique morphisms compose either in \mathcal{C}_1 or in \mathcal{C}_2 as follows:

$$\begin{array}{c}
 \frac{f : P \rightarrow_{\mathcal{Q}} FQ \quad g : Q \rightarrow_{\mathcal{Q}} N}{f : FP \rightarrow FQ \quad g : FQ \rightarrow N} \quad \frac{f : P \rightarrow_{\mathcal{Q}} N \quad g : GN \rightarrow_{\mathcal{Q}} M}{f : P \rightarrow GN \quad g : GN \rightarrow GM} \\
 \hline
 \frac{g \bullet f : FP \rightarrow N}{g \bullet f : P \rightarrow_{\mathcal{Q}} N} \quad \frac{g \circ f : P \rightarrow GM}{g \circ f : P \rightarrow_{\mathcal{Q}} M}
 \end{array}$$

Thus we define morphisms $A \rightarrow_{\mathcal{Q}} B$ as oblique morphisms:

$$\boxed{A^+ \rightarrow_{\mathcal{Q}} B^\ominus}, \quad \text{where} \quad \begin{array}{ll} P^+ \stackrel{\text{def}}{=} P & P^\ominus \stackrel{\text{def}}{=} FP \\ N^+ \stackrel{\text{def}}{=} GN & N^\ominus \stackrel{\text{def}}{=} N \end{array}$$

In other words, we define $|\mathcal{D}| \stackrel{\text{def}}{=} |\mathcal{C}_1| \uplus |\mathcal{C}_2|$ and (taking an irrelevant bias towards \mathcal{C}_1) we define $\mathcal{D}(A, B) \stackrel{\text{def}}{=} \mathcal{C}_1(FA^+, B^\ominus)$. Positive composition is given by the composition in \mathcal{C}_1 . Composition of $f \in \mathcal{D}(A, N)$ and $g \in \mathcal{D}(N, B)$ is given by $g \circ^{\mathcal{D}} f \stackrel{\text{def}}{=} (g^\# \circ^{\mathcal{C}_2} f^\#)^\flat$. Identities are given with $\text{id}_P^{\mathcal{D}} \stackrel{\text{def}}{=} \text{id}_{FP}^{\mathcal{C}_1}$ and $\text{id}_N^{\mathcal{D}} \stackrel{\text{def}}{=} \text{id}_{GN}^{\mathcal{C}_2}{}^\flat$.

Proposition 10. *The above defines a pre-duploid \mathcal{D} .*

Proof (sketch). $\bullet\bullet$ -associativity is given, and $\circ\circ$ -associativity is immediate using the fact that $\#$ and \flat are inverse. $\bullet\circ$ -associativity relies on the fact that the transformations $\#$ and \flat are natural. \blacksquare

Remark 11. In particular \mathcal{P} is the Kleisli category $(\mathcal{C}_2)_{GF}$ of the monad GF and \mathcal{N} is the Kleisli category $(\mathcal{C}_1)_{FG}$ of the co-monad FG .

The pre-duploid has shifts, defined as follows:

$$\begin{aligned} \uparrow P &\stackrel{\text{def}}{=} FP & \Downarrow N &\stackrel{\text{def}}{=} GN \\ \mathcal{D}(P, \uparrow P) &\ni \text{delay}_P \stackrel{\text{def}}{=} \text{id}_{FP}^{\mathcal{C}_1} \in \mathcal{C}_1(FP, FP) \\ \mathcal{D}(\uparrow P, P) &\ni \text{force}_P \stackrel{\text{def}}{=} (\text{id}_{GFP})^\flat \in \mathcal{C}_1(FGFP, FP) \\ \mathcal{D}(N, \Downarrow N) &\ni \text{wrap}_N \stackrel{\text{def}}{=} \text{id}_{FGN}^{\mathcal{C}_1} \in \mathcal{C}_1(FGN, FGN) \\ \mathcal{D}(\Downarrow N, N) &\ni \text{unwrap}_N \stackrel{\text{def}}{=} (\text{id}_{GN})^\flat \in \mathcal{C}_1(FGN, N) \end{aligned}$$

It is easy to see that:

Proposition 12. *Every adjunction determines a duploid as above.*

3.3 Linear and Thinkable Morphisms in Duploids

In duploids, we have the following useful characterisation of linear and thinkable morphisms.

Proposition 13. *In a duploid \mathcal{D} , let $f \in \mathcal{D}(A, P)$. Then f is thinkable if and only if:*

$$(\text{wrap}_{\uparrow P} \circ \text{delay}_P) \bullet f = \text{wrap}_{\uparrow P} \circ (\text{delay}_P \bullet f) \quad (5)$$

Dually, let $f \in \mathcal{D}(N, B)$. Then f is linear if and only if:

$$f \circ (\text{unwrap}_N \bullet \text{force}_{\Downarrow N}) = (f \circ \text{unwrap}_N) \bullet \text{force}_{\Downarrow N}$$

Proof. We establish the non-trivial implication for the first case. The second case is obtained by symmetry. First we prove that any morphism that satisfies (5) also satisfies $(h \circ \text{delay}_P) \bullet f = h \circ (\text{delay}_P \bullet f)$ for any $h \in \mathcal{D}(\uparrow P, A)$. Indeed for any such h we have:

$$\begin{aligned} (h \circ \text{delay}_P) \bullet f &= (h \circ \text{unwrap}_{\uparrow P}) \bullet \underline{(\text{wrap}_{\uparrow P} \circ \text{delay}_P) \bullet f} \\ &= (h \circ \underline{\text{unwrap}_{\uparrow P}}) \bullet \underline{\text{wrap}_{\uparrow P}} \circ (\text{delay}_P \bullet f) \quad (\text{by hypothesis}) \\ &= h \circ (\text{delay}_P \bullet f) \end{aligned}$$

Now we prove that f is thinkable. For any g, h we have:

$$\begin{aligned}
 (h \circ \underline{g}) \bullet f &= \underline{(h \circ (g \bullet \text{force}_P) \circ \text{delay}_P)} \bullet f \\
 &= h \circ \underline{(g \bullet \text{force}_P) \circ (\text{delay}_P \bullet f)} \quad (\text{with the above}) \\
 &= h \circ ((g \bullet \underline{\text{force}_P \circ \text{delay}_P}) \bullet f) \quad (\text{with the above again}) \\
 &= h \circ (g \bullet f) \quad \blacksquare
 \end{aligned}$$

By applying the above proposition to the duploid construction, we easily deduce the following:

Proposition 14. *Let $F \dashv_{(\eta, \varepsilon)} G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ be an adjunction, and consider the associated duploid \mathcal{D} . Then $f \in \mathcal{D}(N, A)$ is linear if and only if $f \circ \varepsilon_{FGN} = f \circ FG\varepsilon_N$ (in \mathcal{C}_1), and $f \in \mathcal{D}(A, P)$ is thinkable if and only if its transpose $f^\sharp \in \mathcal{C}_2(A^+, GFP)$ satisfies $\eta_{GFP} \circ f^\sharp = GF\eta_P \circ f^\sharp$ (in \mathcal{C}_2).*

Now recall that an adjunction $F \dashv G$ that satisfies either of the following equivalent statements is called *idempotent*: the multiplication of the associated monad is an isomorphism; or the co-multiplication of the associated co-monad is an isomorphism; or we have $\varepsilon_{GF} = GF\varepsilon$; or we have $\eta_{FG} = FG\eta$. Thus we deduce the following:

Corollary 15. *Let $F \dashv_{(\eta, \varepsilon)} G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$. The associated duploid \mathcal{D} is a category if and only if the adjunction is idempotent.*

3.4 Structure of Shifts

As we have seen, the Kleisli category of a co-monad is described by a runnable monad; and the Klesli category of a monad is described by a thunk, which is a co-monad. We observe a similar phenomenon with duploids. We show that there is a reversed adjunction, in the sense that the right adjoint \uparrow is from positives to negatives:

$$\boxed{\Downarrow \dashv \uparrow : \mathcal{P} \rightarrow \mathcal{N}}$$

Actually, we state a wider adjunction. First remark that we can extend the shifts \Downarrow, \uparrow to all objects in a straightforward manner:

$$\begin{array}{ll}
 \Downarrow A \stackrel{\text{def}}{=} \begin{cases} \Downarrow N & \text{if } A = N \\ P & \text{if } A = P \end{cases} & \text{delay}_N \stackrel{\text{def}}{=} \text{id}_N : N \rightarrow \uparrow N \\
 \uparrow A \stackrel{\text{def}}{=} \begin{cases} N & \text{if } A = N \\ \uparrow P & \text{if } A = P \end{cases} & \text{force}_N \stackrel{\text{def}}{=} \text{id}_N : \uparrow N \rightarrow N \\
 & \text{wrap}_P \stackrel{\text{def}}{=} \text{id}_P : P \rightarrow \Downarrow P \\
 & \text{unwrap}_P \stackrel{\text{def}}{=} \text{id}_P : \Downarrow P \rightarrow P
 \end{array}$$

By “extend”, we mean that we have for all f, g :

$$\begin{array}{ll}
 (f \circ \text{force}_A) \circ (\text{delay}_A \circ g) = f \circ g & \text{delay}_A \circ \text{force}_A = \text{id}_A \\
 (f \circ \text{unwrap}_A) \bullet (\text{wrap}_A \circ g) = f \circ g & \text{wrap}_A \circ \text{unwrap}_A = \text{id}_A
 \end{array}$$

Also, extending proposition 8, we have, for all objects A , that unwrap_A and wrap_A are thinkable whereas delay_A and force_A are linear.

Proposition 16. *Let \mathcal{D} be a duploid. The following:*

$$\Uparrow f \stackrel{\text{def}}{=} \text{delay}_B \circ f \circ \text{force}_A \qquad \Downarrow f \stackrel{\text{def}}{=} \text{wrap}_B \circ f \circ \text{unwrap}_A$$

define functors $\Uparrow : \mathcal{D}_l \rightarrow \mathcal{N}_l$ and $\Downarrow : \mathcal{D}_l \rightarrow \mathcal{P}_l$ that take part in adjoint equivalences of categories $I \dashv_{(\text{delay}, \text{force})} \Uparrow : \mathcal{D}_l \rightarrow \mathcal{N}_l$ and $I \dashv_{(\text{wrap}, \text{unwrap})} \Downarrow : \mathcal{D}_l \rightarrow \mathcal{P}_l$, where I denotes the inclusion functors.

Proof (sketch). The result follows from the fact that delay and force are inverse natural transformations in \mathcal{D}_l ; likewise for wrap and unwrap in \mathcal{D}_l . ■

We can deduce the following:

Proposition 17. *Let \mathcal{D} be a duploid. We have natural isomorphisms between (pro-)functors $\boxed{\mathcal{D}_l(-, I\Uparrow-) \simeq \mathcal{D}(-, =) \simeq \mathcal{D}_l(I\Downarrow-, =) : \mathcal{D}_l^{\text{op}} \times \mathcal{D}_l \rightarrow \mathbf{Set}}$ where I denotes the inferrable inclusion functors.*

In particular, leaving the inclusion functors implicit, we have the adjunctions:

$$\begin{array}{ccc} & \Downarrow & \\ \mathcal{D}_l & \xrightarrow{\quad} & \mathcal{D}_l \\ & \Uparrow & \end{array} \qquad \begin{array}{ccc} & \Downarrow & \\ \mathcal{N} & \xrightarrow{\quad} & \mathcal{P} \\ & \Uparrow & \end{array}$$

The adjunction $\Downarrow \dashv \Uparrow$ distinguishes our interpretation of polarities from ones based on adjunctions of the form $\Uparrow \dashv \Downarrow$ that appears in the context of focusing in logic and continuation-passing style in programming (see Laurent [11], Zeilberger [25]). Our direct notion of polarities adds a level of granularity. In terms of continuations, our polarities makes the distinction between continuations that are meant to be applied and continuations that are meant to be passed.

3.5 The Category of Duploids

Definition 18. A functor of pre-duploids $F : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ is given by a mapping on objects $|F| : |\mathcal{D}_1| \rightarrow |\mathcal{D}_2|$ that preserves polarities, together with mappings on morphisms $F_{A,B} : \mathcal{D}_1(A, B) \rightarrow \mathcal{D}_2(FA, FB)$, satisfying $\text{Fid}_A = \text{id}_{FA}$ and $F(g \circ f) = Fg \circ Ff$. A functor of duploids $F : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ is a functor of pre-duploids such that $F\text{force}_P$ is linear for all $P \in |\mathcal{P}_1|$, and $F\text{wrap}_N$ is thinkable for all $N \in |\mathcal{N}_1|$.

Proposition 19. *Let \mathcal{D} and \mathcal{D}' be two duploids and let $F : \mathcal{D} \rightarrow \mathcal{D}'$ be a mapping on objects $|F| : |\mathcal{D}| \rightarrow |\mathcal{D}'|$ that preserves polarities, together with mappings on morphisms $F_{A,B} : \mathcal{D}(A, B) \rightarrow \mathcal{D}'(FA, FB)$. Then F is a functor of duploids if and only if F restricts to functors $F_l : \mathcal{D}_l \rightarrow \mathcal{D}'_l$ and $F_l : \mathcal{D}_l \rightarrow \mathcal{D}'_l$, such that the transformation $F : \mathcal{D}(-, =) \rightarrow \mathcal{D}'(F_l-, F_l=)$ is natural.*

Proof. (\Leftarrow) is easy to prove. (\Rightarrow) : Suppose that F is a functor of duploids. First we establish that the full sub-pre-duploid $F\mathcal{D}$ of \mathcal{D}' with objects of the form FA for $A \in |\mathcal{D}|$ has a duploid structure given by $F\text{delay}$, $F\text{force}$, $F\text{wrap}$ and $F\text{unwrap}$. This follows from definition 9, using the hypothesis that $F\text{force}$ is linear and $F\text{wrap}$ is thinkable.

Then, considering proposition 13 applied to the duploid $F\mathcal{D}$, we show that F preserves linearity and thunkability. In other words it restricts to functors $F_l : \mathcal{D}_l \rightarrow \mathcal{D}'_l$ and $F_r : \mathcal{D}_r \rightarrow \mathcal{D}'_r$. That $F : \mathcal{D}(-, =) \rightarrow \mathcal{D}'(F_l-, F_r=)$ is a natural transformation follows from $Fh \circ Fg \circ Ff = F(h \circ g \circ f)$ which makes sense for h linear and f thunkable. ■

Definition 20. *Dupl* is the category whose objects are duploids and whose morphisms are duploid functors. The obvious identity in *Dupl* is written $1_{\mathcal{D}}$.

3.6 Examples of Duploids

The Blass Phenomenon in Conway Games. Melliès [16] comes close to building a duploid using the construction of Blass games. According to his analysis [16, Section 3], the Blass problem comes down to the fact that the (pro-)functor $\mathcal{C}_1(F-, =) : \mathcal{C}_2^{\text{op}} \times \mathcal{C}_1 \rightarrow \mathbf{Set}$, in the terminology of Section 3.2, does not extend into a functor $\mathcal{P}^{\text{op}} \times \mathcal{N} \rightarrow \mathbf{Set}$ where \mathcal{P} and \mathcal{N} are respectively the Kleisli categories of the monad GF and the comonad FG . This is the essence of proposition 15. He then defines a category for an asynchronous variant of Conway games. As he shows, asynchronism is a way to force the double-negation monad to be idempotent, and therefore to recover associativity of composition. He builds this way a game model of linear logic.

Girard’s Polarisation. Girard’s polarised translation of the classical logic **LC** into intuitionistic logic [7], further formulated by Danos, Joinet and Schellinx [5] and Laurent [11], inspired the duploid construction. Girard’s translation corresponds to considering in the duploid construction the self-adjunction of the negation functor $\neg = R^-$ in **Set** for R arbitrary. But obviously, the duploid obtained from the self-adjunction of negation in any response category (in the terminology of Selinger [23]) gives a denotational semantics of **LC**. Thielecke [24] later noticed the importance of this self-adjunction in the understanding of continuation-passing style.

Response categories have recently been refined into dialogue categories by Melliès and Tabareau [17] to provide a denotational semantics of linear logic *via* the polarised translation. This was conceived as an abstract account of the asynchronous games of Melliès [16] mentioned above.

Direct Models of Call by Value and of Call by Name. We defined a pre-duploid with a bijection $\uparrow : |\mathcal{P}| \rightarrow |\mathcal{N}|$ from a thunk-force category $(\mathcal{P}, \bullet, \text{id}, L, \vartheta, \varepsilon)$. We complete the definition into a duploid by defining $\downarrow : |\mathcal{N}| \rightarrow |\mathcal{P}|$ with $\downarrow \uparrow P \stackrel{\text{def}}{=} LP$; and delay, force, wrap, unwrap in an obvious manner. We can show that thunk-force categories are characterised as duploids where \uparrow is bijective on objects. Symmetrically, we can show that categories with a runnable monad are characterised as duploids where \downarrow is bijective on objects.

4 Structure Theorem

4.1 Every Duploid Comes from an Adjunction

Proposition 21. *Let \mathcal{D} be a duploid. We define $\uparrow : \mathcal{P}_l \rightarrow \mathcal{N}_l$ the restriction of \uparrow and $\downarrow : \mathcal{N}_r \rightarrow \mathcal{P}_r$ the restriction of \downarrow . There is an adjunction $\uparrow \dashv \downarrow$ with unit $\text{wrap}_{\uparrow} \circ \text{delay}$ and co-unit $\text{unwrap} \bullet \text{force}_{\downarrow}$.*

Proof. Due to the adjoint equivalences from proposition 16, we have the following natural isomorphisms:

$$\begin{aligned} \mathcal{N}_I(\uparrow I-, =) &\simeq \mathcal{D}_I(I-, I=) : \mathcal{P}_I^{\text{op}} \times \mathcal{N}_I \rightarrow \mathbf{Set} \\ \mathcal{P}_I(-, \Downarrow I=) &\simeq \mathcal{D}_I(I-, I=) : \mathcal{P}_I^{\text{op}} \times \mathcal{N}_I \rightarrow \mathbf{Set}, \end{aligned}$$

where I denotes the inferrable inclusion functors. Since we have $\mathcal{D}_I(P, N) = \mathcal{D}(P, N) = \mathcal{D}_I(P, N)$, we also have $\mathcal{D}_I(I-, I=) = \mathcal{D}_I(I-, I=)$ above. Thus we have a natural isomorphism $\mathcal{N}_I(\uparrow -, =) = \mathcal{N}_I(\uparrow I-, =) \simeq \mathcal{P}_I(-, \Downarrow I=) = \mathcal{P}_I(-, \Downarrow -)$. We can check that the unit is $\text{wrap}_{\uparrow} \circ \text{delay}$ and the co-unit is $\text{unwrap} \bullet \text{force}_{\Downarrow}$. ■

Proposition 22. *There is an isomorphism between \mathcal{D} and the duploid \mathcal{D}' obtained from the above adjunction $\uparrow \dashv \downarrow$.*

Proof (sketch). Recall that \mathcal{D}' is defined with $|\mathcal{D}'| = |\mathcal{D}|$ and $\mathcal{D}'(A, B) = \mathcal{N}_I(\uparrow \Downarrow A, \uparrow B)$. According to propositions 16 and 17, we have natural isomorphisms $\mathcal{D}(-, =) \simeq \mathcal{D}_I(I\Downarrow -, =) \simeq \mathcal{N}_I(\uparrow \Downarrow -, \uparrow =)$, and thus for all $A, B \in |\mathcal{D}|$ we have a bijection $\mathcal{D}(A, B) \rightarrow \mathcal{D}'(A, B)$. It is easy to verify that this mapping defines a functor of duploids $F : \mathcal{D} \rightarrow \mathcal{D}'$. Using the characterisation of proposition 19, its inverse is a functor of duploids. ■

4.2 The Equalising Requirement

Definition 23. *An adjunction $F \dashv_{(\eta, \varepsilon)} G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ satisfies the equalising requirement if and only if for all $P \in |\mathcal{C}_2|$, η_P is an equaliser of $\eta_{GF P}$ and $GF\eta_P$, and for all $N \in |\mathcal{C}_1|$, ε_N is a co-equaliser of $\varepsilon_{FG N}$ and $FG\varepsilon_N$.*

We give an equivalent formulation of this condition in terms of the associated duploid:

Proposition 24. *Let $F \dashv_{(\eta, \varepsilon)} G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ be an adjunction, and consider the associated duploid \mathcal{D} . The adjunction satisfies the equalising requirement if and only if for all objects A, P, N the following three conditions hold:*

1. ε_N is an epimorphism and η_P is a monomorphism; or equivalently G and F are faithful;
2. all linear morphisms $f \in \mathcal{D}(N, A)$ are of the form $g \circ \varepsilon_N$ with $g \in \mathcal{C}_1(N, A^-)$; or equivalently all linear morphisms are in the image of G modulo the adjunction;
3. all thinkable morphisms $f \in \mathcal{D}(A, P)$ are (modulo the adjunction) of the form $\eta_P \circ g$ with $g \in \mathcal{C}_2(A^+, P)$; or equivalently all thinkable morphisms are in the image of F ;

Proof (sketch). Follows from the characterisation in proposition 14. ■

Proposition 25. *Let \mathcal{D} be a duploid and consider the adjunction $\uparrow \dashv \downarrow : \mathcal{N}_I \rightarrow \mathcal{P}_I$. The adjunction satisfies the equalising requirement.*

Proof (sketch). Follows easily from the fact that $\varepsilon = \text{unwrap} \bullet \text{force}_{\Downarrow}$ has a section in \mathcal{N} , namely $\text{delay}_{\Downarrow} \bullet \text{wrap} : 1 \dashrightarrow \uparrow \Downarrow$, and symmetrically for η . ■

4.3 Main Result

We consider pseudo maps of adjunctions as defined by Jacobs [9]:

Definition 26. Let $F \dashv_{(\eta, \varepsilon)} G : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ and $F' \dashv_{(\eta', \varepsilon')} G' : \mathcal{C}'_1 \rightarrow \mathcal{C}'_2$ be two adjunctions. A pseudo map of adjunctions:

$$(H_1, H_2, \phi, \psi) : (F \dashv_{(\eta, \varepsilon)} G) \rightarrow (F' \dashv_{(\eta', \varepsilon')} G')$$

is given by a pair of functors $H_1 : \mathcal{C}_1 \rightarrow \mathcal{C}'_1$ and $H_2 : \mathcal{C}_2 \rightarrow \mathcal{C}'_2$ together with natural isomorphisms $\phi : F'H_2 \xrightarrow{\sim} H_1F$ and $\psi : G'H_1 \xrightarrow{\sim} H_2G$, such that H_1 and H_2 preserve η and ε up to isomorphism: $H_2\eta = \psi_F \circ G'\phi \circ \eta'_{H_2}$ and $H_1\varepsilon = \varepsilon'_{H_1} \circ F'\psi^{-1} \circ \phi_G^{-1}$.

As noted by Jacobs, two pseudo maps (H_1, H_2, ϕ, ψ) and $(H'_1, H'_2, \phi', \psi')$ compose as:

$$(H'_1, H'_2, \phi', \psi') \circ (H_1, H_2, \phi, \psi) = (H'_1H_1, H'_2H_2, H'_1\phi \circ \phi'_{H_2}, H'_2\psi \circ \psi'_{H_1}).$$

Definition 27. The category of adjunctions \mathbf{Adj} has adjunctions between locally small categories as objects and pseudo maps of adjunctions as morphisms. The full subcategory \mathbf{Adj}_{eq} of \mathbf{Adj} consists in adjunctions that satisfy the equalising requirement.

Theorem 28. There are a reflection and an equivalence as follows:

$$\mathbf{Dupl} \simeq \mathbf{Adj}_{\text{eq}} \triangleleft \mathbf{Adj}$$

Proof (sketch). The functor $j : \mathbf{Adj} \rightarrow \mathbf{Dupl}$ is given on objects by the duploid construction. The functor $i : \mathbf{Dupl} \rightarrow \mathbf{Adj}_{\text{eq}}$ is given on objects by proposition 25. Proposition 22 gives the family of isomorphisms $ji\mathcal{D} \simeq \mathcal{D}$.

The complete proof appears in the author’s PhD thesis, Chapter II [20].

Intuitively, theorem 28 together with proposition 24 mean that the duploid construction j completes the values with all the expressions that are pure, and completes the stacks with all the evaluation contexts that are linear. Moreover j identifies any two values that denote the same expression, and any two stacks that denote the same evaluation context.

5 Ongoing Work

This work was developed during a collaboration with Marcelo Fiore and Pierre-Louis Curien, in an effort to connect the **L** system [4,8,19,20] with adjunction models. The calculus suggests that connectives should have an elegant characterisation in terms of duploids, which is the subject of an ongoing work.

Acknowledgements. I am grateful to Pierre-Louis Curien and Marcelo Fiore for their influence and support, as well as for their careful reading and suggestions. I am grateful to Paul-André Melliès for the many discussions. For discussions relevant to this article I also wish to thank Jean-Yves Girard, Hugo Herbelin and Olivier Danvy. Thanks to the anonymous reviewers for their suggestions. This work has been partially supported by the French National Research Agency¹ and by the Fondation Sciences Mathématiques de Paris.

¹ Projects CHOCO (ANR-07-BLAN-0324), COQUAS (ANR-12-JS02-006-01), LOGOI (ANR-10-BLAN-0213) and RECRE (ANR-11-BS02-0010).

References

1. Abramsky, S.: Sequentiality vs. concurrency in games and logic. *Math. Struct. Comput. Sci.* 13(4), 531–565 (2003)
2. Ariola, Z.M., Herbelin, H.: Control Reduction Theories: the Benefit of Structural Substitution. *Journal of Functional Programming* 18(3), 373–419 (2008)
3. Blass, A.: A game semantics for linear logic. *Ann. Pure Appl. Logic* 56(1-3), 183–220 (1992)
4. Curien, P.L., Herbelin, H.: The duality of computation. *ACM SIGPLAN Notices* 35, 233–243 (2000)
5. Danos, V., Joinet, J.B., Schellinx, H.: A New Deconstructive Logic: Linear Logic. *Journal of Symbolic Logic* 62(3), 755–807 (1997)
6. Führmann, C.: Direct Models for the Computational Lambda Calculus. *Electr. Notes Theor. Comput. Sci.* 20, 245–292 (1999)
7. Girard, J.-Y.: A new constructive logic: Classical logic. *Math. Struct. Comp. Sci.* 1(3), 255–296 (1991)
8. Herbelin, H.: C'est maintenant qu'on calcule, au cœur de la dualité, Habilitation thesis (2005)
9. Jacobs, B.: Comprehension categories and the semantics of type dependency. *Theor. Comput. Sci.* 107(2), 169–207 (1993)
10. Lafont, Y., Reus, B., Streicher, T.: Continuation Semantics or Expressing Implication by Negation. Technical report, University of Munich (1993)
11. Laurent, O.: Etude de la polarisation en logique. Thèse de doctorat, Université Aix-Marseille II (March 2002)
12. Laurent, O., Quatrini, M., Tortora de Falco, L.: Polarized and focalized linear and classical proofs. *Ann. Pure Appl. Logic* 134(2-3), 217–264 (2005)
13. Levy, P.B.: Call-by-Push-Value: A Subsuming Paradigm. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 228–243. Springer, Heidelberg (1999)
14. Levy, P.B.: Adjunction models for call-by-push-value with stacks. In: *Proc. Cat. Th. and Comp. Sci. ENTCS*, vol. 69 (2005)
15. Loday, J.-L.: Generalized bialgebras and triples of operads. arXiv preprint math/0611885 (2006)
16. Melliès, P.A.: Asynchronous Games 3 An Innocent Model of Linear Logic. *Electr. Notes Theor. Comput. Sci.* 122, 171–192 (2005)
17. Melliès, P.A., Tabareau, N.: Resource modalities in tensor logic. *Ann. Pure Appl. Logic* 161(5), 632–653 (2010)
18. Moggi, E.: Computational Lambda-Calculus and Monads. In: LICS (1989)
19. Munch-Maccagnoni, G.: Focalisation and classical realisability. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 409–423. Springer, Heidelberg (2009)
20. Munch-Maccagnoni, G.: Syntax and Models of a non-Associative Composition of Programs and Proofs. PhD thesis, Univ. Paris Diderot (2013)
21. Murthy, C.R.: A Computational Analysis of Girard's Translation and LC. In: LICS, pp. 90–101. IEEE Computer Society (1992)
22. Selinger, P.: Re: co-exponential question. Message to the Category Theory mailing list (July 1999), <http://permalink.gmane.org/gmane.science.mathematics.categories/1181>
23. Selinger, P.: Control Categories and Duality: On the Categorical Semantics of the Lambda-Mu Calculus. *Math. Struct. in Comp. Sci.* 11(2), 207–260 (2001)
24. Thielecke, H.: Categorical Structure of Continuation Passing Style. PhD thesis, University of Edinburgh (1997)
25. Zeilberger, N.: On the unity of duality. *Ann. Pure and App. Logic* 153(1) (2008)

Foundations for Decision Problems in Separation Logic with General Inductive Predicates

Timos Antonopoulos¹, Nikos Gorogiannis², Christoph Haase^{3,*},
Max Kanovich⁴, and Joël Ouaknine¹

¹ Department of Computer Science, University of Oxford, UK

² Department of Computer Science, Middlesex University London, UK

³ LSV, CNRS & École Normale Supérieure (ENS) de Cachan, France

⁴ Department of Computer Science, Queen Mary University of London, UK

Abstract. We establish foundational results on the computational complexity of deciding entailment in Separation Logic with general inductive predicates whose underlying base language allows for pure formulas, pointers and existentially quantified variables. We show that entailment is in general undecidable, and EXPTIME-hard in a fragment recently shown to be decidable by Iosif *et al.* Moreover, entailment in the base language is Π_2^P -complete, the upper bound even holds in the presence of list predicates. We additionally show that entailment in essentially any fragment of Separation Logic allowing for general inductive predicates is intractable even when strong syntactic restrictions are imposed.

1 Introduction

Separation Logic (SL) is an extension of Hoare logic for reasoning about programs which manipulate heap data structures. Introduced in the early 2000s by O’Hearn, Ishtiaq, Reynolds and Yang [18,20], it has been the starting point of a line of research that has led to a large body of theoretical and practical work.

In the early days, the potential of Separation Logic was recognised by proving the (partial) correctness of the Schorr-Waite graph marking algorithm [22] and Cheney’s copying garbage collector [6]. Those proofs were essentially carried out in a pen-and-paper fashion and demonstrated the strength of the paradigm underlying Separation Logic: *local reasoning*. The latter means that correctness proofs for heap-manipulating code should only depend on the portions of the heap accessed by the code and not the entire memory. Motivated by these positive results, research has been conducted on automating such proofs. On the one hand, support for Separation Logic has been integrated into proof assistants such as Coq, enabling semi-automated verification of program code, see *e.g.* [2]. On the other hand, a number of fully-automatic tools such as SMALLFOOT, SLAYER, SPACE INVADER or SLAD have been developed and successfully used to show absence of memory errors in low-level real-world code, see *e.g.* [11,7,5].

* Supported by the French ANR, REACHARD (grant ANR-11-BS02-001).

A crucial requirement for any of these tools is the ability to check applications of the consequence rule in Hoare logic, as it is this rule that underpins most methods of proof based on Separation Logic. The consequence rule, in turn, requires the ability to check entailment between Separation Logic formulas. However, entailment checking in full Separation Logic is undecidable [12,10], thus these tools have to work with restricted, decidable fragments. Decidability, though, comes at the cost of reduced expressive power and, often, reduced generality. For instance, the fragment used by SMALLFOOT allows for reasoning about memory shapes built upon the hard-coded primitives of pointers and linked lists, essentially limiting its applicability to programs only involving those data structures; some efforts have been made in order to allow for reasoning about more generic list data structures, see *e.g.* [3]. The limitations of hard-coded inductive predicates have been realised by the community, and recent research has been conducted to enable automated reasoning about generic user-defined inductive predicates, inside the framework of Separation Logic, see *e.g.* [13,9], or in related frameworks such as forest automata [16]. Notable recent progress has been made by Iosif *et al.* who showed decidability of satisfiability and entailment for a syntactic fragment of Separation Logic with general recursively defined predicates by establishing a reduction to Monadic Second Order Logic on graphs with bounded tree width [17]. Finally, Brotherston *et al.* have developed an EXPTIME-complete decision procedure for satisfiability of Separation Logic with general inductively defined predicates [8]. In the same paper it is shown that the problem becomes NP-complete if the arity of all predicates is bounded by a constant.

The goal of this paper is to contribute to this line of research and to establish foundational results on the inherent computational complexity of reasoning problems in Separation Logic with general inductively defined predicates. In order to obtain meaningful lower bounds, we restrict our analysis to the most basic syntactic fragment of Separation Logic comprising (positive) Boolean combinations of judgments on stack variables, both fixed and existentially quantified, and pointers. This fragment also forms the basis of the decidable fragments of Separation Logic from [17,8]. Standard inductive data types are inductively expressible in this fragment, for instance singly-linked lists as used by SMALLFOOT [4] can be defined as follows:

$$\text{ls}(a, b) := \text{emp} \wedge a = b \mid \exists c. \text{pt}(a, c) * \text{ls}(c, b) \wedge a \neq b \quad (1)$$

Informally speaking, supposing that $\text{ls}(x, y)$ holds in a memory model, this definition states that there is a singly-linked list segment from the memory cell labeled with the stack variable x to the memory cell labeled with y if either the heap is empty and x is equal to y , or x is not equal to y and the heap can be split into two disjoint parts, indicated by the $*$ -conjunction, such that on the first part x is allocated and points to some cell a and heap cell a is the starting point of a singly-linked list segment ending in y in the other part.

The main results of our paper are as follows. In the first part, we consider entailment in Separation Logic with general inductive predicates. Given two assertions α, α' and a finite set of inductive predicates P referred to by α and α' ,

entailment is to decide whether the set of memory models of α is contained in the set of memory models of α' with respect to P . We show that this problem is undecidable in general and EXPTIME-hard when restricted to the decidable syntactic fragment defined by Iosif *et al.* In the second part, we take a closer look at entailment in the basic fragment of Separation Logic in the absence of inductive predicates, *i.e.*, Separation Logic with positive pure formulas, existentially quantified variables and pointers. We show that this problem is complete for Π_2^P , the second level of the polynomial hierarchy. The upper bound also holds when allowing for the above list predicate hard-coded in the syntax. Subsequently, we analyse the Π_2^P lower bound and define a natural syntactic fragment for which entailment is decidable in polynomial time, yet NP-hard in the presence of a list predicate, *i.e.*, one of the simplest possible inductive predicates. We discuss the results obtained in the conclusion at the end of the paper.

Some proofs have been omitted due to lack of space, but are included in a longer, online version of the paper, obtainable from the authors' webpages.

2 Preliminaries

Let X and Y be sets, and let $R \subseteq X \times Y$. We say that R is *functional* if for every $x \in X$ there is at most one $y \in Y$ with $(x, y) \in R$. Let Y be a countable, possibly infinite, set. We write $X \subseteq_{\text{fin}} Y$ if X is a finite subset of Y . Moreover, given countable sets X, Y , we write $f : X \rightarrow_{\text{fin}} Y$ if f is a function whose domain is a finite subset of X and its co-domain is Y . Given $f : X \rightarrow Y$, $x \in X$, $y \in Y$, we write $f[x \mapsto y]$ to denote the function f' such that $f'(z) \stackrel{\text{def}}{=} y$ if $z = x$, and $f'(z) \stackrel{\text{def}}{=} f(z)$ otherwise. Finally, given $i \leq j \in \mathbb{N}$, we write $[i, j]$ to denote the set $\{i, \dots, j\} \subseteq \mathbb{N}$ and $[i]$ as an abbreviation for $[1, i]$.

Graphs. Let L be a countable set of *labels*. We define *directed labeled graphs* (just *graphs* in the following) as tuples $G = (V, E, \ell)$, where V denotes the set of *nodes* or *vertices*, E is a subset of $V \times V$, and $\ell : L \rightarrow_{\text{fin}} V$ is a *labeling function*. If L is empty we omit ℓ and just write $G = (V, E)$. A graph $G = (V, E, \ell)$ is *undirected* if E is symmetric. If G is a graph, we also denote its set of nodes by $V(G)$ and its set of edges by $E(G)$. The *size* of G is defined as $|G| \stackrel{\text{def}}{=} |V(G)|$.

For interpretations below, we require a slightly more general class of graphs which we call selector graphs, inspired by [17]. A *selector graph* is a tuple $G = (V, E, \ell, s)$, where V and ℓ are as above, $s : V \rightarrow \mathbb{N}$ assigns an *arity* to each vertex, and $E : V \times \mathbb{N} \rightarrow V$ is a partial function such that E is defined precisely for every pair (v, i) with $v \in V$ and $i \in [s(v)]$. If $s(v) \in \{0, 1\}$ for all $v \in V$, we obtain partial functional graphs.

Formulas of Separation Logic. The subsequent definitions are partly adapted or inspired from [17]. Let **Vars** be a totally ordered countably infinite set of *variable names*, which is partitioned into disjoint infinite sets **EVars** and **FVars** representing sets of *existential variables* and *fixed stack variables*, respectively. We will usually use $\mathbf{a}, \mathbf{b}, \mathbf{c}$ for elements from **EVars**, and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ will usually be elements from **Vars**. Variables in **FVars** will be used to represent fixed stack

variables. The purpose of the distinction between EVars and FVars is to help the reader to easily identify in which context a variable occurs. Let PNames be a finite set of *predicate names*, where each predicate has an associated arity $k \in \mathbb{N}$, and is written as $\text{pred}(\mathbf{a}_1, \dots, \mathbf{a}_k)$ with $\mathbf{a}_i \in \text{EVars}$ for $i \in [k]$. The syntax of *SL-assertions or SL-formulas over PNames* is given by the following grammar, where $\mathbf{x}, \mathbf{x}_1, \dots, \mathbf{x}_m, \mathbf{y}, \mathbf{y}_1, \dots, \mathbf{y}_k \in \text{Vars}$, $\mathbf{a}_1, \dots, \mathbf{a}_n \in \text{EVars}$, $m \geq 1$ and $n \geq 0$:

$$\begin{aligned} \varphi &::= \top \mid \perp \mid \mathbf{x} = \mathbf{y} \mid \neg\varphi \mid \varphi \wedge \varphi && \text{(pure formulas)} \\ \sigma &::= \text{emp} \mid \text{tt} \mid \text{pt}(\mathbf{x}, (\mathbf{x}_1, \dots, \mathbf{x}_m)) \mid \text{pred}(\mathbf{y}_1, \dots, \mathbf{y}_k) \mid \sigma * \sigma \mid \alpha && \text{(spatial formulas)} \\ \alpha &::= \exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma \wedge \varphi && \text{(SL-assertions)} \end{aligned}$$

Here, $\text{pt}(\mathbf{x}, \mathbf{y})$ is the *points-to* or *pointer predicate* of an arbitrary arity m , and the $*$ -conjunction is commutative, *i.e.*, SL-assertions are considered equivalent up to permutations of $*$ -connected subformulas. We say that the SL-assertion $\alpha = \exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma \wedge \varphi$ is *flat* if σ contains no SL-assertion α' as a subformula. Given an SL-assertion $\alpha = \exists \mathbf{a}_1, \dots, \mathbf{a}_m. (\sigma * \exists \mathbf{b}_1, \dots, \mathbf{b}_n. (\sigma' \wedge \varphi')) \wedge \varphi$, and supposing without loss of generality that $\{\mathbf{a}_i : i \in [m]\} \cap \{\mathbf{b}_j : j \in [n]\} = \emptyset$, we can exhaustively rewrite the formula α as $\exists \mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_n. (\sigma * \sigma') \wedge \varphi \wedge \varphi'$. Thus we may assume with no loss of generality that an SL-assertion is flat.

Remark 1. We have imposed flatness and $*$ -commutativity as syntactic properties. This is merely for presentational convenience in order to save space, as these properties follow from the semantics definition below.

We call φ *positive* if φ is a conjunction of literals $\mathbf{x} = \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$. Moreover, we say that α is positive if φ is positive, and that α is *reduced* if no $\text{pred}(\mathbf{y}_1, \dots, \mathbf{y}_k)$ occurs in σ . By $\text{vars}(\alpha) \subseteq \text{Vars}$ we denote the *set of all variables occurring in α* . The *size of α* , denoted by $|\alpha|$, is defined to be the number of symbols in α .

A *set P of inductive predicates over PNames* is a finite set of *definitions*

$$\text{pred}(\mathbf{a}_1, \dots, \mathbf{a}_k) := \alpha_1 \mid \dots \mid \alpha_m$$

such that each $\mathbf{a}_i \in \text{EVars}$, α_i is a flat SL-assertion $\alpha_i = \exists \mathbf{b}_1, \dots, \mathbf{b}_n. \sigma \wedge \varphi$ over PNames such that $\{\mathbf{a}_i : i \in [m]\} \cap \{\mathbf{b}_j : j \in [n]\} = \emptyset$, and each predicate name $\text{pred}(\mathbf{a}_1, \dots, \mathbf{a}_k)$ occurs exactly once on the left-hand side of a definition in P . Moreover, we require that each α_i has no unbounded existential variables, *i.e.*, for each α_m as above, $\text{vars}(\alpha_m) \subseteq \text{FVars} \cup \{\mathbf{a}_i, \mathbf{b}_j : i \in [k], j \in [n]\}$.¹ Given $\mathbf{x}_1, \dots, \mathbf{x}_k \in \text{Vars}$, define $\text{pred}[\mathbf{x}_1/\mathbf{a}_1, \dots, \mathbf{x}_k/\mathbf{a}_k] \stackrel{\text{def}}{=} \{\alpha_i[\mathbf{x}_1/\mathbf{a}_1, \dots, \mathbf{x}_k/\mathbf{a}_k] : i \in [k]\}$, where $\alpha_i[\mathbf{x}_1/\mathbf{a}_1, \dots, \mathbf{x}_k/\mathbf{a}_k]$ is obtained from α_i by replacing each occurrence of \mathbf{a}_j with \mathbf{x}_j for $j \in [k]$. Given a flat assertion $\alpha = \exists \mathbf{c}_1, \dots, \mathbf{c}_p. \sigma \wedge \varphi$, an *unraveling of α with respect to P* is obtained by replacing each $\text{pred}(\mathbf{x}_1, \dots, \mathbf{x}_k)$ occurring as a subformula in σ with some $\alpha' \in \text{pred}[\mathbf{x}_1/\mathbf{a}_1, \dots, \mathbf{x}_k/\mathbf{a}_k]$. We write $\alpha \rightarrow_P \beta$ if β is an unraveling of α with respect to P , and denote the reflexive transitive closure of \rightarrow_P by \rightarrow_P^* . An unraveling $\alpha \rightarrow_P^* \beta$ is *complete* if no $\text{pred}(\mathbf{x}_1, \dots, \mathbf{x}_k)$ occurs in β .

¹ In a slight departure from convention, for presentational convenience we allow free variables to appear in the body of definitions; such predicates can always be transformed to equivalent ones where the previously free variables are parameters, w.l.o.g.

Example 2. Taking P to be the singleton set consisting of the definition of $\text{ls}(\mathbf{a}, \mathbf{b})$ from Equation (1), we have

$$\text{ls}(\mathbf{x}, \mathbf{y}) \rightarrow_P^* \exists c_1, c_2. \text{pt}(\mathbf{x}, c_1) * \text{pt}(c_1, c_2) * \text{ls}(c_2, \mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y} \wedge c_1 \neq \mathbf{y}$$

with respect to P , which is *not* a complete unraveling. A complete unraveling is $\text{ls}(\mathbf{x}, \mathbf{y}) \rightarrow_P^* \exists c. \text{pt}(\mathbf{x}, c) * \text{emp} \wedge \mathbf{x} \neq \mathbf{y} \wedge c = \mathbf{y}$.

For convenience, we sometimes use a generalised $*$ -conjunction and, given spatial formulas $\sigma_1, \dots, \sigma_n$, write $*_{1 \leq i \leq n} \sigma_i$ for $\sigma_1 * \dots * \sigma_n$. Likewise, we write $\text{pred}(\mathbf{a}_1, \dots, \mathbf{a}_k) := \prod_{i \in [n]} \alpha_i$ for $\text{pred}(\mathbf{a}_1, \dots, \mathbf{a}_k) := \alpha_1 \mid \dots \mid \alpha_n$. Moreover, $\exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma$ abbreviates $\exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma \wedge \top$; we may also write *e.g.* $\text{pt}(\mathbf{x}, (-, \mathbf{y}))$ as a shorthand for $\exists \mathbf{a}. \text{pt}(\mathbf{x}, (\mathbf{a}, \mathbf{y}))$, where $\mathbf{a} \in \text{EVars}$ is a fresh existential variable. Furthermore, $\exists_{i \in [n]} \mathbf{a}_i. \sigma \wedge \varphi$ abbreviates $\exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma \wedge \varphi$. If PNames is clear from the context we will omit stating it explicitly.

Interpretations. As stated above, interpretations are given in terms of selector graphs. This diversion from the more commonly found “heap-and-stack model” found in the literature is for technical convenience only, and it is easy to translate between the two interpretation domains.

An *SL-interpretation*, or simply *interpretation*, \mathcal{I} is a selector graph $\mathcal{I} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \ell^{\mathcal{I}}, s^{\mathcal{I}})$ such that $\ell^{\mathcal{I}} : \text{FVars} \rightarrow_{\text{fin}} V$. For $\mathbf{x}_1, \dots, \mathbf{x}_n \in \text{FVars}$ and for $v_1, \dots, v_n \in V^{\mathcal{I}}$, we denote by $\mathcal{I}[\mathbf{x}_1 \mapsto v_1; \dots; \mathbf{x}_n \mapsto v_n]$ the SL-interpretation $\hat{\mathcal{I}} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \hat{\ell}^{\mathcal{I}}, s^{\mathcal{I}})$, where $\hat{\ell}^{\mathcal{I}} \stackrel{\text{def}}{=} \ell^{\mathcal{I}}[\mathbf{x}_1 \mapsto v_1; \dots; \mathbf{x}_n \mapsto v_n]$, and we call such an interpretation an *extension* of \mathcal{I} .

In our interpretations, nodes with arity greater than zero are the equivalent to allocated heap cells in the “heap-and-stack model”, while record fields are represented by the different selectors. We define the $*$ -decomposition of \mathcal{I} as follows: $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$ iff $\mathcal{I}_1 = (V^{\mathcal{I}_1}, E^{\mathcal{I}_1}, \ell^{\mathcal{I}_1}, s^{\mathcal{I}_1})$ and $\mathcal{I}_2 = (V^{\mathcal{I}_2}, E^{\mathcal{I}_2}, \ell^{\mathcal{I}_2}, s^{\mathcal{I}_2})$ such that $V^{\mathcal{I}} = V^{\mathcal{I}_1} \cup V^{\mathcal{I}_2}$; $\ell^{\mathcal{I}}, \ell^{\mathcal{I}_1}$ and $\ell^{\mathcal{I}_2}$ coincide; for $i \in \{1, 2\}$, either $s^{\mathcal{I}_i}(v) = 0$ or $s^{\mathcal{I}_i}(v) = s^{\mathcal{I}}(v)$, and $s^{\mathcal{I}}(v) = s^{\mathcal{I}_1}(v) + s^{\mathcal{I}_2}(v)$ for all $v \in V^{\mathcal{I}}$; for $i \in \{1, 2\}$, if $s^{\mathcal{I}_i}(v) > 0$ then $E^{\mathcal{I}_i}(v, j) = E^{\mathcal{I}}(v, j)$ for all $v \in V^{\mathcal{I}}$ and $j \in [s^{\mathcal{I}_i}(v)]$.

Semantics of SL-assertions. The semantics of flat reduced SL-assertions is defined by structural induction. Let $\mathcal{I} = (V^{\mathcal{I}}, E^{\mathcal{I}}, \ell^{\mathcal{I}}, s^{\mathcal{I}})$ be an SL-interpretation and φ a pure formula only over variable names from FVars , the *satisfaction relation* $\mathcal{I} \models \varphi$ is defined such that $\mathcal{I} \models \top$ holds always, $\mathcal{I} \models \perp$ never holds, $\mathcal{I} \models \mathbf{x} = \mathbf{y}$ iff $\ell^{\mathcal{I}}(\mathbf{x}) = \ell^{\mathcal{I}}(\mathbf{y})$, $\mathcal{I} \models \neg \varphi$ iff $\mathcal{I} \not\models \varphi$, and $\mathcal{I} \models \varphi_1 \wedge \varphi_2$ iff $\mathcal{I} \models \varphi_1$ and $\mathcal{I} \models \varphi_2$.

For reduced flat spatial formulas σ such that $\text{vars}(\sigma) \subseteq \text{FVars}$, we define $\mathcal{I} \models \text{emp}$ iff $s^{\mathcal{I}}(v) = 0$ for all $v \in V^{\mathcal{I}}$ and $\mathcal{I} \models \text{tt}$ holds always. Moreover, $\mathcal{I} \models \sigma_1 * \sigma_2$ iff $\mathcal{I} = \mathcal{I}_1 * \mathcal{I}_2$ such that $\mathcal{I}_1 \models \sigma_1$ and $\mathcal{I}_2 \models \sigma_2$, and finally, $\mathcal{I} \models \text{pt}(\mathbf{x}, (\mathbf{x}_1, \dots, \mathbf{x}_m))$ iff

- $v = \ell^{\mathcal{I}}(\mathbf{x})$, $s^{\mathcal{I}}(v) = m$, $s^{\mathcal{I}}(v') = 0$ for all $v' \in V^{\mathcal{I}} \setminus v$; and
- $E^{\mathcal{I}}(v, i) = \ell^{\mathcal{I}}(\mathbf{x}_i)$ for all $i \in [m]$.

For a flat reduced SL-assertion $\alpha = \exists \mathbf{a}_1, \dots, \mathbf{a}_n. \sigma \wedge \varphi$, we define $\mathcal{I} \models \alpha$ iff there is an extension $\hat{\mathcal{I}} = \mathcal{I}[\mathbf{x}_1 \mapsto v_1, \dots, \mathbf{x}_n \mapsto v_n]$ for fresh variables $\mathbf{x}_1, \dots, \mathbf{x}_n \in \text{FVars}$

such that $\hat{\mathcal{I}} \models \sigma[x_1/a_1, \dots, x_n/a_n]$ and $\hat{\mathcal{I}} \models \varphi[x_1/a_1, \dots, x_n/a_n]$. We call \mathcal{I} a *model* of α if $\mathcal{I} \models \alpha$. Given α and a set of inductive predicates P , we write $\mathcal{I} \models_P \alpha$ if $\mathcal{I} \models \alpha'$ for some α' obtained from a complete unraveling $\alpha \rightarrow_P^* \alpha'$. Given α and α' over a set of inductive predicates P , we write $\alpha \models_P \alpha'$ iff whenever $\mathcal{I} \models_P \alpha$ then $\mathcal{I} \models_P \alpha'$. Given α over a set of inductive predicates P , *satisfiability* is to decide whether there is an interpretation \mathcal{I} such that $\mathcal{I} \models_P \alpha$. Given \mathcal{I} , *model checking* is to decide $\mathcal{I} \models_P \alpha$. The main decision problem of interest in this paper is entailment, defined as follows.

ENTAILMENT

INPUT: SL assertions α, α' with respect to a set P of inductive predicates.

QUESTION: Does $\alpha \models_P \alpha'$?

3 Entailment in the Presence of Inductive Predicates

In this section, we show that entailment with general inductive predicates is undecidable when no restrictions are imposed. Subsequently, we give an EXPTIME lower bound for the fragment introduced by Iosif *et al.* [17].

General undecidability. We show undecidability via a reduction from the undecidable Post Correspondence Problem [19].

POST CORRESPONDENCE PROBLEM (PCP)

INPUT: A finite set of tiles $(v_1, w_1), \dots, (v_k, w_k)$, $v_i, w_i \in \{0, 1\}^*$.

QUESTION: Does there exist a sequence $s_1 s_2 \dots s_\ell \in \{1, \dots, k\}^\ell$, $\ell > 0$ such that $v_{s_1} v_{s_2} \dots v_{s_\ell} = w_{s_1} w_{s_2} \dots w_{s_\ell}$?

For any $u \in \{0, 1\}^*$, denote by $|u|$ the length of each tile, and by $u(i)$ the i -th symbol of u , for $1 \leq i \leq |u|$. For example, if $u = 01101$, we have $|u| = 5$ and $u(3) = 1$. Let $(v_1, w_1), \dots, (v_k, w_k)$ be an instance of PCP. The set of predicates P in Figure 1 establishes a reduction such that this instance has a solution iff $\text{PCP}(x, y) \wedge x_0 \neq x_1 \not\models_P \overline{\text{PCP}}(x, y)$. The intuition behind these definitions is as follows. For $x, y \in \text{FVars}$, $\text{PCP}(x, y)$ generates the set of all possible tilings for a given instance: in any model the v_i -tilings begin at x and the w_i -tilings at y . The fixed stack variables $x_0, x_1 \in \text{FVars}$ are used to represent the corresponding symbols 0 and 1. Likewise, $\overline{\text{PCP}}(x, y)$ generates all tilings which are incorrect. This is the case if the model is empty, there are two symbols at the same position which are different (*cf.* $\text{NEqualPair}(x, y)$), or the length of the strings encoded in a model is different (*cf.* $\text{NEqualLen}(x, y)$).

Theorem 3. *Entailment in Separation Logic with general inductive predicates is undecidable.*

Remark 4. An anonymous referee remarked that our reduction can also be applied for showing that satisfiability in the presence of conjunction over spatial formulas and general inductive predicates is undecidable. The models of the subsequent predicate encode all pairs of equal strings:

$$\text{EqPairs}(a, b) = \text{emp} \mid \parallel_{i \in \{0, 1\}} \exists p, r. \text{pt}(x, (x_i, p)) * \text{pt}(y, (x_i, r)) * \text{EqPairs}(p, r).$$

$$\begin{aligned}
 \text{PCP}(\mathbf{a}, \mathbf{b}) &:= \text{emp} \mid \text{Tile}_1(\mathbf{a}, \mathbf{b}) \mid \cdots \mid \text{Tile}_k(\mathbf{a}, \mathbf{b}) \\
 \text{Tile}_i(\mathbf{a}, \mathbf{b}), i \in [k] &:= \exists \mathbf{p}_0, \dots, \mathbf{p}_{|v_i|}, \mathbf{r}_0, \dots, \mathbf{r}_{|w_i|} \cdot \underset{0 \leq j < |v_i|}{*} \text{pt}(\mathbf{p}_j, (\mathbf{x}_{v_i(j+1)}, \mathbf{p}_{j+1})) * \\
 &\quad * \underset{0 \leq j < |w_i|}{*} \text{pt}(\mathbf{r}_j, (\mathbf{x}_{w_i(j+1)}, \mathbf{r}_{j+1})) * \text{PCP}(\mathbf{p}_{|v_i|}, \mathbf{r}_{|w_i|}) \wedge \mathbf{a} = \mathbf{p}_0 \wedge \mathbf{b} = \mathbf{r}_0 \\
 \text{NEqualPair}(\mathbf{a}, \mathbf{b}) &:= \exists \mathbf{p}, \mathbf{r}. \text{pt}(\mathbf{a}, (_, \mathbf{p})) * \text{pt}(\mathbf{b}, (_, \mathbf{r})) * \text{NEqualPair}(\mathbf{p}, \mathbf{r}) \\
 &\quad \mid \exists \mathbf{c}, \mathbf{d}. \text{pt}(\mathbf{a}, (\mathbf{c}, _)) * \text{pt}(\mathbf{b}, (\mathbf{d}, _)) * \text{tt} \wedge \mathbf{c} \neq \mathbf{d} \\
 \text{Tail}(\mathbf{a}) &:= \text{emp} \mid \exists \mathbf{b}. \text{pt}(\mathbf{a}, (_, \mathbf{b})) * \text{Tail}(\mathbf{b}) \\
 \text{NEqualLen}(\mathbf{a}, \mathbf{b}) &:= \exists \mathbf{x}, \mathbf{p}, \mathbf{r}. \text{pt}(\mathbf{a}, (\mathbf{x}, \mathbf{p})) * \text{pt}(\mathbf{b}, (\mathbf{x}, \mathbf{r})) * \text{NEqualLen}(\mathbf{p}, \mathbf{r}) \\
 &\quad \mid \exists \mathbf{p}. \text{pt}(\mathbf{a}, (_, \mathbf{p})) * \text{Tail}(\mathbf{p}) \mid \exists \mathbf{r}. \text{pt}(\mathbf{b}, (_, \mathbf{r})) * \text{Tail}(\mathbf{r}) \\
 \overline{\text{PCP}}(\mathbf{a}, \mathbf{b}) &:= \text{emp} \mid \text{NEqualPair}(\mathbf{a}, \mathbf{b}) \mid \text{NEqualLen}(\mathbf{a}, \mathbf{b})
 \end{aligned}$$

Fig. 1. The set P of inductive predicates for the reduction from PCP

It is then easy to conjoin $\text{EqPairs}(\mathbf{x}, \mathbf{y})$ with $\text{PCP}(\mathbf{x}, \mathbf{y})$ such that a model exists if, and only if, the given PCP instance has a solution.

Inductive Predicates with Bounded Tree Width. Iosif *et al.* define in [17] a fragment of Separation Logic by syntactically restricting the definitions of inductive predicates such that all models have bounded tree width. In particular, their fragment requires that there is *exactly one* points-to predicate in any definition, which is clearly not the case in the reduction from PCP. Moreover, briefly speaking, further restrictions require that in each predicate definition, if a predicate name occurs in the body of a predicate definition then a points-to predicate occurs in the definition as well, that every existentially quantified variable is eventually allocated, and some further subtle technical conditions. We omit further details for space reason and show that entailment is EXPTIME-hard in this fragment. The reader can easily verify that our reduction fulfils the requirements defined in [17].

Our reduction is from the language inclusion problem for non-deterministic top-down binary finite tree automata. A *prefix closed set of strings* over $\{0, 1\}$ is a set of strings S such that for each $s \in S$ and any prefix s_p of s , s_p is also in S . A *binary ordered tree* t over a finite *alphabet* Σ is a tuple (N, Σ, ℓ) , where N is a prefix closed set of strings over $\{0, 1\}$ denoting the *nodes of the tree*, where for each $s \in N$, $s \cdot 1 \in N$, if and only if $s \cdot 0 \in N$, and $\ell : N \rightarrow \Sigma$ is a function assigning *labels* to nodes of the tree. The root of a tree is the empty string ϵ , and for any two nodes s and $s \cdot i$, for $i \in \{0, 1\}$, $s \cdot i$ is a child node of s . We say that a node $s \in N$ is a *leaf node* if it has no child nodes, and a node is *internal* otherwise.

Recall that a *finite non-deterministic top-down tree automaton (NFTA)* A is a tuple (Σ, Q, δ, I) , where Σ is a finite *alphabet*, Q is a finite set of *states* with a designated state q_{leaf} , $I \subseteq Q$ is the set of *initial or accepting states*, and $\delta : Q \times \Sigma \rightarrow 2^{Q \times Q}$ is the *transition function* such that for all $\sigma \in \Sigma$, $\delta(q_{\text{leaf}}, \sigma) = \emptyset$. A *run of A* on a tree $t = (N, \Sigma, \ell)$ is a function $\rho : N \rightarrow Q$

assigning states to the nodes of t such that for each internal node $s \in N$, $(\rho(s \cdot 0), \rho(s \cdot 1)) \in \delta(\rho(s), \ell(s))$ and for each leaf node $s \in N$, $\rho(s) = q_{leaf}$. A run is *accepting* if $\rho(\epsilon) \in I$, and the *language* $\mathcal{L}(A)$ *accepted by a NFTA* A is the set of trees t for which there is an accepting run of A on t .

NFTA LANGUAGE INCLUSION PROBLEM

INPUT: Two NFTA A_1 and A_2 .

QUESTION: Does $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ hold?

A classical result states that the language inclusion problem for non-deterministic tree automata is complete for EXPTIME [21]. Let $A = (\Sigma, Q, \delta, I)$ be an NFTA. We define the subsequent set P of inductive predicates, where $\text{Tree}_{q,\sigma}(\mathbf{a})$ is defined for every $\sigma \in \Sigma$ and $q \in Q$ for which $\delta(q, \sigma)$ is non-empty:

$$\text{Tree}_{q,\sigma}(\mathbf{a}) := \parallel_{\substack{(q_1, q_2) \in \delta(q, \sigma) \\ \sigma_1, \sigma_2 \in \Sigma \\ \delta(q_1, \sigma_1) \neq \emptyset \vee q_1 = q_{leaf} \\ \delta(q_2, \sigma_2) \neq \emptyset \vee q_2 = q_{leaf}}} \exists l, r. \text{pt}(\mathbf{a}, (\mathbf{s}_\sigma, l, r)) * \text{Tree}_{q_1, \sigma_1}(l) * \text{Tree}_{q_2, \sigma_2}(r)$$

$$\text{Tree}_{q_{leaf}, \sigma}(\mathbf{a}) := \text{pt}(\mathbf{a}, \mathbf{s}_\sigma)$$

$$\text{Trees}_A(\mathbf{a}) := \parallel_{\substack{q \in I \\ \sigma \in \Sigma}} \exists \mathbf{b}. \text{pt}(\mathbf{a}, \mathbf{b}) * \text{Tree}_{q, \sigma}(\mathbf{b})$$

In any model, the predicate $\text{Trees}_A(\mathbf{x})$ encodes all trees in $\mathcal{L}(A)$: apart from the node labeled with \mathbf{x} , each allocated vertex represents a node of the tree, the first selector represents the label of the node, and the subsequent selectors represent respectively the left and right descendants, if the node is an internal node. The additional pointer at \mathbf{x} is for technical reasons in order to comply with the restrictions defined in [17]. It is now easily checked that given two NFTA A_1 and A_2 over some alphabet $\Sigma = \{\sigma_1, \dots, \sigma_n\}$,

$$\text{Trees}_{A_1}(\mathbf{x}) \wedge \bigwedge_{1 \leq i \neq j \leq n} \mathbf{s}_{\sigma_i} \neq \mathbf{s}_{\sigma_j} \models_P \text{Trees}_{A_2}(\mathbf{x})$$

is a valid entailment if, and only if, $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.

Theorem 5. *Deciding entailment in Separation Logic with inductive predicates with bounded tree width as defined in [17] is EXPTIME-hard.*

It is worth emphasizing that hardness already holds if the arity of the pointer predicates is fixed to three. Also note that the EXPTIME-hardness proof for satisfiability provided in [8] does not trivially establish that entailment in the fragment defined in [17] is EXPTIME-hard since the definitions given in [8] are not in the fragment of [17].

Also, note that in [21] it is shown that for two NFTA that accept finite languages, the language inclusion problem is PSPACE-complete, and therefore the proof of Theorem 5 can be adapted to show PSPACE-hardness of entailment with inductive predicates not involving cyclic definitions.

4 Entailment for Fixed Fragments

The primary goal of this section is to first study the complexity of entailment in the base language defined in Section 2, and subsequently in the base language equipped with a fixed list predicate as defined in (1), which is a fragment commonly found in the program verifiers discussed in the introduction.

We first show that entailment in the base language is Π_2^P -complete. Moreover, we additionally outline that the upper bound even holds in the presence of the aforementioned list predicate. This result complements the previous section in that it indicates that for specific and fixed natural decidable fragments involving cyclic definitions of small arity the EXPTIME lower bound can be avoided.

In the second part we analyse the lower bound from the first part and consider natural syntactic fragments defined in terms of structural properties of graphs representing SL-assertions. It has been shown that such restrictions can lead to polynomial-time decision procedures for entailment when dropping existentially quantified variables [14] and also decidability results for more expressive extensions of our base assertion language [7]. We show that basically there is no hope of achieving polynomial-time decision procedures in the presence of list predicates and existentially quantified variables, even when strong syntactic restrictions on the assertions are imposed.

Entailment in the General Case. We begin with the lower bound and show hardness for the base language, *i.e.* the language having only points-to predicates, via a reduction from a generalisation of *graph three-colorability* that has been defined in [1]. Recall that given an undirected graph $G = (V, E)$, graph three-colorability is to decide whether there is a *three-coloring* $f : V \rightarrow \{1, 2, 3\}$ such that $f(v) \neq f(w)$ for all $\{v, w\} \in E$. A *leaf coloring* of G is a function $f : V_l \rightarrow \{1, 2, 3\}$, where V_l is the set of vertices of G with degree one, *i.e.*, those nodes that have exactly one incident edge. The generalisation of graph three-colorability is given as follows.

2-ROUND 3-COLORABILITY

INPUT: Undirected graph $G = (V, E)$.

QUESTION: For every fixed leaf coloring f of G , can f be extended to a three-coloring of G ?

It has been shown in [1] that 2-ROUND 3-COLORABILITY is Π_2^P -complete. We now show hardness of entailment for SL-formulas via a reduction from 2-ROUND 3-COLORABILITY. To this end, we construct flat reduced SL-assertions α, α' such that the graph $G = (V, E)$ is a valid instance of 2-ROUND 3-COLORABILITY iff $\alpha \models \alpha'$. We partition V into disjoint sets $V' = \{v_1, \dots, v_n\}$ of nodes with degree greater than one and $V'' = \{v_{n+1}, \dots, v_m\}$ of nodes with degree equal to one, and define α and α' such that

$$\begin{aligned} \alpha &\stackrel{\text{def}}{=} * \text{pt}(x_{i,j}, y_i) * * \text{pt}(x_j, z_j) \wedge \bigwedge_{n < i \leq m} \bigvee_{j \in [3]} z_i = y_j \wedge \bigwedge_{1 \leq i \neq j \leq 3} y_i \neq y_j \\ \alpha' &\stackrel{\text{def}}{=} \exists_{i \in [n]} \mathbf{a}_i. \exists_{j \in [m]} \mathbf{b}_j. * \text{pt}(\mathbf{a}_i, \mathbf{b}_i) * * \text{pt}(x_j, \mathbf{b}_j) * \text{tt} \wedge \bigwedge_{(v_i, v_j) \in E} \mathbf{b}_i \neq \mathbf{b}_j. \end{aligned}$$

Intuitively speaking, the pointers $\text{pt}(x_j, z_j)$ in α can choose in any model \mathcal{I} of α a coloring of the leaves of G , represented by the variables y_i , $i \in [3]$. The $x_{i,j}$ are allocated in order to enable α' to choose a coloring of the nodes which are not leaves. Consequently in a model \mathcal{I} of α , an extension of \mathcal{I} determining the existentially quantified variables b_i of α' determines a three-coloring of G . Conversely, if $\alpha \not\models \alpha'$ then the counter-model \mathcal{I} encodes a coloring of the leaves which cannot be extended to a three-coloring.

It is not difficult to see that Π_2^P -hardness of entailment can already be established for SL-assertions without disjunction, by only requiring $y_i \neq y_j$ for all $1 \leq i \neq j \leq 3$ in the pure part of α : if $\alpha \models \alpha'$ then, in particular, any leaf coloring can be extended to a three-coloring since a subset of the models of α will encode all possible leaf colorings. The converse direction then follows as above. In addition, by introducing additional existentially quantified slack variables, the hardness proof can also be tightened in a way such that no “tt” is required in spatial formulas, *i.e.*, hardness holds in non-intuitionistic fragments. Finally, this reduction can be reused in order to show NP-hardness of the model checking problem via a reduction from 3-COLORABILITY.

Since the size of all relevant models is *a priori* fixed by the size of the formulas under consideration, a Π_2^P upper bound follows trivially.

Theorem 6. *Entailment for flat reduced SL-assertions is Π_2^P -complete.*

For the remainder of this section, we turn towards entailment in the base language equipped with an additional fixed list predicate as defined in (1) and restrict our attention to pointer predicates of arity one, and consequently to interpretations which are functional graphs.

First, we can also establish a Π_2^P upper bound for entailment in this fragment by showing a small-model property. The main idea is that for a sufficiently large \mathcal{I} such that $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \alpha'$, we can find an instantiation of an $\text{ls}(x, y)$ predicate in \mathcal{I} such that the path between x and y is long enough that we can obtain a new \mathcal{I}' by removing a vertex occurring on this path while ensuring that the newly obtained \mathcal{I}' is still a counter-model witnessing $\alpha \not\models \alpha'$.

Lemma 7. *Let α, α' be SL-assertions, let $n = |\text{vars}(\alpha)| + |\text{vars}(\alpha')|$ and suppose that $\alpha \not\models \alpha'$. Then there exists an \mathcal{I} witnessing $\alpha \not\models \alpha'$ with $|\mathcal{V}^{\mathcal{I}}| \in O(n^2)$.*

This lemma immediately yields a Π_2^P upper bound: in order to show $\alpha \not\models \alpha'$, we can guess a small model \mathcal{I} of α and then verify with an NP oracle that $\mathcal{I} \not\models \alpha'$.

Theorem 8. *Entailment for flat reduced SL-assertions with a fixed list predicate is Π_2^P -complete.*

Entailment under Structural Restrictions. The goal of this section is to argue that entailment in essentially any useful fragment is intractable in the presence of existential quantification and list predicates, even under severe syntactic restrictions. In the following, we will only consider positive SL-assertions, since otherwise non-entailment is trivially coNP-hard.

In order to identify syntactic fragments with potentially polynomial-time entailment problems, we look at properties of graphs representing SL-formulas. Let $G = (V, E)$ be a directed graph and $v \in V$ a vertex of G . Then, define functions $\text{pred}(v) \stackrel{\text{def}}{=} \{v' \in V : (v', v) \in E\}$ and $\text{succ}(v) \stackrel{\text{def}}{=} \{v' \in V : (v, v') \in E\}$. A node $v \in V$ is a *source node* if $\text{pred}(v) = \emptyset$, and v is a *sink node* if $\text{succ}(v) = \emptyset$. Let $\alpha = \sigma \wedge \varphi$ be an SL-assertion, and $x \in \text{vars}(\alpha)$ be a variable of α . Then define the set $\text{Eq}(\varphi, x) = \{y \in \text{vars}(\alpha) : \text{for all } \mathcal{I}, \text{ if } \mathcal{I} \models \varphi \text{ then } \mathcal{I} \models x = y\}$. Next, we define the *graph* $G(\alpha)$ *corresponding to* α as $G(\alpha) = (V_\alpha, E_\alpha, \ell_\alpha)$, where the set of vertices is defined as $V_\alpha \stackrel{\text{def}}{=} \{\text{Eq}(\varphi, x) : x \in \text{vars}(\alpha)\}$, and the set of edges as $E_\alpha \stackrel{\text{def}}{=} \{(\text{Eq}(\varphi, x), \text{Eq}(\varphi, y)) : \text{pt}(x, y) \text{ or } \text{ls}(x, y) \text{ occurs in } \sigma\}$. Finally, ℓ_α is such that $\ell_\alpha(x) = \text{Eq}(\varphi, x)$ for all $x \in \text{vars}(\alpha)$. A node $v \in V_\alpha$ is *fixed* if there is $x \in \text{FVars}$ such that $\ell_\alpha(x) = v$.

Inspecting the Π_2^P -hardness proof above, we see that one fundamental source of complexity comes from having pointers $\text{pt}(a, b)$ with $a, b \in \text{EVars}$, *i.e.*, the graph corresponding to α' above has source and sink nodes which are not fixed. On the one hand, when not allowing for list predicates, if all source nodes of an SL-assertion were to be fixed, entailment between formulas in such a suitably defined fragment would trivially become polynomial-time decidable. The main reason for this is that in any model \mathcal{I} of $\exists a. \text{pt}(x, a)$, a would have to be instantiated with the *unique* successor of the vertex labeled with x . However, such a fragment would only allow for reasoning about models whose size is *a priori* fixed by the size of the antecedent, which limits its usefulness. On the other hand, when allowing for list predicates, the proof of NP-hardness of abduction (given in [15]) can be easily adapted to show that entailment becomes intractable even if source nodes are required to be fixed.

This leaves us with an interesting case, which we introduce by considering an example of an instance of entailment:

$$\text{ls}(x, y) \wedge x \neq y \models \exists a. \text{pt}(x, a) * \text{ls}(a, y).$$

The validity of this entailment rests on the fact that x has a successor in any model containing a non-empty list from x to y . In this example, the consequent is a formula of the fragment of the assertion language which we are going to consider in the remainder of this section: an SL-assertion α is in the *fixed endpoints fragment* if all source and sink nodes of the graph $G(\alpha)$ corresponding to α are fixed. We show CONP-hardness of entailment in this fragment via a reduction from an NP-complete variant of the Hamiltonian path problem.

FIXED VERTEX HAMILTONIAN PATH (FVHP)

INPUT: A directed graph $G = (V, E)$ and $v \in V$.

QUESTION: Does there exist a Hamiltonian path in G ending in v ?

Given a graph $G = (V, E)$, a vertex $v \in V$ and an instance of FVHP such that $V = \{v_1, \dots, v_{N+1}\}$ and $v = v_{N+1}$, we show how to compute in polynomial time SL-formulas α, α' in the fixed endpoints fragment such that $\alpha \not\models \alpha'$ if and only if G is a valid instance of FVHP. For $i \in [N + 1]$ and $j \in [N]$, for the spatial parts of α and α' we define:

$$\begin{aligned} \text{node}_j &\stackrel{\text{def}}{=} \text{ls}(e_j^s, a_j^0) * \bigstar_{k \in [0, N-1]} \text{ls}(a_j^k, b_j^{k+1}) * \bigstar_{k \in [N-1]} \text{ls}(b_j^k, a_j^k) * \text{ls}(b_j^N, e_j^f) \\ \text{order}_i^0 &\stackrel{\text{def}}{=} \text{pt}(s_i^0, f_i^0) \\ \text{order}_i^j &\stackrel{\text{def}}{=} \text{ls}(s_i^j, d_i^j) * \text{ls}(d_i^j, f_i^j) \\ \sigma &\stackrel{\text{def}}{=} \bigstar_{j \in [N]} \text{node}_j * \bigstar_{\substack{i \in [N+1] \\ j \in [0, N]}} \text{order}_i^j \end{aligned}$$

A graphical illustration of the formulas node_j , order_i^j and order_i^0 is given in Figure 2, where list predicates are represented as dashed arrows and pointer predicates as full arrows. Consider the submodels of each of the formulas above. The intuition behind these formulas, in conjunction with the inequalities introduced below, is that there will be a model comprising a long concatenation, loosely speaking, of such submodels, if and only if there is a hamiltonian path in the given graph. The inequalities introduced below, will additionally ensure that such a long concatenation can happen only in the models of α and not of α' , and thus entailment will not hold in such a case. The variables a_j^k and b_j^k are essentially used in a way to count how long the concatenation is.

We now turn towards the pure parts of α and α' . For notational convenience, given $x \in \text{vars}(\alpha)$ and $S \subseteq \text{vars}(\alpha)$, subsequently $x \approx S$ abbreviates $\bigwedge_{y \in \text{vars}(\alpha) \setminus (S \cup \{x\})} x \neq y$. In other words, in any model \mathcal{I} of $x \approx S$, if $\ell^{\mathcal{I}}(x) = \ell^{\mathcal{I}}(y)$ for some $y \in \text{vars}(\alpha)$ then $y \in S$ or $x \equiv y$. We define Dvars to be the set $\{d_k^\ell : k \in [N + 1], \ell \in [0, N]\}$, and we define φ to be the conjunction of the subsequent pure formulas:

$$\begin{aligned} s_i^0 &\approx \emptyset \wedge f_i^N \approx \text{Dvars} \wedge d_i^0 = f_i^0, & i \in [N + 1] \\ s_i^j &\approx \bigcup_{\substack{v_p \in \text{pred}(v_i), \\ N-j < k \leq N-1}} \{a_p^k, b_p^k, b_p^N, e_p^f\} \cup \text{Dvars}, & i \in [N + 1], j \in [N] \\ f_i^j &\approx \{e_i^s, a_i^0, b_i^{N-j}\} \cup \bigcup_{k \in [N-j-1]} \{a_i^k, b_i^k\} \cup \text{Dvars}, & i \in [N], j \in [0, N - 1] \\ f_{N+1}^j &\approx \text{Dvars}, & j \in [0, N - 1] \\ e_i^f &\neq e_j^f, & 1 \leq i \neq j \leq N \\ d_i^j &\neq b_i^{N-j}, & i \in [N], j \in [0, N - 1] \\ \bigwedge_{k \in [N] \setminus \{i\}} d_i^j &\neq b_k^{N-j+1} & i \in [N + 1], j \in [N] \end{aligned}$$

Finally, we define α and α' using the set of variables \mathbf{V} shown below:

$$\begin{aligned} \mathbf{V} &\stackrel{\text{def}}{=} \{a_i^0, a_i^j, b_i^j, b_i^N : i \in [N], j \in [N - 1]\} \cup \{d_i^j : i \in [N + 1], j \in [0, N]\} \\ \alpha &\stackrel{\text{def}}{=} \exists_{x \in \mathbf{V}} x. \sigma \wedge \varphi \quad \text{and} \quad \alpha' \stackrel{\text{def}}{=} \exists_{x \in \mathbf{V}} x. \sigma \wedge \varphi \wedge d_{N+1}^N \neq f_{N+1}^N \end{aligned}$$

Note that φ includes $f_i^N \approx \text{Dvars}$ for all $i \in [N + 1]$. Given the additional disequality in α' , we now have $f_{N+1}^N \approx \text{Dvars} \setminus \{d_{N+1}^N\}$ in the pure part of α' .

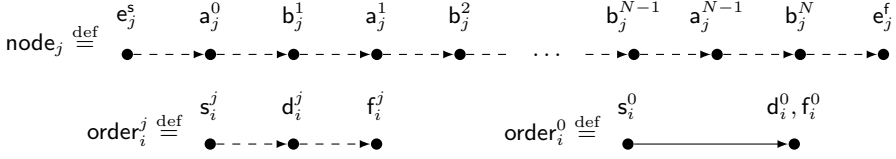


Fig. 2. Graphical representation of the formulas $node_i^j$ and $order_i^j$

Also note that in order to simplify the presentation, we have defined α and α' such that we use the same existentially quantified variables both in α and α' . It is important to note that given a model \mathcal{I} of both α and α' , the extension $\hat{\mathcal{I}}_1$ of \mathcal{I} that witnesses the satisfaction of the formula α and the extension $\hat{\mathcal{I}}_2$ that witnesses the satisfaction of the formula α' do not in general agree on the mapping of those existentially quantified variables. The existentially quantified variables in α could also be seen as fixed variables with names different from the existentially quantified variables of α' . As these variables act in the same way in both formulas, in order to avoid writing the above definitions twice and to simplify our proof, we have decided to treat them as existentially quantified and define them such that they have the same name in both formulas.

Lemma 9. $G = (V, E)$ and $v \in V$ is a valid instance of FVHP iff $\alpha \not\models \alpha'$.

Proof (sketch). First, a crucial observation is that for any \mathcal{I} such that $\mathcal{I} \models \alpha$ and $\mathcal{I} \not\models \alpha'$, in any extension $\hat{\mathcal{I}}$ witnessing $\mathcal{I} \models \alpha$, we have that the instantiation of d_{N+1}^N coincides with f_{N+1}^N . Suppose this was not the case, then $\hat{\mathcal{I}}$ would also witness $\mathcal{I} \models \alpha'$, contradicting our assumption. In this case we say that d_{N+1}^N is forced on f_{N+1}^N . We can show that forcing d_{N+1}^N on f_{N+1}^N is only possible if b_p^1 is forced on s_{N+1}^N for some unique predecessor $v_p \in pred(v_N)$ of v_N . In order to force b_p^1 on s_{N+1}^N , it can then be shown that d_p^{N-1} and therefore f_p^{N-1} is forced on a_p^0 . In summary, we can establish the following chain of inductive reasoning:

- if $d_{i_0}^N$ is forced on $f_{i_0}^N$ then $b_{i_1}^1$ is forced on $s_{i_0}^N$ for some $v_{i_1} \in pred(v_{i_0})$
- if $b_{i_1}^1$ is forced on $s_{i_0}^N$ then $d_{i_1}^{N-1}$ is forced on $f_{i_1}^{N-1}$
- \vdots
- if $d_{i_{N-1}}^N$ is forced on $f_{i_{N-1}}^N$ then $b_{i_N}^N$ is forced on $s_{i_0}^1$ for some $v_{i_N} \in pred(v_{i_{N-1}})$
- if $b_{i_N}^N$ is forced on $s_{i_0}^1$ then $d_{i_N}^0$ is forced on $f_{i_N}^0$

Now considering the variable names b_i^j in the implication chain, we obtain a sequence $b_{i_1}^1, b_{i_2}^2, \dots, b_{i_{N-1}}^{N-1}, b_{i_N}^N$ such that for all $1 \leq j < N$, v_{i_j} is a successor of $v_{i_{j+1}}$ in G . Consequently, the sequence of nodes $\pi = v_{i_N} \cdots v_{i_2} v_{i_1} v_{N+1}$ is a path of length $N + 1$ in G ending in v_{N+1} . Using the definition of φ , it follows that any b_i^j can only be “used” once, hence π does not contain duplicate nodes and thus is a Hamiltonian path ending in v_{N+1} . \square

It is readily checked that source and sink nodes in the graph corresponding to the definition of α and α' are fixed. Hence, we have established the following.

Theorem 10. Entailment in the fixed endpoints fragment is CONP-hard.

5 Conclusion

The results in this paper can be interpreted as follows: when considering fragments of Separation Logic which allow for existential quantification and unbounded data structures, having tractable, polynomial-time decision procedures will require severe syntactic restriction, since entailment is CONP-hard even in the strongly constrained fixed endpoints fragment. In the presence of general inductive predicates, although satisfiability is decidable [8], we have shown that entailment becomes undecidable. This result complements the decidability result obtained by Iosif *et al.* [17] and shows that the syntactic restrictions defined in [17] are not only natural but also crucial for decidability. However, we have shown that entailment in this fragment is EXPTIME-hard. On the more positive side, we have shown that entailment is “only” Π_2^P -complete in the presence of existential quantification, pointers and linked lists. Since this is a fragment that has been shown to be useful in program verifiers, this result may be seen as an argument in favour of supporting the development of decision procedures for domain-specific fragments of Separation Logic with a fixed set of predicates.

A number of problems remain open which we intend to investigate in the future. For instance, an open issue is whether a restriction to a one-selector fragment leads to decidable entailment. Also, although we have shown EXPTIME-hardness for the bounded-tree width fragment, we currently do not know whether this is a tight bound. This is also true for the fixed endpoints fragment and its CONP-hardness. Finally, of great interest would be decision procedures for entailment in these fragments, since most tools use incomplete heuristics.

Acknowledgments. We would like to thank the referees for their helpful comments. In particular, we wish to thank one referee who suggested the reduction from the inclusion problem for tree automata for the proof of Theorem 5.

References

1. Ajtai, M., Fagin, R., Stockmeyer, L.: The closure of monadic NP. *Journal of Computer and System Sciences* 60(3), 660–716 (2000)
2. Bengtson, J., Jensen, J.B., Birkedal, L.: Charge! In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 315–331. Springer, Heidelberg (2012)
3. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O’Hearn, P.W., Wies, T., Yang, H.: Shape analysis for composite data structures. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 178–192. Springer, Heidelberg (2007)
4. Berdine, J., Calcagno, C., O’Hearn, P.W.: A decidable fragment of separation logic. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 97–109. Springer, Heidelberg (2004)
5. Berdine, J., Cook, B., Ishtiaq, S.: SLAYER: Memory safety for systems-level code. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 178–183. Springer, Heidelberg (2011)
6. Birkedal, L., Torp-Smith, N., Reynolds, J.C.: Local reasoning about a copying garbage collector. In: *Principles of Programming Languages*, pp. 220–231. ACM, New York (2004)

7. Bouajjani, A., Drăgoi, C., Enea, C., Sighireanu, M.: Accurate invariant checking for programs manipulating lists and arrays with infinite data. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 167–182. Springer, Heidelberg (2012)
8. Brotherston, J., Fuhs, C., Gorogiannis, N., Navarro Pérez, J.: A decision procedure for satisfiability in separation logic with inductive predicates. Technical Report RN/13/15, University College London (2013)
9. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012)
10. Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. In: Logic in Computer Science, pp. 137–146. IEEE Computer Society (2010)
11. Calcagno, C., Distefano, D., O’Hearn, P.W., Yang, H.: Beyond reachability: Shape abstraction in the presence of pointer arithmetic. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 182–203. Springer, Heidelberg (2006)
12. Calcagno, C., Yang, H., O’Hearn, P.W.: Computability and complexity results for a spatial assertion language for data structures. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 108–119. Springer, Heidelberg (2001)
13. Chin, W.-N., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Science of Computer Programming* 77(9), 1006–1036 (2012)
14. Cook, B., Haase, C., Ouaknine, J., Parkinson, M., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 235–249. Springer, Heidelberg (2011)
15. Gorogiannis, N., Kanovich, M., O’Hearn, P.W.: The complexity of abduction for separated heap abstractions. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 25–42. Springer, Heidelberg (2011)
16. Habermehl, P., Holík, L., Rogalewicz, A., Šimáček, J., Vojnar, T.: Forest automata for verification of heap manipulation. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 424–440. Springer, Heidelberg (2011)
17. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 21–38. Springer, Heidelberg (2013)
18. Ishtiaq, S., O’Hearn, P.: BI as an assertion language for mutable data structures. In: Principles of Programming Languages, pp. 14–26. ACM (2001)
19. Post, E.L.: A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society* 52(4), 264–268 (1946)
20. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Logic in Computer Science. IEEE Computer Society (2002)
21. Seidl, H.: Deciding equivalence of finite tree automata. *SIAM Journal on Computing* 19(3), 424–437 (1990)
22. Yang, H.: Local Reasoning for Stateful Programs. PhD thesis, University of Illinois at Urbana-Champaign (Technical Report UIUCDCS-R-2001-2227) (2001)

A Coalgebraic Approach to Linear-Time Logics

Corina Cîrstea

University of Southampton
cc2@ecs.soton.ac.uk

Abstract. We extend recent work on defining linear-time behaviour for state-based systems with branching, and propose modal and fixpoint logics for specifying linear-time temporal properties of states in such systems. We model systems with branching as coalgebras whose type arises as the composition of a branching monad and a polynomial endofunctor on the category of sets, and employ a set of truth values induced canonically by the branching monad. This yields logics for reasoning about quantitative aspects of linear-time behaviour. Examples include reasoning about the probability of a linear-time behaviour being exhibited by a system with probabilistic branching, or about the minimal cost of a linear-time behaviour being exhibited by a system with weighted branching. In the case of non-deterministic branching, our logic supports reasoning about the *possibility* of exhibiting a given linear-time behaviour, and therefore resembles an existential version of the logic LTL.

1 Introduction

Linear-time temporal logics such as LTL interpreted over non-deterministic transition systems and its probabilistic interpretation over Markov chains (see e.g. [1]) have been used successfully as specification logics in model checking. These logics share the same notion of linear-time behaviour, and employ a set of truth values which depends on the type of branching: a two-valued logic is used for non-deterministic transition systems, whereas elements of the unit interval are the possible truth values in the case of Markov chains. Despite such commonalities, a general and uniform account of linear-time logics is still missing.

The present paper fills this gap by building on recent work on defining linear-time behaviour for states in coalgebras with branching [2]. We model systems as coalgebras whose type incorporates branching, and define modal and fixpoint logics that are *parametric* in both the branching type and the transition type. The branching type canonically induces a set of truth values, whereas the transition type canonically induces the notion of observable linear behaviour. In addition to non-deterministic and probabilistic branching, our approach also instantiates to weighted branching. Our approach can be summarised as follows:

- We model systems as coalgebras of an endofunctor obtained as the composition of a *branching monad* $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ with a *polynomial endofunctor* $F : \mathbf{Set} \rightarrow \mathbf{Set}$. The elements of the final F -coalgebra provide the observable linear-time behaviours, whereas the set $\mathbb{T}1$ (with 1 a one-element set) is taken as domain of truth values.

- Fundamental to our approach is a (partial) semiring structure on the set $\mathbf{T1}$, studied in [8,3,2]. On the one hand, its (partial) addition operation induces an order on $\mathbf{T1}$ which is used to generalise the notion of predicate typically employed in the semantics of modal logics, by considering predicates valued in $\mathbf{T1}$. This subsequently supports the interpretation of fixpoint formulas. On the other hand, the multiplication operation on $\mathbf{T1}$ is used to canonically associate a set of *predicate liftings* to a polynomial endofunctor on \mathbf{Set} .
- We employ two kinds of liftings of endofunctors on \mathbf{Set} to the category of generalised predicates: one is inspired by coalgebraic modal logic (see e.g. [9]) and is used to provide semantics to individual modalities of a linear-time logic, while another is used to abstract away branching.
- We define modal and fixpoint linear-time logics for coalgebras with branching, and provide an alternative relational semantics for these logics that is amenable to model checking. The relational semantics relies on a generalised notion of relation lifting studied in [2], and currently applies to fixpoint formulas with only one type of fixpoints (either least or greatest ones).

While our approach builds on [2], the study of generalised predicate liftings and the definition of linear-time modal and fixpoint logics are new. Our results apply to systems with probabilistic or weighted branching, and yield linear-time logics for reasoning about the probability or the minimal cost of exhibiting a given linear-time behaviour. Our relational semantics provides a *global approach to model-checking linear-time logics*, whereby the truth values of all sub-formulas of a given formula are computed simultaneously. In this approach, computing the truth values of desirable (undesirable) system properties formalised using least (respectively greatest) fixpoints is done through a sequence of approximations, and this computation can be stopped once a satisfactory threshold is reached.

The remainder of this paper is structured as follows. Section 2 introduces basic definitions (Section 2.1) and gives a summary of our previous work on linear-time behaviour (Sections 2.2 and 2.3). Section 3 defines generalised predicate liftings, which are used in Section 4 to define multi-valued, linear-time modal logics for coalgebras with branching. Fixpoint extensions of these logics are considered in Section 5, where an outline of a relational approach to model checking such logics is also given. Section 6 describes ongoing and future work.

2 Preliminaries

2.1 Monads and Semirings

In what follows, we use monads (\mathbb{T}, η, μ) on \mathbf{Set} (where $\eta : \text{Id} \Rightarrow \mathbb{T}$ and $\mu : \mathbb{T} \circ \mathbb{T} \Rightarrow \mathbb{T}$ are the *unit* and *multiplication* of \mathbb{T}) to capture branching in coalgebraic types. Moreover, we assume that these monads are *strong* and *commutative*. A *strong monad* is equipped with a *strength map* $\text{st}_{X,Y} : X \times \mathbb{T}Y \rightarrow \mathbb{T}(X \times Y)$, natural in X and Y and subject to coherence conditions w.r.t. η and μ . For such a monad, one can also define a *swapped strength map* $\text{st}'_{X,Y} : \mathbb{T}X \times Y \rightarrow \mathbb{T}(X \times Y)$ by:

$$\mathbb{T}X \times Y \xrightarrow{\text{tw}_{\mathbb{T}X,Y}} Y \times \mathbb{T}X \xrightarrow{\text{st}'_{Y,X}} \mathbb{T}(Y \times X) \xrightarrow{\mathbb{T}\text{tw}_{Y,X}} \mathbb{T}(X \times Y)$$

where $\text{tw}_{X,Y} : X \times Y \rightarrow Y \times X$ is the *twist map* taking $(x, y) \in X \times Y$ to (y, x) . *Commutative monads* are strong monads where the maps $\mu_{X,Y} \circ \text{Tst}'_{X,Y} \circ \text{st}_{\text{T}X,Y} : \text{T}X \times \text{T}Y \rightarrow \text{T}(X \times Y)$ and $\mu_{X,Y} \circ \text{Tst}_{X,Y} \circ \text{st}'_{X,\text{T}Y} : \text{T}X \times \text{T}Y \rightarrow \text{T}(X \times Y)$ coincide, yielding a *double strength map* $\text{dst}_{X,Y} : \text{T}X \times \text{T}Y \rightarrow \text{T}(X \times Y)$ for each choice of sets X, Y .

Example 1. As examples of monads, we consider:

1. the *powerset monad* $\mathcal{P} : \text{Set} \rightarrow \text{Set}$, given by $\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$, modelling non-deterministic computations, with unit given by singletons and multiplication given by unions. Its strength and double strength are given by

$$\text{st}_{X,Y}(x, V) = \{x\} \times V \qquad \text{dst}_{X,Y}(U, V) = U \times V$$

for $x \in X, U \in \mathcal{P}X$ and $V \in \mathcal{P}Y$.

2. the *sub-probability distribution monad* $\mathcal{S} : \text{Set} \rightarrow \text{Set}$, given by

$$\mathcal{S}(X) = \{\varphi : X \rightarrow [0, 1] \mid \sum_{x \in \text{supp}(\varphi)} \varphi(x) \leq 1\}$$

and modelling probabilistic computations. Here, $\text{supp}(\varphi) = \{x \in X \mid \varphi(x) \neq 0\}$ is called the *support* of φ . The unit of \mathcal{S} is given by the Dirac distributions (i.e. $\eta_X(x) = (x \mapsto 1)$), and its multiplication is given by $\mu_X(\Phi) = \sum_{\varphi \in \text{supp}(\Phi)} \sum_{x \in \text{supp}(\varphi)} \Phi(\varphi) * \varphi(x)$, with $*$ denoting multiplication on $[0, 1]$. Its

strength and double strength are given by

$$\text{st}_{X,Y}(x, \psi)(z, y) = \begin{cases} \psi(y) & \text{if } z = x \\ 0 & \text{otherwise} \end{cases} \qquad \text{dst}_{X,Y}(\varphi, \psi)(z, y) = \varphi(z) * \psi(y)$$

for $x \in X, \varphi \in \mathcal{S}(X), \psi \in \mathcal{S}(Y), z \in X$ and $y \in Y$.

3. the *semiring monad* $\text{T}_S : \text{Set} \rightarrow \text{Set}$ with $(S, +, 0, \bullet, 1)$ a commutative semiring, given by

$$\text{T}_S(X) = \{f : X \rightarrow S \mid \text{supp}(f) \text{ is finite}\}$$

Its unit, multiplication, strength and double strength are defined similarly to the sub-probability distribution monad (see [2] for details). As a concrete example we consider the semiring $W = (\mathbb{N}^\infty, \min, \infty, +, 0)$ (sometimes called the *tropical semiring*), and use T_W to model weighted computations.

We restrict attention to commutative, *partially additive* monads [2], as these have been shown in loc. cit. to induce partial commutative semirings, whose carriers will serve as our domains of truth values. To define partial additivity, we begin by observing that any monad $\text{T} : \text{Set} \rightarrow \text{Set}$ with $\text{T}\emptyset = 1$ is such that, for any X , $\text{T}X$ has a *zero element* $0 \in \text{T}X$, obtained as $(\text{T}1_X)(*)$, where $*$ denotes the unique element of 1. This yields a *zero map* $0 : Y \rightarrow \text{T}X$ for any X, Y , given by the composition

$$Y \xrightarrow{!_Y} \text{T}\emptyset \xrightarrow{\text{T}1_X} \text{T}X$$

with the maps $!_Y : Y \rightarrow T\emptyset$ and $!_X : \emptyset \rightarrow X$ arising by finality and initiality, respectively. Partial additivity is then defined using the following map:

$$T(X + Y) \xrightarrow{\langle \mu_X \circ Tp_1, \mu_Y \circ Tp_2 \rangle} TX \times TY \tag{1}$$

where $p_1 = [\eta_X, 0] : X + Y \rightarrow TX$ and $p_2 = [0, \eta_Y] : X + Y \rightarrow TY$.

Definition 1. A monad $T : \mathbf{Set} \rightarrow \mathbf{Set}$ is called partially additive if $T\emptyset = 1$ and the map in (1) is a monomorphism.

Remark 1. When the map in (1) is an isomorphism, then T is called additive. Additive monads were studied in [8,3].

A (partially) additive monad T induces a (partial) addition operation $+$ on the set TX , given by $T[1_X, 1_X] \circ q_{X,X}$:

$$\begin{array}{ccc}
 TX & \xleftarrow{T[1_X, 1_X]} & T(X + X) \xrightarrow{\langle \mu_X \circ Tp_1, \mu_Y \circ Tp_2 \rangle} TX \times TX \\
 & \searrow & \swarrow \\
 & & q_{X,X} \\
 & \xleftarrow{\quad + \quad} &
 \end{array}$$

where $q_{X,X} : TX \times TX \rightarrow T(X + X)$ is the (partial) left inverse of the map $\langle \mu_X \circ Tp_1, \mu_Y \circ Tp_2 \rangle$. That is, $a + b$ is defined if and only if $(a, b) \in \text{Im}(\langle \mu_X \circ Tp_1, \mu_Y \circ Tp_2 \rangle)$. Hence, when T is additive, $+$ is a total operation.

The next result relates commutative, partially additive monads to *partial commutative semirings*. The latter are given by a set S carrying a partial commutative monoid structure $(S, +, 0)$, as well as a commutative monoid structure $(S, \bullet, 1)$, with \bullet distributing over $+$. Specifically, for all $s, t, u \in S$, $s \bullet 0 = 0$, and whenever $t + u$ is defined, so is $s \bullet t + s \bullet u$ and moreover $s \bullet (t + u) = s \bullet t + s \bullet u$.

Proposition 1 ([2]). Let T be a commutative, (partially) additive monad. Then $(T1, 0, +, \bullet, \eta_1(*))$ is a (partial) commutative semiring.

Example 2. For the monads in Example 1, one obtains the commutative semirings $(\{\perp, \top\}, \vee, \perp, \wedge, \top)$ when $T = \mathcal{P}$ and S when $T = T_S$, and the partial commutative semiring $([0, 1], +, 0, *, 1)$ when $T = \mathcal{S}$ (where in the latter case $a + b$ is defined if and only if $a + b \leq 1$).

2.2 Generalised Relations and Relation Lifting

Throughout this section we fix a partial commutative semiring $(S, +, 0, \bullet, 1)$ and, following [2], define a preorder relation \sqsubseteq on S by

$$x \sqsubseteq y \text{ if and only if there exists } z \in S \text{ such that } x + z = y \tag{2}$$

for $x, y \in S$. It follows immediately from the axioms of a partial commutative semiring (see [2]) that \sqsubseteq has $0 \in S$ as bottom element and is preserved by \bullet in each argument.

We now let Rel denote the category¹ with objects given by triples (X, Y, R) , where $R : X \times Y \rightarrow S$ is a function defining a *multi-valued relation* (or *S-relation*), and with arrows from (X, Y, R) to (X', Y', R') given by pairs of functions (f, g) as below, such that $R \sqsubseteq R' \circ (f \times g)$:

$$\begin{array}{ccc} X \times Y & \xrightarrow{f \times g} & X' \times Y' \\ R \downarrow & \sqsubseteq & \downarrow R' \\ S & \xlongequal{\quad} & S \end{array}$$

Here, the order \sqsubseteq on S has been extended pointwise to S -relations with the same carrier. We write $q : \text{Rel} \rightarrow \text{Set} \times \text{Set}$ for the functor taking (X, Y, R) to (X, Y) and (f, g) to itself. It follows easily that q is a fibration, with reindexing functors $(f, g)^* : \text{Rel}_{X', Y'} \rightarrow \text{Rel}_{X, Y}$ taking $R' : X' \times Y' \rightarrow S$ to $R' \circ (f \times g) : X \times Y \rightarrow S$. We also write $\text{Rel}_{X, Y}$ for the *fibre over* (X, Y) , i.e. the subcategory of Rel with objects given by S -relations over $X \times Y$ and arrows given by $(1_X, 1_Y)$.

[2] shows how to canonically lift *polynomial* endofunctors on Set (that is, endofunctors constructed from identity and constant functors using *finite* products and set-indexed coproducts) to the category of generalised relations. To define such liftings, an additional assumption that the unit 1 of \bullet is a top element for \sqsubseteq is made. The *relation lifting* of $F : \text{Set} \rightarrow \text{Set}$ is an endofunctor $\text{Rel}(F) : \text{Rel} \rightarrow \text{Rel}$ making the following diagram commute:

$$\begin{array}{ccc} \text{Rel} & \xrightarrow{\text{Rel}(F)} & \text{Rel} \\ q \downarrow & & \downarrow q \\ \text{Set} \times \text{Set} & \xrightarrow{F \times F} & \text{Set} \times \text{Set} \end{array}$$

The definition of $\text{Rel}(F)$ is by induction on the structure of F , and makes use of the \bullet operation in the case of products of polynomial functors. The reader is referred to [2] for details.

A special relation lifting, called *extension lifting* and induced canonically by a commutative, partially additive monad \mathbb{T} , is also defined in [2]. This time, relations are valued into the partial commutative semiring induced by \mathbb{T} (i.e. $S = \mathbb{T}1$), and the extension lifting $E_{\mathbb{T}} : \text{Rel} \rightarrow \text{Rel}$ lifts the endofunctor $\mathbb{T} \times \text{Id}$ to Rel

$$\begin{array}{ccc} \text{Rel} & \xrightarrow{E_{\mathbb{T}}} & \text{Rel} \\ q \downarrow & & \downarrow q \\ \text{Set} \times \text{Set} & \xrightarrow{\mathbb{T} \times \text{Id}} & \text{Set} \times \text{Set} \end{array}$$

and takes $R : X \times Y \rightarrow \mathbb{T}1$ to the relation $E_{\mathbb{T}}(R) : \mathbb{T}X \times Y \rightarrow \mathbb{T}1$ given by

$$\mathbb{T}X \times Y \xrightarrow{\text{st}'_{X, Y}} \mathbb{T}(X \times Y) \xrightarrow{\mathbb{T}(R)} \mathbb{T}^2 1 \xrightarrow{\mu_1} \mathbb{T}1$$

¹ To keep notation simple, the dependency on S is left implicit.

(The actual definition of the extension lifting in [2] is given in terms of unique 1-linear extensions of relations of type $X \times Y \rightarrow \mathbb{T}1$ to relations of type $\mathbb{T}X \times Y \rightarrow \mathbb{T}1$. However, as shown in loc. cit., the above is an equivalent characterisation.)

2.3 Linear-Time Behaviour via Relation Lifting

This section summarises the definition of the linear-time behaviour of a state in a coalgebra with branching, as proposed in [2]. The approach in loc. cit. applies to coalgebras of functors obtained as arbitrary compositions of a single branching monad and a finite number of polynomial endofunctors on \mathbf{Set} . Here we restrict attention to compositions of type $\mathbb{T} \circ F$. Thus, we model systems with branching as $\mathbb{T} \circ F$ -coalgebras on \mathbf{Set} , where the partially additive, commutative monad $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ specifies the type of branching, and the polynomial endofunctor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ specifies the structure of individual transitions.

Given an arbitrary endofunctor $F : \mathbf{Set} \rightarrow \mathbf{Set}$, an F -coalgebra is given by a pair (C, γ) with C a set (of states), and $\gamma : C \rightarrow FC$ a function describing the one-step evolution of the states. The notion of *coalgebraic bisimulation* provides a canonical and uniform observational equivalence relation between states of F -coalgebras. One of the many (and under mild assumptions, equivalent) definitions of bisimulation involves lifting the endofunctor F to the category of *standard relations* (obtained in our setting by taking $S = (\{\perp, \top\}, \vee, \perp, \wedge, \top)$). A similar approach is taken in [2] to define the extent to which a state in a coalgebra with branching can exhibit a given linear-time behaviour. The definition in loc. cit. differs from the relational definition of bisimulation (for which we refer the reader to [6]) in two ways: (i) generalised relations are used in place of standard relations, and (ii) the relation lifting employed also involves the extension lifting E_{\top} defined earlier, as the goal is to relate branching-time and linear-time behaviours, as opposed to behaviours of the same coalgebraic type.

Having fixed the branching type \mathbb{T} and the transition type F , the final F -coalgebra (Z, ζ) provides a natural choice as domain of observable linear-time behaviours (which we will also refer to as *maximal traces*), whereas the (partial) commutative semiring $(\mathbb{T}1, +, 0, \bullet, 1)$ induced by \mathbb{T} (see Proposition 1) provides, as argued in [2], a natural choice as set of truth values. Throughout this section, and in the remainder of the paper, we assume that the preorder \sqsubseteq induced by this semiring (defined in (2)) is an ω^{op} -chain complete partial order, and has the unit 1 of \bullet as top element. This assumption is satisfied by the preorders associated to the semirings in Example 2, namely \leq on $\{\perp, \top\}$ for $\mathbb{T} = \mathcal{P}$, \leq on $[0, 1]$ for $\mathbb{T} = \mathcal{S}$, and \geq on \mathbb{N}^{∞} for $\mathbb{T} = \mathbb{T}_W$.

The next definition provides a canonical notion of linear-time behaviour of states in coalgebras with branching. It is inspired by a characterisation of coalgebraic bisimilarity (i.e. the largest bisimulation) between states of coalgebras of the same type as the greatest fixpoint of a monotone operator on the category of standard relations (see e.g. [2][Section 2.2] for a summary). It also resembles partition refinement algorithms for computing largest bisimulations on labelled transition systems with finite state spaces [7].

Definition 2 ([2]). *The linear-time behaviour of a state in a $T \circ F$ -coalgebra (C, γ) is the greatest fixpoint of the operator O on $\text{Rel}_{C,Z}$ given by the composition*

$$\text{Rel}_{C,Z} \xrightarrow{\text{Rel}(F)} \text{Rel}_{FC,FZ} \xrightarrow{E_T} \text{Rel}_{T(FC),FZ} \xrightarrow{(\gamma \times \zeta)^*} \text{Rel}_{C,Z} \quad (3)$$

Monotonicity of the operator O is an immediate consequence of the functoriality of $\text{Rel}(F)$, E_T and $(\gamma \times \delta)^*$. The existence of a greatest fixpoint for O is then guaranteed by the following standard result on the existence of fixpoints in chain-complete partial orders, applied to the *dual* of the order \sqsubseteq .

Theorem 1 ([4, Theorem 8.22]). *Let P be a complete partial order and let $O : P \rightarrow P$ be order-preserving. Then O has a least fixpoint.*

Example 3. For $T = \mathcal{P}$, the greatest fixpoint of O relates a state c in a $\mathcal{P} \circ F$ -coalgebra (C, γ) with a state z of the final F -coalgebra if and only if there exists a sequence of choices in the unfolding of γ starting from c , that results in an F -behaviour bisimilar to z . For $T = T_S$, the greatest fixpoint of O yields, for each state in a $S \circ F$ -coalgebra and each maximal trace, the accumulated probability of this trace being exhibited (across all branches). In particular, for *infinite* traces, the associated probability is often 0. The logics defined later provide the ability to also reason about the probability of exhibiting *finite prefixes* of infinite traces. For $T = T_W$, the greatest fixpoint of O maps a pair (c, z) , with c a state in a $T_W \circ F$ -coalgebra and z a maximal trace, to the minimal cost of exhibiting that trace. Intuitively, this is computed by adding the weights of individual transitions along the same branch, and minimising this sum across the various branches.

Remark 2. We recall from [2] that a relation between states of two $T \circ F$ -coalgebras (C, γ) and (D, δ) can also be defined in a similar way, namely as the greatest fixpoint of the operator $O' : \text{Rel}_{C,D} \rightarrow \text{Rel}_{C,D}$ given by the composition

$$\text{Rel}_{C,D} \xrightarrow{\text{Rel}(F)} \text{Rel}_{FC,FD} \xrightarrow{E'_T} \text{Rel}_{T(FC),T(FD)} \xrightarrow{(\gamma \times \zeta)^*} \text{Rel}_{C,D}$$

where $E'_T : \text{Rel} \rightarrow \text{Rel}$ is the lifting of $T \times T$ to Rel :

$$\begin{array}{ccc} \text{Rel} & \xrightarrow{E_T} & \text{Rel} \\ q \downarrow & & \downarrow q \\ \text{Set} \times \text{Set} & \xrightarrow{T \times T} & \text{Set} \times \text{Set} \end{array}$$

taking $R : X \times Y \rightarrow T1$ to the relation $E'_T(R) : TX \times TY \rightarrow T1$ given by

$$TX \times TY \xrightarrow{\text{dst}_{X,Y}} T(X \times Y) \xrightarrow{T(R)} T^2 1 \xrightarrow{\mu_1} T1$$

Example 4. For non-deterministic systems ($T = \mathcal{P}$), the greatest fixpoint of O' relates two states if and only if they admit a common maximal trace (element of

the final F -coalgebra). For probabilistic systems ($\mathbb{T} = \mathcal{S}$), the greatest fixpoint measures the probability of two states exhibiting the same maximal trace (*any* maximal trace), whereas for weighted systems ($\mathbb{T} = \mathbb{T}_W$), the greatest fixpoint measures the *joint* minimal cost of two states exhibiting the same maximal trace. In the latter case, $E'_W : \text{Rel} \rightarrow \text{Rel}$ takes a relation $R : X \times Y \rightarrow W$ to the relation $E'_W(R) : \mathbb{T}_W X \times \mathbb{T}_W Y \rightarrow W$ given by

$$E'_W(R)(f, g) = \min_{x \in \text{supp}(f), y \in \text{supp}(g)} (f(x) + g(y) + R(x, y))$$

The modal and fixpoint logics we introduce later have a similar flavour to the previous example. In particular, for non-deterministic (respectively probabilistic) systems, the resulting logics will support reasoning about the possibility (respectively likelihood) of a state satisfying a certain linear-time property.

3 Generalised Predicates and Predicate Lifting

The standard approach to defining the semantics of modal logics involves interpreting formulas as predicates over the state space of the system of interest. In the coalgebraic approach to modal logic, individual modal operators are interpreted using so called *predicate liftings* [9]. In order to follow the same approach for linear-time logics, we introduce *generalised predicates*, i.e. predicates valued in a partial commutative semiring $(S, +, 0, \bullet, 1)$ with induced order \sqsubseteq .

We let Pred denote the category with objects given by pairs (X, P) with $P : X \rightarrow S$ a function defining a *multi-valued predicate* (or S -*predicate*), and arrows from (X, P) to (X', P') given by functions $f : X \rightarrow X'$ such that $P \sqsubseteq P' \circ f$:

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ P \downarrow & \sqsubseteq & \downarrow P' \\ S & \equiv & S \end{array}$$

As in the case of generalised relations, we obtain a fibration $p : \text{Pred} \rightarrow \text{Set}$, with p taking (X, P) to X . The fibre over X is denoted Pred_X , and the reindexing functor $f^* : \text{Pred}_{X'} \rightarrow \text{Pred}_X$ takes $P' : X' \rightarrow S$ to $P' \circ f : X \rightarrow S$.

The next definition generalises predicate liftings as used in the semantics of coalgebraic modal logics [9].

Definition 3. A predicate lifting of arity $n \in \omega$ for an endofunctor $F : \text{Set} \rightarrow \text{Set}$ is a functor $L : \text{Pred}^n \rightarrow \text{Pred}$ making the following diagram commute:

$$\begin{array}{ccc} \text{Pred}^n & \xrightarrow{L} & \text{Pred} \\ p \downarrow & & \downarrow p \\ \text{Set} & \xrightarrow{F} & \text{Set} \end{array}$$

where the category Pred^n has objects given by tuples (X, P_1, \dots, P_n) with $P_i : X \rightarrow S$ for $i \in \{1, \dots, n\}$, and arrows from (X, P_1, \dots, P_n) to (X', P'_1, \dots, P'_n) given by functions $f : X \rightarrow X'$ such that $P_i \sqsubseteq P'_i \circ f$ for all $i \in \{1, \dots, n\}$.

We now restrict attention to polynomial functors $F : \mathbf{Set} \rightarrow \mathbf{Set}$, and show how to define a canonical *set* of predicate liftings for F by induction on its structure. Since in \mathbf{Set} finite products distribute over arbitrary coproducts, any polynomial endofunctor is naturally isomorphic to a coproduct of finite (including empty) products of identity functors. The next definition exploits this observation.

Definition 4. Let $F = \coprod_{i \in I} \mathbf{Id}^{j_i}$, with $j_i \in \omega$ for $i \in I$. The set of predicate liftings $\Lambda = \{L_i \mid i \in I\}$ has elements $L_i : \mathbf{Pred}^{j_i} \rightarrow \mathbf{Pred}$ with $i \in I$ given by:

$$(L_i)_X(P_1, \dots, P_{j_i})(f) = \begin{cases} P_1(x_1) \bullet \dots \bullet P_{j_i}(x_{j_i}) & \text{if } f = (x_1, \dots, x_{j_i}) \in \iota_i(\mathbf{Id}^{j_i}) \\ 0 & \text{otherwise} \end{cases}$$

The functoriality of this definition follows from the preservation of \sqsubseteq by \bullet .

Example 5. For $F = 1 + A \times \mathbf{Id} \times \mathbf{Id} \simeq 1 + \coprod_{a \in A} \mathbf{Id} \times \mathbf{Id}$, F -coalgebras are binary trees with internal nodes labelled by elements of A . Definition 4 yields a nullary predicate lifting L_0 and an A -indexed set of binary predicate liftings $(L_a)_{a \in A}$:

$$L_0(f) = \begin{cases} 1 & \text{if } f = \iota_1(*) \\ 0 & \text{otherwise} \end{cases}$$

$$(L_a)_X(P_1, P_2)(f) = \begin{cases} P_1(x_1) \bullet P_2(x_2) & \text{if } f = \iota_a(x_1, x_2) \\ 0 & \text{otherwise} \end{cases}$$

Remark 3. One can also define a single, unary predicate lifting $\mathbf{Pred}(F)$ for each polynomial functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$, again by induction on the structure of F :

- If $F = \mathbf{Id}$, $\mathbf{Pred}(F)$ takes an S -predicate to itself.
- If $F = 1$, $\mathbf{Pred}(F)$ takes an S -predicate to the predicate $* \mapsto 1$.
- If $F = F_1 \times F_2$, $\mathbf{Pred}(F)(P) : F_1 X \times F_2 X \rightarrow S$ is given by

$$\mathbf{Pred}(F)(P)(f_1, f_2) = \mathbf{Pred}(F_1)(P)(f_1) \bullet \mathbf{Pred}(F_2)(P)(f_2), \quad \text{for } P : X \rightarrow S.$$

- if $F = \coprod_{i \in I} F_i$, $\mathbf{Pred}(F)(P) : \coprod_{i \in I} F_i X \rightarrow S$ is given by

$$\mathbf{Pred}(F)(P)(\iota_i(f_i)) = \mathbf{Pred}(F_i)(P)(f_i) \quad \text{for } P : X \rightarrow S, i \in I \text{ and } f_i \in F_i X.$$

Indeed, this is the approach taken in [5]. However, this predicate lifting turns out to yield a modal logic with limited expressive power. We show later how $\mathbf{Pred}(F)$ yields a coinductive interpretation of truth in a system with branching.

Example 6. Let $F : \mathbf{Set} \rightarrow \mathbf{Set}$ be as in Example 5. Then $\mathbf{Pred}(F)$ is given by

$$\mathbf{Pred}(F)(P)(\iota_1(*)) = 1 \quad \mathbf{Pred}(F)(P)(\iota_a(x_1, x_2)) = P(x_1) \bullet P(x_2)$$

As can be seen, the resulting unary modality requires the same property (P) to hold on both the left- and the right subtree.

As we are interested in linear-time logics, a special *extension lifting*, akin to the extension lifting of Section 2.2, will be used to abstract away branching.

Definition 5. Let $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ be a commutative, partially additive monad. The extension lifting $P_{\mathbb{T}} : \mathbf{Pred} \rightarrow \mathbf{Pred}$ is the lifting of $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ to \mathbf{Pred}

$$\begin{array}{ccc} \mathbf{Pred} & \xrightarrow{P_{\mathbb{T}}} & \mathbf{Pred} \\ p \downarrow & & \downarrow p \\ \mathbf{Set} & \xrightarrow{\mathbb{T}} & \mathbf{Set} \end{array}$$

which takes $P : X \rightarrow \mathbb{T}1$ to the predicate $P_{\mathbb{T}}(P) : \mathbb{T}X \rightarrow \mathbb{T}1$ given by $\mu_1(\mathbb{T}(P))$.

Remark 4. As in the case of $E_{\mathbb{T}}$, $P_{\mathbb{T}}(P)$ can alternatively be defined as the unique extension of $P : X \rightarrow \mathbb{T}1$ to a \mathbb{T} -algebra homomorphism $(\mathbb{T}X, \mu_X) \rightarrow (\mathbb{T}1, \mu_1)$.

4 Linear-Time Modal Logics

We are now ready to define linear-time logics for coalgebras of type $\mathbb{T} \circ F$, where the partially additive monad $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ and the polynomial functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ are used as in Section 2.3. Our logics will be valued into the partial semiring $(\mathbb{T}1, +, 0, \bullet, 1)$ induced by the monad \mathbb{T} (see Section 2.1).

We begin by fixing a set A of modal operators with associated predicate liftings $(P_{\lambda})_{\lambda \in A}$ for F . A canonical choice for A is given by the set of predicate liftings in Definition 4. The next definition adapts the definition of coalgebraic modal logics [9] in order to provide reasoning about linear-time behaviours.

Definition 6. The logic \mathcal{L}_A has syntax given by

$$\varphi ::= \mathbb{T} \mid [\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)})$$

with $\lambda \in A$ of arity $\text{ar}(\lambda)$, and semantics $\llbracket _ \rrbracket_{\gamma} : \mathcal{L}_A \rightarrow \mathbf{Pred}_C$ (where (C, γ) is a $\mathbb{T} \circ F$ -coalgebra) defined inductively on the structure of formulas by

- $\llbracket \mathbb{T} \rrbracket_{\gamma}(c) = \mathbb{T}$
- $\llbracket [\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)}) \rrbracket_{\gamma} = \gamma^*(P_{\mathbb{T}}(P_{\lambda}(\llbracket \varphi_1 \rrbracket_{\gamma}, \dots, \llbracket \varphi_n \rrbracket_{\gamma})))$

where $\gamma^* : \mathbf{Pred}_C \rightarrow \mathbf{Pred}_{\mathbb{T}FC}$ performs reindexing of predicates along γ .

The semantics of \mathcal{L}_A resembles that of coalgebraic modal logics (see e.g. [9]), with two differences: (i) the interpretation of a formula is a generalised predicate over the state space as opposed to a subset of the state space, and (ii) the extension lifting $P_{\mathbb{T}} : \mathbf{Pred} \rightarrow \mathbf{Pred}$ of Definition 5 is used to abstract away branching. In particular, the use of $P_{\mathbb{T}}$ is what makes \mathcal{L}_A a *linear-time* logic.

It turns out that an equivalent definition of the semantics of \mathcal{L}_A can be given in terms of relation lifting. To show this, we let $L_A = \sum_{\lambda \in A} \text{Id}^{\text{ar}(\lambda)}$, and note that $L_A(L) \simeq \{[\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)}) \mid \lambda \in A, \varphi_1, \dots, \varphi_{\text{ar}(\lambda)} \in L\}$. We now consider the lifting $D : \mathbf{Rel} \rightarrow \mathbf{Rel}$ of the functor $F \times L_A : \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set} \times \mathbf{Set}$ defined through case analysis by

$$D(R)(f, [\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)})) = P_{\lambda}(R^{\sharp}(\varphi_1), \dots, R^{\sharp}(\varphi_{\text{ar}(\lambda)}))(f)$$

for $R : C \times L \rightarrow \mathbb{T}1$, $f \in FC$ and $\varphi_1, \dots, \varphi_{\text{ar}(\lambda)} \in L$, where $R^{\sharp} : L \rightarrow \mathbf{Pred}_C$ is obtained from R by currying.

Lemma 1. $D : \text{Rel} \rightarrow \text{Rel}$ is a functor making the following diagram commute:

$$\begin{array}{ccc} \text{Rel} & \xrightarrow{D} & \text{Rel} \\ U \downarrow & & \downarrow U \\ \text{Set} \times \text{Set} & \xrightarrow{F \times L_A} & \text{Set} \times \text{Set} \end{array}$$

Proof (Sketch). Functoriality of D follows from the functoriality of P_λ for $\lambda \in A$.

An alternative definition of the semantics of \mathcal{L}_A can now be given by turning the initial $(\{\top\} + L_A)$ -algebra (\mathcal{L}_A, α) into a $(\{\top\} + L_A)$ -coalgebra $(\mathcal{L}_A, \alpha^{-1})$.

Proposition 2. Consider the operator S on $\text{Rel}_{C, \mathcal{L}_A}$ given by the composition:

$$\text{Rel}_{C, \mathcal{L}_A} \xrightarrow{D} \text{Rel}_{FC, L_A \mathcal{L}_A} \xrightarrow{E_\top} \text{Rel}_{\top FC, L_A \mathcal{L}_A} \xrightarrow{X} \text{Rel}_{\top FC, \{\top\} + L_A \mathcal{L}_A} \xrightarrow{(\gamma \times \alpha^{-1})^*} \text{Rel}_{C, \mathcal{L}_A}$$

where E_\top is the extension lifting of Section 2.2, and where the lifting $X : \text{Rel} \rightarrow \text{Rel}$ of $\text{Id} \times (\{\top\} + \text{Id}) : \text{Set} \times \text{Set} \rightarrow \text{Set} \times \text{Set}$ takes $R : C \times L \rightarrow \top 1$ to the relation $X(R) : C \times (\{\top\} + L) \rightarrow \top 1$ given by

$$X(R)(c, \iota_1(\top)) = 1, \quad X(R)(c, \iota_2(l)) = R(c, l) \text{ for } c \in C \text{ and } l \in L.$$

Then, the least and greatest fixpoints of S coincide, and the semantics of \mathcal{L}_A is obtained via currying from this unique fixpoint $\text{fix}(S) \in \text{Rel}_{C, \mathcal{L}_A}$.

Proof (Sketch). Let $\text{fix}(S)^\sharp : \mathcal{L}_A \rightarrow \text{Pred}_C$ be obtained from $\text{fix}(S) : C \times \mathcal{L}_A \rightarrow \top 1$ by currying. It follows by induction on the modal depth of a formula φ (degree of nesting of the modalities) that for φ of depth n , $\text{fix}(S)(c, \varphi)$ can be computed in n steps for any $c \in C$ and moreover, $\text{fix}(S)^\sharp(\varphi) = \llbracket \varphi \rrbracket_\gamma$. The proof of the inductive step exploits the close relationships between D and $(P_\lambda)_{\lambda \in A}$ on the one hand, and between the extension liftings E_\top and P_\top on the other.

In Section 5, a fixpoint extension $\mu \mathcal{L}_A$ of \mathcal{L}_A is defined and a similar result is proved for a fragment of $\mu \mathcal{L}_A$.

We note the absence of conjunction and disjunction from \mathcal{L}_A . Restricted versions of these operators can be incorporated into the modal operators, as illustrated by the next example, and this appears to be sufficient in practice. On the other hand, if the domain of truth values carries a lattice structure (which is the case for all three branching monads considered here), then canonical interpretations for conjunction and disjunction exist. The addition of such operators in the general case, as well as the expressiveness of \mathcal{L}_A , are left as future work.

Example 7. Let $F = 1 + A \times \text{Id} \simeq 1 + \coprod_{a \in A} \text{Id}$, and let the nullary modality $*$, the unary modality $\langle a \rangle$ and the binary modality $[a]$ be defined using the predicate liftings $P_* : 1 \rightarrow \text{Pred}$, $P_{\langle a \rangle} : \text{Pred} \rightarrow \text{Pred}$ and $P_{[a]} : \text{Pred} \times \text{Pred} \rightarrow \text{Pred}$ for F ,

given by

$$\begin{array}{ll}
 P_*(\iota_1(*)) = 1 & P_*(\iota_a(x)) = P_*(\iota_{a'}(x)) = 0 \\
 P_{\langle a \rangle}(P)(\iota_a(x)) = P(x) & P_{\langle a \rangle}(P)(\iota_1(*)) = P_{\langle a \rangle}(P)(\iota_{a'}(x)) = 0 \\
 P_{[a]}(P_1, P_2)(\iota_a(x)) = P_1(x) & P_{[a]}(P_1, P_2)(\iota_{a'}(x)) = P_2(x) \\
 P_{[a]}(P_1, P_2)(\iota_1(*)) = 0 &
 \end{array}$$

where $a' \in A \setminus \{a\}$ in the above. Then, the formula $\langle a \rangle \top$ measures the extent to which the output a is observed in the next step. Also, the formula $[a](\top, *)$ measures the extent to which either the output a is observed in the next step, or an output $a' \neq a$ is observed and following that, the computation terminates.

Modalities of this kind can be defined for an arbitrary polynomial endofunctor, but space limitations prevent us from including the general case here.

We conclude this section with a brief discussion on the expressiveness of \mathcal{L}_A . We immediately note that \mathcal{L}_A is intended as a specification logic, and therefore finding a semantically-defined relation that captures the indistinguishability of states by formulas is not the primary concern. This paper does not provide a definitive answer on the expressiveness of \mathcal{L}_A . However, it does provide an answer in the case when the predicate liftings in A are the canonical ones from Definition 4. In this case, \mathcal{L}_A is (isomorphic to) the initial $(\{\top\} + F)$ -algebra, whose elements can be thought of as *finite trace prefixes*, and formulas of \mathcal{L}_A measure the extent to which finite-trace prefixes are exhibited by states of $\top \circ F$ -coalgebras. Thus, two states are indistinguishable by formulas if and only if the extent to which they can exhibit each *finite* linear-time behaviour is the same.

Example 8. For $F = 1 + A \times \text{Id} \simeq 1 + \coprod_{a \in A} \text{Id}$, finite trace prefixes are in one-to-one correspondence with finite sequences of one of the forms $a_1 \dots a_n \top$ or $a_1 \dots a_n *$ with $n \in \omega$ and $a_1, \dots, a_n \in A$, where the latter sequence is also a maximal trace. For $F = 1 + A \times \text{Id} \times \text{Id} \simeq 1 + \coprod_{a \in A} \text{Id} \times \text{Id}$, finite trace prefixes are given by finite binary trees with internal nodes labelled by elements of A and with leafs labelled by either $*$ or \top .

5 Linear-Time Fixpoint Logics

We now extend the logic \mathcal{L}_A with fixpoints, and describe an approach to model checking a fragment of the resulting logic, whose formulas do not contain both least and greatest fixpoints. In order to interpret both greatest and least fixpoints, we additionally assume that the order \sqsubseteq induced by the (partial) semiring of Proposition 1 is ω -chain complete. This assumption holds in all our examples.

Definition 7. Let \mathcal{V} be a set of variables. The logic $\mu\mathcal{L}_A$ has syntax given by

$$\varphi ::= x \mid \top \mid [\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)}) \mid \mu x. \varphi \mid \nu x. \varphi$$

with $x \in \mathcal{V}$ and $\lambda \in A$, and semantics $\llbracket - \rrbracket_\gamma^V : \mu\mathcal{L}_A \rightarrow \text{Pred}_C$ (where (C, γ) is a $\top \circ F$ -coalgebra and $V : \mathcal{V} \rightarrow \text{Pred}_C$ is a valuation) defined inductively on the structure of formulas by

- $\llbracket x \rrbracket_\gamma^V = V(x)$,
- $\llbracket \mu x. \varphi \rrbracket_\gamma^{V \setminus \{x\}}$ ($\llbracket \nu x. \varphi \rrbracket_\gamma^{V \setminus \{x\}}$) is the least (respectively greatest) fixpoint of the operator on Pred_C defined by $P \mapsto \llbracket \varphi \rrbracket_\gamma^{V[P/x]}$, where the valuation $V[P/x] : \mathcal{V} \rightarrow \text{Pred}_C$ is given by $V[P/x](x) = P$ and $V[P/x](y) = V(y)$ for $y \in \mathcal{V} \setminus \{x\}$

and clauses for \top and $[\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)})$ similar to Definition 6.

The fact that the operator used to interpret fixpoint formulas is order-preserving follows from the functoriality of predicate liftings. Existence of the required least and greatest fixpoints then follows by Theorem 1.

Example 9. For $\top = \mathcal{P}$, predicate liftings for F are as used in the semantics of coalgebraic modal logic [9], and $\mu\mathcal{L}_A$ -formulas can be interpreted on F -coalgebras. In this case, the logic $\mu\mathcal{L}_A$ can be viewed as an *existential* version of the logic LTL, wherein a linear-time formula holds in a state whenever a trace satisfying the formula can be exhibited from that state. Our logic is however more general, as it applies to transition structures defined by an arbitrary polynomial functor F . For $\top = \mathcal{S}$ or $\top = \top_W$, $\mu\mathcal{L}_A$ -formulas measure the likelihood, respectively minimal cost, of satisfying a certain linear-time property.

Example 10. Using the modalities $[a]$ and $\langle a \rangle$ of Example 7, the extent to which $a \in A$ appears (i) eventually, (ii) always and (iii) infinitely many times in the unfolding of a state in a $\top \circ F$ -coalgebra is measured by the formulas $\mu x. [a](\top, x)$, $\nu x. \langle a \rangle x$, and respectively $\nu x. \mu y. [a](x, y)$.

Remark 5. The formula $\nu x. \circ x$, with \circ the predicate lifting defined in Remark 3, can be viewed as providing a coinductive interpretation of truth. When $\top = \mathcal{P}$, $\nu x. \circ x$ holds in a state precisely when there exists a maximal trace from that state, arising from a sequence of choices in the branching behaviour. (Such a path will not exist from a state that offers no choices for proceeding.) For $\top = \top_W$, the truth value associated to $\nu x. \circ x$ in a particular state is the minimum accumulated weight that can be achieved along any maximal trace from that state.

Ongoing work concerns the formulation of a result similar to Proposition 2 for the logic $\mu\mathcal{L}_A$, and its exploitation for model checking $\mu\mathcal{L}_A$ -formulas. Here we only present a restricted version of such a result, which concerns the fragment of $\mu\mathcal{L}_A$ whose formulas do not contain both least and greatest fixpoints.

The following definitions are standard in the fixpoint logic literature.

Definition 8. A formula $\varphi \in \mu\mathcal{L}_A$ is *clean* if every variable is bound at most once in φ , and *guarded* if every occurrence of a bound variable appears within the scope of a modal operator. A set $C \subseteq \mu\mathcal{L}_A$ of formulas is *closed* if

- $\varphi \in C$ whenever $[\lambda]\varphi \in C$, for $\lambda \in A$,
- $\varphi[\eta x. \varphi/x] \in C$ whenever $\eta x. \varphi \in C$, for $\eta \in \{\mu, \nu\}$.

The closure $\text{Cl}(\varphi)$ of a $\mu\mathcal{L}_A$ -formula φ is the smallest closed set containing φ .

We now proceed by observing that the set $\mathcal{F} := \text{Cl}(\varphi)$ carries a $\{\top\} + \mathbf{L}_A + \text{Id}$ -coalgebra structure $\alpha : \mathcal{F} \rightarrow \{\top\} + \mathbf{L}_A \mathcal{F} + \mathcal{F}$, defined by:

- $\alpha(\top) = \iota_1(\top)$,
- $\alpha([\lambda](\varphi_1, \dots, \varphi_{\text{ar}(\lambda)})) = \iota_2(\iota_\lambda(\varphi_1, \dots, \varphi_{\text{ar}(\lambda)}))$,
- $\alpha(\eta x. \varphi) = \iota_3(\varphi[\eta x. \varphi/x])$ for $\eta \in \{\mu, \nu\}$.

Our goal is to characterise the semantics of $\mu\mathcal{L}_\Lambda$ using relation lifting, as this was done for the logic \mathcal{L}_Λ . The slight difficulty here is that an unfolding of the $\top \circ F$ -coalgebra γ (performed along an unfolding of the $\{\top\} + \mathbf{L}_\Lambda + \text{Id}$ -coalgebra α) is only required in the case of formulas whose outer-most operator is a modality. For formulas whose outer-most operator is a fixpoint operator, only an unfolding of the respective fixpoint formula should be performed (by unfolding α). This explains the somewhat involved next definition, which, in particular, replaces the coalgebra γ as used in Proposition 2 by the coalgebra $\langle \gamma, \text{id}_C \rangle : C \rightarrow \top FC \times C$.

Definition 9. *The operator $S_\mu : \text{Rel}_{C, \mathcal{F}} \rightarrow \text{Rel}_{C, \mathcal{F}}$ is defined by the composition*

$$\text{Rel}_{C, \mathcal{F}} \xrightarrow{\mathbf{F}} \text{Rel}_{\top FC \times C, \mathbf{L}_\Lambda \mathcal{F} + \mathcal{F}} \xrightarrow{\mathbf{X}} \text{Rel}_{\top FC \times C, \{\top\} + \mathbf{L}_\Lambda \mathcal{F} + \mathcal{F}} \xrightarrow{(\langle \gamma, \text{id}_C \rangle \times \alpha)^*} \text{Rel}_{C, \mathcal{F}}$$

where the lifting $\mathbf{F} : \text{Rel} \rightarrow \text{Rel}$ of $((\top \circ F) \times \text{Id}) \times (\mathbf{L}_\Lambda + \text{Id}) : \text{Set} \times \text{Set} \rightarrow \text{Set} \times \text{Set}$ takes $R : C \times L \rightarrow \top 1$ to the relation $\mathbf{F}(R) : (\top FC \times C) \times (\mathbf{L}_\Lambda L + L) \rightarrow \top 1$ given by

$$\begin{aligned} \mathbf{F}(R)((u, c), \iota_1(\iota_\lambda(\varphi_1, \dots, \varphi_{\text{ar}(\lambda)}))) &= \mathbf{E}_\top(\mathbf{D}(R))(u, \iota_\lambda(\varphi_1, \dots, \varphi_{\text{ar}(\lambda)})) \\ \mathbf{F}(R)((u, c), \iota_2(\varphi)) &= R(c, \varphi) \end{aligned}$$

The lifting \mathbf{F} of Definition 9 plays a rôle similar to that of $\mathbf{E}_\top \circ \mathbf{D}$ in Proposition 2, only its definition is more involved for the reasons identified above.

Theorem 2. *Let $\varphi \in \mu\mathcal{L}_\Lambda$ be a clean, guarded formula containing no free variables, and only least (or only greatest) fixpoint operators. Let $\mathcal{F} := \text{Cl}(\varphi)$, and let (C, γ) be a $\top \circ F$ -coalgebra. Then, $\llbracket \varphi \rrbracket_\gamma \in \text{Pred}_C$ can be obtained as $\text{fix}(S_\mu)^\sharp(\varphi)$, where $\text{fix}(S_\mu) : C \times \mathcal{F} \rightarrow \top 1$ is the least (respectively greatest) fixpoint of the operator S_μ of Definition 9, and $(_)^\sharp$ denotes currying.*

Proof (Sketch). The statement follows by induction on the nesting depth of fixpoint operators. Once the equivalence in Proposition 2 is taken into account, the only difference between the two characterisations of the fixpoint semantics is that in the relational semantics, the approximations of outer fixpoints used in the computation of the inner fixpoints are updated *while* the computation of the inner fixpoints is taking place. Given that all the fixpoints are of the same nature (either least or greatest), this is not a problem. The proof of the inductive step uses the observation that the above difference only impacts on *how quickly* the fixpoint is reached, and not on the truth value of the outer fixpoint formula. For example, in the case of the formula $\mu x. \mu y. [a](x, y)$, the only effect of updating the truth value of x (with a *more accurate* approximation) during the computation of the inner fixpoint is that the outer fixpoint is potentially reached earlier.

We conclude by describing the relevance of Theorem 2 to model checking $\mu\mathcal{L}_\Lambda$ -formulas. We believe the value of this result stands in providing (so far only for a

fragment of $\mu\mathcal{L}_A$), a global approximation procedure that does not require inner fixpoints to be fully computed before the computation of the outer fixpoints can resume. With this procedure, assuming a finite state space, and since the closure of a formula is itself finite, one obtains increasingly accurate approximations of the truth value of a formula in finite time, and can choose to stop computing these approximations as soon as a satisfactory threshold is reached. This methodology can be applied to desirable properties captured by formulas only involving least fixpoints, as well as to undesirable properties captured by formulas only involving greatest fixpoints. In the latter case, as the approximations decrease the truth values of formulas, computing them can be stopped as soon as the truth value of the property of interest is sufficiently small in the initial state(s) of the system.

6 Conclusions and Future Work

We have described a uniform approach to defining linear-time fixpoint logics for a large class of state-based systems, modelled as coalgebras whose type incorporates branching. In our view, employing a universe of truth values derived from the type of branching yields more natural logics which may in time prove easier to model-check. In particular, our results apply to systems with weighted branching, for which temporal logics and associated model checking techniques have hardly been studied. Such systems can be used to model resources, including time, memory or computational power.

Ongoing work concerns extending Theorem 2 to arbitrary $\mu\mathcal{L}_A$ -formulas. Such an extension will provide the necessary support for model checking algorithms based on the relational semantics. Future work will investigate similar logics for coalgebras of even more general types, including arbitrary compositions of a single branching monad with several polynomial endofunctors, as considered in [2]. The expressiveness of the proposed logics also deserves further study.

References

1. Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press (2008)
2. Cirstea, C.: From Branching to Linear Time, Coalgebraically. In: Baelde, D., Carayol, A. (eds.) Proc. FICS 2013. EPTCS, vol. 126, pp. 11–27 (2013)
3. Coumans, D., Jacobs, B.: Scalars, monads, and categories. In: Heunen, C., Sadrzadeh, M., Grefenstette, E. (eds.) Quantum Physics and Linguistics. A Compositional, Diagrammatic Discourse, pp. 184–216. Oxford Univ. Press (2013)
4. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press (2002)
5. Hermida, C., Jacobs, B.: Structural induction and coinduction in a fibrational setting. *Inf. Comput.* 145(2), 107–152 (1998)
6. Jacobs, B.: Introduction to coalgebra. Towards mathematics of states and observations (version 2.0). Draft (2012)
7. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.* 86(1), 43–68 (1990)
8. Kock, A.: Monads and extensive quantities, arXiv:1103.6009 (2011)
9. Pattinson, D.: Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theor. Comput. Sci.* 309(1-3), 177–193 (2003)

A Relatively Complete Calculus for Structured Heterogeneous Specifications*

Till Mossakowski¹ and Andrzej Tarlecki²

¹ Faculty of Computer Science, Otto-von-Guericke University of Magdeburg

² Institute of Informatics, University of Warsaw

Abstract. Proof calculi for structured specifications have been developed independently of the underlying logical system (formalised as institution). Typically, completeness of these calculi requires interpolation properties of the underlying logic. We develop a relatively complete calculus for structured heterogeneous specifications that does not need interpolation.

1 Introduction

The theory of *institutions* [GB92] provides an excellent framework where the theory of specification and formal software development may be presented in an adequately general and abstract way [ST88a, ST12]. The initial work within this area captured specifications built and developments carried out in an arbitrary but fixed logical system formalised as an institution. However, the practice of software specification and development goes much beyond this. Different logical systems may be appropriate or most convenient for specification of different modules of the same system, of different aspects of system behaviour, or of different stages of system development. This leads to the need for a number of logical systems to be used in the same specification and development project, linked by appropriate notions of morphisms between institutions [GR02]. This observation spurred a substantial amount of research work already, and motivates the research presented here.

In such a framework, one works in a *heterogeneous logical environment* formed by a number of logical systems formalised as institutions and linked with each other in a way captured by various maps between institutions. One such logical environment is the CafeOBJ cube [DF02], another one the HETS family of institutions [Mos05], supported by a tool to build and work with heterogeneous specifications [MML07]. The standard ways of building structured specifications within an institution may then be complemented by heterogeneous specification building constructs, that allow one to move specifications from one institution to another, and then combine specifications originally built in different institutions [Tar00, DF02, MML07, Mos05, MT09].

We study here proof systems for so obtained structured heterogeneous specifications. Of course, we build on the calculi that deal with homogeneous specifications, constructed within a single institution. This topic has been well-studied [ST88a, Bor02, Dia08, ST12], with completeness of the resulting systems being the

* This work has been partially supported by the German Research Foundation (DFG) via the SFB/TR 8 “Spatial Cognition”, project I1-[OntoSpace] (TM) and by the Polish Ministry of Science and Higher Education, grant N206 493138 (AT).

main problematic issue. The completeness results, where they can be achieved, either rely on strong interpolation properties, or sacrifice compositionality of the proof systems, allowing the structure of the specifications to be flattened out entirely. We propose a middle way here, keeping as much as possible of the specification structure, and still ensuring completeness of the resulting calculus. We argue that in many practical situations the structure that is kept is relevant, and the minimal massaging of the specifications we suggest brings no real harm.

We extend this idea to heterogeneous specifications, where the required interpolation property cannot be really expected, and our approach is in fact the only realistically possible. A technical tool here is the notion of modification between institution maps (which we adapt from [Dia02]) and lax compatibility of such maps, which serves us to formulate the necessary (and realistic) compatibility conditions that make the completeness of the resulting calculus for structured heterogeneous specifications achievable.

2 Structured Specifications and Proofs

Let us begin by recalling the notion of an institution, as a formalisation of an arbitrary logical system [GB92], assuming that the reader is familiar with all the intuitions that this notion brings in (see [Mac98] for an introduction to category theory).

Definition 2.1. *An institution \mathcal{I} consists of:*

- a category $\mathbf{Sign}_{\mathcal{I}}$ of signatures;
- a functor $\mathbf{Sen}_{\mathcal{I}}: \mathbf{Sign}_{\mathcal{I}} \rightarrow \mathbf{Set}$,¹ giving a set $\mathbf{Sen}(\Sigma)$ of Σ -sentences for each signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$, and a function $\mathbf{Sen}(\sigma): \mathbf{Sen}(\Sigma) \rightarrow \mathbf{Sen}(\Sigma')$, denoted by $\sigma(_)$, that yields σ -translation of Σ -sentences to Σ' -sentences for each signature morphism $\sigma: \Sigma \rightarrow \Sigma'$;
- a functor $\mathbf{Mod}_{\mathcal{I}}: \mathbf{Sign}_{\mathcal{I}}^{op} \rightarrow \mathbf{Class}$,² giving a class $\mathbf{Mod}(\Sigma)$ of Σ -models for each signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$, and a functor $\mathbf{Mod}(\sigma): \mathbf{Mod}(\Sigma') \rightarrow \mathbf{Mod}(\Sigma)$, denoted by $_|\sigma$, that yields σ -reducts of Σ' -models for each signature morphism $\sigma: \Sigma \rightarrow \Sigma'$; and
- for each $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$, a satisfaction relation $\models_{\mathcal{I}, \Sigma} \subseteq \mathbf{Mod}_{\mathcal{I}}(\Sigma) \times \mathbf{Sen}_{\mathcal{I}}(\Sigma)$

such that for any signature morphism $\sigma: \Sigma \rightarrow \Sigma'$, Σ -sentence $\varphi \in \mathbf{Sen}_{\mathcal{I}}(\Sigma)$ and Σ' -model $M' \in \mathbf{Mod}_{\mathcal{I}}(\Sigma')$:

$$M' \models_{\mathcal{I}, \Sigma'} \sigma(\varphi) \iff M'|\sigma \models_{\mathcal{I}, \Sigma} \varphi \quad [\text{Satisfaction condition}]$$

The satisfaction condition expresses that truth is invariant under change of notation and context.

Example 2.2. Propositional Logic. The institution **Prop** of propositional logic has sets Σ (of propositional symbols) as signatures, and functions $\sigma: \Sigma_1 \rightarrow \Sigma_2$ between such sets as signature morphisms. A Σ -model M is a mapping from Σ to $\{\text{true}, \text{false}\}$. The

¹ The category **Set** has all sets as objects and all functions as morphisms.

² **Class** is the quasi-category of all classes, where “quasi” means that it lives in a higher set-theoretic universe. If model morphisms are needed, one may use categories instead of classes.

reduct of a Σ_2 -model M_2 along $\sigma: \Sigma_1 \rightarrow \Sigma_2$ is the Σ_1 -model given by the composition $\sigma; M_2$.³ Σ -sentences are built from Σ with the usual propositional connectives, and sentence translation along a signature morphism just replaces the propositional symbols along the morphism. Finally, satisfaction of a sentence in a model is defined by the standard truth-table semantics. It is straightforward to see that the satisfaction condition holds.

Example 2.3. Untyped First-order Logic. In the institution $\mathbf{UFOL}^=$ of untyped first-order logic with equality, signatures are first-order signatures, consisting of a set of function symbols with arities, and a set of predicate symbols with arities. Signature morphisms map symbols so that arities are preserved. Models are first-order structures, and sentences are first-order formulas. Sentence translation means replacement of the translated symbols. Model reduct means reassembling the model's components according to the signature morphism. Satisfaction is the usual satisfaction of a first-order sentence in a first-order structure.

Many-sorted First-order Logic. The institution $\mathbf{FOL}^=$ of many-sorted first-order logic with equality is similar to $\mathbf{UFOL}^=$. Signatures are many-sorted first-order signatures, consisting of sorts and typed function and predicate symbols. The rest is similar to $\mathbf{UFOL}^=$. For details, see [GB92].

Many-sorted Partial First-order Logic. The institution $\mathbf{PFOL}^=$ is similar to $\mathbf{FOL}^=$, but functions can be partial. Atomic formulas evaluate to false if some component term involves some undefinedness. See [CoF04].

CASL extends $\mathbf{PFOL}^=$ with subsorting and induction (for datatypes), see [CoF04].

Many-sorted Equational Logic (EqL) is the sublogic of $\mathbf{FOL}^=$ restricting signatures to those without predicate symbols and sentences to universally quantified equations.

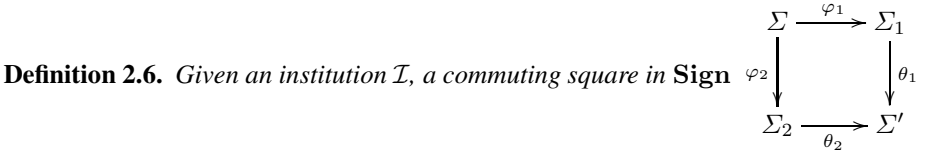
In any institution \mathcal{I} , standard logical notions, like the model class $Mod(\Gamma)$ for any set Γ of sentences, semantic (logical) consequence $\Gamma \models \varphi$ for any set Γ of sentences and sentence φ over the same signature, are defined as usual. In particular, a *theory* is a pair $T = \langle \Sigma, \Gamma \rangle$, where $\Sigma \in \mathbf{Sign}$ and $\Gamma \subseteq \mathbf{Sen}(\Sigma)$. *Theory morphisms* are signature morphisms mapping axioms to logical consequences, leading to a category \mathbf{Th} of theories. It is easy to extend this to an *institution of theories* $\mathcal{I}^{th} = (\mathbf{Th}, \mathbf{Sen}, \mathbf{Mod}, \models)$ over \mathcal{I} .

Definition 2.4. A cocone for a diagram in \mathbf{Sign} is (weakly) amalgamable if it is mapped to a (weak) limit in \mathbf{Class} under \mathbf{Mod} . \mathcal{I} (or \mathbf{Mod}) admits (finite) (weak) amalgamation if (finite) colimits exists in \mathbf{Sign} and colimiting cocones are (weakly) amalgamable, i.e. if \mathbf{Mod} maps (finite) colimits to (weak) limits. An important special case is pushouts: \mathcal{I} (or \mathbf{Mod}) is (weakly) semi-exact, if pushouts exist in \mathbf{Sign} and are (weakly) amalgamable.

Definition 2.5. An institution \mathcal{I} is quasi-exact if for each diagram $D: J \rightarrow \mathbf{Sign}$, there is some weakly amalgamable cocone over D . Quasi-semi-exactness is the restriction of this notion to diagrams of shape $\bullet \longleftarrow \bullet \longrightarrow \bullet$.

³ We write composition in any category in the diagrammatic order and denote it by “;”.

We recall a variant of Craig interpolation that better fits for logics that may have no implication, namely *Craig-Robinson interpolation* [DM00, Sho67].



admits Craig-Robinson interpolation whenever for all finite sets of sentences $\Psi_1 \subseteq \mathbf{Sen}(\Sigma_1)$ and $\Psi_2, \Gamma_2 \subseteq \mathbf{Sen}(\Sigma_2)$, if $\theta_1(\Psi_1) \cup \theta_2(\Gamma_2) \models \theta_2(\Psi_2)$ then there exists a finite set Ψ of Σ -sentences such that $\Psi_1 \models \varphi_1(\Psi)$ and $\varphi_2(\Psi) \cup \Gamma_2 \models \Psi_2$.

\mathcal{I} has Craig-Robinson interpolation if all signature pushouts admit Craig-Robinson interpolation.

This is the category-theoretic generalisation of the usual notion of interpolation. In particular, the usual notion of “common language of Ψ_1 and Ψ_2 ” is generalised to an arbitrary span $\Sigma_1 \xleftarrow{\varphi_1} \Sigma \xrightarrow{\varphi_2} \Sigma_2$ (imagine Σ to be the intersection $\Sigma_1 \cap \Sigma_2$). Craig interpolation is a weaker version of Craig-Robinson interpolation, with $\Gamma_2 = \emptyset$ in Def. 2.6.

Institutions were originally introduced to free the theory of specifications from dependency on any particular logical system. We follow [ST88a] and for any institution \mathcal{I} consider a class $Spec_{\mathcal{I}}$ of specifications built in \mathcal{I} from *basic specifications* (*presentations*, which consist of a signature and a set of sentences over this signature) by means of a number of *specifications-building operations*. Fix an institution $\mathcal{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$. Simultaneously with the notion of structured specification, we define functions *Sig* and *Mod* yielding the signature and the model class for any specification.

presentations: For any signature $\Sigma \in |\mathbf{Sign}|$ and finite set $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ of Σ -sentences, the *presentation* $\langle \Sigma, \Gamma \rangle$ is a specification with:

$$Sig[\langle \Sigma, \Gamma \rangle] := \Sigma \quad Mod[\langle \Sigma, \Gamma \rangle] := \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Gamma\}$$

union: For any signature $\Sigma \in |\mathbf{Sign}|$, given Σ -specifications SP_1 and SP_2 , their *union* $SP_1 \cup SP_2$ is a specification with:

$$Sig[SP_1 \cup SP_2] := \Sigma \quad Mod[SP_1 \cup SP_2] := Mod[SP_1] \cap Mod[SP_2]$$

translation: For any signature morphism $\sigma : \Sigma \rightarrow \Sigma'$ and Σ -specification SP , $\sigma(SP)$ is a specification with:

$$Sig[\sigma(SP)] := \Sigma' \quad Mod[\sigma(SP)] := \{M' \in \mathbf{Mod}(\Sigma') \mid M'|_{\sigma} \in Mod[SP]\}$$

hiding: For any signature morphism $\sigma : \Sigma \rightarrow \Sigma'$ and Σ' -specification SP' , $SP'|_{\sigma}$ is a specification with:

$$Sig[SP'|_{\sigma}] := \Sigma \quad Mod[SP'|_{\sigma}] := \{M'|_{\sigma} \mid M' \in Mod[SP']\}$$

Typical structuring constructs of many existing specification languages like CASL [CoF04], CafeOBJ [DF02] and others can be mapped to this kernel formalism.

The semantics determines specification equivalence: $SP_1 \equiv SP_2$ iff $Sig[SP_1] = Sig[SP_2]$ and $Mod[SP_1] = Mod[SP_2]$. Furthermore, we get the obvious notion of semantic consequence for structured specifications: given a specification SP , a sentence $\varphi \in \mathbf{Sen}(Sig[SP])$ is a *semantic consequence* of SP , written $SP \models \varphi$, if $M \models \varphi$ for all models $M \in Mod[SP]$. Then, given two specifications SP and SP' , SP *refines* to SP' , written $SP \Rightarrow SP'$, if $Sig[SP] = Sig[SP']$ and $Mod[SP'] \subseteq Mod[SP]$. These two simple notions underlie the standard view of properties that specifications ensure and of systematic development of programs from specifications by step-wise refinements, see [ST88b, ST12].

3 Proofs

We very briefly recalled above the semantic concepts developed within the theory of institutions that underlie the methodology for formal specification and systematic development of software, cf. [ST12]. For practical applications they need a proof-theoretic counterpart, whereby the semantic, hard to establish relationships are augmented by calculi to approximate them in an effective way.

Proof-theoretic entailment to approximate semantic entailment in any institution is captured by the following notion, introduced in the institutional context in [FS88] under the name of π -institution, see also [Mes89, HST94].

Definition 3.1. *Given an institution $\mathcal{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$, an entailment system \vdash for \mathcal{I} consists of a relation $\vdash_{\Sigma} \subseteq \mathcal{P}(\mathbf{Sen}(\Sigma)) \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$, such that the following properties are satisfied:*

1. reflexivity: for any $\varphi \in \mathbf{Sen}(\Sigma)$, $\{\varphi\} \vdash_{\Sigma} \varphi$,
2. monotonicity: if $\Gamma \vdash_{\Sigma} \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_{\Sigma} \varphi$,
3. transitivity: if $\Gamma \vdash_{\Sigma} \varphi_i$ for $i \in I$ and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_{\Sigma} \psi$, then $\Gamma \vdash_{\Sigma} \psi$,
4. \vdash -translation: if $\Gamma \vdash_{\Sigma} \varphi$, then for any $\sigma: \Sigma \rightarrow \Sigma'$ in \mathbf{Sign} , $\sigma(\Gamma) \vdash_{\Sigma'} \sigma(\varphi)$,
5. soundness: if $\Gamma \vdash_{\Sigma} \varphi$ then $\Gamma \models_{\Sigma} \varphi$.

The entailment system is complete if, in addition, $\Gamma \models_{\Sigma} \varphi$ implies $\Gamma \vdash_{\Sigma} \varphi$.

A logic $\mathcal{LOG} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models, \vdash)$ is an institution $(\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ equipped with an entailment system \vdash . In an arbitrary logic, it is possible to design a logic independent proof calculus [ST88a] for proving entailments between specifications and sentences, written in the form $SP \vdash \varphi$, where SP is a structured specification and φ is a formula, see Fig. 1.

$$\boxed{
\begin{array}{l}
(CR) \frac{\{SP \vdash \varphi_i\}_{i \in I} \quad \{\varphi_i\}_{i \in I} \vdash \varphi}{SP \vdash \varphi} \quad (basic) \frac{\varphi \in \Gamma}{\langle \Sigma, \Gamma \rangle \vdash \varphi} \quad (sum1) \frac{SP_1 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi} \\
(sum2) \frac{SP_2 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi} \quad (trans) \frac{SP \vdash \varphi}{\sigma(SP) \vdash \sigma(\varphi)} \quad (derive) \frac{SP \vdash \sigma(\varphi)}{SP|_{\sigma} \vdash \varphi}
\end{array}
}$$

Fig. 1. Proof calculus for entailment in structured specifications

$(Basic) \frac{SP \vdash \varphi \text{ for all } \varphi \in \Gamma}{\langle \Sigma, \Gamma \rangle \rightsquigarrow SP}$	$(Sum) \frac{SP_1 \rightsquigarrow SP \quad SP_2 \rightsquigarrow SP}{SP_1 \cup SP_2 \rightsquigarrow SP}$
$(Trans) \frac{SP \rightsquigarrow SP' _\sigma}{\sigma(SP) \rightsquigarrow SP'}$	$(Derive) \frac{SP \rightsquigarrow SP''}{SP _\sigma \rightsquigarrow SP''} \quad \text{if } \sigma: SP' \longrightarrow SP'' \text{ is a conservative extension}$

Fig. 2. Proof calculus for refinement of structured specifications

Fig. 2 shows an extension of this calculus to refinements between specifications, with judgements written as $SP \rightsquigarrow SP'$, where SP and SP' are structured specifications with a common signature. Note that rule (CR) can be limited to a finitary version for compact institutions (where an institution is compact if $\Gamma \models \varphi$ implies the existence of a finite $\Gamma' \subseteq \Gamma$ with $\Gamma' \models \varphi$). The extended calculus relies on an *oracle for conservative extensions*, where given specifications SP and SP' , a signature morphism $\sigma: Sig[SP] \rightarrow Sig[SP']$ is a *conservative extension* if it is a specification morphism $\sigma: SP \rightarrow SP'$ (i.e., $M'|_\sigma \in Mod[SP]$ for all $M' \in Mod[SP']$) and is *conservative* (for all $M \in Mod[SP]$ there is $M' \in Mod[SP']$ with $M'|_\sigma = M$).

Theorem 3.2 (Soundness [ST88a, Bor02]). *The calculi for specification entailment and refinement between structured specifications given above are sound: if $SP \vdash \varphi$ then $SP \models \varphi$, and if $SP \rightsquigarrow SP'$ then $SP \Rightarrow SP'$.*

Theorem 3.3 (Completeness [Bor02, Dia08, ST13]). *Assuming that*

- *the institution has Craig-Robinson interpolation,*
- *the institution is weakly semi-exact,*
- *the entailment system is complete,*

the calculi for specification entailment and refinement between structured specifications are sound and complete: $SP \vdash \varphi$ iff $SP \models \varphi$, and $SP \rightsquigarrow SP'$ iff $SP \Rightarrow SP'$.

Actually, as discussed in [Bor02, ST13], the assumption of Craig-Robinson interpolation and weak amalgamation can be restricted to those pushouts for which it is really needed. Typically, we can limit the classes of morphisms used to build structured specification by hiding and translation, respectively. Under suitable technical conditions, Craig-Robinson interpolation is needed then for pushouts of spans formed by morphisms permitted in hiding on the left and those permitted in translations on the right. Still, the requirement that the institution admits Craig-Robinson interpolation is the strongest assumption in Thm. 3.3. While it holds in many logics, there are prominent examples where it fails. For example, even Craig interpolation fails in **QS5**, the first-order version of the modal logic **S5** [Fin79], which is just one instance of many failures of interpolation in various versions of modal logics. Interpolation also fails in some typical logical systems used in specification formalisms, with interpretation of some types or concepts fixed semantically; for instance, interpolation fails for the logic of CASL due to CASL-style subsorting [Bor00]. Even the standard first-order logic may cause problems here. While untyped first-order logic **UFOL**[≠] has Craig-Robinson interpolation, its many-sorted version **FOL**[≠] admits Craig-Robinson interpolation for pushouts of spans where at least one morphism is injective on sorts. To use Thm. 3.3

even in the refined version hinted at above for specifications in \mathbf{FOL}^- we would have to limit the use of hiding to signature morphisms that are injective on sorts — a seriously limiting restriction. Things get even worse with many-sorted equational logic \mathbf{EqL} , which admits Craig-Robinson interpolation for pushouts of spans with the left morphism satisfying a strong “encapsulation” property, see [Dia08] (Craig, but not necessarily Craig-Robinson interpolation, is also ensured here for pushouts of spans with the right morphism being injective).

As shown in [ST13], Craig-Robinson interpolation is necessary for the completeness of the above calculi, and moreover, the calculus for specification entailments cannot be improved without sacrificing its *compositionality* (consequences of a structured specification are deduced from the consequences of its immediate components).

When we sacrifice compositionality of the calculus, a sound and complete calculus may be obtained also for institutions without interpolation when we agree that specifications are “massaged” before calculating their consequences, so allowing the calculus to reach arbitrarily deep into the specification structure. This is often done using *normal forms* of specifications. The well-known normal form result is that each structured specification SP as considered here can be turned into an equivalent normal form $nf(SP) = \langle \Sigma', \Gamma' \rangle |_\sigma$, thus entirely flattening the specification to a theory with a single use of hiding (this requires the institution to have relevant signature pushouts that admit weak amalgamation). Then an obvious rule

$$(nf) \frac{\Gamma' \vdash \sigma(\varphi)}{SP \vdash \varphi} \quad \text{if } nf(SP) = \langle \Sigma', \Gamma' \rangle |_\sigma$$

yields a sound and complete calculus for specification entailments.

However, this normal form and its use in the above proof rule entirely forgets about any structure that was given in SP . We show how some key aspects of the structure may be maintained without losing the completeness of the calculus. To achieve this we define a *structured normal form* $snf(SP)$ for any structured specification SP , which only pushes out the hiding operations, while retaining the key structure given by union and translation (and, very informally, renaming hidden symbols to avoid unintended name clashes). The definition below requires existence and suitable choice of the relevant signature pushouts:

$$\overline{snf(\langle \Sigma, \Gamma \rangle)} = \langle \Sigma, \Gamma \rangle |_{id}$$

$$\frac{snf(SP_1) = SP'_1 |_{\sigma_1} \quad snf(SP_2) = SP'_2 |_{\sigma_2}}{snf(SP_1 \cup SP_2) = (\theta_1(SP'_1) \cup \theta_2(SP'_2)) |_{\sigma_1; \theta_1}} \quad \text{if} \quad \begin{array}{ccc} Sig[SP_1] & \xrightarrow{\sigma_1} & Sig[SP'_1] \\ \downarrow \sigma_2 & & \downarrow \theta_1 \\ Sig[SP'_2] & \xrightarrow{\theta_2} & \Sigma' \end{array} \quad \begin{array}{l} \text{is a} \\ \text{pushout} \end{array}$$

$$\frac{snf(SP) = SP' |_{\sigma_1}}{snf(\sigma_2(SP)) = (\theta_1(SP')) |_{\theta_2}} \quad \text{if} \quad \begin{array}{ccc} Sig[SP] & \xrightarrow{\sigma_1} & Sig[SP'] \\ \downarrow \sigma_2 & & \downarrow \theta_1 \\ \Sigma_2 & \xrightarrow{\theta_2} & \Sigma' \end{array} \quad \text{is a pushout}$$

$$\frac{snf(SP) = SP' |_\sigma}{snf(SP |_\theta) = SP' |_{\theta; \sigma}}$$

Proposition 3.4. *In any weakly semi-exact institution, SP and $snf(SP)$ are equivalent.*

Moreover, we can obtain a stronger completeness result:

Theorem 3.5. *Under the assumptions that the institution is weakly semi-exact and the entailment system is complete, the calculi for specification entailments and refinement between structured specifications extended by the following structured normal form rule:*

$$(snf) \frac{SP' \vdash \sigma(\varphi)}{SP \vdash \varphi} \quad \text{if } snf(SP) = SP'|_{\sigma}$$

are sound and complete.

Let us stress again that using structured normal forms is much better than using normal forms: the latter flatten out specification structure completely, while the former keep the structure almost intact — only hiding, not so frequently used in typical specifications, is moved outside. In many institutions, the obvious choice of signature pushouts involved in the definition of snf leads to the structured normal forms where all the visible names are kept as in the original structured specification, while only the hidden operations may need to be renamed so that name clashes are avoided. This allows proof search strategies in structured specifications, as discussed for instance in [SB83, HST94], to be easily mimicked in their corresponding structured normal forms.

Consequently, the above proof calculus for specification entailments with the rule (snf) offers a well-balanced choice, maintaining the key advantages of compositionality and keeping the need for restructuring specifications to the necessary minimum.

A complete oracle for conservative extensions is very powerful: it can be used to trivially obtain a complete refinement calculus. Namely, in order to decide whether $SP_1 \Rightarrow SP_2$, it suffices to check whether $SP_1 \cup SP_2$ is a conservative extension of SP_2 . Nevertheless, our completeness theorem is meaningful and useful. This is because the completeness proof uses the oracle for conservative extensions only in a limited way. The extensions considered are those obtained from hidings (pushed along some morphism into a “big” signature collecting everything). This means, for example, that if we use hiding only to hide symbols that have been defined using some logic-specific definition scheme, we will need the oracle for conservative extensions only for checking this definition scheme — and typically all such “definitional extensions” are conservative. We cannot expect in general to check conservativity independently of the underlying institution; institution-specific rules are needed. See e.g. [CMM13] for checking conservativity in CASL.

4 Heterogeneous Specifications

So far, we have covered specifications built and their refinements carried out in an arbitrary but fixed logical system formalised as an institution. In practice though, different logical systems may be appropriate or most convenient for specification of different modules of the same system, of different aspects of system behaviour, or of different stages of system development. This leads to the need for a number of logical systems to be used in the same specification and development project. This makes sense though

only if the logical systems involved (formalised as institutions) are linked appropriately, with links captured by various notions of morphisms between institutions [GR02], yielding heterogeneous specification environments such as those of CafeOBJ [DF02] and HETS [MML07].

Definition 4.1. An institution comorphism $\rho: \mathcal{I} \rightarrow \mathcal{I}'$ consists of:

- a functor $\Phi: \mathbf{Sign} \rightarrow \mathbf{Sign}'$;
- a natural transformation $\alpha: \mathbf{Sen} \rightarrow \Phi; \mathbf{Sen}'$, and
- a natural transformation $\beta: \Phi^{op}; \mathbf{Mod}' \rightarrow \mathbf{Mod}$,

such that for any $\Sigma \in |\mathbf{Sign}|$, for any $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}'(\Phi(\Sigma))$:

$$M' \models'_{\Phi(\Sigma)} \alpha_{\Sigma}(\varphi) \iff \beta_{\Sigma}(M') \models_{\Sigma} \varphi \quad [\text{Satisfaction condition}]$$

Institution comorphisms compose in the obvious, component-wise manner. The category of institutions with institution comorphisms is denoted by $co\mathcal{INS}$.

Example 4.2. Consider the translation of propositional logic into untyped first-order logic, mapping propositions to unary predicates plus a global constant a . An atomic sentence p is mapped to $p(a)$; this is inductively extended to all sentences. A first-order model is translated to a propositional model by inspecting whether the interpretation of a is contained in a given predicate. This can easily be organised as an institution comorphism.

Example 4.3. Many examples for comorphisms arise from *substitutions*, where we follow [Mes89] and define them as comorphisms $\rho = \langle \Phi, \alpha, \beta \rangle$ such that the signature translation Φ is an embedding of categories, all sentence translations α_{Σ} are injective and all model translations β_{Σ} are isomorphisms. For example, propositional logic and many-sorted equational logic are both substitutions of many-sorted first-order logic (but not of untyped first-order logic).

Example 4.4. The encoding of $\mathbf{PFOL}^=$ into $\mathbf{FOL}^=$ that adds definedness predicates to signatures and restricts carrier sets of models to these predicates can easily be formalised as an institution comorphism [Mos02b].

The following properties of institution comorphisms ensure a good interaction with logical consequence:

Definition 4.5. An institution comorphism is model expansive, if all the model translation functors are surjective on objects.

An institution comorphism is (weakly) exact, if the naturality squares for the model translation are (weak) pullbacks.

For example, the comorphism from propositional logic to \mathbf{UFOL} from Example 4.2 is model-expansive and weakly exact. Any substitution comorphism is both model-expansive and exact.

The notion of a *heterogeneous logical environment* (called *indexed coinstitutions* in [Mos02a], dualising the *indexed institutions* of [Dia02]) may be formalised as a collection of institutions linked by institution comorphisms.

Definition 4.6. A heterogeneous logical environment \mathcal{HLE} is a collection of institutions and institution comorphisms between them, that is, a diagram $\mathcal{HLE}: \mathcal{G} \rightarrow \text{coINS}^4$ in the category coINS .

Working in a heterogeneous logical environment, we can enrich the collection of specification-building operations by translation along institution comorphisms, see [ST12]. Somewhat less naturally, we can also define hiding w.r.t. institution comorphisms, but the target signature has to be given explicitly then. Namely, given an institution comorphism $\rho: \mathcal{I} \rightarrow \mathcal{I}'$, we define:

heterogeneous translation: For any \mathcal{I} -specification SP , $\rho(SP)$ is a specification with:

$$\text{Sig}[\rho(SP)] := \Phi(\text{Sig}[SP]) \quad \text{Mod}[\rho(SP)] := \beta_{\text{Sig}[SP]}^{-1}(\text{Mod}[SP])$$

heterogeneous hiding: For any \mathcal{I}' -specification SP' and signature Σ with $\text{Sig}[SP'] = \Phi(\Sigma)$, $SP'|_{\rho}^{\Sigma}$ is a specification with:

$$\text{Sig}[SP'|_{\rho}^{\Sigma}] := \Sigma \quad \text{Mod}[SP'|_{\rho}^{\Sigma}] := \beta_{\Sigma}(\text{Mod}[SP'])$$

These new, inter-institutional specification-building operations may be arbitrarily mixed with other (intra-institutional) operations, yielding heterogeneous specifications. Parts of such specifications may be given in different institutions of the heterogeneous logical environment we work in. However, each such a specification as a whole eventually focuses on a particular institution in this environment, where its overall semantics (signature and the class of models) is given. In essence, viewed from a certain perspective, such *focused heterogeneous specifications* do not differ much from the structured specifications built within a single institution. For instance, the view of a software specification and development process as presented in [ST12] directly adapts to the use of such specifications without much (semantic) change. We will make this view more formal now.

Definition 4.7. Consider institutions \mathcal{I} and \mathcal{I}' and signatures $\Sigma \in |\mathbf{Sign}|$ and $\Sigma' \in |\mathbf{Sign}'|$. A heterogeneous signature comorphism is a pair $\langle \rho, \sigma' \rangle: \Sigma \rightarrow \Sigma'$ that consists of an institution comorphism $\rho: \mathcal{I} \rightarrow \mathcal{I}'$ and a signature morphism $\sigma': \Phi(\Sigma) \rightarrow \Sigma'$ in \mathbf{Sign}' . It induces the heterogeneous reduct $_|_{\langle \rho, \sigma' \rangle}: \mathbf{Mod}'(\Sigma') \rightarrow \mathbf{Mod}(\Sigma)$ defined as the composition $\mathbf{Mod}'(\sigma'); \beta_{\Sigma}$, i.e., $M'|_{\langle \rho, \sigma' \rangle} = \beta_{\Sigma}(M'|_{\sigma'})$, for all $M' \in \mathbf{Mod}'(\Sigma')$. Heterogeneous sentence translations are defined similarly.

Heterogeneous signature comorphisms compose as expected: $\langle \rho_1, \sigma_1 \rangle; \langle \rho_2, \sigma_2 \rangle = \langle \rho_1; \rho_2, \Phi_2(\sigma_1); \sigma_2 \rangle$. For any heterogeneous logical environment $\mathcal{HLE}: \mathcal{G} \rightarrow \text{coINS}$ this yields the heterogeneous category $\mathbf{Sign}^{\mathcal{HLE}}$ of signatures in institutions in \mathcal{HLE} with heterogeneous comorphisms that involve institution translation comorphisms in \mathcal{HLE} . Then model functors extend to $\mathbf{Mod}^{\mathcal{HLE}}: (\mathbf{Sign}^{\mathcal{HLE}})^{\text{op}} \rightarrow \mathbf{Class}$ using the reducts defined above. Similarly, we obtain $\mathbf{Sen}^{\mathcal{HLE}}: \mathbf{Sign}^{\mathcal{HLE}} \rightarrow \mathbf{Set}$.

Proposition 4.8 ([Mos02a]). The constructions in Def. 4.7 augmented with the family of satisfaction relations defined component-wise yield an institution

$$\mathcal{I}^{\mathcal{HLE}} = \langle \mathbf{Sign}^{\mathcal{HLE}}, \mathbf{Sen}^{\mathcal{HLE}}, \mathbf{Mod}^{\mathcal{HLE}}, \models^{\mathcal{HLE}} \rangle.$$

⁴ We introduce the following notation: the objects $n \in |\mathcal{G}|$ carry institutions $\mathcal{HLE}(n) = \mathcal{I}^n = \langle \mathbf{Sign}^n, \mathbf{Sen}^n, \mathbf{Mod}^n, \langle \models^n \rangle_{\Sigma \in |\mathbf{Sign}^n|} \rangle$ linked by institution comorphisms $\mathcal{HLE}(e) = \rho^e = \langle \Phi^e, \alpha^e, \beta^e \rangle: \mathcal{HLE}(n) \rightarrow \mathcal{HLE}(m)$ for each morphism $e: n \rightarrow m$ in \mathcal{G} .

The institution $\mathcal{I}^{\mathcal{HLE}}$ is known as the *Grothendieck institution* [Dia02, Mos02a].

For full formality, signatures in the heterogeneous categories of signatures defined above should really be written as pairs $\langle i, \Sigma \rangle$, with $i \in |\mathcal{G}|$.

The inter-institutional specification-building operations given above arise as hiding w.r.t. and translation along heterogeneous signature comorphisms within the Grothendieck institution $\mathcal{I}^{\mathcal{HLE}}$. Namely, given an institution comorphism $\rho: \mathcal{I} \rightarrow \mathcal{I}'$ and \mathcal{I} -specification SP , $\rho(SP)$ can be captured as $\langle \rho, id_{\Sigma'} \rangle(SP)$, where $\Sigma' = \Phi(\text{Sig}[SP])$. Similarly, for \mathcal{I}' -specification SP' and \mathcal{I} -signature Σ such that $\Phi(\Sigma) = \text{Sig}[SP']$, $SP'|_{\rho}^{\Sigma}$ becomes now $SP'|_{\langle \rho, id_{\Sigma} \rangle}$. Conversely, the “intra-institutional” specification-building operations introduced in Sect. 2 in the Grothendieck institution $\mathcal{I}^{\mathcal{HLE}}$ may be presented using the inter-institutional operations introduced above in combination with intra-institutional operations in component institutions. In particular, translation along $\langle \rho, \sigma \rangle$ is the composition of heterogeneous translation along ρ with (intra-institutional) translation along σ , and analogously for hiding w.r.t. $\langle \rho, \sigma \rangle$.

Consequently, the proof calculi introduced in Sect. 3 can be directly used for heterogeneous specifications by considering them for specifications built in the Grothendieck institution. The soundness (as given by Thm 3.2) carries over without change. The completeness theorems (Thm. 3.3 and 3.5) carry over as well, but the problem is that the assumptions under which they guarantee completeness of the calculi typically fail in Grothendieck institutions for many logical environments. Craig-Robinson interpolation was problematic even for truly homogeneous logical systems — it will fail in Grothendieck institutions for heterogeneous logical environments that contain even one institution where it fails. If all the institutions in the environment have interpolation, it still is likely to fail for the Grothendieck institution, even if [Dia04, Dia08] offer results which carry over Craig-Robinson interpolation from component institutions to the Grothendieck institution — under rather strong assumptions though.

The other key assumption in Thm. 3.3 and, especially, Thm 3.5, the weak amalgamation property, carries over from the heterogeneous logical environment to the Grothendieck institution rather naturally:

Proposition 4.9 ([Mos02a]). *Let $\mathcal{HLE}: \mathcal{G} \rightarrow \text{coINS}$ be a heterogeneous logical environment consisting of comorphisms with cocontinuous signature translation functors. Its Grothendieck institution is (weakly) semi-exact if and only if*

- \mathcal{HLE} is (weakly) locally semi-exact, i.e., each institution in \mathcal{HLE} is (weakly) semi-exact,
- \mathcal{HLE} is (weakly) semi-exact, i.e., pushouts in \mathcal{G} exist and are for each signature, (weak) pullbacks of model translation functors, and
- all institution comorphisms in \mathcal{HLE} are (weakly) exact.

Unfortunately, again, the conditions of Prop. 4.9 are not fulfilled in many typical logical environments. For example, neither the CASL institution nor the HETS logical environment are weakly semi-exact. Indeed, in HETS, there are many spans of institution comorphisms which can only be complemented to squares that do not even commute — see [Mos06] for an example.

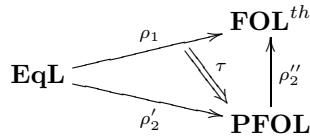
5 Lax Heterogeneous Logical Environments

Sometimes it is useful to indicate that two institution comorphisms differ only in an inessential way. This in particular applies when the comorphisms arise as compositions of other comorphisms. We therefore introduce the notion of *modification*. Modifications are also useful for solving the problem mentioned at the end of the previous section (see Def. 5.6 below). Moreover, they naturally arise when representing comorphisms in some “universal” logic, along with the representation of source and target logic. Following [Mos02a], we dualise and strengthen the original notion from [Dia02] to *discrete* modifications (but we omit the qualifier “discrete” henceforth):

Definition 5.1. *Given two institution comorphisms $\rho_1, \rho_2: \mathcal{I} \rightarrow \mathcal{J}$, an institution comorphism modification $\tau: \rho_1 \rightarrow \rho_2$ is a natural transformation $\tau: \Phi_1 \rightarrow \Phi_2$ such that $\alpha_1; (\mathbf{Sen}_{\mathcal{J}} \cdot \tau) = \alpha_2$ and $(\mathbf{Mod}_{\mathcal{J}} \cdot \tau); \beta_2 = \beta_1$.*

Together with obvious identities and compositions, modifications can serve as 2-cells, leading to a 2-category which we also denote by $co\mathcal{INS}$.

Example 5.2. There are two ways to go from equational logic to first-order logic: one is the obvious substitution comorphism ρ_1 from Example 4.3, the other one is the composition ρ_2 of the obvious substitution comorphism ρ'_2 from equational logic to partial first-order logic with the encoding ρ''_2 of partial first-order logic into first-order logic from Example 4.4. (Actually, the latter ends in \mathbf{FOL}^{th} .) These comorphisms are different: ρ_2 adds some (superfluous) coding of partiality. The comorphism modification $\tau: \rho_1 \rightarrow \rho_2$ is just the pointwise inclusion of an algebraic signature viewed as first-order signature into the theory coding a partial variant of that signature.



This motivates the following extension of the notion of heterogeneous logical environment:

Definition 5.3. *A lax heterogeneous logical environment is a 2-functor $\mathcal{HLE}: \mathcal{G} \rightarrow co\mathcal{INS}$, where both \mathcal{G} and $co\mathcal{INS}$ are 2-categories.⁵*

We can then use the institution comorphism modifications to obtain a congruence on Grothendieck signature morphisms: the congruence is generated by

$$\langle d', \tau_{\Sigma}^u: \Phi^{d'}(\Sigma) \rightarrow \Phi^d(\Sigma) \rangle \equiv \langle d, id: \Phi^d(\Sigma) \rightarrow \Phi^d(\Sigma) \rangle: \langle i, \Sigma \rangle \rightarrow \langle j, \Phi^d(\Sigma) \rangle$$

for $\Sigma \in \mathbf{Sign}^i$, $d, d': i \rightarrow j \in \mathcal{G}$, and $u: d' \Rightarrow d \in \mathcal{G}$. This congruence has the following crucial property:

⁵ Extending the notation introduced in footnote 4, a 2-cell $u: d \Rightarrow d'$ determines the corresponding modification $\mathcal{HLE}(u) = \tau^u: \rho^d \Rightarrow \rho^{d'}$.

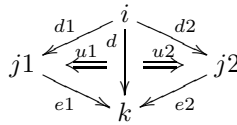
Proposition 5.4. *Equivalent signature morphisms have identical sentence translation and model reduct functors.*

Let $q^{\mathcal{HLE}} : \mathbf{Sign}^{\mathcal{HLE}} \rightarrow \mathbf{Sign}^{\mathcal{HLE}}/\equiv$ be the quotient functor induced by \equiv (see [Mac98] for the definition of quotient category). Note that it is the identity on objects. We easily obtain that the model and sentence functors of the Grothendieck institution $\mathcal{I}^{\mathcal{HLE}}$ factor through the quotient category $\mathbf{Sign}^{\mathcal{HLE}}/\equiv$:

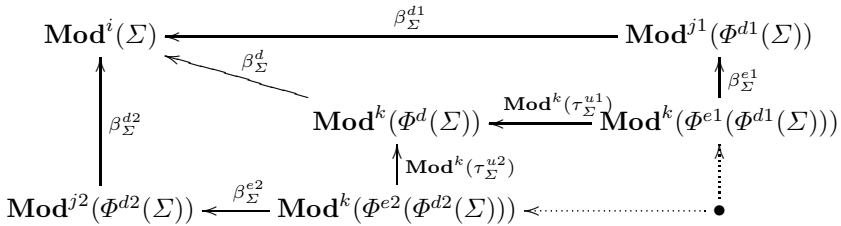
Corollary 5.5. *The components of the Grothendieck institution $\mathcal{I}^{\mathcal{HLE}}$ factor through the equivalence \equiv , yielding the quotient Grothendieck institution, which by abuse of notation we write as $\mathcal{I}^{\mathcal{HLE}}/\equiv = \langle \mathbf{Sign}^{\mathcal{HLE}}/\equiv, \mathbf{Sen}^{\mathcal{HLE}}, \mathbf{Mod}^{\mathcal{HLE}}, \models^{\mathcal{HLE}} \rangle$.*

When considering e.g. the comorphism going from partial first-order logic $\mathbf{PFOL}^=$ to first-order logic $\mathbf{FOL}^=$, and the composite comorphism going from $\mathbf{PFOL}^=$ to CASL and then to $\mathbf{FOL}^=$, we end up in different comorphisms, which are however related by a comorphism modification. The above identification process in the Grothendieck institution now tells us that it does not matter which way we choose.

Definition 5.6. *Given a lax heterogeneous logical environment $\mathcal{HLE} : \mathcal{G} \rightarrow \mathit{coINS}$, a square consisting of two lax triangles of index morphisms*



is called (weakly) amalgamable, if the following outer square is a (weak) pullback



where the lower right square is a pullback.

\mathcal{HLE} is called lax-quasi-exact, if each pair of arrows $j_1 \xleftarrow{d_1} i \xrightarrow{d_2} j_2$ in \mathcal{G} may

be completed to a weakly amalgamable square of lax triangles $j_1 \xleftarrow{d_1} i \xrightarrow{d_2} j_2$

□

6 A Proof Calculus for Heterogeneous Specifications

We obtain a proof calculus for entailment between heterogeneous specifications and sentences by extending the proof calculus in for structured specifications in Sect. 3, Fig. 1, with the following rules:

$$\begin{array}{ll}
(\text{het-trans}) \frac{SP \vdash \varphi}{\rho(SP) \vdash \alpha(\varphi)} & (\text{het-derive}) \frac{SP \vdash \alpha(\varphi)}{SP|_{\rho}^{\Sigma} \vdash \varphi} \\
(\text{borrowing}) \frac{\rho(SP) \vdash \alpha(\varphi)}{SP \vdash \varphi} & \text{if } \rho \text{ is model-expansive} \\
(\text{Het-snf}) \frac{SP' \vdash \sigma(\alpha(\varphi))}{SP \vdash \varphi} & \text{if } \text{hsnf}(SP) = (SP'|_{\sigma})|_{\rho}^{\Sigma}
\end{array}$$

(where hsnf is snf for the Grothendieck institution) and the calculus for refinements between heterogeneous specifications in Fig. 2 is extended as follows:

$$(\text{Het-Trans}) \frac{SP \rightsquigarrow SP'|_{\rho}^{\Sigma}}{\rho(SP) \rightsquigarrow SP'} \quad (\text{Het-Derive}) \frac{SP \rightsquigarrow SP''}{SP|_{\rho}^{\Sigma} \rightsquigarrow SP'} \quad \text{if } \rho: SP' \longrightarrow SP'' \text{ is a conservative extension}$$

Conservativity of $\rho = (\Phi, \alpha, \beta): SP' \longrightarrow SP''$ means that for each model $M' \in \text{Mod}(SP')$, there is a model $M'' \in \text{Mod}(SP'')$ with $\beta(M'') = M'$.

Theorem 6.1. *For a lax heterogeneous logical environment $\mathcal{HLE}: \mathcal{G} \longrightarrow \text{coINS}$ (with some of the institutions also being logics), the proof calculi for heterogeneous specifications are sound for $\mathcal{I}^{\mathcal{HLE}}/\equiv$. If*

1. \mathcal{HLE} is lax-quasi-exact,
2. all institution comorphisms in \mathcal{HLE} are weakly exact,
3. there is a set \mathcal{L} of institutions in \mathcal{HLE} that come as complete logics,
4. all institutions in \mathcal{L} are quasi-semi-exact,
5. from each institution in \mathcal{HLE} , there is some model-expansive comorphism in \mathcal{HLE} going into some logic in \mathcal{L} ,

then the proof calculus for entailments between heterogeneous specifications and sentences is complete over $\mathcal{I}^{\mathcal{HLE}}/\equiv$. If, moreover, the rule system is extended with a (sound and complete) oracle for conservative extension, then the proof calculus for refinements between heterogeneous specifications is also complete.

The oracle for conservative extensions cannot be resigned (not even in the homogeneous case, see [MAH06]). One crucial achievement here is that, in contrast to Prop. 4.9, we need *neither* cocontinuity *nor* exactness of the comorphisms. Moreover, we need quasi-exactness only for some of the logics; this allows us to include logics which are not quasi-exact, such as CASL. Our proof calculus is related to, but different from and conceptually simpler than the one for heterogeneous development graphs in [Mos02a, Mos05]: it is defined along the structure of heterogeneous structured specifications. A similar proof calculus has been implemented in the heterogeneous tool set HETS [MML07].

7 Final Remarks

Building on the standard approach to structured specifications in an arbitrary institution, we extend it to deal with heterogeneous specifications constructed in a heterogeneous logical environment, formalised as a diagram of institutions with institution comorphisms. The focus in this paper is on the proof systems for consequences of so obtained

structured heterogeneous specifications and for refinements between such specifications. We put forward a modification of the standard proof systems of homogeneous structured specifications that strike a proper balance between compositionality and the need for completeness. This system is then extended to heterogeneous specifications. The key result is the (soundness and) completeness of the system under assumptions considerably milder than those that guarantee completeness of purely compositional calculi.

In order to make the work in this paper practically useful for formal software development with heterogeneous logics, the implementation of heterogeneous specifications and proofs in HETS [MML07, Mos05] needs to be generalised to the lax case (see Sect. 5). It also would be important to generalise the present work to further practically relevant notions of maps between institutions, studied in [GR02]. Future work will apply the presented approach to the heterogeneous logical environment arising from UML (see [CKTW08] for initial promising steps in this direction).

References

- [Bor00] Borzyszkowski, T.: Generalized interpolation in CASL. *Information Processing Letters* 79, 19–24 (2000)
- [Bor02] Borzyszkowski, T.: Logical systems for structured specifications. *Theoretical Computer Science* 286, 197–245 (2002)
- [CKTW08] Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A heterogeneous approach to UML semantics. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) *Concurrency, Graphs and Models*. LNCS, vol. 5065, pp. 383–402. Springer, Heidelberg (2008)
- [CM97] Cerioli, M., Meseguer, J.: I borrow your logic? (transporting logical structures along maps). *Theor. Comput. Sci.* 173(2), 311–347 (1997)
- [CMM13] Codescu, M., Mossakowski, T., Maeder, C.: Checking conservativity with HETS. In: Heckel, R., Milius, S. (eds.) *CALCO 2013*. LNCS, vol. 8089, pp. 315–321. Springer, Heidelberg (2013)
- [CoF04] Mosses, P.D. (ed.): *CASL Reference Manual*. LNCS, vol. 2960. Springer, Heidelberg (2004), <http://www.cofi.info>
- [DF02] Diaconescu, R., Futatsugi, K.: Logical foundations of CafeOBJ. *Theoretical Computer Science* 285, 289–318 (2002)
- [Dia02] Diaconescu, R.: Grothendieck institutions. *J. Applied Categorical Structures* 10, 383–402 (2002)
- [Dia04] Diaconescu, R.: Interpolation in Grothendieck Institutions. *Theoretical Computer Science* 311(1-3), 439–461 (2004)
- [Dia08] Diaconescu, R.: *Institution-independent Model Theory*. Birkhäuser (2008)
- [DM00] Dimitrakos, T., Maibaum, T.: On a generalized modularization theorem. *Information Processing Letters* 74, 65–71 (2000)
- [Fin79] Fine, K.: Failures of the Interpolation Lemma in Quantified Modal Logic. *J. of Symbolic Logic* 44(2), 201–206 (1979)
- [FS88] Fiadeiro, J., Sernadas, A.: Structuring theories on consequence. In: Sannella, D., Tarlecki, A. (eds.) *Abstract Data Types 1987*. LNCS, vol. 332, pp. 44–72. Springer, Heidelberg (1988)
- [GB92] Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. *Journal of the ACM* 39(1), 95–146 (1992)

- [GR02] Goguen, J.A., Rosu, G.: Institution morphisms. *Formal Aspects of Computing* 13(3-5), 274–307 (2002)
- [HST94] Harper, R., Sannella, D., Tarlecki, A.: Structured presentations and logic representations. *Annals of Pure and Applied Logic* 67, 113–160 (1994)
- [Mac98] Mac Lane, S.: *Categories for the Working Mathematician*, 2nd edn. Springer (1998)
- [MAH06] Mossakowski, T., Autexier, S., Hutter, D.: Development graphs – proof management for structured specifications. *J. of Logic and Algebraic Programming* 67(1-2), 114–145 (2006)
- [Mes89] Meseguer, J.: General logics. In: *Logic Colloquium 1987*, pp. 275–329. North Holland (1989)
- [MML07] Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set, HETS. In: Grumberg, O., Huth, M. (eds.) *TACAS 2007*. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)
- [Mos02a] Mossakowski, T.: Comorphism-based Grothendieck logics. In: Diks, K., Rytter, W. (eds.) *MFCS 2002*. LNCS, vol. 2420, pp. 593–604. Springer, Heidelberg (2002)
- [Mos02b] Mossakowski, T.: Relating CASL with other specification languages: the institution level. *Theoretical Computer Science* 286, 367–475 (2002)
- [Mos05] Mossakowski, T.: *Heterogeneous Specification and the Heterogeneous Tool Set*. Habilitation thesis, Universität Bremen (2005)
- [Mos06] Mossakowski, T.: Institutional 2-cells and grothendieck institutions. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) *Algebra, Meaning, and Computation*. LNCS, vol. 4060, pp. 124–149. Springer, Heidelberg (2006)
- [MT09] Mossakowski, T., Tarlecki, A.: Heterogeneous logical environments for distributed specifications. In: Corradini, A., Montanari, U. (eds.) *WADT 2008*. LNCS, vol. 5486, pp. 266–289. Springer, Heidelberg (2009)
- [SB83] Sannella, D., Burstall, R.: Structured theories in LCF. In: Ausiello, G., Protasi, M. (eds.) *CAAP 1983*. LNCS, vol. 159, pp. 377–391. Springer, Heidelberg (1983)
- [Sho67] Shoenfield, J.: *Mathematical Logic*. Addison-Wesley (1967)
- [ST88a] Sannella, D., Tarlecki, A.: Specifications in an arbitrary institution. *Information and Computation* 76, 165–210 (1988)
- [ST88b] Sannella, D., Tarlecki, A.: Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica* 25, 233–281 (1988)
- [ST12] Sannella, D., Tarlecki, A.: *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer (2012)
- [ST13] Sannella, D., Tarlecki, A.: Property-oriented semantics of structured specifications. In: *Mathematical Structures in Computer Science* (2013)
- [Tar00] Tarlecki, A.: Towards heterogeneous specifications. In: Gabbay, D., de Rijke, M. (eds.) *Frontiers of Combining Systems 2*, *Studies in Logic and Computation*, pp. 337–360. Research Studies Press (2000)

Author Index

- Adámek, Jiří 366
Almagor, Shaull 226
Antonopoulos, Timos 411
Avni, Guy 119
- Bertrand, Nathalie 29, 134
Bonchi, Filippo 351
Bonnet, Rémi 43
Bonsangue, Marcello 381
- Carraro, Alberto 103
Chatterjee, Krishnendu 210, 242
Cîrstea, Corina 426
Cleaveland, Rance 304
- D'Argenio, Pedro R. 289
Desel, Jörg 258
Doyen, Laurent 58, 210, 242
- Erbatur, Serdar 274
Esparza, Javier 258
- Fabre, Éric 29
Ferlez, James 304
Fournier, Paulin 134
Fu, Hongfei 73
- Gebler, Daniel 289
Gimbert, Hugo 210
Gorogiannis, Nikos 411
Guerrieri, Giulio 103
- Haar, Stefan 29
Haase, Christoph 411
Haddad, Serge 29
Hélouët, Loïc 29
Hennessy, Matthew 320
- Inaba, Kazuhiro 149
- Jančar, Petr 1
- Kanovich, Max 411
Kapur, Deepak 274
Kiefer, Stefan 43
- Kobayashi, Naoki 149, 180
Koutavas, Vasileios 320
Kupferman, Orna 119, 226
- Lang, Martin 195
Lee, Matias David 289
Lin, Anthony Widjaja 43
- Marcus, Steve 304
Marshall, Andrew M. 274
Massart, Thierry 58
Meadows, Catherine 274
Milius, Stefan 366
Mio, Matteo 335
Mossakowski, Till 441
Munch-Maccagnoni, Guillaume 396
Murawski, Andrzej S. 164
Myers, Robert S.R. 366
- Nain, Sumit 242
Narendran, Paliath 274
- Ouaknine, Joël 411
Oualhadj, Youssouf 210
- Padovani, Luca 88
- Ringeissen, Christophe 274
Rot, Jurriaan 381
- Sangnier, Arnaud 134
Shirmohammadi, Mahsa 58
Sobociński, Paweł 351
Spaccasassi, Carlo 320
- Tamir, Tami 119
Tarlecki, Andrzej 441
Tsukada, Takeshi 149, 180
Tzevelekos, Nikos 164
- Urbat, Henning 366
- Vardi, Moshe Y. 242
- Zanasi, Fabio 351