Roberto Amadio (Ed.)

LNCS 4962

# Foundations of Software Science and Computational Structures

11th International Conference, FOSSACS 2008
Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008
Budapest, Hungary, March/April 2008, Proceedings


Springer

Lecture Notes in Computer Science 4962

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

Roberto Amadio (Ed.)

# Foundations of Software Science and Computational Structures

11th International Conference, FOSSACS 2008
Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008
Budapest, Hungary, March 29 – April 6, 2008
Proceedings

Springer

Volume Editor

Roberto Amadio
Université Paris 7, PPS, Case 7014
75205 Paris Cedex 13, France
E-mail: Roberto.Amadio@pps.jussieu.fr

Library of Congress Control Number: 2008922352

CR Subject Classification (1998): F.3, F.4.2, F.1.1, D.3.3-4, D.2.1

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

ISSN 0302-9743
ISBN-10 3-540-78497-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-78497-5 Springer Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12235414 06/3180 5 4 3 2 1 0

# Foreword

ETAPS 2008 was the 11th instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference that was established in 1998 by combining a number of existing and new conferences. This year it comprised five conferences (CC, ESOP, FASE, FOSSACS, TACAS), 22 satellite workshops (ACCAT, AVIS, Bytecode, CMCS, COCV, DCC, FESCA, FIT, FORMED, GaLoP, GT-VMT, LDTA, MBT, MOMPES, PDMC, QAPL, RV, SafeCert, SC, SLA++P, WGT, and WRLA), nine tutorials, and seven invited lectures (excluding those that were specific to the satellite events). The five main conferences received 571 submissions, 147 of which were accepted, giving an overall acceptance rate of less than 26%, with each conference below 27%. Congratulations therefore to all the authors who made it to the final programme! I hope that most of the other authors will still have found a way of participating in this exciting event, and that you will all continue submitting to ETAPS and contributing to make of it the best conference in the area.

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

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

ETAPS 2008 was organized by the John von Neumann Computer Society jointly with the Budapest University of Technology and the Eötvös University, in cooperation with:

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

and with support from Microsoft Research and Danubius Hotels.

The organizing team comprised:

| | |
|---|---|
| Chair | Dániel Varró |
| Director of Organization | István Alföldi |
| Main Organizers | Andrea Tósoky, Gabriella Aranyos |
| Publicity | Joost-Pieter Katoen |
| Advisors | András Pataricza, Joaõ Saraiva |
| Satellite Events | Zoltán Horváth, Tihamér Levendovszky, Viktória Zsók |
| Tutorials | László Lengyel |
| Web Site | Ákos Horváth |
| Registration System | Victor Francisco Fonte, Zsolt Berényi, Róbert Kereskényi, Zoltán Fodor |
| Computer Support | Áron Sisak |
| Local Arrangements | László Gönczy, Gábor Huszerl, Melinda Magyar, several student volunteers. |

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

Vladimiro Sassone (Southampton, Chair), Luca de Alfaro (Santa Cruz), Roberto Amadio (Paris), Giuseppe Castagna (Paris), Marsha Chechik (Toronto), Sophia Drossopoulou (London), Matt Dwyer (Nebraska), Hartmut Ehrig (Berlin), Chris Hankin (London), Laurie Hendren (McGill), Mike Hinchey (NASA Goddard), Paola Inverardi (L'Aquila), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Kim Larsen (Aalborg), Gerald Luettgen (York) Tiziana Margaria (Göttingen), Ugo Montanari (Pisa), Martin Odersky (Lausanne), Catuscia Palamidessi (Paris), Anna Philippou (Cyprus), CR Ramakrishnan (Stony Brook), Don Sannella (Edinburgh), João Saraiva (Minho), Michael Schwartzbach (Aarhus), Helmut Seidl (Munich), Perdita Stevens (Edinburgh), and Dániel Varró (Budapest).

I would like to express my sincere gratitude to all of these people and organizations, the Programme Committee Chairs and members of the ETAPS conferences, the organizers of the satellite events, the speakers themselves, the many reviewers, and Springer for agreeing to publish the ETAPS proceedings. Finally, I would like to thank the Organizing Chair of ETAPS 2008, Dániel Varró, for arranging for us to have ETAPS in the most beautiful city of Budapest

January 2008 Vladimiro Sassone

# Preface

The present volume contains the proceedings of the 11th international conference on the Foundations of Software Science and Computations Structures (FOSSACS) 2008, held in Budapest, Hungary, April 2–4, 2008. FOSSACS is an event of the *Joint European Conferences on Theory and Practice of Software* (ETAPS). The previous ten FOSSACS conferences took place in Lisbon (1998), Amsterdam (1999), Berlin (2000), Genoa (2001), Grenoble (2002), Warsaw (2003), Barcelona (2004), Edinburgh (2005), Vienna (2006), and Braga (2007).

FOSSACS presents original papers on foundational research with a clear significance to software science. The Programme Committee invited papers on theories and methods to support analysis, synthesis, transformation and verification of programs and software systems. In particular, we identified the following topics: algebraic models, automata and language theory, behavioral equivalences, categorical models, computation processes over discrete and continuous data, infinite state systems, computation structures, logics of programs, modal, spatial, and temporal logics, models of concurrent, reactive, distributed, and mobile systems, process algebras and calculi, semantics of programming languages, software specification and refinement, type systems and type theory, fundamentals of security, semi-structured data, program correctness and verification.

We ultimately received 124 submissions. This proceedings volume consists of the abstract of an invited talk by Igor Walukiewicz together with 33 contributed papers. The contributed papers were selected for publication by the Programme Committee during a two-week electronic discussion. We sincerely thank all the authors of papers submitted to FOSSACS 2008; we were pleased indeed by the number and quality of the submissions. Moreover, we would like to thank the members of the Programme Committee for their excellent job during the selection process. Clearly, all this would not have been possible without the valuable and detailed reports provided by the sub-reviewers. Also, through the phases of submission, evaluation, and production of the proceedings we relied on the invaluable assistance of the EasyChair system.

Last but not least, we would also like to thank the ETAPS 2008 Organizing Committee chaired by Dániel Varró and the ETAPS Steering Committee chaired by Vladimiro Sassone for their efficient coordination of all the activities leading up to FOSSACS 2008.

January 2008 Roberto M. Amadio

# Organization

## Programme Chair

Roberto Amadio

## Programme Committee

Luca Aceto, Reykjavik University
Roberto Amadio (Chair), Paris Diderot University
Lars Birkedal, Copenhagen IT University
Roberto Bruni, Pisa University
Hubert Comon, ENS Cachan
Thierry Coquand, Göteborg University
Zoltan Esik, Szeged University
Dan Ghica, Birmingham University
Jürgen Giesl, RWTH Aachen
Martin Hofmann, Munich University
Radha Jagadeesan, DePaul University
Petr Jančar, Ostrava Technical University
Leonid Libkin, Edinburgh University
Dale Miller, INRIA Saclay
Eugenio Moggi, Genoa University
Anca Muscholl, LABRI, Bordeaux
Vincent van Oostrom, Utrecht University
Prakash Panangaden, McGill University
Jean-François Raskin, Brussels Free University
David Sands, Göteborg University
Colin Stirling, Edinburgh University
Pawel Urzyczyn, Warsaw University
Thomas Wilke, Kiel University
Nobuko Yoshida, Imperial College, London

## External Reviewers

Parosh Abdulla
Andreas Abel
Samson Abramsky
Lucia Acciai
Rajeev Alur
David Baelde
Christel Baier
Adam Bakewell
Paolo Baldan
Pablo Barcelo
Miklos Bartha
Marcin Benke

Martin Berger
Lennart Beringer
Marco Bernardo
Nathalie Bertrand
Dietmar Berwanger
Bodil Biering
Stephen L. Bloom
Benedikt Bollig
Filippo Bonchi
Michele Boreale
Patricia Bouyer
Tomas Brazdil
Thomas Brihaye
Vaclav Brozek
Véronique Bruyère
Mikkel Bundgaard
Marzia Buscemi
Marco Carbone
Koen Claessen
David Clark
Brendan Cordy
Flavio Corradini
Silvia Crafa
Deepak D'Souza
Troels Damgaard
Søren Debois
Stéphanie Delaune
Yannick Delbecque
Stéphane Demri
Josee Desharnais
Raymond Devillers
Volker Diekert
Ernst-Erich Doberkat
Daniel Dougherty
Laurent Doyen
Ebbe Elsborg
Joerg Endrullis
Javier Esparza
Kousha Etessami
Alain Finkel
Eric Fabre
John Fearnley
Andrzej Filinski
Wan Fokkink
Vojtech Forejt
Cédric Fournet
Adrian Francalanza
Oliver Friedmann
Murdoch Gabbay
Andrew Gacek
Fabio Gadducci
Tjalling Gelsema
Thomas Genet
Hugo Gimbert
Arne Glenstrup
Andrew Gordon
Daniele Gorla
Clemens Grabmayer
Hermann Gruber
Erich Grädel
Stefano Guerrini
Vineet Gupta
Ichiro Hasuo
Dimitri Hendriks
Thomas Hildebrandt
Jane Hillston
Jan Holeček
Clement Houtmann
Helle Hvid Hansen
Szabolcs Ivan
Florent Jacquemard
Alan Jeffrey
Joost-Pieter Katoen
Tomasz Kazana
Klaus Keimel
Carsten Kern
Claude Kirchner
Vladimir Klebanov
Jetty Kleijn
Bartek Klin
Bartosz Klin
Eryk Kopczynski
Vaclav Koubek
Steve Kremer
Antonin Kucera
Ralf Kuesters
Alexander Kurz
Detlef Köhler
Barbara König
Anna Labella

Ivan Lanese
Martin Lange
Francois Laroussinie
Sławomir Lasota
James J. Leifer
Stéphane Lengrand
Marina Lenisa
Giacomo Lenzi
Jérôme Leroux
Michael Leuschel
Paul Levy
Alberto Lluch-Lafuente
Christof Loeding
Markus Lohrey
Hans-Wolfgang Loidl
Etienne Lozes
Christof Löding
Sergio Maffeis
Patrick Maier
Luc Maranget
Carbone Marco
Radu Mardare
Nicolas Markey
Andrea Masini
Thierry Massart
Hernan Melgratti
Massimo Merro
Marino Miculan
Michael Mislove
Faron Moller
David Monniaux
Larry Moss
Andrzej Murawski
Anca Muscholl
Rasmus Møgelberg
Gopalan Nadathur
Sebastian Nanz
Uwe Nestmann
Linh Anh Nguyen
Damian Niwinski
Thomas Noll
Ulf Norell
Gethin Norman
David Nowak
Russell O'Connor
Jan Obdrzalek
Luke Ong
Karol Ostrovsky
Joel Ouaknine
Iain Phillips
Nir Piterman
Adam Poswolsky
Damien Pous
John Power
K.V.S. Prasad
R. Ramanujam
Julian Rathke
Pierre-Alain Reynier
Jussi Rintanen
Eike Ritter
Enric Rodríguez-Carbonell
Philipp Ruemmer
Michal Rutkowski
Alexis Saurin
Philippe Schnoebelen
Pierre-Yves Schobbens
Aleksy Schubert
Roberto Segala
Sebastian Seibert
Damien Sereni
Pawel Sobocinski
Jiri Srba
Ian Stark
Sam Staton
Lutz Strassburger
Jan Strejcek
Grégoire Sutre
Josef Svenningsson
Morten Heine Sørensen
Jonathan Taylor
Hayo Thielecke
René Thiemann
Sophie Tison
Alwen Tiu
Richard Trefler
Ralf Treinen
Tomasz Truderung
Theodoros Tsokos
Emilio Tuosto
Irek Ulidowski

Michael Ummels
Christian Urban
Sandor Vagvolgyi
Frank Valencia
Laurent Van Begin
Daniele Varacca
Björn Victor
Maria Vigliotti
Emanuele Viola
Janis Voigtlaender
David Wahlstedt
Volker Weber
Anthony Widjaja To
James Worrell
Francesco Zappa Nardelli
Gianluigi Zavattaro
Wieslaw Zielonka
Elena Zucca
Vojtěch Řehák
Franck van Breugel

# Table of Contents

# Finding Your Way in a Forest: On Different Types of Trees and Their Properties

Igor Walukiewicz*

CNRS LaBRI
Université de Bordeaux
351, Cours de la Libération
33405 Talence, France

We see trees in almost any part of computer science. Traditionally, ranked trees, that are nothing else but terms, captured most attention, although exceptions could have been found in graph theory or linguistics [9]. Recently unranked trees are a subject of renewed interest, mainly because of the developement of XML [22]. It is also quite comon nowadays to see trees with infinite paths, especailly in the context of verification. We will omit this aspect, as for the questions we want to discuss finite trees are sufficiently interesting. We prefer instead to make distinction between ordered and unordered trees, i.e., distinguish situations when syblings are ordered or not. Thus we will deal with four types of trees depending on two parameters: ranked/unranked, and ordered/unordered.

There are even more formalisms to describe tree properties than there are tree types. Here, the main reference point for us will be monadic second-order logic which captures recognizable sets of trees. This logic has a binary predicate interpreted as a descendant relation in a tree and a monadic predicate for every possible node label. If models are ordered trees then the logic may also have another binary predicate interpreted as the sybling order. Important logics are obtained by restricting the range of quantification: when quantifying only over elements we obtain first-order logic (FOL), when quantifying over chians (sets where every pair of nodes is in a descendant relation) we get chain logic, finally antichain logic is obtained when quatifiers range over sets where no two elements are in the descendant relation. Appart from these classical logics we have important variants of modal and temporal logics over trees: CTL, CTL*, PDL, the $\mu$-calculus.

Given this multitude of formalisms, the first question one can ask is to compare their expressive power, i.e., establish if all properties expressible in a logic A can be also expressed in a logic B. We know the answers to this kind of questions for the formalism listed above. Figure 1 presents the case of ranked ordered trees. Some of the presented inclusions are nontrivial. For insatance, the inclusions of the MSOL in the $\mu$-calculus [18] or FOL in CTL* [14] show that one can obtain the same expressive power using very different means. For other types of trees the picture is not the same. For example, if we consider ordered unranked

* Work supported by project DOTS (ANR-06-SETI-003).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 1–4, 2008.

$$\begin{array}{ccccccc} & & \mu\text{-calculus} & = & \text{MSOL} & = & \text{antichain logic} \\ & & \cup\!\!\!\!\!\!\neq & & \cup\!\!\!\!\!\!\neq & & \cup\!\!\!\!\!\!\neq \\ \text{CTL} & \subsetneq & \text{CTL}^* & = & \text{FOL} & \subsetneq & \text{chain logic} \end{array}$$

**Fig. 1.** Relations between logics over ordered binary trees

trees, all equalities on the left change to strict inclusions, and chain logic becomes incomparable with antichaing logic.

While diagrams as that on Figure 1 are important, they are far from giving a complete explanation of the expressive power of the logics in question. Consider the following situation. Separating FOL from MSOL over ordered binary trees is easy. We know that over words FOL cannot express counting modulo properties [20]. For example, "even length" is not first order definable. A short argument shows that the language of trees with the leftmost path of even length is not FOL expressible. This observation though, does not tell much about what properties are expressible in FOL. Take the language of ordered binary trees "the depths of all leaves are even". Somehow surprisingly, it turns out that this language is FOL-expressible [19]. Indeed it is a major open question to find an algorithm deciding if a given regular tree language is experssible in FOL. At present decidable characterisations are known for very few fragments of MSOL [1,7,4,8].

Apart from the mathematical curiosity, there is a growing number of reasons for looking closer at tree formalisms. In the context of XML, the aspect of data (infnite set of labels with some operations on them) is important. It is rather difficult to come with a decidable nontrivial formalism [6,16]. Understanding well expressive power in the case without data can help substantially.

Another reason is the study of order invariance. A property is order invariant if does not distingush two trees that differ only in the order of syblings. It is natural to ask if such a property can be expressed without referring to this order. Over unranked trees, order invariant properties are exactly those expressible in MSOL extended with counting modulo quantifiers [11]. The language "the depths of all leaves are even" described above cannot be defined in FOL without a sybling order. So the situation is much less clear for FOL. Curiously enough if we restrict to FOL[succ] where we allow only successor relation in place of descendant relation then all order invariant FOL[succ] properites can be expressed in FOL[succ] [2].

Finally, there is a question of automata and grammars for trees. In the literature one can find many proposals of differen automata on trees. Even for ordered binary trees we have for example several versions of tree-walking automata [12,5,13]. For other types of trees the number of variants is even bigger [10]. The similar situation is with regular expressions for trees [21,15,17,3]. Better understanding of logical formalisms is indenspensable to classify and clarify all these notions.

In this talk we will survey some recent results in the field. We will start with a unifying presentation of formalisms discussed above. For this a small detour to algebra will be useful [8]. We will present dependencies between different

formalisms, and known decidability results. Some less common questions as order invariance, or existance of a finite base will be also considered. The talk will present joint work with Mikolaj Bojańczyk.

## References


1. Benedikt, M., Segoufin, L.: Regular languages definable in FO. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 327–339. Springer, Heidelberg (2005)
2. Benedikt, M., Segoufin, L.: Towards a characterization of order-invariant queries over tame structures. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 276–291. Springer, Heidelberg (2005)
3. Bojanczyk, M.: Forest expressions. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 146–160. Springer, Heidelberg (2007)
4. Bojanczyk, M.: Two-way unary temporal logic over trees. In: LICS 2007, pp. 121–130 (2007)
5. Bojanczyk, M., Colcombet, T.: Tree-walking automata do not recognize all regular languages. In: STOC 2005, pp. 234–243. ACM, New York (2005)
6. Bojanczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: PODS 2006, pp. 10–19. ACM, New York (2006)
7. Bojanczyk, M., Walukiewicz, I.: Characterizing EF and EX tree logics. Theoretical Computer Science 358(2–3), 255–272 (2006)
8. Bojanczyk, M., Walukiewicz, I.: Forest algebras. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata. Texts in Logic and Games, vol. 2, pp. 107–132. Amsterdam University Press (2007)
9. Carpenter, B.: The Logic of Typed Future Structures. Cambridge University Press, Cambridge (1992)
10. Comon, H., Dauchet, M., Gilleron, R., Lugiez, F.J.D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2002), http://www.grappa.univ-lille3.fr/tata/
11. Courcelle, B.: The monadic second-order logic of graphs V: On closing the gap between definability and recognizability. Theor. Comput. Sci. 80(2), 153–202 (1991)
12. Engelfriet, J., Hoogeboom, H.J.: Tree–walking pebble automata. In: Karhumäki, J., et al. (eds.) Jewels are forever, pp. 72–83. Springer, Heidelberg (1999)
13. Engelfriet, J., Hoogeboom, H.J., Samwel, B.: XML transformation by tree-walking transducers with invisible pebbles. In: PODS 2007, pp. 63–72. ACM, New York (2007)
14. Hafer, T., Thomas, W.: Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 269–279. Springer, Heidelberg (1987)
15. Heuter, U.: First-order properties of trees, star-free expressions, and aperiodicity. In: Cori, R., Wirsing, M. (eds.) STACS 1988. LNCS, vol. 294, pp. 136–148. Springer, Heidelberg (1988)
16. Jurdzinski, M., Lazic, R.: Alternation-free modal mu-calculus for data trees. In: LICS 2007, pp. 131–140. IEEE Computer Society Press, Los Alamitos (2007)
17. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Trans. Database Syst. 31(3), 770–813 (2006)
18. Niwiński, D.: Fixed points vs. infinite generation. In: LICS 1988, pp. 402–409 (1988)

19. Potthoff, A.: First-order logic on finite trees. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) TAPSOFT 1995. LNCS, vol. 915, pp. 125–139. Springer, Heidelberg (1995)
20. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control 8, 190–194 (1965)
21. Thomas, W.: Logical aspects in the study of tree languages. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 31–50. Springer, Heidelberg (1984)
22. Vianu, V.: A web odyssey: From CODD to XML. In: PODS 2001, ACM, New York (2001)

# Simple Stochastic Games with Few Random Vertices Are Easy to Solve*

Hugo Gimbert[1] and Florian Horn[2]

[1] LaBRI, Université Bordeaux 1, France
hugo.gimbert@labri.fr
[2] LIAFA, Université Paris 7, France
florian.horn@liafa.jussieu.fr

**Abstract.** We present a new algorithm for solving Simple Stochastic Games (SSGs). This algorithm is based on an exhaustive search of a special kind of positional optimal strategies, the *f-strategies*. The running time is $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$, where $|V|, |V_{\mathrm{R}}|, |E|$ and $|p|$ are respectively the number of vertices, random vertices and edges, and the maximum bit-length of a transition probability.

Our algorithm improves existing algorithms for solving SSGs in three aspects. First, our algorithm performs well on SSGs with few random vertices, second it does not rely on linear or quadratic programming, third it applies to all SSGs, not only stopping SSGs.

## 1 Introduction

Simple Stochastic Games (SSGs for short) are played by two players Max and Min in a sequence of steps. Players move a pebble along edges of a directed graph $(V, E)$. There are three type of vertices: $V_{\mathrm{Max}}$ is the set of vertices of player Max, $V_{\mathrm{Min}}$ the set of vertices of player Min and $V_{\mathrm{R}}$ the set of random vertices. When the pebble is on a vertex of $V_{\mathrm{Max}}$ or $V_{\mathrm{Min}}$, the corresponding player chooses an outgoing edge and moves the pebble along it. When the pebble is on a random vertex, the successor is chosen randomly according to some fixed probability distribution: from vertex $v \in V_{\mathrm{R}}$ the pebble moves towards vertex $w \in V$ with some probability $p(w|v)$ and the probability that the game stops is 0, *i.e.* $\sum_{w \in V} p(w|v) = 1$. An SSG is depicted on Figure 1, with vertices of $V_{\mathrm{Max}}$ represented as $\bigcirc$, vertices of $V_{\mathrm{Min}}$ represented as $\square$, and vertices of $V_{\mathrm{R}}$ represented as $\lozenge$.

Player Max and Min have opposite goals, indeed player Max wants the pebble to reach a special vertex $t \in V$ called the *target vertex*, if this happens *the play is won by player* Max. In the opposite case, the play proceeds forever without reaching $t$ and is *won by player* Min. For technical reasons, we assume that $t$ is a vertex of player Max and is absorbing. *Strategies* are used by players to choose

* This research was partially supported by french project ANR "DOTS".

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 5–19, 2008.

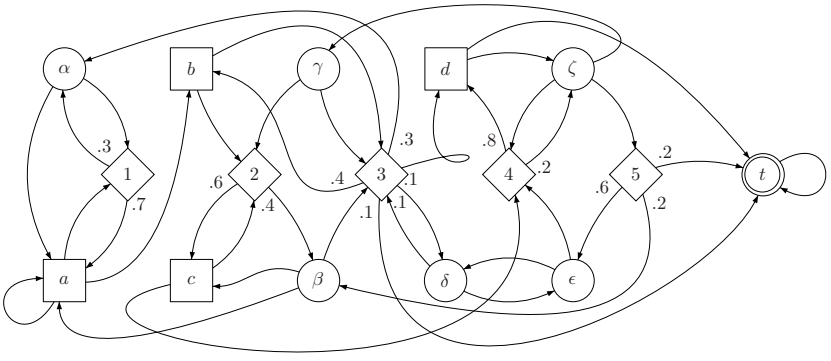



**Fig. 1.** A Simple Stochastic Game

their moves, a strategy tells where to move the pebble depending on the sequence of previous vertices, *i.e.* the finite path followed by the pebble from the beginning of the play. The *value* of a vertex $v$ is the maximal probability with which player Max can enforce the play to reach the target vertex. When player Max, respectively player Min, uses an *optimal strategy* he ensures reaching the target with a probability greater, respectively smaller, than the value of the initial vertex. Notions of value and optimal strategies are formally defined in Section 2.

We are interesting in *solving* SSGs, that is computing values and optimal strategies.

*Applications.* SSGs are a natural model of reactive systems, and algorithms for solving SSGs may be used for synthesizing controllers of such systems. For example an hardware component may be modelled as an SSG whose vertices are the global states of the component, the target is some error state to avoid, random states are used to model failure probabilities and stochastic behaviours of the environment, choices of player Min corresponds to actions available to the software driver and player Max corresponds to non-deterministic behaviour of the environment. Then an optimal strategy for player Min in the SSG will correspond to a software driver minimizing the probability to enter the error state, whatever be the behaviour of the environment.

*Existing algorithms for solving SSGs.* The complexity of solving SSGs was first considered by Condon [Con92], who proved that deciding whether the value of an SSG is greater than $\frac{1}{2}$ is in NP $\cap$ co-NP. The algorithm provided in [Con92] consists in first transforming the input SSG in a *stopping SSG* where the probability to reach a sink vertex is 1. The transformation keeps unchanged the fact that the initial vertex has value strictly greater than $\frac{1}{2}$ but induces a quadratic blowup of the size of the SSG. The algorithm then non-deterministically guesses the values of vertices, which are rational numbers of linear size, and checks that these values are the unique solutions of some *local optimality equations*.

Three other kinds of algorithms for solving SSGs are presented in [Con93]. These algorithms require transformation of the initial SSG into an equivalent

*stopping* SSG and are based on local optimality equations. First algorithm computes values of vertices using a quadratic program with linear constraints. Second algorithm computes iteratively from below the values of the SSGs, and the third is a strategy improvement algorithm *à la* Hoffman-Karp. These two last algorithms require solving an exponential number of linear programs, as it is the case for the algorithm recently proposed in [Som05].

Finally, these four algorithms suffer three main drawbacks.

First, these algorithms rely on solving either an exponential number of linear programs or a quadratic program, which may have prohibitive algorithmic cost and makes the implementation tedious.

Second, these algorithms only apply to the special class of *stopping* SSGs. Although it is possible to transform any SSG into a stopping SSG with arbitrarily small change of values, computing exact values this way requires to modify drastically the original SSG, introducing either $|V|^2$ new random vertices or new transition probabilities of bit-length quadratic in the original bit-length. This also makes the implementation tedious.

Third, the running time of these algorithms may *a priori* be exponential whatever be the number of random vertices of the input SSG, including the case of SSGs with no random vertices at all, also known as reachability games on graphs. However it is well-known that reachability games on graphs are solvable in quadratic time.

Notice that randomized algorithms do not perform much better since the best randomized algorithms [Lud95, Hal07] known so far run in sub-exponential expected time $e^{O(\sqrt{n})}$.

*Our results.* In this paper we present an algorithm that computes values and optimal strategies of an SSG in time $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$, where $|V_{\mathrm{R}}|$ is the number of random vertices, $|V|$ is the number of vertices and $|p|$ is the maximal bit-length of transition probabilities.

The key point of our algorithm is the fact that optimal strategies may be looked for in a strict subset of positional strategies, called the class of **f**-strategies. The **f**-strategies are in correspondence with permutations of random vertices. Our algorithm does an exhaustive search of optimal **f**-strategies among the $|V_{\mathrm{R}}|!$ available ones and check their optimality. Optimality is easy to check, it consists in computing a reachability probability in a Markov Chain with $V_{\mathrm{R}}$ states, which amounts to solving a linear system with at most $|V_{\mathrm{R}}|$ equations.

*Comparison with existing work.* We improve existing results by three aspects: our algorithm performs better on SSGs with few random vertices, it is arguably much simpler, and we provide new insight about the structure of optimal strategies.

Our algorithm performs much better on SSGs with few random vertices than previously known algorithms. Indeed, its complexity is $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$, hence when there are no random vertices at all, our algorithm matches the usual quadratic complexity for solving reachability games on graphs. When the number of random vertices is fixed, our algorithm performs in polynomial time, and on the class of SSGs such that $|V_{\mathrm{R}}| \leq \sqrt{|V_{\mathrm{Max}}| + |V_{\mathrm{Min}}|}$ our algorithm is sub-exponential.

Our algorithm is arguably simpler than previously known algorithms. Indeed, it does not require use of linear or quadratic programming. Although linear programs can be solved in polynomial time [Kac79, Ren88], this requires high-precision arithmetic. By contrast, our algorithm is very elementary: it enumerates permutations of the random vertices and for each permutation, it solves a linear system of equations.

Our algorithm is also simpler because it applies directly to any kind of SSGs, whereas previously known algorithms require the transformation of the input SSG into a stopping SSG of quadratic size.

*Plan.* The paper is organised as follows. In the first section, we introduce formally SSGs, values and optimal strategies. In the second section, we present the notion of **f**-strategies. In the third section, we focus on two properties of **f**-strategies: self-consistency and progressiveness. We prove that **f**-strategies that are both self-consistent and progressive are also optimal, and we prove the existence of such strategies. In the fourth section we describe our algorithm for solving SSGs.

Omitted proofs can be found in the full version of the paper [GH07].

## 2 Simple Stochastic Games

In this section we give formal definitions of an SSG, values and optimal strategies.

An SSG is a tuple $(V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_{\mathrm{R}}, E, t, p)$, where $(V, E)$ is a graph, $(V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_{\mathrm{R}})$ is partition of $V$, $t \in V$ is the target vertex and for every $v \in V_{\mathrm{R}}$ and $w \in V$, $p(w|v)$ is the transition probability from $v$ to $w$, with the property $\sum_{w \in V} p(w|v) = 1$.

A *play* is an infinite sequence $v_0 v_1 \cdots \in V^\omega$ of vertices such that if $v_n \in (V_{\mathrm{Max}} \cup V_{\mathrm{Min}})$ then $(v_n, v_{n+1}) \in E$ and if $v_n \in V_{\mathrm{R}}$ then $p(v_{n+1}|v_n) > 0$. A play is *won by* Max if it visits the target vertex; otherwise the play is won by Min. A *finite play* is a finite prefix of a play.

A *strategy* for player Max is a mapping $\sigma : V^* V_{\mathrm{Max}} \to V$ such that for each finite play $h = v_0 \dots v_n$ such that $v_n \in V_{\mathrm{Max}}$, we have $(v_n, \sigma(h)) \in E$. A play $v_0 v_1 \cdots$ is *consistent with* $\sigma$ if for every $n$, if $v_n \in V_{\mathrm{Max}}$ then $v_{n+1}$ is $\sigma(v_0 \cdots v_n)$. A strategy for player Min is defined similarly, and is generally denoted $\tau$.

Once the initial vertex $v$ and two strategies $\sigma, \tau$ for player Max and Min are fixed, we can measure the probability that a given set of plays occurs. This probability measure is denoted $\mathbb{P}_v^{\sigma,\tau}$. For every $n \in \mathbb{N}$, we denote by $V_n$ the random variable defined by $V_n(v_0 v_1 \cdots) = v_n$, the set of plays is equipped with the $\sigma$-algebra generated by random variables $(V_n)_{n \in \mathbb{N}}$. Then there exists a probability measure $\mathbb{P}_v^{\sigma,\tau}$ with the following properties:

$$\mathbb{P}_v^{\sigma,\tau}(V_0 = v) = 1 \tag{1}$$

$$\mathbb{P}_v^{\sigma,\tau}(V_{n+1} = \sigma(V_0 \cdots V_n) \mid V_n \in V_{\mathrm{Max}}) = 1, \tag{2}$$

$$\mathbb{P}_v^{\sigma,\tau}(V_{n+1} = \tau(V_0 \cdots V_n) \mid V_n \in V_{\mathrm{Min}}) = 1, \tag{3}$$

$$\mathbb{P}_v^{\sigma,\tau}(V_{n+1} \mid V_n \in V_{\mathrm{R}}) = p(V_{n+1}|V_n). \tag{4}$$

Expectation of a real-valued, measurable and bounded function $\phi$ under $\mathbb{P}_v^{\sigma,\tau}$ is denoted $\mathbb{E}_v^{\sigma,\tau}[\phi]$. We will often use implicitly the following formula, which

gives the expectation of $\phi$ once a finite prefix $h = v_0 v_1 \cdots v_n$ of the play is fixed:

$$\mathbb{E}_v^{\sigma,\tau}\left[\ \phi \mid V_0 \cdots V_n = h\right] = \mathbb{E}_{v_n}^{\sigma[h],\tau[h]}\left[\ \phi[h]\ \right], \tag{5}$$

where $\sigma[h](w_0 w_1 w_2 \cdots) = \sigma(v_0 \cdots v_n w_1 w_2 \cdots)$ and $\tau[h]$ and $\phi[h]$ are defined similarly.

*Values and positional optimal strategies.* The goal of player Max is to reach the target vertex $t$ with the highest probability possible, whereas player Min has the opposite goal. Given a starting vertex $v$ and a strategy $\sigma$ for player Max, whatever strategy $\tau$ is chosen by Min, the target vertex $t$ will be reached with probability at least:

$$\inf_\tau \mathbb{P}_v^{\sigma,\tau}\left(\mathrm{Reach}(t)\right),$$

where $\mathrm{Reach}(t)$ is the event $\{\exists n \in \mathbb{N}, V_n = t\}$. Thus, starting from $v$, player Max can ensure to win the game with probability arbitrarily close to:

$$\mathrm{val}_*(v) = \sup_\sigma \inf_\tau \mathbb{P}_v^{\sigma,\tau}\left(\mathrm{Reach}(t)\right),$$

and symmetrically, player Min can ensure that player Max cannot win with a probability much higher than:

$$\mathrm{val}^*(v) = \inf_\tau \sup_\sigma \mathbb{P}_v^{\sigma,\tau}\left(\mathrm{Reach}(t)\right).$$

Clearly $\mathrm{val}_*(v) \leq \mathrm{val}^*(v)$. In fact these values are equal, and this common value is called the value of vertex $v$ and denoted $\mathrm{val}(v)$. A much stronger result is known about SSGs: the infimum and supremum used in the definition of $\mathrm{val}(v)$ are attained for some strategies called *optimal strategies*. Moreover, there exists optimal strategies of a simple kind, called *positional strategies*. A strategy $\sigma$ is said to be positional if it depends only on the current vertex, *i.e.* for every finite play $v_0 \cdots v_n$ $\sigma(v_0 \cdots v_n) = \sigma(v_n)$. The following results are well-known [Sha53, Con92].

**Theorem 1.** *In any SSG, for every vertex $v \in V$, the values* $\mathrm{val}_*(v)$ *and* $\mathrm{val}^*(v)$ *are equal. This common value is called the value of vertex $v$ and denoted* $\mathrm{val}(v)$. *There exists strategies $\sigma^\#$ and $\tau^\#$ that are* optimal *i.e. for every vertex $v$ and every strategies $\sigma, \tau$:*

$$\mathbb{P}_v^{\sigma,\tau^\#}\left(\mathrm{Reach}(t)\right) \leq \mathrm{val}(v) \leq \mathbb{P}_v^{\sigma^\#,\tau}\left(\mathrm{Reach}(t)\right).$$

*Moreover there exists strategies that are both optimal and positional.*

## 3 Playing with f-Strategies

Existence of positional optimal strategies is a key property of SSGs, used for designing all known algorithms solving SSGs. The algorithm we propose relies on a refinement of this result, we will prove that optimal strategies can be looked for in a strict subset of positional strategies called the set of **f**-strategies.

In this section, we describe what are **f**-strategies.

### 3.1 Informal Description of f-Strategies

With every permutation $\mathbf{f} = (r_0, \ldots, r_m)$ of random vertices $V_{\mathrm{R}}$, we associate a couple $\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}$ of positional strategies, called the $\mathbf{f}$-strategies. We give intuition about what are $\mathbf{f}$-strategies, before giving their formal construction.

A permutation $\mathbf{f} = (r_0, \ldots, r_m)$ of $V_{\mathrm{R}}$ intuitively represents preferences of Max and Min over the random vertices: player Max prefers the play to start from a random vertex of index as high as possible, *i.e.* starting from vertex $r_{l+1}$ is better for player Max than starting from vertex $r_l$, whereas the opposite holds for player Min.

When both players agree on the preferences given by $\mathbf{f}$, the $\mathbf{f}$-strategies $\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}$ are natural behaviours of player Max and Min. Indeed, strategy $\sigma_{\mathbf{f}}$ for player Max consists in attracting the pebble either in the target vertex or in a random vertex of index as high as possible in $\mathbf{f}$. By opposite, strategy $\tau_{\mathbf{f}}$ for player Min consists in keeping the pebble away from the target and from random vertices of high index in $\mathbf{f}$.

The $\mathbf{f}$-strategies can be described more precisely, introducing a non-stochastic game. In this non-stochastic game, plays can be either of finite or infinite duration, and after a play, players Max and Min have to give coins to each other. This game is played on the game graph $(V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, E)$, where all random vertices are terminal vertices. When the play reaches a random vertex $r_l \in V_{\mathrm{R}}$, the play immediately stops and player Min has to give $l$ coins to Max, where $l$ is the index of the random vertex $r_l$ in the permutation $f = (r_0, \ldots, r_m)$. Of course player Min wants to give as few coins as possible to player Max, whereas the goal of player Max is the opposite. The worst for player Min is when the play does not reach a random vertex but reaches the target vertex instead, in that case player Min has to give $m + 1$ coins to Max. The best for player Min is the remaining case, when the play never reaches any random vertex nor the target,

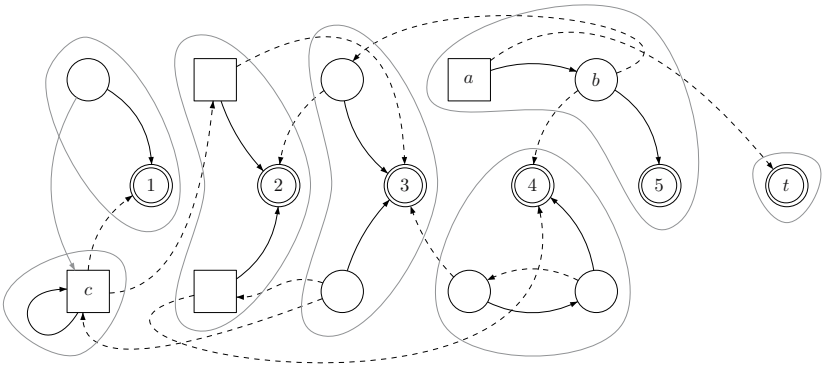



**Fig. 2.** The permutation is $\mathbf{f} = (1, 2, 3, 4, 5)$. Edges consistent with $\mathbf{f}$-strategies are black, other edges are dotted. For example from vertex $a$, player Min prefers moving the pebble on vertex $b$ than on the target vertex $t$. Indeed in the former case player Min has to pay 5 coins to Max whereas in the latter case he would have to pay 6 coins.

then instead of giving coins to Max player Min *receives* 1 coin from player Max. In this game there exist positional optimal strategies $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ for both players. These strategies are precisely the **f**-strategies. This intuitive interpretation of **f**-strategies is depicted on Figure 2, for the permutation $(1, 2, 3, 4, 5)$.

In the rest of this section, we define formally the notion of **f**-strategies. For this we need to introduce in the next subsection the notion of deterministic attractor.

### 3.2 Deterministic Attractors

Let $W \subseteq V$ be a subset of vertices. The *deterministic attractor* in $W$ is the set of vertices $\mathrm{Att}(W) \subseteq V$ from which Max has a strategy for attracting the play in $W$ and avoiding at the same time any visit to a random vertex before the first visit to $W$. An *attraction strategy in* $W$ is a positional strategy $\sigma^{\#}$ for Max which guarantees that every play starting from $\mathrm{Att}(W)$ has this property. A *trapping strategy out of* $W$ is a positional strategy $\tau^{\#}$ for player Min which guarantees than any play starting outside $\mathrm{Att}(W)$ will avoid a visit to $W$ before the first visit to a random vertex. These notions are formalized in the following proposition.

**Proposition 1.** *Let $W \subseteq V$ be a subset of vertices. There exists a subset* $\mathrm{Att}(W)$ *called the deterministic attractor in $W$, a positional strategy $\sigma^{\#}$ for* Max *called the attraction strategy in $W$ and a positional strategy $\tau^{\#}$ for* Min *called the trapping strategy out of $W$ such that:*

1. *For every $v_0 \in \mathrm{Att}(W)$, for every play $v_0 v_1 \cdots \in V^\omega$ consistent with $\sigma^{\#}$, there exists $n \in \mathbb{N}$ such that $v_n \in W$ and for every $0 \leq k < n, v_k \notin V_R$.*
2. *For every $v_0 \notin \mathrm{Att}(W)$, for every play $v_0 v_1 \cdots \in V^\omega$ consistent with $\tau^{\#}$, for every $n \in \mathbb{N}$, if $v_n \in \mathrm{Att}(W)$ then there exists $0 \leq k < n$ such that $v_k \in V_R$.*

*There exists an algorithm that computes* $\mathrm{Att}(W)$, *$\sigma^{\#}$ and $\tau^{\#}$ in time $\mathcal{O}(|E| \cdot |V|)$.*

### 3.3 Computing the f-Strategies

We now describe formally how to compute the **f**-strategies $\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}$ associated with a permutation $\mathbf{f} = (r_0, \ldots, r_m)$ of random vertices. Intuitively, the strategy $\sigma_{\mathbf{f}}$ for Max consists in attracting the play in the target vertex $t$ or in a random vertex whose index is as high as possible in $\mathbf{f}$, while the strategy $\tau_{\mathbf{f}}$ for Min aims at the opposite.

We start with defining a sequence $W_-, W_0, \ldots, W_{m+1}$ of subsets of $V$:

$$\begin{aligned} & & W_{m+1} &= \mathrm{Att}(\{t\}), \\ \text{for all } 0 \leq l \leq m, & & W_l &= \mathrm{Att}(\{r_l, r_{l+1}, \ldots, r_m, t\}), \\ & & W_- &= V \setminus W_0. \end{aligned} \tag{6}$$

An example is given on Figure 2, where relative frontiers of the sets $W_-, W_0, \ldots, W_{m+1}$ are delimited by gray lines. On this example, the set $W_{m+1}$

only contains the target vertex $t$, the set $W_5$ is $\{a, b, 5, t\}$, the set $W_-$ is the singleton $\{c\}$.

The $\mathbf{f}$-strategies are constructed by combining different positional strategies together. Let $\sigma_{m+1}$ be the attraction strategy in $\{t\}$, then on $W_{m+1}$ $\sigma_{\mathbf{f}}$ coincides with $\sigma_{m+1}$ and $\tau_{\mathbf{f}}$ is any positional strategy. For $0 \leq l \leq m$, let $\sigma_l$ be the attraction strategy in $\{r_l, r_{l+1}, \ldots, r_m, t\}$ and let $\tau_l$ be the trapping strategy out of $\{r_{l+1}, \ldots, t\}$, then on $W_l \setminus W_{l+1}$ $\sigma_{\mathbf{f}}$ coincides with $\sigma_l$ and $\tau_{\mathbf{f}}$ coincides with $\tau_l$. Let $\tau_-$ be the trapping strategy out of $\{r_0, \ldots, r_m, t\}$, then on $W_- = V \setminus W_0$, $\tau_{\mathbf{f}}$ coincides with $\tau_-$ and $\sigma_{\mathbf{f}}$ is any positional strategy.

We will use the following properties of $\mathbf{f}$-strategies:

**Lemma 1.** *Let $\mathbf{f} = (r_0, \ldots, r_m)$ a permutation of random vertices. Every play consistent with $\sigma_{\mathbf{f}}$ and starting from $W_{m+1}$ stays in $W_{m+1}$ and reaches the target vertex $t$. Every play consistent with $\tau_{\mathbf{f}}$ and starting from $W_-$ stays in $W_-$ and never reaches $t$. Let $\phi : V \to \mathbb{R}$ defined by $\phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}}(\mathrm{Reach}(t))$. For every $0 \leq l \leq m$, for every $v \in W_l \setminus W_{l+1}$, $\phi(v) = \phi(r_l)$.*

## 4 Optimality of f-Strategies

A key property of $\mathbf{f}$-strategies is given in the following theorem.

**Theorem 2.** *In every SSG, there exists a permutation $\mathbf{f}$ of random vertices such that the $\mathbf{f}$-strategies $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ are optimal.*

This theorem suggests the following algorithm for solving SSGs. It consists in testing, for each possible permutation $\mathbf{f}$ of random vertices, whether the $\mathbf{f}$-strategies are optimal. Since $\mathbf{f}$-strategies are positional, their optimality can be tested in polynomial time using linear programming [Der72, Con92]. Finally, the corresponding algorithm can find values and optimal strategies solving at most $|V_{\mathrm{R}}|!$ linear programs.

Testing optimality of $\mathbf{f}$-strategies can be done in a more elegant and efficient way, without making use of linear programming. Indeed, Theorem 3 shows that it is sufficient to test whether the $\mathbf{f}$-strategies are *self-consistent* and *progressive*, in the following sense.

**Definition 1 (Self-consistent and progressive permutations).** *Let $\mathbf{f} = (r_0, \ldots, r_m)$ be a permutation of random vertices and $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ the $\mathbf{f}$-strategies. For $v \in V$, let $\phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}}(\mathrm{Reach}(t))$.*

*Permutation $\mathbf{f}$ is* self-consistent *if*

$$\phi(r_0) \leq \phi(r_1) \leq \ldots \leq \phi(r_m). \tag{7}$$

*Permutation $\mathbf{f}$ is* progressive *if for every $0 \leq i \leq j \leq m$, if $\phi(r_i) > 0$ then there exists $w \in \mathrm{Att}(\{r_{j+1}, \ldots, r_m, t\})$ such that $p(w|r_j) > 0$.*

Both properties can be reformulated in term of the Markov chain induced by $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$, see Proposition 2.

Intuitively, both players use their $\mathbf{f}$-strategies when they both agree on the preference order given by $\mathbf{f}$ and play consistently with this preference order. Self-consistency states that plays starting from random vertices of higher rank in $\mathbf{f}$ have greater probabilities of reaching the target. The progressive property states that if some random vertex $r_i$ gives non-zero probability to reach the target, then, from every random vertex $r_j$ of higher rank in $\mathbf{f}$, also with non-zero probability either the target vertex or a random vertex of higher rank than $r_j$ will be reached prior to any visit to a random vertex.

Next theorem states that together with self-consistency, the progressive property ensures optimality of the $\mathbf{f}$-strategies.

**Theorem 3.** *Let* $\mathbf{f}$ *be a permutation of random vertices. If* $\mathbf{f}$ *is self-consistent and progressive then the* $\mathbf{f}$*-strategies* $\sigma_{\mathbf{f}}$ *and* $\tau_{\mathbf{f}}$ *are optimal. Moreover there exists a permutation* $\mathbf{f}$ *of random vertices which is self-consistent and progressive.*

Notice that self-consistency alone is not sufficient for ensuring that $\mathbf{f}$-strategies are optimal, a counter-example is given on Figure 3.

The progressive property forces any wrong guess about preferences of the players to propagate among random vertices and to lead to a violation of self-consistency. Somehow, the progressive property plays a role similar to the halting hypothesis used in [Con92]. The halting hypothesis states that any play of the SSG should reach a sink vertex, which ensures uniqueness of a solution to the local optimality equations, see [Con92] for more details.

An algorithm for solving SSGs and based on Theorem 3 is described in the next section. The rest of this section is dedicated to the proof of Theorem 3, which relies on a few preliminary lemmas.

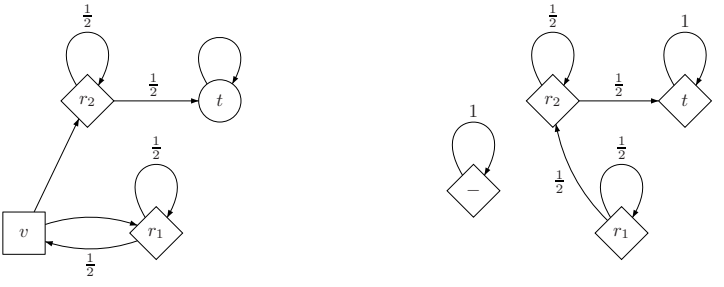



**Fig. 3.** On the left is depicted an SSG with two random vertices $\{r_1, r_2\}$, a Min vertex $v$ and the target vertex $t$. Player Min has only one optimal strategy: moving the pebble to vertex $r_1$ whenever it is on $v$, this way the play never reaches the target vertex $t$. This is exactly the $\mathbf{f}$-strategy $\tau_{\mathbf{f}}$ associated with the permutation $\mathbf{f} = (r_1, r_2)$. Suppose now the permutation is $\mathbf{f} = (r_2, r_1)$, then following its $\mathbf{f}$-strategy $\tau_{\mathbf{f}}$, player Min will go to vertex $r_2$ from $v$. In that case, $t$ is reached from both $r_1$ and $r_2$ with probability 1, hence permutation $(r_2, r_1)$ is self-consistent, although strategy $\tau_{\mathbf{f}}$ is *not* optimal. However $\mathbf{f}$ is *not* progressive since from $r_1$ the play reaches $r_2$ before reaching $t$. On the right is depicted the Markov chain $\mathcal{M}_{\mathbf{f}}$ associated with $\mathbf{f} = (r_2, r_1)$.

Under the hypothesis that $\mathbf{f}$ is self-consistent, first lemma states that during a play consistent with $\sigma_{\mathbf{f}}$, the values of vertices relatively to $\mathbf{f}$-strategies is a super-martingale, and symmetrically for player Min.

**Lemma 2.** *Let* $\mathbf{f}$ *a permutation of* $V_R$ *and* $\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}$ *the* $\mathbf{f}$*-strategies. Let* $\phi : V \to \mathbb{R}$ *defined by* $\phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t))$. *Suppose* $\mathbf{f}$ *is self-consistent. Then for every strategies* $\sigma, \tau$ *for* Max *and* Min*, for every* $v \in V$ *and* $n \in \mathbb{N}$*,*

$$\mathbb{E}_v^{\sigma_{\mathbf{f}},\tau}[\ \phi(V_{n+1}) \ \mid V_0 \cdots V_n] \geq \phi(V_n), \tag{8}$$

$$\mathbb{E}_v^{\sigma,\tau_{\mathbf{f}}}[\ \phi(V_{n+1}) \ \mid V_0 \cdots V_n] \leq \phi(V_n). \tag{9}$$

Under the hypothesis that $\mathbf{f}$ is progressive, next lemma gives a necessary and sufficient condition for a play consistent with $\sigma_{\mathbf{f}}$ to reach the target vertex.

**Lemma 3.** *Let* $\mathbf{f}$ *be a permutation of* $V_R$ *and* $\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}$ *the* $\mathbf{f}$*-strategies. Let* $\phi : V \to \mathbb{R}$ *defined by* $\phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t))$. *Suppose* $\mathbf{f}$ *is progressive. Then for every vertex* $v \in V$ *and every strategy* $\tau$ *for* Min*:*

$$\mathbb{P}_v^{\sigma_{\mathbf{f}},\tau}\left(\mathrm{Reach}(t) \,\middle|\, \phi(V_n) > 0 \text{ for infinitely many } n \in \mathbb{N}\right) = 1. \tag{10}$$

Next lemma is the main ingredient for constructing iteratively a self-consistent and progressive permutation.

**Lemma 4.** *Let* $X \subseteq V$ *be a subset of vertices of an SSG and let* $W = \mathrm{Att}(X)$. *Suppose* $W$ *contains the target vertex. Then either all vertices* $v \in V \setminus W$ *have value* 0 *in the SSG or there exists a random vertex* $r \in V_R \cap (V \setminus W)$ *such that* $\mathrm{val}(r) = \max\{\mathrm{val}(v) \mid v \in V \setminus W\}$ *and* $\sum_{v \in W} p(v|r) > 0$.

We now give a proof of Theorem 3.

*Proof (of Theorem 3)*
We start with proving that if $\mathbf{f}$ is self-consistent and progressive then $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ are optimal. Let $v \in V$ and $\sigma, \tau$ be some strategies for Max and Min.

We first prove that starting from $v$, $\sigma_{\mathbf{f}}$ ensures to reach $t$ with probability at least $\mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t))$:

$$\begin{aligned}
\mathbb{P}_v^{\sigma_{\mathbf{f}},\tau}(\mathrm{Reach}(t)) &\geq \mathbb{P}_v^{\sigma_{\mathbf{f}},\tau}(\phi(V_n) > 0 \text{ for infinitely many } n \in \mathbb{N}) \\
&\geq \mathbb{E}_v^{\sigma_{\mathbf{f}},\tau}\left[\limsup_{n\in\mathbb{N}} \phi(V_n)\right] \geq \limsup_{n\in\mathbb{N}} \mathbb{E}_v^{\sigma_{\mathbf{f}},\tau}[\phi(V_n)] \\
&\geq \mathbb{E}_v^{\sigma_{\mathbf{f}},\tau}[\phi(V_0)] = \phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t)),
\end{aligned} \tag{11}$$

where the first inequality comes from Lemma 3, the second because values of $\phi$ are between 0 and 1, the third is a property of expectations, the fourth comes from Lemma 2 and the last two equalities hold because $V_0$ is equal to the starting vertex $v$ and by definition of $\phi$.

We now prove that starting from $v$, $\tau_{\mathbf{f}}$ ensures to reach $t$ with probability no more than $\mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t))$:

$$\begin{aligned}\mathbb{P}_v^{\sigma,\tau_{\mathbf{f}}}(\mathrm{Reach}(t)) &\leq \mathbb{P}_v^{\sigma,\tau_{\mathbf{f}}}\left(\left(\liminf_{n\in\mathbb{N}}\phi(V_n)\right)=1\right) \leq \mathbb{E}_v^{\sigma,\tau_{\mathbf{f}}}\left[\liminf_{n\in\mathbb{N}}\phi(V_n)\right]\\ &\leq \liminf_{n\in\mathbb{N}}\mathbb{E}_v^{\sigma,\tau_{\mathbf{f}}}\left[\phi(V_n)\right] \leq \mathbb{E}_v^{\sigma,\tau_{\mathbf{f}}}\left[\phi(V_0)\right]\\ &= \phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t))\,, \end{aligned} \tag{12}$$

where the first inequality holds because $t$ is an absorbing state and $\phi(t)=1$, the second holds because values of $\phi$ are between 0 and 1, the third is a property of expectations, the fourth comes from Lemma 2 and the two last equalities hold because $V_0$ is equal to the starting vertex $v$ and by definition of $\phi$.

Finally, (11) and (12) together prove that $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ are optimal.

We now prove the existence of a permutation $\mathbf{f}$ which is self-consistent and progressive. For a set $W$ and a random vertex $r \in V_{\mathrm{R}}$ we denote $p(W|r) = \sum_{w\in W} p(w|r)$ the probability of going to $W$ from $r$.

We build a self-consistent and progressive permutation $\mathbf{f} = (r_0, r_1, \ldots, r_m)$ by iteration of the following iterative step.

Let $0 \leq l \leq m$, suppose that vertices $(r_{l+1}, \ldots, r_m)$ have already been chosen, let $X_{l+1} = \{r_{l+1}, \ldots, r_m, t\}$ and let $W_{l+1} = \mathrm{Att}(X_{l+1})$. If $l > 0$ and all vertices in $V \setminus W_{l+1}$ have value 0, choose $r_l$ to be any random vertex in $V_{\mathrm{R}} \setminus X_{l+1}$. Otherwise, according to Lemma 4, there exists a random vertex $r_l$ in $V \setminus W_{l+1}$ whose value is maximal in $V \setminus W_{l+1}$ and such that

$$p(W_{l+1}|r_l) > 0. \tag{13}$$

This achieves the inductive step.

Let $\mathbf{f} = (r_0, r_1, \ldots, r_m)$ be a permutation built according to this iterative procedure, we now prove that $\mathbf{f}$ is self-consistent and progressive.

By construction of $\mathbf{f}$,

$$\mathrm{val}(r_0) \leq \ldots \leq \mathrm{val}(r_n). \tag{14}$$

By definition of the $\mathbf{f}$-strategies, for any $0 \leq l \leq m$:

(A) $\sigma_{\mathbf{f}}$ coincides on $W_l \setminus W_{l+1}$ with an attraction strategy in $\{r_l, \ldots, r_m, t\}$,
(B) $\tau_{\mathbf{f}}$ coincides on $W_l \setminus W_{l+1}$ with a trapping strategy out of $\{r_{l+1}, \ldots, r_m, t\}$,
(C) $\sigma_{\mathbf{f}}$ coincides on $W_{m+1}$ with an attraction strategy in $\{t\}$,
(D) $\tau_{\mathbf{f}}$ coincides on $W_- = V \setminus W_0$ with a trapping strategy out of $W_0$,
(E) any play consistent with $\tau_{\mathbf{f}}$ starting form a vertex of value 0 stays in the set of vertices of value 0.

We start with proving that $\mathbf{f}$ is progressive. Let $0 \leq k \leq m$ such that $\mathbb{P}_{r_k}^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t)) > 0$. According to (E), $\mathrm{val}(r_k) > 0$, hence according to (14), for every $k \leq l \leq m$, $\mathrm{val}(r_l) > 0$. Hence, for every $k \leq l \leq m$, eq. (13) holds, which proves that $\mathbf{f}$ is progressive.

Now we prove that $\mathbf{f}$ is self-consistent. According to (14), for proving that $\mathbf{f}$ is self-consistent it is enough to prove that for any $0 \leq l \leq m$,

$$\mathbb{P}_{r_l}^{\sigma_{\mathbf{f}},\tau_{\mathbf{f}}}(\mathrm{Reach}(t)) = \mathrm{val}(r_l). \tag{15}$$

We start with proving for every $0 \leq l \leq m$,

$$\text{val is constant equal to } \mathrm{val}(r_l) \text{ on } W_l \setminus W_{l+1}. \tag{16}$$

Let $0 \leq l \leq m$ and $v \in W_l \setminus W_{l+1}$. According to (A), $\sigma_{\mathbf{f}}$ guarantees any play starting from $v$ to reach set $\{r_l, \ldots, r_m, t\}$ hence $\mathrm{val}(v) \geq \min\{\mathrm{val}(r_l), \ldots, \mathrm{val}(r_m), \mathrm{val}(t)\}$, and together with (14) we get $\mathrm{val}(v) \geq \mathrm{val}(r_l)$. The converse inequality holds because according to (B), strategy $\tau_{\mathbf{f}}$ guarantees any play starting from $v$ to either stay forever in $V \setminus W_{l+1}$ and never reach $t$ or to reach a random vertex in $\{r_0, \ldots, r_l\}$, hence $\mathrm{val}(v) \leq \max\{\mathrm{val}(r_0), \ldots, \mathrm{val}(r_l)\} = \mathrm{val}(r_l)$. This achieves to prove (16).

Now we prove that for every $v, w \in V$,

$$\mathbb{E}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} [\mathrm{val}(V_{n+1}) \mid V_n = w] = \mathrm{val}(w). \tag{17}$$

According to (C), val is constant equal to 1 on $W_{m+1}$ and $W_{m+1}$ is stable under $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ hence (17) holds for $w \in W_{m+1}$. According to (D), val is constant equal to 0 on $W_-$ and $W_-$ is stable under $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$ hence (17) holds for $w \in W_-$. Let $0 \leq l \leq m$ and $w \in W_l \setminus W_{l+1}$. According to (16), val is constant on $W_l \setminus W_{l+1}$ and according to (A) and (B), $W_l \setminus W_{l+1}$ is stable under $\sigma_{\mathbf{f}}$ and $\tau_{\mathbf{f}}$, hence (17) holds if $w \in V_{\mathrm{Max}} \cup V_{\mathrm{Min}}$. If $w \in V_{\mathrm{R}}$ then (17) also holds because $w \neq t$ hence according to (5), $\mathrm{val}(w) = \sum_{v \in V} p(v|w) \cdot \mathrm{val}(v)$.

Now we prove that for any $v \in V$,

$$(\mathbb{P}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} (\mathrm{Reach}(t)) = 0) \implies (\mathrm{val}(v) = 0). \tag{18}$$

For every $v \in V$ let $\phi(v) = \mathbb{P}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} (\mathrm{Reach}(t))$. Let $Z = \{v \in V \mid \phi(v) = 0\}$. We start with proving that $Z = V \setminus W_l$ for some $l$. According to Lemma 1, for each $0 \leq l \leq m$, the value of $\phi$ is constant equal to $\phi(w_l)$ on the set $W_l \setminus W_{l+1}$. Since $\phi$ has value 0 on $V \setminus W_0$, and since $W_0 \subseteq W_1 \subseteq \ldots \subseteq W_{m+1}$, there exists $0 \leq l \leq m$ such that $Z = V \setminus W_l$. Now we prove that $Z$ is stable under random moves. Indeed, let $r \in V_{\mathrm{R}} \cap Z$ then since $\phi(r) = \sum_{v \in V} p(v|r)\phi(v)$ and since $r \in Z$, $\phi(r) = 0$, hence all successors of $r$ have value 0 and are in $Z$. Now we prove that $\tau_{\mathbf{f}}$ guarantees that every play starting from $Z$ never leaves $Z$. Indeed according to (B), strategy $\tau_{\mathbf{f}}$ traps the play in $V \setminus W_{l+1} = Z$ until it reaches a random vertex, but $Z$ is stable under random moves. Finally, since $Z$ does not contain the target, any play consistent with strategy $\tau_{\mathbf{f}}$ and starting from $Z$ will never reach the target. This proves that vertices in $Z$ have value 0 and achieves the proof of (18).

Now we can achieve the proof that $\mathbf{f}$ is self-consistent. We already proved that $\mathbf{f}$ is progressive hence according to Lemma 3, there are two types of play consistent with $\sigma_{\mathbf{f}}$: those reaching the target and those staying ultimately in the set where $\phi$ has value 0. In the former case, since $t$ is absorbing, $\lim_n \mathrm{val}(V_n) = 1$ and in the latter case according to (18), $\lim_n \mathrm{val}(V_n) = 0$. Hence:

$$\mathbb{P}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} (\mathrm{Reach}(t)) = \mathbb{E}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} \left[\lim_n \mathrm{val}(V_n)\right] = \lim_n \mathbb{E}_v^{\sigma_{\mathbf{f}}, \tau_{\mathbf{f}}} [\mathrm{val}(V_n)] = \mathrm{val}(v),$$

where the second equality holds because val is bounded and the third comes from (17). This proves (15) and achieves the proof that $\mathbf{f}$ is self-consistent.

## 5 An Algorithm for Computing Values of SSGs

In this section, we give an algorithm that computes values and optimal strategies of an SSG. This algorithm, based on Theorem 3, looks for a permutation $\mathbf{f}$ which is self-consistent and progressive.

### 5.1 Testing Whether a Permutation Is Self-consistent and Progressive

For testing whether a permutation $\mathbf{f}$ is self-consistent and progressive, it is enough to compute some reachability probabilities in the following Markov chain.

**Definition 2 (Markov chain associated with f).** *Let $\mathbf{f} = (r_0, \dots, r_m)$ a permutation of $V_R$, and $W_-, W_0, \dots, W_{m+1}$ the subsets of $V$ defined by (6). Let $\mathcal{M}_\mathbf{f}$ be the Markov chain with states $S = \{-, 0, \dots, m, m+1\}$ such that both states $-$ and $m+1$ are absorbing and for every $i \in \{0, \dots, m\}$ and $j \in S$, the transition probability from $i$ to $j$ in $\mathcal{M}_\mathbf{f}$ is given by:*

$$x_{i,j} = \sum_{v \in W_j} p(v|r_i).$$

The Markov chain $\mathcal{M}_\mathbf{f}$ is designed to mimic behaviour of the play when the players use their $\mathbf{f}$-strategies: it is obtained by removing edges which are not consistent with $\mathbf{f}$-strategies and shrinking each set $W_l$ to the vertex $r_l$.

The following proposition gives an effective procedure for testing self-consistency and progressiveness of a permutation.

**Proposition 2.** *Let $\mathbf{f} = (r_0, \dots, r_m)$ be a permutation of random vertices, with $|V_R| = m+1$. Let $\mathcal{M}_\mathbf{f}$ the Markov chain associated with $\mathbf{f}$, with transition probabilities $(x_{i,j})_{i,j \in S}$. For $0 \leq i \leq m$, let $x_i^*$ the probability of eventually reaching state $m+1$ starting from state $i$ in the Markov chain $\mathcal{M}_\mathbf{f}$. Then $\mathbf{f}$ is* self-consistent *iff for every $0 \leq i, j \leq m$,*

$$(i \leq j) \implies (x_i^* \leq x_j^*), \tag{19}$$

*and $\mathbf{f}$ is* progressive *iff for every $0 \leq i \leq j \leq m$,*

$$(x_i^* > 0) \implies (\text{there exists } j < k \leq m+1 \text{ such that } x_{j,k} > 0). \tag{20}$$

*Let $I \subseteq S$ the set of states from which $m+1$ is reachable in $\mathcal{M}_\mathbf{f}$* i.e. *such that $x_i^* > 0$. Then $(x_i^*)_{i \in I}$ is the unique solution of the following linear system:*

$$\begin{cases} x_i^* & = \sum_{j \in I} x_{i,j} \cdot x_j^*, \quad (i \in I \setminus \{m+1\}) \\ x_{m+1}^* & = 1. \end{cases} \tag{21}$$

*Proof.* Uniqueness of a solution of the linear system (21) is proved for example in [Con92], Lemma 1.

### 5.2 Solving SSGs in Time $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$

Bringing together results about optimality of $\mathbf{f}$-strategies and characterization of self-consistent and progressive permutations given by Proposition 2, we obtain an algorithm for solving SSG:

**Theorem 4.** *Values and optimal strategies of a simple stochastic game* $G = (V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_R, E, t, p)$ *are computable in time* $\mathcal{O}(\ |V_R|! \cdot (|V||E| + |p|)\ )$*, where* $|p|$ *is the maximal bit-length of a transition probability in* $p$*.*

This algorithm enumerates all possible permutations $\mathbf{f}$ of $V_{\mathrm{R}}$. For each permutation, the algorithm tests whether $\mathbf{f}$ is self-consistent and progressive. This is done by computing the sets $W_-, W_0, \ldots, W_m, W_{m+1}$ defined by (6), computing the transition probabilities $(x_{i,j})_{i,j \in S}$ of the Markov chain $\mathcal{M}_{\mathbf{f}}$ associated with $\mathbf{f}$, solving the linear system (21) and testing conditions (19) and (20). If the permutation fails the test then the algorithm proceeds to the next permutation. If the permutation passes the test, then the algorithm outputs the $\mathbf{f}$-strategies and the mapping $\mathrm{val} : V \to [0, 1]$ which associates 0 to the vertices in $W_-$, 1 to the vertices in $W_{m+1}$ and $x_l^*$ to the vertices in $W_l, 0 \leq l \leq m$.

Correctness of this algorithm comes from Theorem 3, which ensures the existence of a self-consistent and progressive permutation $\mathbf{f}$ and the optimality of $\mathbf{f}$-strategies associated with any such permutation. Proposition 2 validates the procedure used for testing self-consistency and progressiveness.

The complexity of the algorithm is $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$. Indeed, there are exactly $|V_{\mathrm{R}}|!$ permutations of random vertices. For each permutation $\mathbf{f}$, the algorithm builds the Markov chain $\mathcal{M}_{\mathbf{f}}$. This is done by computing the deterministic attractors $W_{m+1}, \ldots, W_-$, which according to Proposition 1 takes time $\mathcal{O}(|E||V|)$. Then the algorithms solves the linear system (21), which can be done in time $|V_{\mathrm{R}}|^3|p|$, see [Dix82]. The two tests (19) and (20) can be performed in time $\mathcal{O}(|V_{\mathrm{R}}|)$.

## 6 Conclusion

We presented an algorithm computing values and optimal strategies of an SSG in time $\mathcal{O}(\ |V_{\mathrm{R}}|! \cdot (|V||E| + |p|)\ )$. Our algorithm is particularly efficient for the SSGs with few random vertices and does not rely on quadratic or linear programming solvers.

A natural way of improving our algorithm would be to design a smart way of updating a permutation $\mathbf{f}$ in case it is not self-consistent or progressive, this way one would obtain a new kind of strategy improvement algorithm for solving SSGs.

## References

[Con92] Condon, A.: The complexity of stochastic games. Information and Computation 96, 203–224 (1992)

[Con93] Condon, A.: On algorithms for simple stochastic games. In: Advances in computational complexity theory. DIMACS series in discrete mathematics and theoretical computer science, vol. 13, pp. 51–73 (1993)

[Der72] Derman, C.: Finite State Markov Decision Processes. Academic Press, London (1972)

[Dix82] Dixon, J.D.: Exact solution of linear equations using $p$-adic expansions. Numerische Mathematik 40, 137–141 (1982)

[GH07] Gimbert, H., Horn, F.: Solving simple stochastic games with few random vertices, http://hal.archives-ouvertes.fr/hal-00195914/fr/

[Hal07] Halman, N.: Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. Algorithmica 49, 37–50 (2007)

[Kac79] Kachiyan, L.G.: A polynomial time algorithm for linear programming. Soviet Math. Dokl. 20, 191–194 (1979)

[Lud95] Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Information and Computation 117, 151–155 (1995)

[Ren88] Renegar, J.: A polynomial-time algorithm, based on newton's method, for linear programming. Mathematical Programming 40, 59–93 (1988)

[Sha53] Shapley, L.S.: Stochastic games. In: Proceedings of the National Academy of Science USA, vol. 39, pp. 1095–1100 (1953)

[Som05] Somla, R.: New algorithms for solving simple stochastic games. Electr. Notes Theor. Comput. Sci. 119(1), 51–65 (2005)

# The Complexity of Nash Equilibria in Infinite Multiplayer Games*

Michael Ummels

Mathematische Grundlagen der Informatik, RWTH Aachen, Germany
ummels@logic.rwth-aachen.de

**Abstract.** We study the complexity of Nash equilibria in infinite (turn-based, qualitative) multiplayer games. Chatterjee & al. showed the existence of a Nash equilibrium in any such game with $\omega$-regular winning conditions, and they devised an algorithm for computing one. We argue that in applications it is often insufficient to compute just *some* Nash equilibrium. Instead, we enrich the problem by allowing to put (qualitative) constraints on the payoff of the desired equilibrium. Our main result is that the resulting decision problem is NP-complete for games with co-Büchi, parity or Streett winning conditions but fixed-parameter tractable for many natural restricted classes of games with parity winning conditions. For games with Büchi winning conditions we show that the problem is, in fact, decidable in polynomial time.

We also analyse the complexity of strategies realising a Nash equilibrium. In particular, we show that pure finite-state strategies as opposed to arbitrary mixed strategies suffice to realise any Nash equilibrium of a game with $\omega$-regular winning conditions with a qualitative constraint on the payoff.

## 1 Introduction

We study *infinite games of perfect information* [10] played by multiple players on a finite directed graph. Intuitively, a *play* of such a game evolves by moving a token along edges of the graph. Every vertex of the graph is controlled by precisely one player. Whenever the token arrives at some vertex, the player who controls this vertex must move the token to a successor vertex. Thus a play of such a game is an infinite path through the graph. Plays are mapped to *payoffs*, one for each player. In the simplest case, which we discuss here, payoffs are just 0 and 1, i.e. each player either wins or loses a given play of the game. In this case, the payoff function of each player can be described by the set of plays where she receives payoff 1, her *winning condition*.

Infinite games have been successfully applied in the verification and synthesis of reactive systems. Such a system is usually modelled as a game between the system and its environment where the environment's objective is the complement

* This research has been supported by the DFG Research Training Group "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems" (AlgoSyn).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 20–34, 2008.

of the system's objective, so the environment is considered hostile. Therefore, traditionally, the research in this area has mostly looked at two-player games where each play is won by precisely one of the two players, so-called *two-player zero-sum games*. However, motivated by the modelling of distributed systems, interest in the general case has increased in recent years [3,4].

The most common interpretation of rational behaviour in multiplayer games is captured by the notion of a *Nash equilibrium*. In a Nash equilibrium, no player can improve her payoff by unilaterally switching to a different strategy. Chatterjee & al. [4] showed that any infinite multiplayer game with $\omega$-regular winning conditions has a Nash equilibrium in pure strategies, and they also gave an algorithm for computing one. We argue that this is not satisfactory. Indeed, it can be shown that their algorithm may compute an equilibrium where all player lose when there exist other equilibria where all players win.

In applications, one might look for an equilibrium where as many players win as possible or where it is guaranteed that certain players win while certain others lose. Formulated as a decision problem, we want to know, given a $k$-player game $\mathcal{G}$ with initial vertex $v_0$ and two payoff thresholds $\overline{x}, \overline{y} \in \{0,1\}^k$, whether $(\mathcal{G}, v_0)$ has a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$.

When restricted to two-player zero-sum games, this problem, which we call NE for short, is nothing else than the classical game problem of deciding the winner. For parity games, the latter problem is known to be in UP $\cap$ co-UP [12] and therefore unlikely to be NP-hard. Moreover, if e.g. the number of priorities is bounded (as it is the case for Büchi and co-Büchi winning conditions), the problem is known to be decidable in polynomial time.

Our main result is that NE is NP-complete for games with Streett, parity or co-Büchi winning conditions. However, if each player has a Büchi winning condition, the problem becomes decidable in polynomial time.

For the proof of NP-hardness, it is essential that the number of players is unbounded. In applications, the number of players may be much smaller than the size of the game graph. So it is interesting to know the complexity of the problem if the number of players is small. A more fine-grained parameter is the *Hamming distance* of the two thresholds, i.e. the number of non-matching bits. Clearly, if the number of players is bounded, then the Hamming distance of the thresholds can be bounded as well, but even if the number of players is not bounded, the distance of the thresholds may be.

We show that, for games with parity winning conditions, NE retains the complexity of the classical parity game problem when restricted to payoff thresholds of bounded Hamming distance: In general, the complexity goes down to UP $\cap$ co-UP, but for restricted classes of games we get membership in P. It follows that there exists an FPT algorithm for NE on these restricted classes if one considers the Hamming distance of the payoff thresholds as the parameter.

Another interesting question is about the complexity of strategies realising a Nash equilibrium. We show that not only pure strategies (as opposed to mixed strategies, which allow for randomisation between actions) suffice, but also ones that require only a finite amount of memory. In particular, we show that in a

multiplayer Streett game if there exists an arbitrary Nash equilibrium with a payoff between two (qualitative) thresholds, then there exists one that can be implemented by a finite automaton with $O(kdn)$ states, where $k$ is the number of players, $d$ is the maximal number of Streett pairs for each player and $n$ is the size of the arena.

**Related Work**

Determining the complexity of Nash Equilibria has attracted much interest in recent years. Daskalakis & al. [7] showed that the problem of computing *some* Nash equilibrium of a game in strategic form is complete for the complexity class PPAD (a class of function problems which lies between FP and TFNP), and Chen and Deng [5] showed that this remains true for two-player games. More in the spirit of our work, Conitzer and Sandholm [6] showed that deciding whether there exists a Nash equilibrium in a two-player game in strategic form where player 1 receives payoff at least $x$ and related decision problems are all NP-hard.

For infinite games, not much is known. Chatterjee & al. [4] showed that one can compute a Nash equilibrium of a multiplayer parity game in nondeterministic polynomial time. They also showed that it is NP-hard to determine whether a multiplayer game with reachability conditions has a Nash equilibrium where all players win.[1] However, translating a multiplayer reachability game into a game with (co-)Büchi or parity conditions typically entails an exponential blowup, so their results do not transfer to games with these winning conditions. In fact, it follows from our results that the problem is unlikely to be NP-hard in the case of multiplayer parity games, since we show that the problem is in UP $\cap$ co-UP for multiplayer parity games and even in P for multiplayer Büchi or co-Büchi games.

## 2 Preliminaries

The definition of an infinite (two-player zero-sum) game played on a finite directed graph easily generalises to the multiplayer setting. Formally, we define an *infinite multiplayer game* as a tuple $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, (\mathrm{Win}_i)_{i\in\Pi})$ where

- $\Pi$ is a finite set of *players*;
- $(V, E)$ is a finite directed graph;
- $(V_i)_{i\in\Pi}$ is a partition of $V$;
- $\mathrm{Win}_i$ is a Borel set over $V^\omega$ for all $i \in \Pi$.

The structure $G = (V, (V_i)_{i\in\Pi}, E)$ is called the *arena* of $\mathcal{G}$, and $\mathrm{Win}_i$ is called the *winning condition* of player $i \in \Pi$. For the sake of simplicity, we assume that $vE := \{w \in V : (v, w) \in E\} \neq \emptyset$ for all $v \in V$, i.e. each vertex of $G$ has at least one outgoing edge. We call $\mathcal{G}$ a *zero-sum game* if the sets $\mathrm{Win}_i$ define a partition

[1] In fact, they consider concurrent games, but it is easy to see that their reduction can be modified to work for the kind of games we consider here.

of $V^\omega$. Thus if $\mathcal{G}$ is an infinite two-player zero-sum game with players 0 and 1 it suffices to define $V_0$ and $\mathrm{Win}_0$, and we just write $\mathcal{G} = (V, V_0, E, \mathrm{Win}_0)$.

A *play* or *history* of $\mathcal{G}$ is an infinite or finite path in $G$, respectively. We say that a play $\pi$ is *won* by player $i \in \Pi$ if $\pi \in \mathrm{Win}_i$. The *payoff* of a play $\pi$ of $\mathcal{G}$ is the vector $\mathrm{pay}(\pi) \in \{0,1\}^\Pi$ defined by $\mathrm{pay}(\pi)(i) = 1$ if $\pi$ is won by player $i$.

A *(mixed) strategy of player $i$ in $\mathcal{G}$* is a total function $\sigma : V^*V_i \to \mathcal{D}(V)$ assigning to each nonempty sequence $xv$ of vertices ending in a vertex $v$ of player $i$ a (discrete) probability distribution over $V$ such that $\sigma(xv)(w) > 0$ only if $(v,w) \in E$. We say that a play $\pi$ of $\mathcal{G}$ is *consistent* with a strategy $\sigma$ of player $i$ if $\sigma(\pi(0)\dots\pi(k))(\pi(k+1)) > 0$ for all $k < \omega$ with $\pi(k) \in V_i$. A *(mixed) strategy profile of $\mathcal{G}$* is a tuple $\overline{\sigma} = (\sigma_i)_{i\in\Pi}$ where $\sigma_i$ is a strategy of player $i$ in $\mathcal{G}$. Note that a strategy profile can be identified with a function $V^+ \to \mathcal{D}(V)$. Given a strategy profile $\overline{\sigma} = (\sigma_j)_{j\in\Pi}$ and a strategy $\tau$ of player $i$, we denote by $(\overline{\sigma}_{-i}, \tau)$ the strategy profile resulting from $\overline{\sigma}$ by replacing $\sigma_i$ with $\tau$.

A strategy $\sigma$ of player $i$ is called *pure* if for each $xv \in V^*V_i$ there exists $w \in vE$ with $\sigma(xv)(w) = 1$. Note that a pure strategy of player $i$ can be identified with a function $\sigma : V^*V_i \to V$. Finally, a strategy profile $\overline{\sigma}$ is called *pure* if each one of the strategies $\sigma_i$ is pure. A strategy $\sigma$ of player $i$ in $\mathcal{G}$ is called *positional* if $\sigma$ depends only on the current vertex, i.e. if $\sigma(xv) = \sigma(v)$ for all $xv \in V^*V_i$. A strategy profile $\overline{\sigma}$ of $\mathcal{G}$ is called *positional* if each $\sigma_i$ is positional.

It is sometimes convenient to designate an initial vertex $v_0 \in V$ of the game. We call the tuple $(\mathcal{G}, v_0)$ an *initialised game*. A *play (history) of* $(\mathcal{G}, v_0)$ is a play (history) of $\mathcal{G}$ starting with $v_0$. A strategy (strategy profile) of $(\mathcal{G}, v_0)$ is just a strategy (strategy profile) of $\mathcal{G}$. Note that if $\overline{\sigma}$ is a pure strategy profile then there is a unique play of $(\mathcal{G}, v_0)$ consistent with each $\sigma_i$, which we denote by $\langle\overline{\sigma}\rangle$.

Given a strategy profile $\overline{\sigma}$ and an initial vertex $v_0$, the *probability* of a basic open set $v_0v_1\dots v_k \cdot V^\omega$ is defined as the product of the probabilities $\overline{\sigma}(v_0\dots v_{j-1})(v_j)$ for $j = 1,\dots,k$. It is a classical result of measure theory that this extends to a unique probability measure over Borel sets, which we denote by $\Pr_{\overline{\sigma}}$. The *payoff* of a strategy profile $\overline{\sigma}$ is the vector $\mathrm{pay}(\overline{\sigma}) \in [0,1]^\Pi$ defined by $\mathrm{pay}(\overline{\sigma})(i) = \Pr_{\overline{\sigma}}(\mathrm{Win}_i)$.

A strategy profile $\overline{\sigma}$ is called a *Nash equilibrium* of $(\mathcal{G}, v_0)$ if $\Pr_{(\overline{\sigma}_{-i},\tau)}(\mathrm{Win}_i) \leq \Pr_{\overline{\sigma}}(\mathrm{Win}_i)$ for each player $i$ and each strategy $\tau$ of player $i$. Thus, in a Nash equilibrium no player can improve her payoff by (unilaterally) switching to a different strategy. The following proposition rephrases the condition for the case that $\overline{\sigma}$ is pure. In this case, it suffices to check that no player can gain from switching to another pure strategy. This follows from the fact that pure strategy suffice to win one-player games.

**Proposition 1.** *Let $\overline{\sigma}$ be a pure strategy profile of a game $(\mathcal{G}, v_0)$. The profile $\overline{\sigma}$ is Nash equilibrium iff $\langle\overline{\sigma}_{-i}, \tau\rangle \in \mathrm{Win}_i \Rightarrow \langle\overline{\sigma}\rangle \in \mathrm{Win}_i$ for each player $i$ and each pure strategy $\tau$ of player $i$.*

A strategy $\sigma$ of player $i$ is called *winning* if $\Pr_{(\overline{\tau}_{-i},\sigma)}(\mathrm{Win}_i) = 1$ for any strategy profile $\overline{\tau}$. For a game $\mathcal{G}$ we call the set of all vertices $v \in V$ such that player $i$ has a winning strategy for $(\mathcal{G}, v)$ the *winning region of player $i$*, and a strategy

of player $i$ that is winning in $(\mathcal{G}, v)$ for each vertex $v$ in the winning region of player $i$ an *optimal* strategy.

A celebrated theorem due to Martin [14] states that any two-player zero-sum game $\mathcal{G}$ with a Borel set as its winning condition is *determined* by pure strategies, i.e. both players have pure optimal strategies, and the union of the winning regions is the set of all vertices.

**Winning Conditions**

We have introduced winning conditions as abstract Borel sets of infinite sequences of vertices. In verification winning conditions are usually *$\omega$-regular sets*. Special cases are the following well-studied winning conditions:

- *Büchi* (given by $F \subseteq V$): the set of all $\alpha \in V^\omega$ such that $\alpha(k) \in F$ for infinitely many $k < \omega$.
- *co-Büchi* (given by $F \subseteq V$): the set of all $\alpha \in V^\omega$ such that $\alpha(k) \in F$ for all but finitely many $k < \omega$.
- *Parity* (given by a *priority function* $\Omega : V \to \omega$): the set of all $\alpha \in V^\omega$ such that the least number occurring infinitely often in $\Omega(\alpha)$ is even.
- *Streett* (given by a set $\Omega$ of *pairs* $(L, R)$ where $L, R \subseteq V$): the set of all $\alpha \in V^\omega$ such that for all pairs $(L, R) \in \Omega$ with $\alpha(k) \in L$ for infinitely many $k < \omega$ it is the case that $\alpha(k) \in R$ for infinitely many $k < \omega$.

Note that any Büchi condition is a parity conditions with two priorities and that any parity condition is a Streett condition. In fact, the intersection of any two parity conditions is a Streett condition. Moreover, the complement of a Büchi condition is a co-Büchi condition and vice versa, whereas the class of parity conditions is closed under complementation. Finally, note that any Streett condition is *prefix independent*, i.e. for every $\alpha \in V^\omega$ and $x \in V^*$ it is the case that $\alpha$ satisfies the condition if and only if $x\alpha$ does.

We call a game $\mathcal{G}$ a *multiplayer $\omega$-regular, (co-)Büchi, parity or Streett game* if the winning condition of *each* player is of the respective type. This differs somehow from the usual convention for two-player zero-sum games where a Büchi, co-Büchi or Streett game is a game where the winning condition of the *first* player is a Büchi, co-Büchi or Streett condition, respectively.

## 3 Characterising Nash Equilibria

In this section we aim to characterise the existence of a Nash equilibrium with a qualitative constraint on the payoff. The characterisation works for any prefix-independent Borel winning condition, though we will only use it for games with (co-)Büchi, parity or Streett winning conditions in the following sections.

For the rest of this section, let $(\mathcal{G}, v_0)$ be any $k$-player game with prefix-independent Borel winning conditions $\mathrm{Win}_i$; let $W_i$ be the winning region of player $i$ in $\mathcal{G}$, and let $\tau_i$ be a pure optimal strategy of player $i$. For each player $i \in \Pi$, we define $\mathrm{Hit}_i := V^* \cdot W_i \cdot V^\omega$. Moreover, for $\overline{x}, \overline{y} \in \{0,1\}^k$ let $\mathrm{Plays}(\overline{x}, \overline{y})$ be the set of all plays $\pi$ such that $\overline{x} \leq \mathrm{pay}(\pi) \leq \overline{y}$.

**Lemma 2.** *Let $\overline{\sigma}$ be a strategy profile such that $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$ for $\overline{x}, \overline{y} \in \{0,1\}^k$. Then $\Pr_{\overline{\sigma}}(\mathrm{Plays}(\overline{x}, \overline{y})) = 1$.*

For each strategy $\sigma$ of player $i$, we define a new strategy $\sigma^*$ of player $i$ by

$$\sigma^*(v_1 \dots v_k) = \begin{cases} \sigma(v_1 \dots v_k) & \text{if } v_j \notin W_i \text{ for all } 0 < j \leq k, \\ \tau_i(v_j v_{j+1} \dots v_k) & \text{if } v_j \in W_i \text{ and } v_l \notin W_i \text{ for all } 0 < l < j. \end{cases}$$

Intuitively, $\sigma^*$ is at least as good as $\sigma$, since $\sigma^*$ behaves like $\sigma$ as long as $W_i$ is not visited and switches to $\tau_i$ after the first visit to $W_i$, which guarantees a win with probability 1. The exact gain in probability is given by the following lemma.

**Lemma 3.** *For any strategy profile $\overline{\sigma}$ and each player $i \in \Pi$, $\Pr_{(\overline{\sigma}_{-i}, \sigma_i^*)}(\mathrm{Win}_i) = \Pr_{\overline{\sigma}}(\mathrm{Win}_i) + \Pr_{\overline{\sigma}}(\overline{\mathrm{Win}_i} \cap \mathrm{Hit}_i)$.*

The next lemma allows to infer the existence of a Nash equilibrium from the existence of a certain play. The proof uses so-called *threat strategies* (also known as *trigger strategies*), which are the basis of the *folk theorems* in the theory of repeated games (cf. [1] and [17, Chapter 8]).

**Lemma 4.** *If there exists a play $\pi$ such that $\pi \in \mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}$ for each player $i \in \Pi$, then there exists a pure Nash equilibrium $\overline{\sigma}$ with $\pi = \langle \overline{\sigma} \rangle$.*

*Proof.* Let $\pi$ be a play of $(\mathcal{G}, v_0)$ such that $\pi \in \mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}$ for each player $i \in \Pi$. Moreover, let $\tau_{\Pi \setminus \{j\}}$ be an optimal pure strategy of the coalition $\Pi \setminus \{j\}$ in the two-player zero-sum game $\mathcal{G}_j$ where player $j$ plays with her winning condition against all other players in $\mathcal{G}$; let $\tau_{i,j}$ be the corresponding strategy of player $i \neq j$ in $\mathcal{G}$ (i.e. $\tau_{i,j}(xv) = \tau_{\Pi \setminus \{j\}}(xv)$ for each $v \in V_i$), and for each player $i$ let $\tau_{i,i}$ be an arbitrary pure strategy. For each player $i$, we define a new pure strategy $\sigma_i$ in $\mathcal{G}$ as follows:

$$\sigma_i(xv) = \begin{cases} \pi(k+1) & \text{if } xv = \pi(0) \dots \pi(k) \prec \pi, \\ \tau_{i,j}(x_2 v) & \text{otherwise,} \end{cases}$$

where, in the latter case, $x = x_1 x_2$ such that $x_1$ is the longest prefix of $x$ still being a prefix of $\pi$, and $j$ is the player whose turn it was after that prefix (i.e. $x_1$ ends in $V_j$), where $j = i$ if $x_1 = \varepsilon$.

Clearly, we have $\pi = \langle \overline{\sigma} \rangle$. We claim that $\overline{\sigma}$ is a Nash equilibrium. Towards a contradiction, assume that some player $i \in \Pi$ with $\pi \notin \mathrm{Win}_i$ can improve her payoff by playing according to some (w.l.o.g. pure) strategy $\tau$ instead of $\sigma_i$. Then there exists $k < \omega$ such that $\tau(\pi(k)) \neq \sigma_i(\pi(k))$, and consequently from this point onwards $\langle \overline{\sigma}_{-i}, \tau \rangle$ is consistent with $\tau_{-i}$, the optimal strategy of the coalition $\Pi \setminus \{i\}$ in $\mathcal{G}_i$. Hence, it must be the case that $\tau_{-i}$ is not winning from $\pi(k)$. By determinacy, this implies that $\pi(k) \in W_i$, so $\pi \in \mathrm{Hit}_i$, a contradiction to the assumption that $\pi \in \mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}$. □

Now we have all the ingredients to prove the following proposition, which characterises the existence of a Nash equilibrium with a payoff between two thresholds $\overline{x}, \overline{y} \in \{0,1\}^k$.

**Proposition 5.** *Let $\overline{x}, \overline{y} \in \{0,1\}^k$. The following statements are equivalent:*

1. *There exists a Nash equilibrium $\overline{\sigma}$ with payoff $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$;*
2. *There exists a strategy profile $\overline{\sigma}$ with payoff $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$ such that $\Pr_{\overline{\sigma}}(\mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}) = 1$ for each player $i \in \Pi$;*
3. *There exists a play $\pi$ with $\overline{x} \leq \mathrm{pay}(\pi) \leq \overline{y}$ such that $\pi \in \mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}$ for each player $i \in \Pi$;*
4. *There exists a pure Nash equilibrium $\overline{\sigma}$ with payoff $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$.*

*Proof.* (1. $\Rightarrow$ 2.) Let $\overline{\sigma}$ be a Nash equilibrium with $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$. We claim that we have $\Pr_{\overline{\sigma}}(\mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}) = 1$ for each player $i \in \Pi$. Otherwise, $\Pr_{\overline{\sigma}}(\overline{\mathrm{Win}_i} \cap \mathrm{Hit}_i) > 0$ for some player $i \in \Pi$, and by Lemma 3 we would have $\Pr_{(\overline{\sigma}_{-i}, \sigma_i^*)}(\mathrm{Win}_i) > \Pr_{\overline{\sigma}}(\mathrm{Win}_i)$, so player $i$ could improve her payoff by switching from $\sigma$ to $\sigma^*$, a contradiction to the fact that $\overline{\sigma}$ is a Nash equilibrium.

(2. $\Rightarrow$ 3.) Assume that there exists a strategy profile $\overline{\sigma}$ with $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$ such that $\Pr_{\overline{\sigma}}(\mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}) = 1$ for each player $i \in \Pi$. Moreover, by Lemma 2 we have $\Pr_{\overline{\sigma}}(\mathrm{Plays}(\overline{x}, \overline{y})) = 1$. Hence, by elementary probability theory, also $\Pr_{\overline{\sigma}}(\mathrm{Plays}(\overline{x}, \overline{y}) \cap \bigcap_{i \in \Pi}(\mathrm{Win}_i \cup \overline{\mathrm{Hit}_i})) = 1$. But then, in particular, there must exist a play $\pi \in \mathrm{Plays}(\overline{x}, \overline{y})$ such that $\pi \in \mathrm{Win}_i \cup \overline{\mathrm{Hit}_i}$ for each player $i$.

(3. $\Rightarrow$ 4.) The claim follows from Lemma 4.

(4. $\Rightarrow$ 1.) Trivial. □

As a corollary we can infer that randomised strategies are not more powerful than pure ones as far as the existence of a Nash equilibrium with a qualitative constraint on the payoff is concerned.

**Corollary 6.** *Let $\overline{x}, \overline{y} \in \{0,1\}^k$. There exists a Nash equilibrium $\overline{\sigma}$ with $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$ iff there exists a pure Nash equilibrium $\overline{\sigma}$ with $\overline{x} \leq \mathrm{pay}(\overline{\sigma}) \leq \overline{y}$.*

*Remark 7.* Corollary 6 fails if the thresholds $\overline{x}$ and $\overline{y}$ are not bitvectors: One can easily construct an example of a two-player game where there is a Nash equilibrium with payoff $(1, x)$ for each real number $x \in [0,1]$, whereas any pure Nash equilibrium has payoff $(1,0)$ or $(1,1)$.

## 4 Computational Complexity

Previous research on algorithms for finding Nash equilibria in infinite games has focused on computing *some* Nash equilibrium [4]. However, a game may have several Nash equilibria with different payoffs, so one might not be interested in *any* Nash equilibrium but in one that fulfils certain requirements. For example, one might look for a Nash equilibrium where certain players win while certain other players lose. Or one might look for a *maximal* Nash equilibrium, i.e. a Nash equilibrium such that there is no Nash equilibrium with a higher payoff. This idea is captured by the following decision problem, which we call NE:

Given a multiplayer game $(\mathcal{G}, v_0)$ with $\omega$-regular winning conditions and thresholds $\overline{x}, \overline{y} \in \{0,1\}^{\Pi}$, decide whether there exists a Nash equilibrium of $(\mathcal{G}, v_0)$ with a payoff $\geq \overline{x}$ and $\leq \overline{y}$.

### 4.1 Upper Bounds

Our first result on the complexity of NE is that there is a nondeterministic polynomial-time algorithm to decide NE for multiplayer Streett games. This may surprise the reader, since the problem of deciding whether player 0 has a winning strategy in a two-player zero-sum game with a Streett winning condition is, in fact, co-NP-complete [8]. However, recall that, according to our definition, a two-player zero-sum game where one player has a Streett winning condition is, in general, not a Streett game, since we require both players to have the same type of winning condition.

**Theorem 8.** *For multiplayer Strett games, NE is in NP.*

*Proof.* To decide whether there exists a Nash equilibrium of a multiplayer Strett game $(\mathcal{G}, v_0)$ with payoff $\geq \overline{x}$ and $\leq \overline{y}$, a nondeterministic algorithm can guess a payoff $\overline{z} \in \{0,1\}^{\Pi}$ with $\overline{x} \leq \overline{z} \leq \overline{y}$ and, for each player $i \in \Pi$ with $z_i = 0$, a set $Z_i$ together with a pure positional strategy $\tau_{-i}$ for the coalition $\Pi \setminus \{i\}$. Finally, the algorithm guesses a strongly connected set $U \subseteq \bigcap_{z_i=0}(V \setminus Z_i)$ that is reachable from $v_0$ inside $\bigcap_{z_i=0}(V \setminus Z_i)$. If there is no such set, the algorithm rejects immediately. In the next step, the algorithm checks for each $i$ with $z_i = 0$ whether $\tau_{-i}$ is winning on $V \setminus Z_i$ as well as whether $U$ fulfils the winning condition (i.e. $U \cap L \neq \emptyset \Rightarrow U \cap R \neq \emptyset$ for each Streett pair $(L, R)$) of each player $i$ with $z_i = 1$ and whether $U$ does not fulfil the winning condition of each player $i$ with $z_i = 0$. If all checks are successful, the algorithm accepts; otherwise it rejects.

The correctness of the algorithm follows from Proposition 5: If there is a positional winning strategy of the coalition $\Pi \setminus \{i\}$ on $V \setminus Z_i$, then $Z_i$ must be a superset of the winning region of player $i$. So we can build a play with payoff $\overline{z}$ staying outside of the winning region of each player $i$ with $z_i = 0$ by going from $v_0$ to $U$ and visiting each vertex of $U$ again and again, which is possible because $U$ is strongly connected. On the other hand, if there is a play $\pi$ with payoff $\overline{z}$ and staying inside $V \setminus W_i$ for each player $i$ with $z_i = 0$, then the checks will succeed for the guesses $Z_i = W_i$, $\tau_i$ an optimal pure positional strategy of the coalition $\Pi \setminus \{i\}$ in $\mathcal{G}$ and $U = \mathrm{Inf}(\pi)$, since pure positional strategies suffice for player 1 to win a two-player zero-sum game with a Streett winning condition for player 0 [13].

It remains to show that the algorithm runs in polynomial time. The only critical steps are the checks whether a pure positional strategy of the coalition $\Pi \setminus \{i\}$ is winning on $V \setminus Z_i$. In order to check this, the algorithm can construct the one-player Strett game that arises from fixing the transitions taken by the positional strategy and check whether there exists a winning play for the remaining player from a vertex outside $Z_i$. Emerson and Lei [9] showed that there exists a polynomial-time algorithm for the latter problem. □

As an immediate consequence of Theorem 8, we get that for multiplayer parity games the problem NE is in NP, as well. However, in many cases, we can do better: For two payoff vectors $\overline{x}, \overline{y} \in \{0,1\}^{\Pi}$, let $\mathrm{dist}(\overline{x}, \overline{y})$ be the *Hamming distance* of $\overline{x}$ and $\overline{y}$, i.e. the number $\sum_{i \in \Pi} |y_i - x_i|$ of nonmatching bits. We show that if $\mathrm{dist}(\overline{x}, \overline{y})$ is bounded then NE retains the complexity of the parity game problem, which is known to be in UP $\cap$ co-UP [12].

**Theorem 9.** *For multiplayer parity games and bounded* $\mathrm{dist}(\overline{x}, \overline{y})$*, NE is in UP* $\cap$ *co-UP.*

*Proof.* In the following, let us assume that $\mathrm{dist}(\overline{x}, \overline{y})$ is bounded. A UP algorithm for NE works as follows: On input $(\mathcal{G}, v_0)$ the algorithm starts by guessing the winning region $W_i$ of each player. Then, for each vertex $v$ and each player $i$, the guess whether $v \in W_i$ or $v \notin W_i$ is verified by running the UP algorithm for the respective problem. If one guess was incorrect, the algorithm rejects immediately. Otherwise, the algorithm checks for each payoff $\overline{z} \in \{0,1\}^{\Pi}$ with $\overline{x} \leq \overline{z} \leq \overline{y}$ whether there exists a winning play from $v_0$ in the one-player Streett game with winning condition $\bigcap_{z_i=1} \mathrm{Win}_i \cap \bigcap_{z_i=0} \overline{\mathrm{Win}_i}$ on the arena $G \restriction \bigcap_{z_i=0}(V \setminus W_i)$. The algorithm accepts if this is the case for at least one such payoff. Analogously, a UP algorithm for the complement of NE accepts if there is *no* winning play in the same game from $v_0$ for each $\overline{x} \leq \overline{z} \leq \overline{y}$.

It is easy to see that both algorithms are indeed UP algorithm. The correctness of the two algorithms follows again from Proposition 5 with a similar reasoning as in the proof of Theorem 8. □

It is a major open problem whether winning regions of two-player zero-sum parity games can be computed in polynomial time, in general. This would allow us to decide NE for multiplayer parity games and bounded $\mathrm{dist}(\overline{x}, \overline{y})$ in polynomial time, as well.

Though it could net yet be shown that the parity game problem is in P, researchers have succeeded in identifying structural subclasses of parity games that allow to solve the game in polynomial time. First of all, it is well known that the parity game problem is in polynomial time for games with a bounded number of priorities (cf. [19]). Other classes of parity games with this property were shown to be any class of games played on graphs of bounded DAG- or Kelly width [2,15,11] (and thus also on graphs of bounded tree width or bounded entanglement), and on graphs of bounded clique width [16]. We give a general theorem that alllows to transfer these results to the multiplayer case. Formally, for any class $\mathcal{C}$ of two-player zero-sum parity games, let $\mathcal{C}^*$ be the class of all multiplayer parity games of the form $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, E, (\Omega_i)_{i \in \Pi})$ where, for each player $i \in \Pi$, the two-player zero-sum game $\mathcal{G}_i = (V, V_i, E, \Omega_i)$ is in $\mathcal{C}$.

**Theorem 10.** *Let* $\mathcal{C}$ *be a class of two-player zero-sum parity games such that the parity game problem is decidable in P for games in* $\mathcal{C}$*. Then NE is in P for games in* $\mathcal{C}^*$ *and bounded* $\mathrm{dist}(\overline{x}, \overline{y})$*.*

*Proof.* Consider the algorithm given in the proof of Theorem 9. The winning region $W_i$ of player $i$ in $\mathcal{G}$ is precisely the winning region of player 0 in the game $\mathcal{G}_i$. So, if $\mathcal{G}_i \in \mathcal{C}$, we can compute the set $W_i$ in polynomial time, and there is no need to guess this set. So we can make the algorithm deterministic while keeping its running time polynomial. □

As a corollary, we can deduce that there exists an FPT algorithm for NE when restricted to games in one of the aforementioned classes w.r.t. the parameter $\mathrm{dist}(\overline{x}, \overline{y})$. In particular, if the general parity game problem is in P, then NE for multiplayer parity games is fixed-parameter tractable.

**Corollary 11.** *Let $\mathcal{C}$ be a class of two-player zero-sum parity games such that the parity game problem is decidable in P for games in $\mathcal{C}$. Then NE for games in $\mathcal{C}*$ admits an FPT algorithm w.r.t. the parameter* $\mathrm{dist}(\overline{x}, \overline{y})$.

*Proof.* To decide whethere there is a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$, it suffices to check for each of the $2^{\mathrm{dist}(\overline{x},\overline{y})}$ many payoffs $\overline{z} \in \{0,1\}^{\Pi}$ with $\overline{x} \leq \overline{z} \leq y$ whether there exists a Nash equilibrium with payoff $\overline{z}$. By Theorem 10, each of these checks can be done in polynomial time for games in $\mathcal{C}^*$.

The natural question at this point is whether NE is actually decidable in polynomial time. In the next section, we will see that this is probably not the case. However, we claim that there exists a polynomial-time algorithm for NE if all winning conditions are Büchi winning conditions. Towards this, we describe a polynomial-time algorithm that computes, given a multiplayer Büchi game $\mathcal{G}$ and payoff thresholds $\overline{x}, \overline{y} \in \{0,1\}^{\Pi}$, the set of vertices from where there exists a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$.

The algorithm is similar to the algorithm by Emerson and Lei [9] for deciding one-player Streett games and works as follows: By Proposition 5, the game $(\mathcal{G}, v)$ has a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$ if and only if there exists a play staying outside the winning region $W_i$ of each player $i$ with $\mathrm{pay}(\pi)(i) = 0$. Clearly, this is the case if and only if there exists a payoff $\overline{z} \in \{0,1\}^{\Pi}$ and a strongly connected set $U \subseteq \bigcap_{z_i=0}(V \setminus W_i)$ reachable from $v$ inside $\bigcap_{z_i=0}(V \setminus W_i)$ such that $U \cap F_i \neq \emptyset$ iff $z_i = 1$. The essential part of the algorithm is the procedure SolveSubgame. On input $X$ its task is to find any such set contained in $X$.

As we are only interested in strongly connected sets that do not intersect with $F_i$ for each player $i$ such that $y_i = 0$, SolveSubgame is firstly called for the subgraph of $G$ that results from $G$ by removing all these sets $F_i$.

**Theorem 12.** *NE is decidable in polynomial time for multiplayer Büchi games.*

*Proof.* We claim that Algorithm 1 computes precisely the set of vertices from where there is a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$. Note that the procedure SolveSubgame calls itself at most $|X| - 1$ times on input $X$. So the procedure is called at most $|V|$ times in total. As furthermore any graph can be decomposed into its SCCs in linear time and winning regions of Büchi games can be computed in polynomial time, this implies that the algorithm runs in polynomial time. □

**Algorithm 1.** Computing the set of vertices from where there is a Nash equilibrium in a multiplayer Büchi game with a payoff $\geq x$ and $\leq y$.

**input** multiplayer Büchi game $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, (F_i)_{i\in\Pi})$, $\overline{x}, \overline{y} \in \{0,1\}^\Pi$
**for each** $i \in \Pi$ **do**
  Compute the winning region $W_i$ of player $i$ in $\mathcal{G}$
**end for**
$X := \bigcap_{y_i=0}(V \setminus F_i)$
**return** SolveSubgame($X$)

**procedure** SolveSubgame($X$)
  $Z := \emptyset$
  Decompose $G \restriction X$ into strongly connected components (SCC's)
  **for each** non-trivial SCC $C$ of $G \restriction X$ **do**
    $L := \{i \in \Pi : C \cap F_i = \emptyset\}$
    **if** $i \notin L$ for all $i$ with $x_i = 1$ **then**
      $Y := C \cap \bigcap_{i\in L}(V \setminus W_i)$
      **if** $Y = C$ **then**
        $Z := Z \cup \{v \in V : C$ reachable from $v$ in $G \restriction \bigcap_{i\in L}(V \setminus W_i)\}$
      **else**
        $Z := Z \cup$ SolveSubgame($Y$)
      **end if**
    **end if**
  **end for**
  **return** $Z$
**end procedure**

*Remark 13.* In fact, by combining the proofs of Theorem 12 and Theorem 10, one can show that NE is decidable in polynomial time for games with an arbitrary number of Büchi winning conditions and a bounded number of co-Büchi winning conditions (or even a bounded number of parity winning conditions with a bounded number of priorities).

### 4.2 Lower Bounds

The question remains whether, in general, NE is NP-hard. We answer this question affirmatively by showing that NE is not only NP-hard for two-player Streett games, but also for multiplayer co-Büchi games with an unbounded number of players even if we only want to know whether there is an equilibrium where a certain player wins. Note that, in the light of Theorem 9, it is very unlikely that NE is NP-hard for parity games when restricted to a bounded number of players.

**Theorem 14.** *NE is NP-hard for two-player Streett games.*

*Proof.* The proof is a variant of the proof for NP-hardness of the problem of deciding whether player 1 has a winning strategy in a two-player zero-sum game with a Streett winning condition [8] and by a reduction from SAT.

Given a Boolean formula $\varphi$ in conjunctive normal form, we construct a two-player Streett game $\mathcal{G}_\varphi$ as follows: For each clause $C$ the game $\mathcal{G}_\varphi$ has a vertex

$C$, which is controlled by player 1, and for each literal $X$ or $\neg X$ occurring in $\varphi$ there is a vertex $X$ or $\neg X$, respectively, which is controlled by player 0. There are edges from a clause to each literal that occurs in this clause, and from a literal to each clause occurring in $\varphi$. Player 1 wins every play of the game whereas player 2 wins if for each variable $X$ either neither $X$ nor $\neg X$ or both $X$ and $\neg X$ have been visited infinitely often (clearly a Streett condition).

Obviously, $\mathcal{G}_\varphi$ can be constructed from $\varphi$ in polynomial time. We claim that $\varphi$ is satisfiable if and only if $(\mathcal{G}_\varphi, C)$ has a Nash equilibrium where player 2 loses (with $C$ being an arbitrary clause). □

Since $\mathrm{dist}(\overline{x}, \overline{y})$ is always bounded by the number of players, it follows from Theorem 14 that NE is not fixed-parameter tractable for Streett games w.r.t. this parameter.

**Theorem 15.** *NE is NP-hard for multiplayer co-Büchi games, even with the thresholds* $\overline{x} = (1, 0, \ldots, 0)$ *and* $\overline{y} = (1, \ldots, 1)$.

*Proof.* Again, the proof is by a reduction from SAT. Given a Boolean formula $\varphi = C_1 \wedge \ldots \wedge C_m$ in CNF over variables $X_1, \ldots, X_n$, we build a game $\mathcal{G}_\varphi$ played by players $0, 1, \ldots, n$ as follows. $\mathcal{G}_\varphi$ has vertices $C_1, \ldots, C_m$ controlled by player 0, and for each clause $C$ and each literal $X_i$ or $\neg X_i$ that occurs in $C$, a vertex $(C, X_i)$ or $(C, \neg X_i)$, respectively, controlled by player $i$. Additionally, there is a sink vertex $\bot$. There are edges from a clause $C_j$ to each vertex $(C_j, L)$ such that $L$ occurs as a literal in $C_j$ and from there to $C_{(j \bmod m)+1}$. Additionally, there is an edge from each vertex $(C, \neg X_i)$ to the sink vertex $\bot$. As $\bot$ is a sink vertex, the only edge leaving $\bot$ leads to $\bot$ itself. The arena of $\mathcal{G}_\varphi$ is schematically depicted in Fig. 1. The co-Büchi winning conditions are as follows:

- Player 0 wins if the sink vertex is visited only finitely often (i.e. never);
- Player $i \in \{1 \ldots, n\}$ wins if each vertex $(C, X_i)$ is visited only finitely often.

Clearly, $\mathcal{G}_\varphi$ can be constructed from $\varphi$ in polynomial time. We claim that $\varphi$ is satisfiable if and only if $(\mathcal{G}_\varphi, C_1)$ has a Nash equilibrium where player 0 wins. □

## 5 Strategy Complexity

Recall that we have already shown that pure strategies suffice to realise any Nash equilibrium with a qualitative constraint on the payoff (Corollary 6). In this section we aim to analyse the memory requirements of these pure strategies.

A *finite-state transducer* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \delta, \tau)$ where $Q$ is a finite set of states, $\Sigma$ and $\Gamma$ finite sets of letters, $q_0 \in Q$ the initial state, $\delta : Q \times \Sigma \to Q$ the transition function, and $\tau : Q \times \Sigma \to \Gamma$ the output function. The function $\delta$ is naturally extended to a function $\delta^* : \Sigma^* \to Q$ by setting $\delta^*(\varepsilon) = q_0$ and $\delta^*(xa) = \delta(\delta^*(x), a)$. The transducer $\mathcal{A}$ computes the function $f : \Sigma^+ \to \Gamma$ defined by $f(xa) = \tau(\delta^*(x), a)$. So, if $\Sigma = \Gamma = V$ and $\tau(q, v) \in vE$ for each $q \in Q$ and $v \in V$ we can interpret $f$ as a pure strategy profile of a game played on $G = (V, (V_i)_{i \in \Pi}, E)$. We call a pure strategy profile $\overline{\sigma}$ a *finite-state strategy profile* if there exists a finite-state transducer that computes $\overline{\sigma}$.

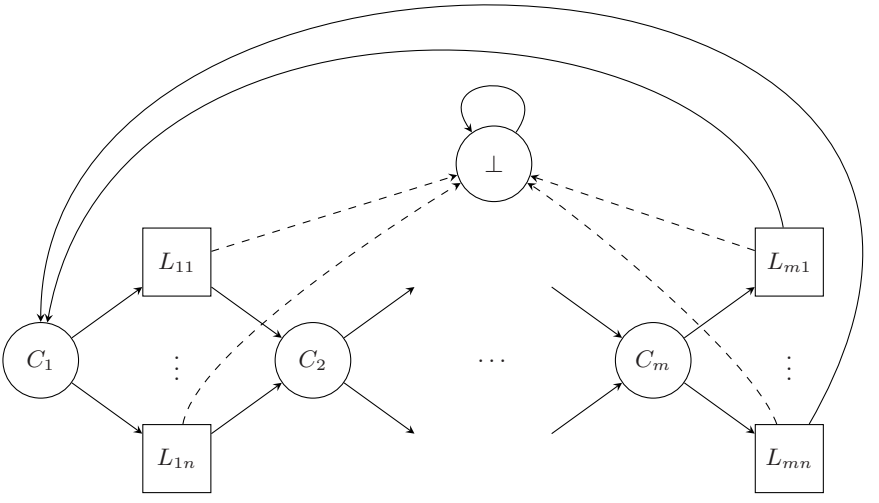



**Fig. 1.** The arena of the game $\mathcal{G}_\varphi$

**Theorem 16.** *Let $(\mathcal{G}, v_0)$ be a multiplayer Streett game with $k$ players, at most $d$ Streett pairs for each player and $n$ vertices, and let $\overline{x}, \overline{y} \in \{0,1\}^k$. If there exists a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$, then there exists a pure Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$ that can be computed by a finite automaton with $\mathrm{O}(kdn)$ (or, alternatively, $\mathrm{O}(n^2)$) states.*

*Proof.* Assume that there exists a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$. By Proposition 5, this implies that there exists a play $\pi$ with payoff $\overline{x} \leq \mathrm{pay}(\pi) \leq \overline{y}$ such that $\pi$ avoids the winning region $W_i$ of each player $i$ with $\pi \notin \mathrm{Win}_i$. Now consider the set $U$ of vertices visited infinitely often in $\pi$. In fact, it suffices to attract the token to $U$ (and thereby avoiding each $W_i$ with $\pi \notin \mathrm{Win}_i$) and then, while staying inside $U$, to visit again and again for each player $i$ and for each Streett pair $(L, R)$ in the condition for player $i$ a vertex $u \in U \cap L$ if $U \cap L \neq \emptyset$ and a vertex $u' \in U \cap R$ if $U \cap R \neq \emptyset$. The resulting play $\pi'$ has the same payoff as $\pi$ and will still satisfy the requirement that $\pi'$ avoids each $W_i$ with $\pi' \notin \mathrm{Win}_i$. It is easy to see that there exists a pure strategy profile $\overline{\tau}$ with $\pi' = \langle \overline{\tau} \rangle$ that can be computed by an automaton with $\mathrm{O}(kd)$ states. Alternatively, using $\mathrm{O}(n)$ states, one can guarantee to visit every vertex in $U$ (and only vertices in $U$) again and again.

Now consider the equilibrium $\overline{\sigma}$ constructed from $\pi'$ in the proof of Lemma 4. Since player 1 has an optimal pure positional strategy in any two-player zero-sum game where player 0 has a Streett winning condition [13], the strategies $\tau_{\Pi \setminus \{j\}}$ defined there can be assumed to be positional. For each one of the states needed to realise $\overline{\tau}$, an automaton that implements $\overline{\sigma}$ needs one state for each vertex to detect a deviation. Additionally, it needs $\min(k, n)$ more states to remember which one of the strategies $\tau_{\Pi \setminus \{j\}}$ it executes after a deviation. So $\overline{\sigma}$ can be computed by an automaton with $\mathrm{O}(kdn)$ (or, alternatively, $\mathrm{O}(n^2)$) states. □

With a small modification in the proof, one can show that for multiplayer parity games one can get rid of the factor $d$, so in this case $O(kn)$ states suffice. Finally, since any game with $\omega$-regular winning conditions can be reduced to a multiplayer parity game using finite memory [18], we can conclude that pure finite-state strategies suffice to realise any Nash equilibrium with a qualitative constraint on the payoff in these games.

**Corollary 17.** *Let $(\mathcal{G}, v_0)$ be a game with $\omega$-regular winning conditions, and let $\overline{x}, \overline{y} \in \{0,1\}^{\Pi}$. If there exists a Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$, then there exists a pure finite-state Nash equilibrium with a payoff $\geq \overline{x}$ and $\leq \overline{y}$.*

## 6 Conclusion

We have analysed the complexity of Nash equilibria in infinite multiplayer games with (co-)Büchi, parity and Streett objectives. We remark that with the same idea as in the proof of Theorem 8 one can show that NE is in $\mathrm{P}^{\mathrm{NP}}$ (in fact $\mathrm{P}^{\mathrm{NP[log]}}$) for games with only Rabin or mixed Rabin and Streett winning conditions and in PSPACE for games with Muller winning conditions.

Apart from studying other winning conditions, three obvious directions for further research come to mind: Firstly, it would be interesting to know the complexity of other solution concepts in the context of infinite games. For *subgame perfect equilibria*, preliminary results were obtained in [18]. Secondly, one could consider more general game models like *stochastic* or *concurrent* games. Thirdly, there is the *quantitative* version of NE, where the thresholds contain arbitrary probabilities rather than just 0 and 1. In fact, this problem arises naturally in the context of stochastic games.

**Acknowledgements.** I would like to thank an anonymous reviewer for clarifying the notion of a threat/trigger strategy and for pointing out [1].

## References

1. Aumann, R.J.: Survey of repeated games. In: Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern. Bibliographisches Institut Mannheim/Wien/Zürich, pp. 11–42 (1981)
2. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: DAG-width and parity games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 436–524. Springer, Heidelberg (2006)
3. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: Proceedings of the 19th Annual Symposium on Logic in Computer Science, LICS 2004, pp. 160–169. IEEE Computer Society Press, Los Alamitos (2004)
4. Chatterjee, K., Jurdziński, M., Majumdar, R.: On Nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)

5. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, pp. 261–272. IEEE Computer Society Press, Los Alamitos (2006)
6. Conitzer, V., Sandholm, T.: Complexity results about Nash equilibria. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 2003, pp. 765–771. Morgan Kaufmann, San Francisco (2003)
7. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 71–78. ACM Press, New York (2006)
8. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs (extended abstract). In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, FoCS 1988, pp. 328–337. IEEE Computer Society Press, Los Alamitos (1988)
9. Emerson, E.A., Lei, C.-L.: Modalities for model checking: Branching time strikes back. In: Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages, POPL 1985, pp. 84–96. ACM Press, New York (1985)
10. Gale, D., Stewart, F.M.: Infinite games with perfect information. In: Contributions to the Theory of Games II. Annals of Mathematical Studies, vol. 28, pp. 245–266. Princeton University Press, Princeton (1953)
11. Hunter, P., Kreutzer, S.: Digraph measures: Kelly decompositions, games, and orderings. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 637–644. ACM Press, New York (2007)
12. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Information Processing Letters 68(3), 119–124 (1998)
13. Klarlund, N.: Progress measures, immediate determinacy, and a subset construction for tree automata. In: Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science, LICS 1992, pp. 382–393. IEEE Computer Society Press, Los Alamitos (1992)
14. Martin, D.A.: Borel determinacy. Annals of Mathematics 102, 363–371 (1975)
15. Obdržálek, J.: DAG-width – connectivity measure for directed graphs. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 814–821. ACM Press, New York (2006)
16. Obdržálek, J.: Clique-width and parity games. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 54–68. Springer, Heidelberg (2007)
17. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
18. Ummels, M.: Rational behaviour and strategy construction in infinite multiplayer games. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 212–223. Springer, Heidelberg (2006)
19. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200(1–2), 135–183 (1998)

# Stochastic Games with Lossy Channels

Parosh Aziz Abdulla[1], Noomene Ben Henda[1], Luca de Alfaro[2], Richard Mayr[3], and Sven Sandberg[1]

[1] Uppsala University, Sweden
[2] University of California, Santa Cruz, USA
[3] NC State University, USA

**Abstract.** We consider turn-based stochastic games on infinite graphs induced by game probabilistic lossy channel systems (GPLCS), the game version of probabilistic lossy channel systems (PLCS). We study games with Büchi (repeated reachability) objectives and almost-sure winning conditions. These games are pure memoryless determined and, under the assumption that the target set is regular, a symbolic representation of the set of winning states for each player can be effectively constructed. Thus, turn-based stochastic games on GPLCS are decidable. This generalizes the decidability result for PLCS-induced Markov decision processes in [10].

## 1 Introduction

*Background.* It is natural to model a reactive system as a 2-player game between the "controller" or player 0, who makes nondeterministic choices of the system, and the "environment" or player 1, who provides malicious inputs to the system. In this model, each state belongs to one of the players, who selects an outgoing transition that determines the next state. Starting in some initial state, the players jointly construct an infinite sequence of states called a *run*. The winning condition is specified as a predicate on runs. Verifying properties of the system corresponds to finding the winner of the game, where the winning condition depends on the property to check.

Systems that have a probabilistic component give rise to stochastic games. These are games where some states belong to "player random", who selects the next state according to a pre-defined probability distribution. Randomness is useful to model stochastic loss of information such as unreliable communication, as well as randomized algorithms.

Previous work on algorithms for stochastic games has mostly focused on finite-state systems (see, e.g., [26,14,16,12]). However, many systems can only be faithfully modeled using infinitely many states. A lot of recent research has therefore been concerned with probabilistic infinite-state models. Probabilistic versions of lossy channel systems [11,7] and pushdown automata [18,19] use unbounded queues and stacks, respectively. Probabilistic Petri nets [4] model systems with an unbounded number of processes which run in parallel. The recently introduced Noisy Turing machines [8] model computer memories subject to stochastic errors.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 35–49, 2008.

We consider infinite-state stochastic games induced by lossy channel systems (LCS) [1,10,24]. LCS consist of finite-state control parts and unbounded channels (queues), i.e., automata where transitions are labeled by send and receive operations. They can model communication protocols such as the sliding window protocol and HDLC [6], where the communication medium is unreliable. In this paper, we introduce *game probabilistic LCS (GPLCS)*. GPLCS are *probabilistic* in the sense that the channels may randomly lose messages; and they are *games* in the sense that the next transition in the control part is selected by one of the players, depending on the current state. We can use player 0 to model nondeterminism in a communication protocol and player 1 to model a malicious cracker trying to break the protocol.

We consider Büchi (repeated reachability) objectives with almost-sure winning conditions. In other words, the goal for player 0 is to guarantee that with probability one, a given set of target states is visited infinitely many times. In the example of the malicious cracker, this corresponds to checking that the system can respond in such a way that it always eventually returns to a "ready state" with probability 1, no matter how the cracker acts.

*Related Work.* The work closest to ours is [9] where the authors consider the same model. They study GPLCS with simple reachability objectives and different winning conditions; i.e., almost-sure, with positive probability, etc. However, they do not consider GPLCS with Büchi objectives. Previous work on LCS considers several types of nondeterministic [20,6] and probabilistic systems (Markov chains) [22,1,24], as well as Markov decision processes [10] and non-stochastic games [3]. Of these, the work most closely related to ours is [10], which concerns LCS where messages are lost probabilistically and control transitions are taken nondeterministically (i.e., PLCS-induced Markov decision processes). This is a special case of our model in the sense that the game is restricted to only one player. It was shown in [10] that such 1-player Büchi-games are decidable (while coBüchi-games are undecidable). We generalize the decidability result of [10] for PLCS-induced Markov decision processes to 2-player stochastic games. The scheme presented in [10] also differs from ours in the fact that the target set is defined by control-states, while we consider more general *regular sets*. Thus our result is not a direct generalization of [10].

Stochastic games on infinite-state probabilistic recursive systems were studied in [18,19]. However, recursive systems are incomparable to the GPLCS model considered in this paper.

In [3], a model similar to ours is studied. It differs in that the system is not probabilistic, and instead one of the players controls message losses. For this model, [3] proves that safety games are decidable and parity games (which generalize Büchi games) are undecidable.

Two-player *concurrent* (but non-stochastic) games with infinite state spaces are studied in [17]. Concurrency means that the two players independently and simultaneously select actions, and the next state is determined by the combination of the actions and the current state. [17] describes schemes for computing winning sets and strategies for Büchi games (as well as reachability games and

some more general games). The article characterizes classes of games where the schemes terminate, based on properties of certain equivalence relations on states. However, this approach does not work for GPLCS (not even for non-probabilistic LCS), since LCS do not satisfy the necessary preconditions. Unlike the process classes studied in [17], LCS do not have a finite index w.r.t. the equivalences considered in [17].

In [28], a scheme is given to solve non-stochastic *parity* games on infinite state spaces of arbitrary cardinality. The parity condition is more general than the Büchi condition, so the scheme applies to Büchi games too. However, stochastic games are not considered. In fact, if our scheme is instantiated on the special case of non-stochastic Büchi games, it will coincide with the scheme in [28]. Furthermore, [28] does not suggest any class of infinite-state systems for which termination is guaranteed.

Our algorithms are related to the algorithms presented in [16,15] for solving concurrent games with respect to probability-1 $\omega$-regular properties. However, the proofs in [16,15] apply only to finite-state games; we will need to develop entirely new arguments to prove the correctness of our approach for GPLCS.

*Contribution.* We prove that the almost-sure Büchi-GPLCS problem is decidable: we can compute symbolic representations of the winning sets and winning strategies for both players. The symbolic representations are based on regular expressions, and the result holds under the assumption that the set of target states is also regular. The winning strategies are pure memoryless, i.e., the next state depends only on the current state and is not selected probabilistically. Our result generalizes the decidability result for PLCS-induced Markov decision processes (i.e., 1-player games) in [10].

We now give an overview of our method. First, we give a scheme to compute the winning sets in simple reachability games, where the goal of player 0 is to reach a regular set of target states with a *positive probability*. Next, we give a scheme to construct the winning sets in almost-sure Büchi-games, using the scheme for reachability games as a subroutine. We prove that for GPLCS, both schemes terminate and we show how to instantiate them using *regular state languages* to effectively represent the infinite sets.

*Outline.* In Section 2, we define stochastic games. In Section 3, we describe GPLCS and show how they induce an infinite-state stochastic game. In Section 4, we show how to construct the winning sets in simple reachability games on GPLCS. In Section 5, we show how to construct the winning sets in Büchi games on GPLCS. Due to space limitations, some proofs are omitted and can be found in [2]; however, the intuitions are given in the main text.

## 2 Preliminaries

We use $\mathbb{R}, \mathbb{N}$ for the real and natural numbers. If $X$ is a set then $X^*$ and $X^\omega$ denote the sets of finite and infinite sequences over $X$, respectively. The empty word is denoted by $\varepsilon$. For partial functions $f, g : X \rightharpoonup Y$ which have the same

value when both are defined, we use $f \cup g$ to denote the smallest function that extends both $f$ and $g$.

A *probability distribution* on a countable set $X$ is a function $f : X \to [0,1]$ such that $\sum_{x \in X} f(x) = 1$. We will sometimes need to pick an arbitrary element from a set. To simplify the exposition, we let $select(X)$ denote an arbitrary but fixed element of the nonempty set $X$.

*Turn-Based Stochastic Games.* A *turn-based stochastic game* (or a *game* for short) is a tuple $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$ where:

- $S$ is a countable set of *states*, partitioned into the pairwise disjoint sets of *random states* $S^R$, states $S^0$ of player 0, and states $S^1$ of player 1.
- $\longrightarrow \subseteq S \times S$ is the *transition relation*. We write $s \longrightarrow s'$ to denote that $(s, s') \in \longrightarrow$. Let $\mathrm{Post}(s) := \{s' : s \longrightarrow s'\}$ denote the set of *successors* of $s$ and extend it to sets $Q \subseteq S$ of states by $\mathrm{Post}(Q) := \bigcup_{s \in Q} \mathrm{Post}(s)$. We assume that games are deadlock-free, i.e., each state has at least one successor ($\forall s \in S. \mathrm{Post}(s) \neq \emptyset$).
- The *probability function* $P : S^R \times S \to [0,1]$ satisfies both $\forall s \in S^R. \forall s' \in S. (P(s,s') > 0 \iff s \longrightarrow s')$ and $\forall s \in S^R. \sum_{s' \in S} P(s,s') = 1$. Note that for any given state $s \in S^R$, $P(s, \cdot)$ is a probability distribution over $\mathrm{Post}(s)$.

For any set $Q \subseteq S$ of states, we let $\overline{Q} := S - Q$ denote its complement. We define $[Q]^R := Q \cap S^R$, $[Q]^0 := Q \cap S^0$, $[Q]^1 := Q \cap S^1$, and $[Q]^{01} := Q \cap (S^0 \cup S^1)$.

A *run* $\rho$ in a game is an infinite sequence $s_0 s_1 \cdots$ of states s.t. $s_i \longrightarrow s_{i+1}$ for all $i \geq 0$. We use $\rho(i)$ to denote $s_i$. A *path* $\pi$ is a finite sequence $s_0 \cdots s_n$ of states s.t. $s_i \longrightarrow s_{i+1}$ for all $i : 0 \leq i < n$. For any $Q \subseteq S$, we use $\Pi_Q$ to denote the set of paths that end in some state in $Q$.

Informally, the two players 0 and 1 construct an infinite run $s_0 s_1 \cdots$, starting in some initial state $s_0 \in S$. Player 0 chooses the successor $s_{i+1}$ if $s_i \in S^0$, player 1 chooses $s_{i+1}$ if $s_i \in S^1$, and the successor $s_{i+1}$ is chosen randomly according to the probability distribution $P(s_i, \cdot)$ if $s_i \in S^R$.

*Strategies.* For $\sigma \in \{0,1\}$, a *strategy* of player $\sigma$ is a partial function $f^\sigma : \Pi_{S^\sigma} \rightharpoonup S$ s.t. $s_n \longrightarrow f^\sigma(s_0 \cdots s_n)$ if $f^\sigma$ is defined. The strategy $f^\sigma$ prescribes for player $\sigma$ the next move, given the current prefix of the run. We say that $f^\sigma$ is total if it is defined for every $\pi \in \Pi_{S^\sigma}$.

A strategy $f^\sigma$ of player $\sigma$ is *memoryless* if the next state only depends on the current state and not on the previous history of the game, i.e., for any path $s_0 \cdots s_k \in \Pi_{S^\sigma}$, we have $f^\sigma(s_0 \cdots s_k) = f^\sigma(s_k)$. A memoryless strategy of player $\sigma$ can be regarded simply as a function $f^\sigma : S^\sigma \rightharpoonup S$, such that $s \longrightarrow f^\sigma(s)$ whenever $f^\sigma$ is defined.

Consider two total strategies $f^0$ and $f^1$ of player 0 and 1. A path $\pi = s_0 \cdots s_n$ in $\mathcal{G}$ is said to be *consistent* with $f^0$ and $f^1$ if the following holds. For all $0 \leq i \leq n-1$, $s_i \in S^0$ implies $f^0(s_0 \cdots s_i) = s_{i+1}$ and $s_i \in S^1$ implies $f^1(s_0 \cdots s_i) = s_{i+1}$. We define similarly *consistent* runs. In the sequel, whenever the strategies are known from the context, we assume that all mentioned paths and runs are consistent with them.

*Probability Measures.* We use the standard definition of the probability measure for a set of runs [23]. First, we define the measure for total strategies, and then extend it to general (partial) strategies. We let $\Omega^s = sS^\omega$ denote the set of all infinite sequences of states starting from $s$. Consider a game $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$, an initial state $s$, and total strategies $f^0$ and $f^1$ of player 0 and 1. For a measurable set $\mathfrak{R} \subseteq \Omega^s$, we define $\mathcal{P}^s_{f^0,f^1}(\mathfrak{R})$ to be the probability measure of $\mathfrak{R}$ under the strategies $f^0, f^1$. It is well-known that this measure is well-defined [23]. When the state $s$ is known from context, we drop the superscript and write $\mathcal{P}_{f^0,f^1}(\mathfrak{R})$. For (partial) strategies $f^0$ and $f^1$ of player 0 and 1, $\sim \in \{<, \leq, =, \geq, >\}$, and any measurable set $\mathfrak{R} \subseteq \Omega^s$, we define $\mathcal{P}^s_{f^0,f^1}(\mathfrak{R}) \sim x$ iff $\mathcal{P}^s_{g^0,g^1}(\mathfrak{R}) \sim x$ for all total strategies $g^0$ and $g^1$ which are extensions of $f^0$ resp. $f^1$. For a single strategy $f^\sigma$ of player $\sigma$, we define $\mathcal{P}^s_{f^\sigma}(\mathfrak{R}) \sim x$ iff $\mathcal{P}^s_{f^0,f^1}(\mathfrak{R}) \sim x$ for all strategies $f^{1-\sigma}$ of player $(1-\sigma)$. If $\mathcal{P}_{f^0,f^1}(\mathfrak{R}) = 1$, then we say that $\mathfrak{R}$ happens *almost surely* under the strategies $f^0, f^1$.

We assume familiarity with the syntax and semantics of the temporal logic *CTL** (see, e.g., [13]). We use $(s \models \varphi)$ to denote the set of runs starting in $s$ that satisfy the *CTL** path-formula $\varphi$. We use $\mathcal{P}_{f^0,f^1}(s \models \varphi)$ to denote the measure of $(s \models \varphi)$ under strategies $f^0, f^1$, i.e., we measure the probability of those runs which start in $s$, are consistent with $f^0, f^1$ and satisfy the path-formula $\varphi$. This set is measurable by [27].

*Traps.* For a player $\sigma \in \{0, 1\}$ and a set $Q \subseteq S$ of states, we say that $Q$ is a *$\sigma$-trap* if player $(1-\sigma)$ has a strategy that forces all runs to stay inside $Q$. Formally, all successors of states in $[Q]^\sigma \cup [Q]^R$ are in $Q$ and every state in $[Q]^{1-\sigma}$ has some successor in $Q$.

*Winning Conditions.* Our main result considers *Büchi* objectives: player 0 wants to visit a given set $F \subseteq S$ infinitely many times. We consider games with *almost-sure* winning condition. More precisely, given an initial state $s \in S$, we want to check whether player 0 has a strategy $f^0$ such that for all strategies $f^1$ of player 1, it is the case that $\mathcal{P}_{f^0,f^1}(s \models \Box\Diamond F) = 1$.

*Determinacy and Solvability.* A game is said to be *determined* if, from every state, one of the players has a strategy that wins against all strategies of the opponent. Notice that determinacy implies that there is a partitioning $W^0, W^1$ of $S$, such that players 0 and 1 have winning strategies from $W^0$ and $W^1$, respectively. A game is *memoryless determined* if it is determined and there are memoryless winning strategies. By *solving* a determined game, we mean giving an algorithm to check, for any state $s \in S$, whether $s \in W^0$ or $s \in W^1$.

## 3 Game Probabilistic Lossy Channel Systems (GPLCS)

A *lossy channel system (LCS)* [6] is a finite-state automaton equipped with a finite number of unbounded FIFO channels (queues). The system is *lossy* in the sense that, before and after a transition, an arbitrary number of messages may

be lost from the channels. *Probabilistic lossy channel system (PLCS)* [11,7,4] define a probabilistic model for message losses. The standard model assumes that each individual message is lost independently with probability $\lambda$ in every step, where $\lambda > 0$ is a parameter of the system.

We consider *game probabilistic LCS (GPLCS)*, the 2-player game extension of PLCS. The set of states is partitioned into states belonging to player 0 and 1, and the transitions are controlled by the players. The player who owns the current control-state chooses an enabled outgoing transition. However, message losses occur randomly. While our definition of GPLCS (see below) assumes the same model of independent message loss as in [11,7,4], this is not necessary for our results. We only require the existence of a finite attractor, in the sense described in Section 5. In fact, many other probabilistic message loss models (e.g., burst disturbances, where groups of messages in close proximity are more often affected) satisfy this attractor condition [5].

The players have conflicting goals: player 0 wants to reach a given set of states infinitely often, and player 1 wants to visit it at most finitely many times. This is called a Büchi objective.

Formally, a GPLCS is a tuple $\mathcal{L} = \left(\mathtt{S}, \mathtt{S}^0, \mathtt{S}^1, \mathtt{C}, \mathtt{M}, \mathtt{T}, \lambda\right)$ where $\mathtt{S}$ is a finite set of *control-states* partitioned into states $\mathtt{S}^0, \mathtt{S}^1$ of player 0 and 1; $\mathtt{C}$ is a finite set of *channels*, $\mathtt{M}$ is a finite set called the *message alphabet*, $\mathtt{T}$ is a set of *transitions*, and $0 < \lambda < 1$ is the *loss rate*. Each transition $\mathtt{t} \in \mathtt{T}$ is of the form $\mathtt{s} \xrightarrow{\mathtt{op}} \mathtt{s}'$, where $\mathtt{s}, \mathtt{s}' \in \mathtt{S}$ and $\mathtt{op}$ is one of $\mathtt{c!m}$ (send message $\mathtt{m} \in \mathtt{M}$ in channel $\mathtt{c} \in \mathtt{C}$), $\mathtt{c?m}$ (receive message $\mathtt{m}$ from channel $\mathtt{c}$), or $\mathtt{nop}$ (do not modify the channels).

A GPLCS $\mathcal{L} = \left(\mathtt{S}, \mathtt{S}^0, \mathtt{S}^1, \mathtt{C}, \mathtt{M}, \mathtt{T}, \lambda\right)$ induces a game $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$, where $S = \mathtt{S} \times (\mathtt{M}^*)^{\mathtt{C}} \times \{0,1\}$. That is, each state in the game consists of a control-state, a function that assigns a finite word over the message alphabet to each channel, and one of the symbols 0 or 1. States where the last symbol is 0 are random: $S^R = \mathtt{S} \times (\mathtt{M}^*)^{\mathtt{C}} \times \{0\}$. The other states belong to a player according to the control-state: $S^\sigma = \mathtt{S}^\sigma \times (\mathtt{M}^*)^{\mathtt{C}} \times \{1\}$. Transitions out of states of the form $s = (\mathtt{s}, \mathtt{x}, 1)$ model transitions in $\mathtt{T}$ leaving state $\mathtt{s}$. On the other hand, transitions leaving states of the form $s = (\mathtt{s}, \mathtt{x}, 0)$ model message losses.

If $s = (\mathtt{s}, \mathtt{x}, 1), s' = (\mathtt{s}', \mathtt{x}', 0) \in S$, then there is a transition $s \longrightarrow s'$ in the game iff one of the following holds:

- $\mathtt{s} \xrightarrow{\mathtt{nop}} \mathtt{s}'$ and $\mathtt{x} = \mathtt{x}'$;
- $\mathtt{s} \xrightarrow{\mathtt{c!m}} \mathtt{s}'$, $\mathtt{x}'(\mathtt{c}) = \mathtt{x}(\mathtt{c})\mathtt{m}$, and for all $\mathtt{c}' \in \mathtt{C} - \{\mathtt{c}\}$, $\mathtt{x}'(\mathtt{c}') = \mathtt{x}(\mathtt{c}')$;
- $\mathtt{s} \xrightarrow{\mathtt{c?m}} \mathtt{s}'$, $\mathtt{x}(\mathtt{c}) = \mathtt{m}\mathtt{x}'(\mathtt{c})$, and for all $\mathtt{c}' \in \mathtt{C} - \{\mathtt{c}\}$, $\mathtt{x}'(\mathtt{c}') = \mathtt{x}(\mathtt{c}')$.

To model message losses, we introduce the subword ordering $\preceq$ on words: $x \preceq y$ iff $x$ is a word obtained by removing zero or more messages from arbitrary positions of $y$. This is extended to channel states $\mathtt{x}, \mathtt{x}' : \mathtt{C} \to \mathtt{M}^*$ by $\mathtt{x} \preceq \mathtt{x}'$ iff $\mathtt{x}(\mathtt{c}) \preceq \mathtt{x}'(\mathtt{c})$ for all channels $\mathtt{c} \in \mathtt{C}$, and to game states $s = (\mathtt{s}, \mathtt{x}, i), s' = (\mathtt{s}', \mathtt{x}', i') \in S$ by $s \preceq s'$ iff $\mathtt{s} = \mathtt{s}'$, $\mathtt{x} \preceq \mathtt{x}'$, and $i = i'$. For any $s = (\mathtt{s}, \mathtt{x}, 0)$ and any $\mathtt{x}'$ such that $\mathtt{x}' \preceq \mathtt{x}$, there is a transition $s \longrightarrow (\mathtt{s}, \mathtt{x}', 1)$. The probability of random transitions is given by $P((\mathtt{s}, \mathtt{x}, 0), (\mathtt{s}, \mathtt{x}', 1)) = a \cdot \lambda^b \cdot (1-\lambda)^c$, where $a$ is the number of ways

to obtain $\mathtt{x}'$ by losing messages in $\mathtt{x}$, $b$ is the total number of messages lost in all channels, and $c$ is the total number of messages in all channels of $\mathtt{x}'$.

Every state on the form $(\mathtt{s}, \mathtt{x}, 0)$ has at least one successor, namely $(\mathtt{s}, \mathtt{x}, 1)$. If a state $(\mathtt{s}, \mathtt{x}, 1)$ does not have successors according to the rules above, then we add a transition $(\mathtt{s}, \mathtt{x}, 1) \longrightarrow (\mathtt{s}, \mathtt{x}, 0)$, to avoid deadlocks. Intuitively, this means that the run stays in the same control state and only loses messages.

Observe that the game is *bipartite*: every transition goes from a player state to a probabilistic state or the other way around, i.e., $\longrightarrow \subseteq ((S^0 \cup S^1) \times S^R) \cup (S^R \times (S^0 \cup S^1))$.

*Problem Statement.* We study the problem Büchi-GPLCS, defined as follows. The game graph is induced by a GPLCS; and we consider the almost-sure Büchi objective: player 0 wants to ensure that a given target set is visited infinitely often with probability one.

## 4 Reachability Games on GPLCS

We consider the reachability game where the winning condition is to reach a given target set with positive probability. Reachability games on GPLCS (with this and various other winning conditions) have been studied in [9], where the winning sets are expressed in terms of the target set in a variant of the $\mu$-calculus.

Nevertheless, we give below a more ad-hoc scheme for computing the winning set, in order to keep the article self-contained. Furthermore, many definitions and some more detailed results on the structure of the winning sets and strategies will be needed in the following section on Büchi-games.

We give a scheme for characterizing sets of states from which a player can, with a positive probability, force the game into a given set of target states, while preserving a given invariant. We show that the scheme always terminates for GPLCS, and then give a symbolic representation of the winning sets, based on regular languages. The symbolic representation is valid under the assumption that the set of target states is also regular. Finally, we show correctness of the construction by describing the winning strategies. In fact, we show that if a player can win, then a *memoryless* strategy is sufficient to win.

*Scheme.* Fix a game $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$ and two sets of states $F, I \subseteq S$, called the *target* and *invariant* sets, respectively. For a player $\sigma \in \{0, 1\}$, we give a scheme for constructing the set $\mathrm{Force}^\sigma(I, F)$ of states where player $\sigma$ can, with a positive probability, force the run to eventually reach $F$, while also preserving the property that the run will always remain within $I$ (i.e., states outside $I$ are not visited before $F$).

The idea of the scheme is to perform backward reachability analysis using the basic operations $\mathrm{Pre}^\sigma$ and $\widetilde{\mathrm{Pre}}^\sigma$, defined as follows. Given $\sigma \in \{0, 1, R\}$ and a set $Q \subseteq S$ of states, let $\mathrm{Pre}^\sigma(Q) := \{s \in S^\sigma : \exists s' \in Q . s \longrightarrow s'\}$ denote the set of states of player $\sigma$ where it is possible to go to $Q$ in the next step. Define $\widetilde{\mathrm{Pre}}^\sigma(Q) := S^\sigma - \mathrm{Pre}^\sigma(\overline{Q})$ to be the set of states where player $\sigma$ cannot avoid going to $Q$ in the next step.

The construction is inductive. For $\sigma \in \{0,1\}$, we define two sequences $\{D_i\}_{i\in\mathbb{N}} : D_0 \subseteq D_1 \subseteq \cdots$ and $\{E_i\}_{i\in\mathbb{N}} : E_0 \subseteq E_1 \subseteq \cdots$ of sets of states as follows:

$$D_0 := [F]^R \cap I \qquad E_0 := [F]^{01} \cap I$$
$$D_{i+1} := \left(D_i \cup \mathrm{Pre}^R(E_i)\right) \cap I \qquad E_{i+1} := \left(E_i \cup \mathrm{Pre}^\sigma(D_i) \cup \widetilde{\mathrm{Pre}}^{1-\sigma}(D_i)\right) \cap I.$$

We let $\mathrm{Force}^\sigma(I,F) := \bigcup_{i\geq 0} D_i \cup E_i$. Intuitively, the set $D_i$ contains those states in $S^R$ from which player $\sigma$ can force the game to $F$ with positive probability (while remaining in $I$) within $i$ steps. The set $E_i$ contains the states in $S^0 \cup S^1$ satisfying the same property[1].

Below, we instantiate the above described scheme for GPLCS. In the rest of this section, we consider the game $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$ induced by a GPLCS $\mathcal{L} = \left(\mathtt{S}, \mathtt{S}^0, \mathtt{S}^1, \mathtt{C}, \mathtt{M}, \mathtt{T}, \lambda\right)$.

*Termination.* We recall from [21] that the relation $\preceq$ is a *well quasi-ordering*, i.e., for each infinite sequence $w_0, w_1, w_2, \ldots$ of words over $\mathtt{M}$, there are $j < k$ such that $w_j \preceq w_k$. A set $U \subseteq \mathtt{M}^*$ is said to be *upward closed* if $w \in U$ implies that $w' \in U$ for each $w' \succeq w$. A *channel language* $L$ is a mapping from $\mathtt{C}$ to $2^{\mathtt{M}^*}$. In other words, $L$ maps each channel to a language over $\mathtt{M}$. We say that $L$ is *upward closed* resp. *regular* if $L(\mathtt{c})$ is upward closed resp. regular for each $\mathtt{c} \in \mathtt{C}$. A *state language* $L$ is of the form $(\mathtt{s}, L')$ where $\mathtt{s} \in \mathtt{S}$ and $L'$ is a channel language. We say that $L$ is upward closed (regular) if $L'$ is upward closed (regular). We generalize the definitions above to finite sets $M$ of state languages by $M$ upward closed (regular) if each $L \in M$ is upward closed (regular). The well quasi-ordering property carries directly over from words (mentioned earlier) to our finite sets of state languages (when required to hold for every control-state).

To prove termination of the scheme, we show that sets $D_i$ are "almost" upward closed in the sense that they are closely related to other sets which are upward closed. More precisely, we consider the sequence $D'_0 \subseteq D'_1 \subseteq \cdots$ of sets of states where $D'_0 := [F]^R$ and $D'_{i+1} := \mathrm{Pre}^R(E_i)$. Since $\mathrm{Pre}^R(Q)$ is upward closed for any set $Q$ of states, it follows that $D'_i$ is upward closed for each $i > 0$. Upward closedness, together with the well quasi-ordering of $\preceq$, implies that there is a $j$ such that $D'_j = D'_{j+1} = \cdots$. We also observe that $D_i = (D'_0 \cup D'_1 \cup \cdots \cup D'_i) \cap I$. This means that $D_{j+1} = D_j$ and consequently $E_{j+2} = E_{j+1}$. Hence, we have the following lemma.

**Lemma 1.** *For any GPLCS and sets $F, I \subseteq S$ of states, the sequences $\{D_i\}_{i\in\mathbb{N}}$ and $\{E_i\}_{i\in\mathbb{N}}$ converge.*

*Forms of Winning Sets.* The above termination argument relied on upward closedness of the sets $D'_i$. In fact, we can derive more information about the structure of the winning sets for games induced by GPLCS. Assuming that the

[1] It is possible to define only one sequence, not separating player states from random states. In later proofs, it will be technically convenient to have the sequence $\{D_i\}_{i\in\mathbb{N}}$ defined, since $\{D_i\}_{i\in\mathbb{N}}$ has properties not shared by $\{E_i\}_{i\in\mathbb{N}}$.

sets $F$ and $I$ are regular state languages, it follows that each set $D_i$ or $E_i$ is also a regular state language. This follows from the fact that regular state languages are closed under the application of $\mathrm{Pre}^\sigma$ and the Boolean operations. Since the scheme terminates (by Lemma 1), the winning set $Q := \mathrm{Force}^\sigma(I,F)$ is also regular. Furthermore, if $I$ and $F$ are upward closed then $[Q]^R$ is also upward closed. This follows from the fact that $\mathrm{Pre}^R(Q)$ is upward closed for any set $Q$ and that the class of upward closed sets is closed under intersection and union. We summarize these properties as properties (1)–(2) of the following lemma. (Properties (2)–(5) are not needed until the next section).

**Lemma 2.** *Let $Q = Force^\sigma(I,F)$. Then:*

*(1) If $F$ and $I$ are regular then $Q$ is regular.*
*(2) If $F$ and $I$ are upward closed then $[Q]^R$ is upward closed.*
*(3) Let $s \in I - Q$. If $s \in S^\sigma \cup S^R$, then $Post(s) \subseteq \overline{Q}$. If $s \in S^{1-\sigma}$, then $Post(s) \cap \overline{Q} \neq \emptyset$.*
*(4) $\underline{Force^\sigma(Q,F)} = Q$.*
*(5) $\overline{Force^\sigma(S,F)}$ is a $\sigma$-trap.*

*Correctness.* First, we describe a partial memoryless winning strategy $\underline{\mathrm{force}}^\sigma(I,F)$ for player $\sigma$ from the states in $[\mathrm{Force}^\sigma(I,F)]^\sigma$. Recall that a memoryless strategy can simply be described as a function that assigns one successor to each state. We define a sequence $e_0 \subseteq e_1 \subseteq e_2 \subseteq \cdots$ of strategies for player $\sigma$. Let $e_0 := \emptyset$ and define $e_{i+1}$ as follows:

– If $e_i(s)$ is defined then $e_{i+1}(s) := e_i(s)$.
– If $e_i(s)$ is undefined and $s \in [E_{i+1} - E_i]^\sigma$ then $e_{i+1}(s) := select(\mathrm{Post}(s) \cap D_i)$.

Let $\underline{\mathrm{force}}^\sigma(I,F) := \bigcup_{i \geq 0} e_i$. From the definitions, we derive the following lemma.

**Lemma 3.** *In any GPLCS, for any $I, F \subseteq S$, $\sigma \in \{0,1\}$, and $s \in Force^\sigma(I,F)$, there exists an $\epsilon_s > 0$ such that $\mathcal{P}_{\underline{force}^\sigma(I,F)}(s \models \Diamond F) \geq \epsilon_s$.*[2]

*Proof.* We recall the construction of the force sets and use induction on $i$ to prove that $\forall i \in \mathbb{N}$ and for any state $s \in (D_i \cup E_i)$ the following holds: There exists an $\epsilon_s > 0$ such that for any extension $f^\sigma$ of the $\underline{\mathrm{force}}^\sigma(I,F)$ and any strategy $f^{1-\sigma}$ of the opponent, $\mathcal{P}_{\underline{\mathrm{force}}^\sigma(I,F),f^{1-\sigma}}(s \models \Diamond Q) \geq \epsilon_s$. Observe that $\forall i. D_i \cap E_i = \emptyset$.

The base case $s \in (D_0 \cup E_0) \subseteq F$ holds trivially (take $\epsilon_s := 1$).

Now assume that the claim holds for $i \in \mathbb{N}$. Consider $s$ to be in $D_{i+1} \cup E_{i+1}$. In the case $s \in (D_i \cup E_i)$ the claim already holds by induction hypothesis. The remaining cases are described below.

**Case $s \in D_{i+1} - D_i$:** This implies that $s \in \mathrm{Pre}^R(E_i)$. Thus there is a state $s' \in E_i$ such that $s \longrightarrow s'$ and $\epsilon_{s'} > 0$ by induction hypothesis. We define $\epsilon_s := P(s,s') * \epsilon_{s'} > 0$.

[2] The weaker statement, i.e., $\mathcal{P}_{\underline{\mathrm{force}}^\sigma(I,F)}(s \models \Diamond F) > 0$, suffices for correctness. However, this stronger version is needed in the sequel.

**Case** $s \in E_{i+1} - E_i$**:** This implies one of the following two cases.

- If $s \in [S]^\sigma$ then $s \in \mathrm{Pre}^\sigma(D_i)$. Thus there is a state $s' \in D_i$ which is chosen as successor state to $s$ by the $\underline{\mathrm{force}}^\sigma(I,F)$ strategy, i.e., $s \longrightarrow s'$ and $s' = \underline{\mathrm{force}}^\sigma(I,F)(s)$. By induction hypothesis $\epsilon_{s'} > 0$. So we obtain $\epsilon_s := \epsilon_{\underline{\mathrm{force}}^\sigma(I,Q)(s)} = \epsilon_{s'} > 0$.
- If $s \in [S]^{1-\sigma}$ then $s \in \widetilde{\mathrm{Pre}}^{1-\sigma}(D_i)$. It follows that $\mathrm{Post}(s) \subseteq D_i$. The set $\mathrm{Post}(s)$ is finite, since the system is finitely branching. Furthermore, by induction hypothesis, $\epsilon_{s'} > 0$ for all $s' \in D_i$. Thus we obtain $\epsilon_s := \min_{s' \in \mathrm{Post}(s)}(\epsilon_{s'}) > 0$.

The main result follows since for any $s \in \mathrm{Force}^\sigma(I,Q)$, there exists a finite minimal $i \in \mathbb{N}$ such that $s \in (D_i \cup E_i)$. □

In the sequel, we use $\mathrm{Force}^\sigma(F)$ to denote $\mathrm{Force}^\sigma(S,F)$, i.e., we do not mention $I$ in case it is equal to $S$. We define $\underline{\mathrm{force}}^\sigma(F)$ analogously.

## 5 Büchi-Games on GPLCS

In this section we consider the BÜCHI-GPLCS problem. We give a scheme for characterizing the winning sets in almost-sure Büchi games, and then instantiate the scheme for GPLCS. In a similar manner to Section 4, we first show that the scheme always terminates for GPLCS, and then describe the winning sets using a symbolic representation based on regular languages. Again, the symbolic representation is valid under the assumption that the set of final states is also regular. We show the correctness of the construction by describing the memoryless winning strategies. Observe that this implies that BÜCHI-GPLCS are memoryless determined and solvable. Throughout this section, we fix a GPLCS $\mathcal{L} = \big(\mathtt{S}, \mathtt{S}^0, \mathtt{S}^1, \mathtt{C}, \mathtt{M}, \mathtt{T}, \lambda\big)$ and the induced game $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$. Take $F \subseteq S$; we consider the Büchi goal for player 0 consisting in visiting $F$ infinitely often.

*Scheme.* We define a sequence $\{X_i\}_{i\in\mathbb{N}} : X_0 \subseteq X_1 \subseteq \cdots$ of sets of states which are winning for player 1 with a positive probability. In the definition of $\{X_i\}_{i\in\mathbb{N}}$, we use an auxiliary sequence $\{M_i\}_{i\in\mathbb{N}} : M_0 \supseteq M_1 \supseteq \cdots$ of sets of states. The construction is inductive where $X_0 := \emptyset$, $M_0 := S$ and

$$M_{i+1} := \mathrm{Force}^0(\overline{X_i}, F) \qquad\qquad X_{i+1} := \mathrm{Force}^1(\overline{M_{i+1}})$$

for each $i \geq 0$. Intuitively, the set $X_i$ consists of states "already classified as losing for player 0". We add states iteratively to these sets. We define $M_{i+1}$ such that $\overline{M_{i+1}}$ is the set of states where player 0 cannot reach $F$ with positive probability while staying always in $\overline{X_i}$. Finally, we claim that the winning states for player 0 are given by $W^0 := \bigcap_{i\geq 0} M_i$, and thus complementarily, the winning states for player 1 are given by $W^1 := \overline{W^0} = \bigcup_{i\geq 0} X_i$.

This property holds by the definitions and will be used later in this section.

**Lemma 4.** $X_0 \subseteq \overline{M_1} \subseteq X_1 \subseteq \overline{M_2} \subseteq X_2 \subseteq \cdots$

The following lemma shows that this construction terminates.

**Lemma 5.** *The sequence $\{X_i\}_{i\in\mathbb{N}}$ converges for any set $F \subseteq S$ of states.*

*Proof.* (Sketch; details in [2]) Consider the sequence in Lemma 4. We perform the proof in four steps; namely, we show that (i) there is a $K$ such that $[X_K]^R = [X_{K+1}]^R$; (ii) $X_{K+1} = \overline{M_{K+1}}$; (iii) $M_{K+1} = M_{K+2}$; (iv) $X_{K+1} = X_{K+2}$.

(i) We show that each $[X_i]^R$ is upward closed, using induction on $i$. The base case is trivial since $X_0 = \emptyset$. For the induction step we let $Y := \left[\overline{M_{i+1}}\right]^{01} \cup [X_i]^R$. Using the definitions of $X_i$, $X_{i+1}$, and $M_{i+1}$, it can be shown that $X_{i+1} = \text{Force}^1(Y)$. Since $[X_i]^R$ is upward closed by the induction hypothesis it follows by Lemma 2(2) that $[X_{i+1}]^R$ is upward closed. From this and well quasi-ordering of $\preceq$, we get $\exists K.\, [X_K]^R = [X_{K+1}]^R$. We will use $K$ in the rest of the analysis below.

(ii) From Lemma 4 and the fact that $[X_K]^R = [X_{K+1}]^R$, we know that $[X_K]^R = \left[\overline{M_{K+1}}\right]^R = [X_{K+1}]^R$. This is used to show that $\text{Pre}^R(\overline{M_{K+1}}), \text{Pre}^1(\overline{M_{K+1}}), \widetilde{\text{Pre}}^0(\overline{M_{K+1}}) \subseteq \overline{M_{K+1}}$ which by the definition of $X_{K+1}$ implies $X_{K+1} \subseteq \overline{M_{K+1}}$. Hence, $X_{K+1} = \overline{M_{K+1}}$, by Lemma 4.

(iii) Since $M_{K+2} = \text{Force}^0(\overline{X_{K+1}}, F)$ and $X_{K+1} = \overline{M_{K+1}}$, we have that $M_{K+2} = \text{Force}^0(M_{K+1}, F)$. From Lemma 2(4) and the fact that $M_{K+1} = \text{Force}^0(\overline{X_K}, F)$, it follows that $M_{K+2} = M_{K+1}$.

(iv) $X_{K+2} = \text{Force}^1(\overline{M_{K+2}}) = \text{Force}^1(\overline{M_{K+1}}) = X_{K+1}$. □

*Forms of Winning Sets.* From Lemma 2(1), it follows that if $F$ is regular then each $X_i$ and $M_i$ is regular. From Lemma 5 we get the following:

**Lemma 6.** *If $F$ is regular then $W^0$ and $W^1$ are regular.*

*Winning Strategy for Player* 1. We define a sequence $\{x_i\}_{i\in\mathbb{N}}$ of strategies for player 1, such that $x_0 \subseteq x_1 \subseteq \cdots$. For each $i$, the strategy $x_i : [X_i]^1 \to S$ is memoryless and winning for player 1 from states in $X_i$. The sequence $\{x_i\}_{i\in\mathbb{N}}$ converges to a memoryless strategy $w^1 := \bigcup_{i\in\mathbb{N}} x_i$ for player 1 which is winning from states in $W^1$. We define the sequence using induction on $i$. We will also motivate why the strategy is winning for player 1. Define $x_0 := \emptyset$. For all $i \geq 0$, we define $x_{i+1}(s)$ by case analysis. By Lemma 4, we know that $X_i \subseteq \overline{M_{i+1}} \subseteq X_{i+1}$. There are three cases, reflecting the membership of $s$ in these three sets:

(i) If $s \in X_i$ then $x_{i+1}(s) := x_i(s)$. Here, we know by the induction hypothesis that a winning strategy $x_i$ for player 1 has already been defined in $s$.

(ii) If $s \in \overline{M_{i+1}} - X_i$ then $x_{i+1}(s) := \mathit{select}(\text{Post}(s) \cap \overline{M_{i+1}})$. The idea is that player 1 uses a strategy which guarantees that any run either (A) will stay in $\overline{M_{i+1}} - X_i$; or (B) will eventually enter $X_i$. In (A), player 1 wins since $\overline{M_{i+1}} - X_i$ does not have any states in $F$ by the definition of $M_{i+1}$. In (B), player 1 wins by the induction hypothesis.

More precisely, we observe that player 1 selects a successor of $s$ which belongs to $\overline{M_{i+1}}$. Such a successor exists by the following argument. First, observe that (by set operations) $\overline{M_{i+1}} - X_i = \overline{X_i} - M_{i+1}$. The result follows

by instantiating Lemma 2(3) with $I = \overline{X_i}$ and $Q = M_{i+1}$. By the same argument, for each $s' \in \left[\overline{M_{i+1}}\right]^R \cup \left[\overline{M_{i+1}}\right]^0$, all successors of $s'$ belong to $\overline{M_{i+1}}$. This guarantees that either (A) or (B) holds.

(iii) If $s \in X_{i+1} - \overline{M_{i+1}}$ then $x_{i+1}(s) := \underline{\text{force}}^1(\overline{M_{i+1}})(s)$. Since, by definition, $X_{i+1} = \text{Force}^1(\overline{M_{i+1}})$, player 1 can use $\underline{\text{force}}^1(\overline{M_{i+1}})$ to take the game with a positive probability to $\overline{M_{i+1}}$ (Lemma 3). From there, player 1 wins as described above.

Now, consider a state $s \in W^1$. By definition, we know that $s \in X_i$ for some $i \geq 0$. This means that $w^1 = x_i$ is winning for player 1 from $s$ according to the above argument. Hence:

**Lemma 7.** *For each $s \in W^1$, $\mathcal{P}_{w^1}(s \models \neg\Box\Diamond F) > 0$.*

*Winning Strategy for Player* 0. In this paragraph, we define a memoryless strategy $w^0$ and we prove that it is winning.

To describe how $w^0$ is defined, we rely on two auxiliary results on games induced by GPLCS. First, we recall the definition of *an attractor*. A set $A \subseteq S$ is called an attractor if $\mathcal{P}(s \models \Diamond A) = 1$ for any $s \in S$. In other words, from any state $s \in S$, $A$ is almost surely visited regardless of the strategies of the players. The following result was shown in [11,7,4] for *probabilistic LCS*, where moves in the control graph are taken probabilistically instead of by two competing players. The results straightforwardly generalize to GPLCS.

**Lemma 8.** *Let $\mathcal{L} = \left(\mathtt{S}, \mathtt{S}^0, \mathtt{S}^1, \mathtt{C}, \mathtt{M}, \mathtt{T}, \lambda\right)$ be a GPLCS and let $\mathcal{G}$ be the game induced by $\mathcal{L}$. The set $A = (\mathtt{S} \times \boldsymbol{\varepsilon} \times \{0,1\})$ is a finite attractor in $\mathcal{G}$.*

The second result follows from Lemma 8 and Lemma 3 as described below.

**Lemma 9.** *Let $\mathcal{G} = (S, S^0, S^1, S^R, \longrightarrow, P)$ be a game induced by a GPLCS. For any $Q, I \subseteq S$ and $\sigma \in \{0,1\}$, the following holds: For any $s \in \mathit{Force}^\sigma(I,Q)$, $\mathcal{P}_{\underline{\mathit{force}}^\sigma(I,Q)}(s \models \Box \mathit{Force}^\sigma(I,Q) \wedge \neg\Box\Diamond Q) = 0$.*

*Proof.* Given $Q, I \subseteq S$, $\sigma \in \{0,1\}$, and $s \in \text{Force}^\sigma(I,Q)$. We assume that player $\sigma$ uses an extension $f^\sigma$ of the $\underline{\text{force}}^\sigma(I,Q)$ strategy and player $(1-\sigma)$ uses a strategy $f^{1-\sigma}$. By Lemma 8, the game has a finite attractor $A$. By definition of the attractor, almost all runs must visit $A$ infinitely often. We define $A' := A \cap \text{Force}^\sigma(I,Q)$.

If $A' = \emptyset$, then $\mathcal{P}_{f^\sigma, f^{1-\sigma}}(s \models \Box\text{Force}^\sigma(I,Q) \wedge \neg\Box\Diamond Q) \leq \mathcal{P}_{f^\sigma, f^{1-\sigma}}(s \models \neg\Diamond A) = 0$, where the inequality follows from the assumption and the equality from the definition of an attractor.

Consider now the case where $A' \neq \emptyset$. By Lemma 3 and finiteness of $A$ (and thus $A'$), we obtain that $\epsilon := \min_{s' \in A'}(\epsilon'_s) > 0$. Almost every run in $(s \models \Box\text{Force}^\sigma(I,Q) \wedge \neg\Box\Diamond Q)$ must visit $A'$ infinitely many times, but $Q$ only finitely many times (and thus have an infinite suffix which never visits $Q$). Thus,

$$\mathcal{P}_{f^\sigma, f^{1-\sigma}}(s \models \Box\text{Force}^\sigma(I,Q) \wedge \neg\Box\Diamond Q) \leq (1-\epsilon)^\infty = 0. \tag{1}$$

Therefore $\mathcal{P}_{\underline{\text{force}}^\sigma(I,Q)}(s \models \Box\text{Force}^\sigma(I,Q) \wedge \neg\Box\Diamond Q) = 0$. □

*Remark 1.* Observe that the inequality (1) holds for any strategy $f^{1-\sigma}$ of the opponent and any extension $f^{\sigma}$ of the $\underline{\text{force}}^{\sigma}(I,Q)$ strategy. In particular, we do **not** require that $f^{1-\sigma}$ is finite-memory. It is possible that $f^{1-\sigma}$ acts quite differently after each of the (possibly infinitely many) visits to the same state in the attractor. The crucial fact is that the quantity $\epsilon_s > 0$ in Lemma 3 is independent of $f^{1-\sigma}$.

Now we are ready to describe the winning strategy $w^0$ for player 0. The idea of the strategy $w^0$ is to keep the run within a force set of $F$ with probability 1. This implies that $F$ will be visited infinitely often with probability 1, by Lemma 9. In order to do that, player 0 exploits certain properties of $W^0$: By Lemmas 5 and 4, there is an $i$ such that $X_i = \overline{M_{i+1}} = X_{i+1}$. From this and the definition of $W^0$ it follows that $W^0 = \overline{X_i} = M_{i+1}$. From $W^0 = \overline{X_i}$ and Lemma 2(5) it follows that $W^0$ is a 1-trap. From $M_{i+1} = \text{Force}^0(\overline{X_i}, F)$, it follows that $W^0 = \text{Force}^0(W^0, F)$. We define $w^0$ on any state $s \in \left[W^0\right]^0$ as follows:

- If $s \in W^0 - F$ then $w^0(s) := \underline{\text{force}}^0(W^0, F)(s)$. This definition is possible since $W^0 = \text{Force}^0(W^0, F)$.
- If $s \in W^0 \cap F$ then $w^0(s) := \mathit{select}(\text{Post}(s) \cap W^0)$. This is possible since $W^0$ is a 1-trap, and therefore $s$ has at least one successor in $W^0$.

Consider any run $\rho$ starting from a state inside $W^0$, where player 0 follows $w^0$. Since $W^0$ is a 1-trap, $\rho$ will always remain inside $W^0$ regardless of the strategy of player 1. This implies that $\mathcal{P}_{w^0}(s \models \Box W^0) = 1$. Furthermore, by Lemma 9 and the definitions of $w^0$ and $W^0$, it follows that $\mathcal{P}_{w^0}(s \models \Box W^0 \wedge \neg\Box\Diamond F) = 0$, which gives the following lemma:

**Lemma 10.** *For any* $s \in W^0$, $\mathcal{P}_{w^0}(s \models \Box\Diamond F) = 1$.

*Determinacy and Solvability.* Memoryless determinacy of almost-sure Büchi-GPLCS follows from Lemmas 7 and 10. By Lemma 6, for any state $s \in S$, we can check whether $s \in W^0$ or $s \in W^1$. This gives the main result:

**Theorem 1.** *Büchi-GPLCS are memoryless determined and solvable, for any regular target set* $F$.

## 6 Conclusions and Future Work

We have introduced GPLCS and given a terminating algorithm to compute symbolic representations of the winning sets in almost-sure Büchi-GPLCS. The strategies are memoryless, and our construction implies that the games we consider are memoryless determined.

The problem of deciding GPLCS games is not primitive recursive, since it is harder than the control-state reachability problem for LCS, which was shown to be non-primitive recursive by Schnoebelen in [25]. (For a given LCS and control-state $q$ we can construct a GPLCS by defining $\mathtt{S}^0 = \emptyset$, $\mathtt{S}^1 = \mathtt{S}$, making the state $q$ absorbing and defining $F$ as all configurations where the control-state is

not $q$. Then player 1 has a winning strategy in the GPLCS iff control-state $q$ is reachable in the LCS.)

There are five immediate extensions of our result. (1) Each winning strategy $w^0, w^1$ wins against any *mixed* strategy of the opponent (i.e. the opponent chooses a probability distribution over the successor states rather than one of them). (2) Our algorithm is easily adapted to almost-sure *reachability*-GPLCS. This is achieved by replacing all outgoing transitions of states in $F$ by self-loops, or, equivalently, replacing the definition of $X_{i+1}$ by $X_{i+1} := \text{Force}^1(\overline{F}, \overline{M_{i+1}})$. (3) Our algorithm can be modified to construct symbolic representations of the winning strategies. A strategy is represented as a finite set $\{L_i, L'_i\}_{i=0}^n$ of pairs of regular state languages, where all $L_i$ are disjoint. Such a finite set represents the strategy $f^0$ where $f^0(s) = \mathit{select}(L'_i)$ if $s \in L_i$. (4) We can extend the scheme to *concurrent* games, where the two players move simultaneously, by an appropriate extension of the Pre operator, as in [15]. (5) The algorithm also works when there are *probabilistic* control states in the GPLCS (see, e.g., [1] for definitions), as well as control states owned by the players and probabilistic message losses.

## References


1. Abdulla, P.A., Baier, C., Iyer, P., Jonsson, B.: Reasoning about probabilistic lossy channel systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 320–333. Springer, Heidelberg (2000)
2. Abdulla, P.A., Ben Henda, N., Mayr, R., Sandberg, S., de Alfaro, L.: Stochastic games with lossy channels. Technical Report 2007-005, Dept. of Information Technology, Uppsala University, Sweden (February 2007)
3. Abdulla, P.A., Bouajjani, A., d'Orso, J.: Deciding monotonic games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 1–14. Springer, Heidelberg (2003)
4. Abdulla, P.A., Henda, N.B., Mayr, R.: Verifying infinite Markov chains with a finite attractor or the global coarseness property. In: Proc. LICS 2005, 21st IEEE Int. Symp. on Logic in Computer Science, pp. 127–136 (2005)
5. Abdulla, P.A., Henda, N.B., Mayr, R., Sandberg, S.: Eager Markov chains. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 24–38. Springer, Heidelberg (2006)
6. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
7. Abdulla, P.A., Rabinovich, A.: Verification of probabilistic systems with faulty communication. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 39–53. Springer, Heidelberg (2003)
8. Asarin, E., Collins, P.: Noisy Turing machines. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1031–1042. Springer, Heidelberg (2005)
9. 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)
10. Baier, C., Bertrand, N., Schnoebelen, P.: Verifying nondeterministic probabilistic channel systems against $\omega$-regular linear-time properties. ACM Transactions on Comp. Logic (to appear, 2006)

11. Bertrand, N., Schnoebelen, P.: Model checking lossy channels systems is probably decidable. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 120–135. Springer, Heidelberg (2003)
12. Chatterjee, K., Jurdziński, M., Henzinger, T.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
13. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
14. Condon, A.: The complexity of stochastic games. Information and Computation 96(2), 203–224 (1992)
15. de Alfaro, L., Henzinger, T.: Concurrent omega-regular games. In: Proc. LICS 2000, 16th IEEE Int. Symp. on Logic in Computer Science, pp. 141–156. IEEE Computer Society Press, Los Alamitos (2000)
16. de Alfaro, L., Henzinger, T., Kupferman, O.: Concurrent reachability games. In: Proc. 39th Annual Symp. Foundations of Computer Science, pp. 564–575. IEEE Computer Society Press, Los Alamitos (1998)
17. de Alfaro, L., Henzinger, T., Majumdar, R.: Symbolic algorithms for infinite-state games. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 536–550. Springer, Heidelberg (2001)
18. Esparza, J., Kučera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: Proc. LICS 2004, 20th IEEE Int. Symp. on Logic in Computer Science, pp. 12–21 (2004)
19. 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)
20. Finkel, A.: Decidability of the termination problem for completely specified protocols. Distributed Computing 7(3), 129–135 (1994)
21. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. (3) 2(7), 326–336 (1952)
22. Iyer, P., Narasimha, M.: Probabilistic lossy channel systems. In: Bidoit, M., Dauchet, M. (eds.) TAPSOFT 1997. LNCS, vol. 1214, pp. 667–681. Springer, Heidelberg (1997)
23. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. D Van Nostad Co. (1966)
24. Rabinovich, A.: Quantitative analysis of probabilistic lossy channel systems. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1008–1021. Springer, Heidelberg (2003)
25. Schnoebelen, P.: Verifying lossy channel systems has nonprimitive recursive complexity. Information Processing Letters 83(5), 251–261 (2002)
26. Shapley, L.S.: Stochastic games. Proceedings of the National Academy of Sciences 39(10), 1095–1100 (1953)
27. Vardi, M.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. FOCS 1985, 26th Annual Symp. Foundations of Computer Science, pp. 327–338 (1985)
28. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200, 135–183 (1998)

# Simulation Hemi-metrics between Infinite-State Stochastic Games*

Jean Goubault-Larrecq

LSV, ENS Cachan, CNRS, INRIA Futurs 61, av. du président-Wilson, 94230 Cachan, France
goubault@lsv.ens-cachan.fr

**Abstract.** We investigate simulation hemi-metrics between certain forms of turn-based $2\frac{1}{2}$-player games played on infinite topological spaces. They have the desirable property of bounding the difference in payoffs obtained by starting from one state or another. All constructions are described as the special case of a unique one, which we call the Hutchinson hemi-metric on various spaces of continuous previsions. We show a directed form of the Kantorovich-Rubinstein theorem, stating that the Hutchinson hemi-metric on spaces of continuous probability valuations coincides with a notion of trans-shipment hemi-metric. We also identify the class of so-called sym-compact spaces as the right class of topological spaces, where the theory works out as nicely as possible.

## 1 Introduction

Given two (stochastic) transition systems, or two states in the same transition system, we may evaluate whether have the same behavior by testing whether they are bisimilar. A finer measure of closeness is obtained by computing *distances* between states, so that two states are at distance 0 if and only they are bisimilar—so-called *bisimulation distances*. This was pioneered for (infinite-state) labeled Markov processes (LMP) by Desharnais *et al.* [7]. One may see LMPs as turn-based stochastic $1\frac{1}{2}$-player games, where, at each state, one player chooses a probability distribution $p$ on states by its label, and the half-player picks the next state by drawing at random along $p$.

In [11] we explored so-called ludic transition systems, which are essentially LMPs, where the probability $p$ is replaced by a so-called *continuous game* $\nu$. When $\nu$ is a belief function, this naturally models a form of turn-based $2\frac{1}{2}$-player games, where player 1 chooses a continuous game $\nu$, then some state is drawn at random along $\nu$—the latter step being the same thing as a half-player picking at random some set from which player 2 picks non-deterministically, demonically. In such transition systems, a simple modal logic (that of Desharnais *et al.* [5], plus binary disjunction) was shown to characterize similarity. It is a natural question to extend the notion of bisimulation distance to "simulation metrics". This is what we do here, in the general case of (infinite-state) topological spaces and slightly more general *prevision* transition systems.

**Related Work.** Bisimulation metrics were explored for LMPs by Desharnais *et al.* [7], then extended to (finite state) systems mixing probabilistic and non-deterministic choice

* Partially supported by the INRIA ARC ProNoBis.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 50–65, 2008.

[6]. An elegant treatment of such was given by Ferns *et al.* [8], which also applies to infinite (measurable) state spaces. We use the topological (instead of measurable) setting of [11,12], which, as we hope to demonstrate again here, has a certain elegance. Notions of simulation rather than bisimulation are more natural here, and correspondingly, we shall develop hemi-metrics rather than metrics. Hemi-metrics were also considered recently by de Alfaro *et al.* [4]. Their paper is both more general than ours (we only consider certain forms of turn-based games) and less general (they only consider finite state spaces). Our last theorem (Theorem 7, non trivial) is that our hemi-metric coincides, in the finite case, with their a posteriori and a priori metrics in the demonic, resp. angelic case, respectively.

The main object of study of this paper is a hemi-metric we call the *Hutchinson hemi-metric* $d_{\mathrm{H}}$. It turns out to have many good properties, including a duality theorem à la Kantorovich-Rubinstein, and the fact that it generalizes several variants of the Hausdorff metric. The closest notion we know of is due to Baddeley [2], whose generalized Hausdorff metric generalizes the Levy-Prohorov metric on spaces of measures instead of the Hutchinson (or Kantorovich) metric. Also, Baddeley only considers $T_2$ topological spaces, despite admitting that it appears that the $T_0$ case is more interesting (which this paper should confirm). Our theorems relating $d_{\mathrm{H}}$ to variants of the Hausdorff metric in the case of classical powerdomains (Section 4) confirm those of Bonsangue *et al.* [3].

**Outline.** We need quite a lot of preliminaries. Well-known facts are recapped in Section 2, while the basic theory of hemi-metric spaces is laid out in Section 3. We develop Hausdorff-like hemi-metrics on classical powerdomains for demonic, angelic, and chaotic non-determinism in Section 4, so as to appreciate how $d_{\mathrm{H}}$ will generalize them later on. Section 5 introduces prevision transition systems, a natural generalization of our ludic transition systems [11], then defines $d_{\mathrm{H}}$, and shows how a hemi-metric computed from $d_{\mathrm{H}}$ bounds errors in payoff evaluations. Section 6 is the core of this paper, and develops the mathematical theory of $d_{\mathrm{H}}$. We conclude in Section 7.

## 2 Preliminaries

We work on general topological spaces $X$, and let $\mathcal{O}(X)$ be the lattice of open subsets of $X$. The notion of *continuous valuation* is a natural alternative to the more well-known notion of measure [16]. Taking the conventions of [11], a *game* $\nu$ on $X$ is a map from $\mathcal{O}(X)$ to $\mathbb{R}^+$ such that $\nu(\emptyset) = 0$, and which is *monotonic*, i.e., such that $\nu(U) \leq \nu(V)$ whenever $U \subseteq V$; $\nu$ is *modular* (resp., *convex*, resp. *concave*) iff $\nu(U \cup V) + \nu(U \cap V) = \nu(U) + \nu(V)$ (resp. $\geq$, resp. $\leq$) for all opens $U, V$. The terms supermodular and submodular are sometimes used in lieu of convex, concave. A modular game is called a *valuation*. A game $\nu$ is *continuous* iff $\nu(\bigcup_{i \in I} U_i) = \sup_{i \in I} \nu(U_i)$ for every directed family $(U_i)_{i \in I}$ of opens. A *(sub)probability valuation* $\nu$ is additionally such that $\nu$ is *(sub)normalized*, i.e., that $\nu(X) = 1$ ($\leq 1$). Continuous valuations extend to measures on the Borel $\sigma$-algebra of the topology, under mild assumptions [19], showing that the two notions are close.

Each topological space $X$ has a *specialization quasi-ordering* $\leq$: $x \leq y$ iff every open containing $x$ also contains $y$. $X$ is $T_0$ iff $\leq$ is an ordering, i.e., $x \leq y$ and $y \leq x$ imply $x = y$. $X$ is $T_2$ iff for any two distinct elements $x$ and $y$, there are disjoint open

subsets containing $x$ and $y$ respectively. A subset $Q$ of $X$ is *compact* iff one may extract a finite subcover from every open cover of $Q$, and is *saturated* iff it is upward-closed.

Each poset can be equipped with the so-called *Scott topology*, whose opens are the upward-closed subsets $U$ such that, for every directed family $(x_i)_{i\in I}$ that has a least upper bound (a.k.a., a *sup*) in $U$, then $x_i \in U$ for some $i \in I$ already. The specialization ordering of the Scott topology is always the original ordering $\leq$. A *cpo* is a poset in which every directed family has a sup. A function $f$ from the poset $X$ to the poset $Y$ is Scott-continuous iff it is continuous for the respective Scott topologies, or equivalently, iff $f$ is monotonic and for every directed family $(x_i)_{i\in I}$ of elements of $X$ with a sup $x \in X$, the directed family $(f(x_i))_{i\in I}$ has $f(x)$ as sup. See [1,10,23] for background material on domain theory and topology.

Let $\langle X \to \mathbb{R}^+\rangle$ be the space of bounded continuous function from $X$ to $\mathbb{R}^+$, with the Scott topology of the pointwise ordering; $\mathbb{R}^+$ is equipped with its Scott-topology, whose non-trivial opens are the open intervals $(r, +\infty)$, $r \in \mathbb{R}^+$. The *Choquet integral* of $f \in \langle X \to \mathbb{R}^+\rangle$ along the continuous game $\nu$, which we shall write $\oint_{x\in X} f(x)d\nu$, is the Riemann integral $\int_0^{+\infty} \nu(f^{-1}(t, +\infty))dt$. There is a more complex formula defining the Choquet integral of functions $f$ taking values in $\mathbb{R}$ rather than $\mathbb{R}^+$ [11], but we shall only note that this can be alternatively defined by $\oint_{x\in X} f(x)d\nu = -a\nu(X) + \oint_{x\in X}(f(x) + a)d\nu$ for some arbitrary $a \geq -\inf_{x\in X} f(x)$. The Choquet integral is Scott-continuous and linear in $\nu$, Scott-continuous and *positively homogeneous* ($\oint_{x\in X} af(x)d\nu = a\oint_{x\in X} f(x)d\nu$ for every $a \in \mathbb{R}^+$) in $f$. It is not in general additive in $f$ (i.e., $\oint_{x\in X}(f(x) + g(x))d\nu$ is not in general equal to $\oint_{x\in X} f(x)d\nu + \oint_{x\in X} g(x)d\nu$), unless $\nu$ is a valuation. The *Dirac valuation* $\delta_x$ maps each open $U$ to 1 if $x \in U$, to 0 otherwise: then $\oint_{x'\in X} f(x')d\delta_x = f(x)$. (See [11].)

For each continuous map $f : X \to Y$, and game $\nu$ on $X$, the *image*, a.k.a. *push-forward* game $f[\nu]$ on $Y$ is defined by $f[\nu](V) = \nu(f^{-1}(V))$ for every $V \in \mathcal{O}(Y)$. Then $f[\nu]$ is convex, concave, modular, continuous respectively as soon as $\nu$ is, and $\oint_{y\in Y} g(y)df[\nu] = \oint_{x\in X} g(f(x))d\nu$. For any game $\nu$ on $X \times Y$, call $\pi_1[\nu]$ and $\pi_2[\nu]$ the first and second *marginals* of $\nu$, where $\pi_1$ and $\pi_2$ are the first and second projections onto $X$, resp. $Y$.

A *continuous prevision* $F$ on a topological space (e.g., a cpo) $X$ is a Scott-continuous map from $\langle X \to \mathbb{R}^+\rangle$ to $\mathbb{R}^+$ such that $F(af) = aF(f)$ for every $a \in \mathbb{R}^+$ (*positive homogeneity*). (The term "prevision" refers to Walley [32], as explained in [12].) A prevision $F$ is *lower* iff $F(h + h') \geq F(h) + F(h')$ for every $h, h'$, *upper* iff $F(h + h') \leq F(h) + F(h')$ for every $h, h'$, *linear* iff $F(h + h') = F(h) + F(h')$, *normalized* iff $F(a + h) = a + F(h)$ for every function $h$ and constant $a \in \mathbb{R}^+$, *subnormalized* iff $F(a + h) \leq a + F(h)$ for every $h$ and constant $a$. The integration functional $\alpha_{\mathcal{C}}$, defined as $\alpha_{\mathcal{C}}(\nu) = \lambda h \in \langle X \to \mathbb{R}^+\rangle \cdot \oint_{x\in X} h(x)d\nu$, maps continuous (resp., and convex, concave, modular) games to continuous previsions (resp., lower, upper, linear). Conversely, define $\gamma_{\mathcal{C}}(F)$ for any prevision $F$ as the game such that $\gamma_{\mathcal{C}}(F)(U) = F(\chi_U)$ for each open $U$, where $\chi_U$ is the (continuous) map sending every $x \in U$ to 1, and every $x \notin U$ to 0. Then $\alpha_{\mathcal{C}}$ and $\gamma_{\mathcal{C}}$ define an isomorphism between the space $\mathbf{V}_1(X)$ (resp., $\mathbf{V}_{\leq 1}(X)$) of continuous (sub)probability valuations and the space $\mathbf{P}_1^{\triangle}(X)$ (resp., $\mathbf{P}_{\leq 1}^{\triangle}(X)$) of continuous (sub)normalized linear previsions, both ordered pointwise. (See [12].) We shall write $\bigtriangledown \mathbf{P}_1(X)$, resp. $\bigtriangleup \mathbf{P}_1(X)$, the space

of all normalized continuous lower (resp., upper) previsions on $X$. We write $\mathbf{F}_1(X)$ the space of all normalized *forks*, where a *fork* is a pair $(F^-, F^+)$ of a continuous lower prevision $F^-$ and a continuous upper prevision $F^+$ satisfying *Walley's condition* $F^-(h+h') \leq F^-(h) + F^+(h') \leq F^+(h+h')$, for every $h, h' \in \langle X \to \mathbb{R}^+ \rangle$; a fork is *normalized* iff both $F^-$ and $F^+$ are. While $\bigtriangledown \mathbf{P}_1(X)$ (resp., $\bigtriangleup \mathbf{P}_1(X)$) is an adequate model of mixed probabilistic and demonic (resp., angelic) non-deterministic choice, $\mathbf{F}_1(X)$ is one of probabilistic and chaotic non-deterministic choice [12,15].

On spaces of previsions, the *weak topology* plays an important role [12,15]. On any space of previsions $Y$ over $X$, this is the least one containing the subbasic opens $[f > r] = \{F \in Y | F(f) > r\}$, $f \in \langle X \to \mathbb{R}^+ \rangle$, $r \in \mathbb{R}$. Similarly, the weak topology on spaces $Y$ of games on $X$ [11] has as subbasic open sets $[f > r] = \{\nu \in Y | \oint_{x \in X} f(x) d\nu > r\}$. This coincides with the product topology, whose subbasic open sets are $[U > r] = [\chi_U > r] = \{\nu \in Y | \nu(U) > r\}$, for each open subset $U$ of $X$.

A *hemi-metric* $d$ on $X$ is a function from $X \times X$ to $\overline{\mathbb{R}^+} = \mathbb{R}^+ \cup \{+\infty\}$ such that $d(x,x) = 0$ for every $x \in X$, and satisfying the *triangular inequality*: $d(x,y) \leq d(x,z) + d(z,y)$ for every $x, y, z \in X$. A *metric* also satisfies $d(x,y) = 0 \Rightarrow x = y$, and enjoys symmetry: $d(x,y) = d(y,x)$. Names vary in the literature: Hemi-metrics (not taking the value $+\infty$) are called directed metrics by de Alfaro *et al.* [4], semi-metrics by Nachbin [25] (although semi-metrics tend to refer nowadays to symmetric hemi-metrics), generalized metrics by Bonsangue *et al.* [3], and just metrics by Lawvere [22]. *Quasi-metrics* refer to $T_0$ hemi-metrics. Here are a few basic properties [13, Appendix A]. Every hemi-metric $d$ induces a topology $\mathcal{O}^d$, the smallest containing all *open balls* $B^d_{x,<\epsilon} = \{y \in X | d(x,y) < \epsilon\}$. As in the metric case, a subset $U$ is open in $\mathcal{O}^d$ iff for every $x \in U$, some open ball $B^d_{x,<\epsilon}$ is contained in $U$. The specialization quasi-ordering of $\mathcal{O}^d$ is such that $x \leq y$ iff $d(x,y) = 0$. So $\mathcal{O}^d$ is a $T_0$ topology iff $d(x,y) = d(y,x) = 0$ implies $x = y$; and if $d$ is a metric, then $\mathcal{O}^d$ is $T_2$. The canonical hemi-metric on $\mathbb{R}$, or $\overline{\mathbb{R}^+}$, is $\mathsf{d}_{\mathbb{R}}$ defined by $\mathsf{d}_{\mathbb{R}}(s,t) = \max(s-t, 0)$ (on $\overline{\mathbb{R}^+}$, we agree that $\mathsf{d}_{\mathbb{R}}(+\infty, +\infty) = 0$). For any two hemi-metric spaces $X, Y$ with hemi-metrics $d$ and $d'$ respectively, and any function $f : X \to Y$, $f$ is *continuous at* $x \in X$ iff for every $\epsilon > 0$, there is an $\eta > 0$ such that for any $x' \in X$ such that $d(x,x') < \eta$, $d'(f(x), f(x')) < \epsilon$. It is equivalent to require that $f$ is continuous, in the usual topological sense (the inverse image of every open is open), or to require that $f$ be continuous at every element $x$ of $X$. For any $c \in \mathbb{R}^+$, say that $f$ is *c-Lipschitz* iff $d(f(x), f(x')) \leq c d(x,x')$ for all $x, x' \in X$. 1-Lipschitz functions are sometimes called *non-expansive*. We say that $f$ is *Lipschitz* iff $f$ is $c$-Lipschitz for some $c$. Every Lipschitz function is continuous.

The hemi-metric $d$ is *bounded* iff there is fixed real $a$ such that $d(x,x') \leq a$ for all $x, x' \in X$. Every hemi-metric is *topologically equivalent* to a bounded one, i.e., the two hemi-metrics generate the same topology. The *opposite* hemi-metric $d^{op}$ is defined by $d^{op}(x,y) = d(y,x)$, and the *symmetrized* hemi-metric $d^{sym}$ is defined by $d^{sym}(x,y) = \max(d(x,y), d(y,x))$. The specialization quasi-ordering of $d^{op}$ is the opposite $\geq$ of that, $\leq$, of $d$ ($x \geq y$ iff $y \leq x$). We shall write $X^{op}$, $X^{sym}$ the space $X$ equipped with $d^{op}$, resp. $d^{sym}$. The topology of $X^{sym}$ is finer (has at least as many opens as) those of $X$ and $X^{op}$. A hemi-metric space $X$ is *totally bounded* [21] iff for every $\epsilon > 0$, there are finitely many elements $x_1, \ldots, x_n$ of $X$ such that $X = \bigcup_{i=1}^n B^{d^{sym}}_{x_i,<\epsilon}$. (Beware: $d^{sym}$,

not $d$.) We call $X$ *sym-compact* iff $X$ is $T_0$ and $X^{sym}$ is compact. Clearly, every sym-compact space is totally bounded. The converse fails, e.g., $X = \{1/(n+1) | n \in \mathbb{N}\}$ with the usual metric on the reals is totally bounded but not (sym-)compact.

## 3 More on Hemi-metric Spaces

There is an obvious duality in hemi-metric spaces: replace $d$ by $d^{op}$, getting $X^{op}$. Then $(X^{op})^{op} = X$. Another well-known duality, this time at the topological level, is given by Nachbin's (1948) theory of stably compact spaces and compact pospaces (see Jung [17] for an excellent introduction). Theorem 1 below shows that these two dualities match, in a precise sense, on sym-compact spaces.

First recall the theory of stably compact spaces [17]. A topological space $X$ is *stably compact* iff $X$ is $T_0$, *well-filtered* (for every filtered family $(Q_i)_{i \in I}$ of compact saturated subsets, for every open $U$, if $\bigcap_{i \in I} Q_i \subseteq U$ then $Q_i \subseteq U$ already for some $i \in I$), *locally compact* (whenever $x \in U$ with $U$ open, there is a compact saturated subset $Q$ such that $x \in int(Q) \subseteq Q \subseteq U$, where $int(Q)$ denotes the interior of $Q$), *coherent* (the intersection of any two compact saturated subsets is again so) and *compact*. When $X$ is stably compact, the *de Groot dual* $X^{\mathsf{d}}$ of $X$ is just $X$, only with the so-called *cocompact topology*, whose opens are the cocompacts, i.e., subsets of the form $X \setminus Q$, $Q$ compact saturated subset of $X$. Well-filternedness, coherence, and compactness imply that this is indeed a topology. Then $X^{\mathsf{d}}$ is again stably compact, and $X^{\mathsf{dd}} = X$. The theory relies on the study of the *patch space* $X^{\mathsf{patch}}$, which is $X$ with the *patch* topology (the least collection of opens containing both the topology of $X$ and that of $X^{\mathsf{d}}$). If $X$ is stably compact, then $X^{\mathsf{patch}}$ is compact, $T_2$, and the graph of $\leq$ is closed in $X \times X$, where $\leq$ is the specialization quasi-ordering of $X$: $(X^{\mathsf{patch}}, \leq)$ is a so-called *compact pospace*. Conversely, whenever $(X', \leq)$ is a compact pospace, the *upper space* $X$ of $X'$, defined as $X'$ with the topology consisting of just the $\leq$-upward-closed open subsets of $X'$, is stably compact, has $\leq$ as specialization ordering, and $X^{\mathsf{patch}} = X'$. Also, $X^{\mathsf{d}}$ is exactly the *lower space* of $X'$, i.e., $X'$ with the topology consisting of $\leq$-downward-closed opens of $X'$. This duality extends to the hemi-metric case:

**Theorem 1.** *Let $X$ be a $T_0$ hemi-metric space. Then $X$ is sym-compact iff: $(*)$ $X$, qua topological space, is stably compact, and $X^{op}$, qua topological space, coincides with $X^{\mathsf{d}}$. In this case, $(X^{sym}, \leq)$ is exactly the patch space of $X$, where $\leq$ is the specialization ordering of $X$.*

*Proof.* As noticed by an anonymous referee, this is well-known. We quote her/his argument. Using results by Kopperman [20] (to which we also refer for missing definitions), a $T_0$ hemi-metric space can be viewed as a special case of a $T_0$-quasi-uniform space $X$, just looking at the entourages determined by the $\varepsilon > 0$. The associated bitopological space $(X, \tau, \tau^*)$ is pairwise $T_2$. Kopperman's Theorem 3.7 then says that $X$ is compact with respect to the associated symmetric topology iff $(X, \tau, \tau^*)$ is joincompact. Joincompact here means that the $\tau^*$-closed sets are $\tau$-compact and that the $\tau$-closed sets are $\tau^*$-compact. And in Proposition 3.4(d) it is stated that in this situation $\tau$ and $\tau^*$ are mutual de Groot duals. A slightly more elementary proof can also be found in [13, Theorem 1]. □

We shall also use the function $d^2 : X \times X \to \overline{\mathbb{R}^+}$ defined by $d^2((x,y),(x',y')) = d(x,x') + d(y',y)$, where $d$ is a hemi-metric on $X$. It is easy to check that $d^2$ is a hemi-metric on $X \times X$. Let $X^{(2)}$ be $X \times X$, equipped with $d^2$. It is also easy to see that the hemi-metric $d$ is a 1-Lipschitz function from $X^{(2)}$ to $\overline{\mathbb{R}^+}$. However, the topology of $X^{(2)}$ is *not* the product topology of $X \times X$ in general [13, Appendix B]:

**Lemma 1.** *Let $X$ be equipped with a bounded hemi-metric $d$. The topology of $X^{(2)}$ is that of the topological product $X \times X^{op}$. Its specialization ordering is $\leq \times \geq$.*

Let $d(x,A)$, the distance of $x$ to a subset $A$ of $X$ be $\inf_{y \in A} d(x,y)$, taking this to be $+\infty$ when $A$ is empty. The function $x \mapsto d(x,A)$ is 1-Lipschitz in the sense that $d(x,A) \leq d(x,y) + d(y,A)$, and $d(x,A) = 0$ iff $x$ is in the topological closure $cl(A)$ of $A$ [13, Appendix C]. Using this, we may define the *thinning* $U^{d,-(\epsilon)} = \{x \in X | d(x, X \setminus U) > \epsilon\}$ of the open set $U$ by $\epsilon$: $U^{d,-(\epsilon)}$ is open, contained in $U$, grows larger as $\epsilon$ decreases, and for any family of non-negative reals $(\epsilon_i)_{i \in I}$ having 0 as inf, $U^{d,-(\epsilon_i)}$ is a directed family of opens whose union is $U$. We may also define the *thickening* of a subset $A$ of $X$ as $A^{d,+(\epsilon)} = \bigcup_{x \in A} B^d_{x,<\epsilon}$. This is always open. Moreover, $\left(U^{d,-(\epsilon)}\right)^{d,+(\epsilon)} \subseteq U$ for every open $U$ and $\epsilon > 0$.

## 4 Hemi-metrics on Powerdomains

It is standard to model non-determinism in domain theory through the use of *powerdomains* [1, Section 6.2]. The *Smyth powerdomain* $\mathcal{Q}(X)$ is the set of all non-empty compact saturated subsets $Q$ of $X$, ordered by reverse inclusion $\supseteq$. This models *demonic* non-determinism, $Q \in \mathcal{Q}(X)$ denoting the set of all possible choices of elements $x \in Q$. (The theory of [11] probably enlightens what "demonic" means, in the sense that the choice of $x \in Q$ is resolved by an *adversary* who picks some $x \in Q$ that pleases you least. Mathematically, $Q$ gives rise to the so-called unanimity game $\mathfrak{u}_Q$, which maps each $U$ containing $Q$ to 1, every other to 0; assuming you get $f(x)$ dollars if you pick $x$, your expected payoff will be the Choquet integral of $f(x)$ along $\mathfrak{u}_Q$, which is exactly $\min_{x \in Q} f(x)$—i.e., you will get the least possible amount of money.) The *Hoare powerdomain* $\mathcal{H}(X)$ is the set of all non-empty closed subsets $F$ of $X$, ordered by ordinary inclusion $\subseteq$, and models *angelic* non-determinism. (Each such $F$ gives rise to the *example game* $\mathfrak{e}_F$, mapping each open $U$ that meets $F$ to 1, every other open to 0; it is an easy exercise that the Choquet integral of $f(x)$ along $\mathfrak{e}_F$ equals $\sup_{x \in F} f(x)$—i.e., you get the most money you can.) The *Plotkin powerdomain* $\mathcal{P}\ell(X)$ is the set of all *lenses* $L$, where a lens is the non-empty intersection of a compact saturated subset $Q$ (which we can take equal to the upward-closure $\uparrow L$ of $L$) and a closed subset $F$ (which we can take equal to the topological closure $cl(L)$ of $L$) of $X$, ordered by the topological Egli-Milner ordering $\sqsubseteq_{\mathrm{EM}}$, defined by $L \sqsubseteq_{\mathrm{EM}} L'$ iff $\uparrow L \supseteq \uparrow L'$ and $cl(L) \subseteq cl(L')$.

These powerdomains are endowed with their Scott topology. More relevant topologies when $X$ is a general topological space, not just a cpo, are the *Vietoris* topologies. Let $\mathcal{Q}_V(X)$ be $\mathcal{Q}(X)$ with the smallest topology containing the basic opens $\Box U = \{Q \in \mathcal{Q}(X) | Q \subseteq U\}$, $U$ open in $X$, $\mathcal{H}_V(X)$ be $\mathcal{H}(X)$ with the smallest topology containing the subbasic opens $\Diamond U = \{F \in \mathcal{H}(X) | F \cap U \neq \emptyset\}$, and $\mathcal{P}\ell_V(X)$ be $\mathcal{P}\ell(X)$ with the smallest topology containing both $\Box U = \{L | L \subseteq U\}$ and $\Diamond U = \{L | L \cap U \neq$

$\emptyset\}$. When $X$ is well-filtered and locally compact, $\mathcal{Q}_V(X) = \mathcal{Q}(X)$. When $X$ is a continuous cpo, $\mathcal{H}_V(X) = \mathcal{H}(X)$, and when $X$ is also coherent, then $\mathcal{P}\ell_V(X) = \mathcal{P}\ell(X)$.

We observe that, when $X$ is a (nice enough) hemi-metric space, all these spaces can be equipped with hemi-metrics that generate the Vietoris topology. These hemi-metrics will be asymmetric variants of the well-known Hausdorff metric on $T_2$ spaces. We shall see later that these are special cases of the Hutchinson hemi-metric, to be defined later. Proofs can be found in [13, Appendix D].

**Proposition 1.** *Let $X$ be equipped with the hemi-metric $d$. The* Hausdorff-Smyth hemi-metric $d_{\mathcal{Q}}$ *on $\mathcal{Q}(X)$ is defined by $d_{\mathcal{Q}}(Q, Q') = \sup_{x' \in Q'} \inf_{x \in Q} d(x, x')$. The bounds are attained, i.e., $d_{\mathcal{Q}}(Q, Q') = \max_{x' \in Q'} \min_{x \in Q} d(x, x')$. This defines a hemi-metric, whose topology is exactly the Vietoris topology of $\mathcal{Q}_V(X)$.*

Observe also that the function $\eta_{\mathcal{Q}} : X \to \mathcal{Q}_V(X)$ mapping $x$ to $\uparrow x = \{y \in X | x \leq y\}$, is an *isometric embedding*, namely $d_{\mathcal{Q}}(\eta_{\mathcal{Q}}(x), \eta_{\mathcal{Q}}(x')) = d(x, x')$.

The case of the Hoare powerdomain requires us to assume $d$ to be totally bounded.

**Proposition 2.** *Let $X$ be equipped with the hemi-metric $d$. The* Hausdorff-Hoare hemi-metric $d_{\mathcal{H}}$ *on $\mathcal{H}(X)$ is defined by $d_{\mathcal{H}}(F, F') = \sup_{x \in F} \inf_{x' \in F'} d(x, x')$. This defines a hemi-metric, whose topology is finer than the Vietoris topology of $\mathcal{H}_V(X)$, and coincides with it as soon as $d$ is totally bounded.*

**Proposition 3.** *Let $X$ be equipped with the hemi-metric $d$. The* Hausdorff hemi-metric $d_{\mathcal{P}\ell}$ *on $\mathcal{P}\ell(X)$ is defined by $d_{\mathcal{P}\ell}(L, L') = \max(\sup_{x' \in L'} \inf_{x \in L} d(x, x'), \sup_{x \in L} \inf_{x' \in L'} d(x, x'))$. Then $d_{\mathcal{P}\ell}(L, L') = \max(d_{\mathcal{Q}}(\uparrow L, \uparrow L'), d_{\mathcal{H}}(cl(L), cl(L')))$, $d_{\mathcal{P}\ell}$ is a hemi-metric, its topology is finer than the Vietoris topology of $\mathcal{P}\ell_V(X)$, and coincides with it as soon as $d$ is totally bounded.*

Then $\eta_{\mathcal{H}} : X \to \mathcal{H}_V(X)$, which sends $x$ to $\downarrow x = cl(\{x\}) = \{y \in X | y \leq x\}$, and $\eta_{\mathcal{P}\ell} : X \to \mathcal{P}\ell_V(X)$, which sends $x$ to $(\uparrow x, \downarrow x)$, are isometric embeddings.

## 5 Prevision Transition Systems

In [11], we defined ludic transition systems on the state space $X$ by analogy with LMPs, as collections $\sigma$ of maps $\sigma_\ell$, where $\ell$ ranges over a finite set $\mathcal{L}$ of labels, where $\sigma_\ell : X \to \mathbf{J}_{\leq 1\ wk}(X)$, and where $\mathbf{J}_{\leq 1\ wk}(X)$ is the space of all continuous subnormalized games over the topological space $X$, with the weak topology. Intuitively, the system evolves from state $x \in X$ by letting one player $\mathsf{P}$ pick a label $\ell \in \mathcal{L}$, then the other player draws a next state at random along the continuous game $\sigma_\ell(x)$. When $\sigma_\ell(x)$ is a belief function, results from [11] imply that this second player can be thought of as one half-player picking some $Q \in \mathcal{Q}(X)$ at random, along some subprobability distribution $(\sigma_\ell(x))^*$, then the second player $\mathsf{C}$ picking the next state in a (demonically) non-deterministic way out of $Q$.

Since any continuous game $\nu$ can be seen as a continuous prevision, namely $\alpha_{\mathcal{C}}(\nu)$, we only define a larger class of transition systems by considering *prevision transition systems* (PrTS), which are collections $\pi$ of maps $\pi_\ell : X \to \mathbf{P}_{\leq 1\ wk}(X)$, where $\mathbf{P}_{\leq 1\ wk}(X)$ is the space of all continuous subnormalized previsions on $X$, with the

weak topology. This lends itself to a sleeker mathematical treatment. Call a PrTS lower, upper, normalized, when $\pi_\ell(x)$ is so, for every $\ell \in \mathcal{L}$ and $x \in X$. When $\pi_\ell(x)$ is normalized, for every $h \in \langle X \to \mathbb{R}\rangle$ (i.e., with values in $\mathbb{R}$, not $\mathbb{R}^+$), write $\widehat{\pi}_\ell(x)(h) = \pi_\ell(x)(h+a) - a \in \mathbb{R}$, for any constant $a \geq -\inf_{x\in X} h(x)$. Then $\widehat{\pi}_\ell(x)$ is monotonic, positively homogeneous, normalized, Scott-continuous, and lower, resp. upper when $\pi_\ell(x)$ is [15]. Note the similarity with the Choquet integral of functions in $\langle X \to \mathbb{R}\rangle$.

As in the theory of Markov decision processes and in [11], we may add rewards and discounts to PrTSes. Imagine P plays according to a finite-state program $\Pi$, i.e., an automaton with *internal states* $q, q'$ and transitions $q \xrightarrow{\ell} q'$. Let $r_{q\xrightarrow{\ell}q'} : X \to \mathbb{R}$ be a family of bounded continuous *reward* functions: we may think that $r_{q\xrightarrow{\ell}q'}(x)$ is the amount of money P gains if she fires her internal transition $q \xrightarrow{\ell} q'$, drawing the next state $y$ at random along $\pi_\ell(x)$. Let $\gamma_{q\xrightarrow{\ell}q'} \in (0,1]$ be a family of so-called *discounts*. Define the average payoff, starting from state $x$ when P is in its internal state $q$, by:

$$V_q(x) = \sup_{\ell, q'/q\xrightarrow{\ell}q'} \left[ r_{q\xrightarrow{\ell}q'}(x) + \gamma_{q\xrightarrow{\ell}q'} \widehat{\pi}_\ell(x)(V_{q'}) \right] \tag{1}$$

This is obtained from formula (4) of [11] by replacing the mean $\oint_{y\in X} V_{q'}(y) d\sigma_\ell(x)$, i.e., $\alpha_{\mathcal{C}}(\sigma_\ell(x))(V_{q'})$, by the more general term $\widehat{\pi}_\ell(x)(V_{q'})$. When $\pi$ is lower and normalized, by [12, Theorem 5], we obtain $\pi_\ell(x)(h) = \min_{G\in CCoeur_1(\pi_\ell(x))} G(h) = \min_{\substack{p\in \mathbf{V}_1(X)\\ \alpha_{\mathcal{C}}(p)\geq \pi_\ell(x)}} \oint_{y\in X} h(y)dp$ for all $h \in \langle X \to \mathbb{R}^+\rangle$. It follows that $\widehat{\pi}_\ell(x)(V_{q'}) = \min_{\substack{p\in \mathbf{V}_1(X)\\ \alpha_{\mathcal{C}}(p)\geq \pi_\ell(x)}} \oint_{y\in X} V_{q'}(y)dp$. Intuitively, P plays first, maximizing her gains, then C picks a distribution $p$ (from some set) to minimize gains, from which a next state $y$ is chosen at random. So C plays before random choice is effected, contrarily to the case of ludic transition systems, but as in other proposals [24,28]. As in [11], we have:

**Theorem 2.** *Assume $\pi$ is* standard*, i.e., $\pi_\ell(x)(\chi_X)$ is always either $0$ or $1$, and the set $\{x \in X | \pi_\ell(x)(\chi_X) = 0\}$ of* deadlock states *is open; or that $r_{q\xrightarrow{\ell}q'}(x) \geq 0$ for all $q$, $\ell$, $q'$, $x \in X$. Assume also that the rewards are* uniformly bounded*, i.e., there are $a, b \in \mathbb{R}$ with $a \leq r_{q\xrightarrow{\ell}q'}(x), \gamma_{q\xrightarrow{\ell}q'} \leq b$ for all $q$, $\ell$, $q'$, $x \in X$. Then (1) has a unique uniformly bounded solution in any of the following two cases:*

*[Finite Horizon] If all paths in $\Pi$ have bounded length.*
*[Discount] If there is a constant $\gamma \in (0,1)$ such that $\gamma_{q\xrightarrow{\ell}q'} \leq \gamma$ for every $q$, $\ell$, $q'$.*

*Proof.* (Sketch.) This is by induction on the length of paths in the Finite Horizon case. In the Discount case, let $\mathcal{V}$ be the operator that maps the family $\boldsymbol{V} = (V_q)_q$ of functions to the family of functions of $x$ indexed by $q$ defined as the right-hand side of (1). We let this operator work on families that are uniformly bounded in $q$, i.e., such that there is a unique interval $[a', b']$ such that $V_q(x) \in [a', b']$ for all $x \in X$, and internal state $q$. Then $\mathcal{V}$ is Scott-continuous, and $\gamma$-Lipschitz in the sense that $\mathcal{V}(\boldsymbol{V})_q(x) - \mathcal{V}(\boldsymbol{V'})_q(x) \leq \gamma . \sup_{\ell, q'/q\xrightarrow{\ell}q'} \sup_{y\in X} \mathsf{d}_{\mathbb{R}}(V_{q'}(y), V'_{q'}(y)) \leq \gamma . \sup_{q',y\in X} \mathsf{d}_{\mathbb{R}}(V_{q'}(y), V'_{q'}(y))$. Since $\gamma < 1$, this implies that any two uniformly bounded solutions $\boldsymbol{V}$ and $\boldsymbol{V'}$ to (1) must be such that $z \leq \gamma z$, where $z = \sup_{q',y\in X} \mathsf{d}_{\mathbb{R}}(V_{q'}(y), V'_{q'}(y))$, so $z = 0$, whence

$V_{q'} \leq V'_{q'}$ for all $q'$, and by symmetry, $V_{q'} = V'_{q'}$ for all $q'$. Then start with $V_q(x)$ defined as some constant $\alpha \leq 0$ for all $q$, sufficiently low that $\mathcal{V}(\boldsymbol{V})_q(x) \geq V_q(x)$ for all $q$ and $x \in X$, i.e., $\alpha \leq \frac{\min(a,0)}{(1-\gamma)}$. Then $(\mathcal{V}^n(\boldsymbol{V}))_{n\in\mathbb{N}}$ is an increasing chain, and converges to some family, since $\mathcal{V}$ is Scott-continuous and $\gamma$-Lipschitz. Since rewards are uniformly bounded, it is easy to see that each iterate, as well as the limit, is, too. □

It is interesting to check whether two states $x, y$ are close to each other in some metric that would inform us about the difference between the payoffs $V_q(x)$ and $V_q(y)$. Ferns *et al.* [8] introduce a nice bisimulation metric, which is computed as a fixed point. Probabilistic distributions are compared in the so-called Hutchinson (or Kantorovich) metric, while sets of non-deterministic choices are compared in the Hausdorff metric.

We introduce a similar construction, with the following differences. First, we work with hemi-metrics instead of metrics, and this will provide us a measure of *simulation*, not bisimulation. Second, we generalize the Hutchinson (hemi-)metric to work not just on probabilities, but on games and even on previsions. This way, the generalized notion will actually encompass both the Hutchinson and the Hausdorff (hemi-)metrics, as used in [8]; this encompassment is the first part of our Theorem 7 (see later).

For any two continuous games $\nu$ and $\nu'$ on $X$, define the *Hutchinson hemi-metric* $d_{\mathrm{H}}(\nu, \nu')$ as $\sup_{f\in\langle X\to\mathbb{R}^+\rangle_1} \mathsf{d}_{\mathbb{R}}\left(\oint_{x\in X} f(x)d\nu, \oint_{x\in X} f(x)d\nu'\right)$, where $\langle X \to \mathbb{R}^+\rangle_1$ is the space of all bounded 1-Lipschitz functions from $X$ to $\mathbb{R}^+$. The classical Hutchinson *metric* uses $\mathsf{d}^{sym}_{\mathbb{R}}$ instead of $\mathsf{d}_{\mathbb{R}}$, i.e., would be the sup of $\left|\oint_{x\in X} f(x)d\nu - \oint_{x\in X} f(x)d\nu'\right|$. Note that our definition is valid for general continuous games, not just valuations.

We can, in turn, extend this to continuous previsions by $d_{\mathrm{H}}(F, F') = \sup_{f\in\langle X\to\mathbb{R}^+\rangle_1} \mathsf{d}_{\mathbb{R}}(F(f), F'(f))$, so that $d_{\mathrm{H}}(\alpha_{\mathcal{C}}(\nu), \alpha_{\mathcal{C}}(\nu')) = d_{\mathrm{H}}(\nu, \nu')$ for every continuous games $\nu$ and $\nu'$; and to forks by $d_{\mathrm{H}}((F^-, F^+),(F'^-, F'^+)) = \max(d_{\mathrm{H}}(F^-, F'^-), d_{\mathrm{H}}(F^+, F'^+))$.

Let $\mathcal{M}(X)$ be the space of all hemi-metrics on $X$. This is a complete lattice, ordered pointwise. Define the operator $\mathcal{D} : \mathcal{M}(X) \to \mathcal{M}(X)$ that maps $d$ to $\mathcal{D}(d)$ defined as $\mathcal{D}(d)(x, y) = \sup_{q,\ell,q'/q\xrightarrow{\ell}q'} (\mathsf{d}_{\mathbb{R}}(r_{q\xrightarrow{\ell}q'}(x), r_{q\xrightarrow{\ell}q'}(y)) + \gamma_{q\xrightarrow{\ell}q'} d_{\mathrm{H}}(\pi_\ell(x), \pi_\ell(y)))$. Let $d_\pi$ be the least fixed point of $\mathcal{D}$ on $\mathcal{M}(x)$. We check that $d_\pi$ bounds the discrepancy of payoffs $V_q(x)$, $V_q(y)$, starting from states $x$ and $y$.

**Proposition 4.** *Under the assumptions of Theorem 2, let $\boldsymbol{V} = (V_q)_q$ be the unique solution of (1). Then $V_q(x) - V_q(y) \leq d_\pi(x, y)$ for every $x, y \in X$.*

*Proof.* By induction on the length of paths in the Finite Horizon case. We deal with the more interesting Discount case. When $\boldsymbol{V}$ consists of constant functions, the inequality $(*)$ $V_q(x) - V_q(y) \leq d_\pi(x, y)$ for all $x, y$ is clear. If $(*)$ holds of $\boldsymbol{V}$, then we claim it holds for $\mathcal{V}(\boldsymbol{V})$. Let $\boldsymbol{V}' = \mathcal{V}(\boldsymbol{V})$. Since by $(*)$ $V_q$ is 1-Lipschitz wrt. $d_\pi$, by definition of $d_{\mathrm{H}}$ we obtain $\widehat{\pi}_\ell(x)(V_q) - \widehat{\pi}_\ell(y)(V_q) \leq (d_\pi)_{\mathrm{H}}(\pi_\ell(x), \pi_\ell(y))$. Also, $V'_q(x) = \sup_{\ell,q'/q\xrightarrow{\ell}q'} \left[r_{q\xrightarrow{\ell}q'}(x) + \gamma_{q\xrightarrow{\ell}q'}\widehat{\pi}_\ell(x)(V_{q'})\right]$. Now $r_{q\xrightarrow{\ell}q'}(x) + \gamma_{q\xrightarrow{\ell}q'}\widehat{\pi}_\ell(x)(V_{q'}) \leq r_{q\xrightarrow{\ell}q'}(y) + \mathsf{d}_{\mathbb{R}}(r_{q\xrightarrow{\ell}q'}(x), r_{q\xrightarrow{\ell}q'}(y)) + \gamma_{q\xrightarrow{\ell}q'}\widehat{\pi}_\ell(y)(V'_{q'}) + \gamma_{q\xrightarrow{\ell}q'}(d_\pi)_{\mathrm{H}}(\pi_\ell(x), \pi_\ell(y)) \leq r_{q\xrightarrow{\ell}q'}(y) + \gamma_{q\xrightarrow{\ell}q'}\widehat{\pi}_\ell(y)(V'_{q'}) + \mathcal{D}(d_\pi)(x, y)$. Taking sups over $\ell$ and $q'$, and since $\mathcal{D}(d_\pi) = d_\pi$, $(*)$ holds for $\boldsymbol{V}'$. So it holds of $\mathcal{V}^n(\boldsymbol{V})$ where $\boldsymbol{V}$ is constant, by induction on $n$. Hence it holds for the unique solution of (1). □

In other words, the payoff $V_q(x)$ at $q$ cannot exceed $V_q(y)$ by more than $d_\pi(x,y)$. The corresponding symmetrized distance $d_\pi^{sym}$ measures, as usual, the absolute value of the difference: $|V_q(x) - V_q(y)| \leq d_\pi^{sym}(x,y)$.

## 6 The Hutchinson Hemi-metric on Games, on Previsions

While our notion of Hutchinson hemi-metric extends a classical notion on measures, we said earlier that the Hutchinson hemi-metric would also extend hemi-metrics on powerdomains [13, Appendix E]. for a proof):

**Proposition 5.** *For all $Q, Q' \in \mathcal{Q}(X)$, $d_{\mathrm{H}}(\mathfrak{u}_Q, \mathfrak{u}_{Q'}) = d_{\mathcal{Q}}(Q,Q')$. For all $F, F' \in \mathcal{H}(X)$, $d_{\mathrm{H}}(\mathfrak{e}_F, \mathfrak{e}_{F'}) = d_{\mathcal{H}}(F,F')$. For every $L \in \mathcal{P}\ell(X)$, $F_L = (\alpha_{\mathcal{C}}(\mathfrak{u}_{\uparrow L}), \alpha_{\mathcal{C}}(\mathfrak{e}_{cl(L)}))$ is a normalized fork, and for all $L, L' \in \mathcal{P}\ell(X)$, $d_{\mathrm{H}}(F_L, F_{L'}) = d_{\mathcal{P}\ell}(L,L')$.*

It follows that the maps $Q \in \mathcal{Q}(X) \mapsto \alpha_{\mathcal{C}}(\mathfrak{u}_Q) \in \bigtriangledown \mathbf{P}_1(X)$, $F \in \mathcal{H}(X) \mapsto \alpha_{\mathcal{C}}(\mathfrak{e}_F) \in \bigtriangleup \mathbf{P}_1(X)$, and $L \in \mathcal{P}\ell(X) \mapsto F_L \in \mathbf{F}_1(X)$ are isometric embeddings, where the powerdomains are equipped with their respective variant of the Hausdorff hemi-metric, and spaces of previsions or forks are equipped with the Hutchinson hemi-metric.

We now note that the Hutchinson hemi-metric defines the weak topology on spaces of continuous normalized games—including $\mathbf{V}_1(X)$, the space of all continuous probability valuations. This is similar to a famous theorem in measure theory stating that the Hutchinson metric on the space of all probability measures over a Polish space has the weak topology (sometimes called the weak$^*$ or the narrow topology), defined as above.

**Theorem 3.** *Let $X$ be equipped with a hemi-metric $d$, $Y$ a space of continuous games over $X$. The topology of $d_{\mathrm{H}}$ on $Y$ is finer than the weak topology on $Y$, and coincides with it when $Y$ is a space of normalized games and $d$ is bounded and totally bounded.*

We omit the proof (see [13, Appendix F]). This requires defining Lipschitz approximations to the indicator functions $\chi_U$: for every $\epsilon > 0$, let $\chi_U^\epsilon(x) = \min(\epsilon, d(x, X \setminus U))$. Then $(\frac{1}{\epsilon}\chi_U^\epsilon)_{\epsilon>0}$ is a directed family of Lipschitz functions with $\chi_{U^{d,-(\epsilon)}} \leq \frac{1}{\epsilon}\chi_U^\epsilon \leq \chi_U$—in particular, has $\chi_U$ as sup. The proof also relies on another hemi-metric, the Levy-Prohorov hemi-metric $d_{\mathrm{LP}}$ (imitated from the classical Levy-Prohorov metric in measure theory), defined by $d_{\mathrm{LP}}(\nu,\nu') = \inf\{\epsilon > 0 | \forall U \in \mathcal{O}(X) \cdot \nu(U) \leq \nu'(U^{d,+(\epsilon)}) + \epsilon\}$. That $d_{\mathrm{LP}}$ is a hemi-metric depends on the fact that $(U^{d,+(\epsilon)})^{d,+(\epsilon')} \subseteq U^{d,+(\epsilon+\epsilon')}$ for every open $U$. We then show that whenever $a$ is an upper bound of $d$, then $d_{\mathrm{H}}(\nu,\nu') \leq (a+1)d_{\mathrm{LP}}(\nu,\nu')$, which implies that the topology of $d_{\mathrm{LP}}$ is finer than that of $d_{\mathrm{H}}$, and finally, that when $d$ is totally bounded, the weak topology is even finer than $d_{\mathrm{LP}}$. So all topologies coincide when $d$ is bounded and totally bounded.

Note also that the map $\eta_{\mathbf{V}} : X \to \mathbf{V}_1(X)$ sending $x$ to $\delta_x$ is (again) an isometric embedding, where $\mathbf{V}_1(X)$ is equipped with $d_{\mathrm{H}}$.

It is well-known that, on Polish spaces, the Hutchinson metric between two probability measures $p$, $p'$ coincides with the so-called *trans-shipment distance*, defined as the least value of $\oint_{(x,y)\in X\times X} d(x,y)dp^2$, where $p^2$ ranges over all probability measures on $X \times X$ having $p$ as first marginal and $p'$ as second marginal. This is the so-called

*Kantorovich-Rubinstein Theorem* (on compact metric spaces, this was first proved in [18]). When $X$ is finite, $p$ can be written $\sum_{x\in X} a_x\delta_x$, $p'$ can be written $\sum_{y\in X} a'_y\delta_y$, and $p^2 = \sum_{x,y\in X} b_{xy}\delta_{(x,y)}$. Interpreting $a_x$ (resp. $a'_x$) as some mass located at $x$, saying that $p^2$ should have $p$ and $p'$ as marginals means that we can distribute the mass $a_x$ at $x$ into small chunks $b_{xy}$ ($a_x = \sum_{y\in X} b_{xy}$) that will each contribute to give total mass $a'_y$ at $y$ ($a'_y = \sum_{x\in X} b_{xy}$). Then $\oint_{(x,y)\in X\times X} d(x,y)dp^2$ is the total amount of work $\sum_{x,y\in X} b_{xy}d(x,y)$ needed to move these chunks, where moving one unit of mass from $x$ to $y$ costs $d(x,y)$. The finite case is an easy case of duality in linear programming (see e.g., van Breugel and Worrell [31]; the subject has a long history [27]).

We prove a similar theorem for continuous probability valuations, in a non-$T_2$ setting. The main difficulty is that the hemi-metric $d$ is continuous, not from $X \times X$ to $\overline{\mathbb{R}^+}$, but from $X^{(2)} = X \times X^{op}$ to $\overline{\mathbb{R}^+}$, so that the integral $\oint_{(x,y)\in X\times X} d(x,y)dp^2$ is in fact *meaningless*. So $p^2$ should be a valuation on $X \times X^{op}$, but its second marginal would then be a valuation on $X^{op}$, while $p'$ is on $X$.

We solve the difficulty in two steps. First, we replace $p^2$ by a linear prevision on $X^{(2)}$—or rather something looking like it, but taking Lipschitz functions instead of continuous functions as arguments, which we call *LL-previsions* (for Linear, Lipschitz). Formally, an LL-prevision over $X$ is a map $\mathbf{k}$ from the space $\langle X \to \mathbb{R}\rangle_L$ of bounded Lipschitz functions from $X$ to $\mathbb{R}$ that is additive, positively homogeneous, monotonic, and *positive* (i.e., if $\varphi(x) \geq 0$ for every $x$, then $\mathbf{k}(\varphi) \geq 0$). It is *normalized* iff $\mathbf{k}(\chi_X)=1$. Next, we shall observe that, in sym-compact spaces, any normalized LL-prevision on $X^{(2)}$ actually defines a continuous probability valuation $p^2$ on $X \times X^{op}$ with the desired properties. We shall require that the problematic second marginal of $p$ equal the *convex-concave dual* $p'^{\perp}$ of $p'$, see below.

For the first step, let $f \ominus g$ abbreviate the function mapping $(x,y) \in X^{(2)}$ to $f(x) - g(y)$. If $f$ and $g$ are Lipschitz, then so is $f \ominus g$ (with the hemi-metric $d^2$ on $X^{(2)}$). Abusing the concept slightly, we shall say that the *first marginal* of an LL-prevision $\mathbf{k}$ on $X^{(2)}$ is the valuation $p$ on $X$ iff $\mathbf{k}(f\ominus 0) = \oint_{x\in X} f(x)dp$ for every bounded 1-Lipschitz function $f$ from $X$ to $\mathbb{R}^+$. (Note that, if $\mathbf{k}$ were obtained by integrating along $p^2$, and if we allowed for more general continuous functions $f$, this would define $p$ as $\pi_1[p^2]$.) We say that the *second marginal* of $\mathbf{k}$ is the valuation $p'$ on $X$ iff $\mathbf{k}(0\ominus g) = -\oint_{y\in X} g(y)dp'$ for every bounded 1-Lipschitz function $g$ from $X$ to $\mathbb{R}^+$. The minus sign copes for the problem with $d$ mentioned above. Let $\mathbf{K}(p,p')$ be the set of all normalized LL-previsions on $X^{(2)}$ whose first marginal is $p$, and whose second marginal is $p'$:

**Theorem 4.** *Let $X$ be equipped with a bounded hemi-metric $d$. For every normalized probability valuations $p$ and $p'$ on $X$, $d_{\mathrm{H}}(p,p') = \min_{\mathbf{k}\in\mathbf{K}(p,p')} \mathbf{k}(d)$.*

*Proof.* (Sketch. See [13, Appendix G].) It is easy to see that $d_{\mathrm{H}}(p,p') \leq \mathbf{k}(d)$ for every $\mathbf{k} \in \mathbf{K}(p,p')$. Conversely, we must build some $\mathbf{k} \in \mathbf{K}(p,p')$ such that $\mathbf{k}(d) = d_{\mathrm{H}}(p,p')$. We use an extension theorem on ordered cones akin to the Hahn-Banach Theorem on normed vector spaces, and derived from Roth's Sandwich Theorem [26,30], to extend the monotonic linear functional $\mathbf{f} : Z \to \overline{\mathbb{R}^+}$, defined by $\mathbf{f}(f \ominus g) = \oint_{x\in X} f(x)dp - \oint_{y\in X} g(y)dp'$ on the convex subset $Z = \{f \ominus g | f,g \in \langle X \to \mathbb{R}^+\rangle_L, f \ominus g \geq 0\}$ of the ordered cone $C = \langle X^{(2)} \to \mathbb{R}^+\rangle$, to a monotonic linear

functional $\mathbf{k}_0$ on the whole of $C$. Then we let $\mathbf{k}(\varphi) = \mathbf{k}_0(b + \varphi) - b$ for any $b \geq -\inf_{(x,y)\in X^{(2)}} \varphi(x, y)$. □

For the second step, for any continuous probability valuation $p'$ on a stably compact space $X$, define its *dual* $p'^{\perp}$ on $X^{\mathsf{d}}$ by $p'^{\perp}(X \setminus Q) = 1 - p'^{\dagger}(Q)$, for any cocompact $X \setminus Q$ of $X$, where $p'^{\dagger}(Q) = \inf_{U\in\mathcal{O}(X),U\supseteq Q} p'(U)$. Using a result by Tix [29, Satz 3.4], $p'^{\perp}$ is a continuous probability valuation on $X^{\mathsf{d}}$. By [15, Section 5], in particular Definition 1 and the fact that $\alpha_{\mathcal{C}}(p'^{\perp}) = \alpha_{\mathcal{C}}(p')^{\perp}$, (see also [14, Section 6.2.2]), $\fint_{y\in X^{op}} g(y)dp'^{\perp} = \alpha_{\mathcal{C}}(p')^{\perp}(g) = -\inf_{f \text{ step function}, f\geq -g} \alpha_{\mathcal{C}}(p')(f) = -\inf_{f \text{ step function}, f\geq -g} \fint_{x\in X} f(x)dp'$. (Step functions are those continuous maps that only take finitely many values.) The formula simplifies in the case of Lipschitz maps and sym-compact spaces, observing that $-h$ is Lipschitz from $X^{op}$ to $\mathbb{R}^+$ whenever $h$ is Lipschitz from $X$ to $\mathbb{R}^+$ [13, Appendix H]:

**Lemma 2.** *Let $X$ be sym-compact, and $p'$ a continuous probability valuation on $X$. For any bounded Lipschitz map $h$ from $X$ to $\mathbb{R}$, $-\fint_{x\in X} h(x)dp' = \fint_{x\in X^{op}} -h(x)dp'^{\perp}$.*

Note that this makes sense: the right-hand side, notably, integrates $-f$, which is Lipschitz from $X^{op}$ to $\mathbb{R}$, while $p'^{\perp}$ is defined on $X^{\mathsf{d}}$. But $X^{op} = X^{\mathsf{d}}$ by Theorem 1.

**Theorem 5.** *Let $X$ be a sym-compact space, with bounded hemi-metric $d$. For all continuous probability valuations $p$, $p'$ on $X$, $d_{\mathrm{H}}(p, p') = \min_{p^2} \fint_{(x,y)\in X^{(2)}} d(x, y)dp^2$, where $p^2$ ranges over the elements of $\mathbf{V}_1(X^{(2)})$ with $\pi_1[p^2] = p$, $\pi_2[p^2] = p'^{\perp}$.*

*Proof.* (Sketch. See [13, Appendix I].) We find $p^2$ from the $\mathbf{k}$ gotten in Theorem 4. It would be tempting to define $p^2(W) = \mathbf{k}(\chi_W)$ for all opens $W$ of $X^{(2)}$, however $\chi_W$ is continuous but not Lipschitz. Instead, we approximate it by the directed family of Lipschitz functions $\frac{1}{\epsilon}\chi^{\epsilon}_{U^{d,-(\epsilon)}}$, $\epsilon > 0$, i.e., we define $p^2(W)$ as $\sup_{\epsilon>0} \mathbf{k}(\frac{1}{\epsilon}\chi^{\epsilon}_{W^{d,-(\epsilon)}})$. The key to the continuity of $p^2$ is the fact that, not only we can write every open subset $U$ as the directed union of all opens $V \Subset U$, where $V \Subset U$ means that there is a saturated compact $Q$ such that $V \subseteq Q \subseteq U$ (as in every locally compact space [10]), but in fact that we can choose $V$ of the special form $U^{d,-(\epsilon)}$, $\epsilon > 0$. (That $U^{d,-(\epsilon)} \Subset U$ relies on Theorem 1: we let $Q = \{x \in X | d(x, X\setminus U) \geq \epsilon/2\}$, and realize this is closed in $X^{op} = X^{\mathsf{d}}$.) It is then enough to check that $p^2(\bigcup_{\epsilon>0} U^{d,-(\epsilon)}) = \sup_{\epsilon>0} p^2(U^{d,-(\epsilon)})$. The hardest part is to show that $p^2$ is modular. Note that $\frac{1}{\epsilon}\chi^{\epsilon}_{(U\cap V)^{d,-(\epsilon)}} = \min(\frac{1}{\epsilon}\chi^{\epsilon}_{U^{d,-(\epsilon)}}, \frac{1}{\epsilon}\chi^{\epsilon}_{V^{d,-(\epsilon)}})$, so that $p^2(U \cap V) = \sup_{\epsilon>0} \mathbf{k}(\min\left(\frac{1}{\epsilon}\chi^{\epsilon}_{U^{d,-(\epsilon)}}, \frac{1}{\epsilon}\chi^{\epsilon}_{V^{d,-(\epsilon)}}\right))$. For unions, the best we can say in general is that $\frac{1}{\epsilon}\chi^{\epsilon}_{(U\cup V)^{d,-(\epsilon)}} \geq \max\left(\frac{1}{\epsilon}\chi^{\epsilon}_{U^{d,-(\epsilon)}}, \frac{1}{\epsilon}\chi^{\epsilon}_{V^{d,-(\epsilon)}}\right)$. However, because $X$ is sym-compact, we can show that for every $\epsilon > 0$, there is an $\epsilon' > 0$ such that for all $\epsilon'' > 0$, $\frac{1}{\epsilon''}\chi^{\epsilon''}_{(U^{d,-(\epsilon)}\cup V^{d,-(\epsilon)})^{d,-(\epsilon'')}} \leq \max\left(\frac{1}{\epsilon'}\chi^{\epsilon'}_{U^{d,-(\epsilon')}}, \frac{1}{\epsilon'}\chi^{\epsilon'}_{V^{d,-(\epsilon')}}\right)$, from which we conclude that $p^2(U \cup V) = \sup_{\epsilon>0} \mathbf{k}(\max(\frac{1}{\epsilon}\chi^{\epsilon}_{U^{d,-(\epsilon)}}, \frac{1}{\epsilon}\chi^{\epsilon}_{V^{d,-(\epsilon)}}))$. That $p^2$ is modular then follows from the linearity of $\mathbf{k}$ and the fact that $\min(a, b) + \max(a, b) = a + b$. We use Lemma 2 to show that the second marginal of $p^2$ is $p'^{\perp}$. □

To characterize $d_{\mathrm{H}}$ over spaces of continuous previsions (including other brands of continuous games), we need a form of the so-called *minimax theorem*, whose proof,

inspired from Frenk and Kassay [9], can be found in [13, Appendix J]. Say that $f : X \times Y \to \mathbb{R}$ is *convex in* $X$ (in the sense of Ky Fan) iff for every $\alpha \in (0,1)$, for every $x_1, x_2$ in $X$, there is an $x_0 \in X$ such that for every $y \in Y$, $f(x_0, y) \leq \alpha f(x_1, y) + (1-\alpha) f(x_2, y)$. Say that $f$ is *concave in* $Y$ iff for every $\alpha \in (0,1)$, for every $y_1, y_2 \in Y$, there is a $y_0 \in Y$ such that for all $x \in X$, $f(x, y_0) \geq \alpha f(x, y_1) + (1-\alpha) f(x, y_2)$.

**Theorem 6 (Minimax).** *Let $X$ be a non-empty compact space, $Y$ a set. Let $f$ be any map from $X \times Y$ to $\mathbb{R}$, such that $\lambda x \in X \cdot f(x,y)$ is continuous for every $y \in Y$. If $f$ is convex in $X$ and concave in $Y$, then* $\sup_{y \in Y} \inf_{x \in X} f(x,y) = \inf_{x \in X} \sup_{y \in Y} f(x,y)$. *Moreover, the inf on the right is attained.*

For disambiguation purposes, write $d^Y_{\mathrm{H}}$ the Hutchinson distance on $Y$, where $Y$ is any space of previsions. By [12], each $F \in \bigtriangledown \mathbf{P}_1(X)$ has a *heart* $CCoeur_1(F) = \{G \in \mathbf{P}^{\triangle}_1(X) | G \geq F\} \in \mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$, where $\mathbf{P}^{\triangle}_{1\,wk}(X)$ is $\mathbf{P}^{\triangle}_1(X)$ with the weak topology; each $F \in \bigtriangleup \mathbf{P}_1(X)$ has a *skin* $CPeau_1(F) = \{G \in \mathbf{P}^{\triangle}_1(X) | G \leq F\} \in \mathcal{H}(\mathbf{P}^{\triangle}_{1\,wk}(X))$; and each $F = (F^-, F^+) \in \mathbf{F}_1(X)$ has a *body* $CCorps_1(F) = CCoeur_1(F^-) \cap CPeau_1(F^+) \in \mathcal{P}\ell(\mathbf{P}^{\triangle}_{1\,wk}(X))$. Then:

**Theorem 7.** *Let $X$ be a sym-compact hemi-metric space, with bounded hemi-metric $d$. Then $CCoeur_1$ is an isometric embedding of $\bigtriangledown \mathbf{P}_1(X)$ (with $d_{\mathrm{H}}^{\bigtriangledown \mathbf{P}_1(X)}$) into $\mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$ (with $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{Q}}$); $CPeau_1$ is an isometric embedding of $\bigtriangleup \mathbf{P}_1(X)$ (with $d_{\mathrm{H}}^{\bigtriangleup \mathbf{P}_1(X)}$) into $\mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$ (with $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{H}}$); and $CCorps_1$ is an isometric embedding of $\mathbf{F}_1(X)$ (with $d_{\mathrm{H}}^{\mathbf{F}_1(X)}$) into $\mathcal{P}\ell(\mathbf{P}^{\triangle}_{1\,wk}(X))$ (with $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{P}\ell}$).*

*Proof.* By [12, Theorem 5], for every continuous normalized lower prevision $F$ on $X$, $F(f) = \min_{G \in \mathbf{P}^{\triangle}_1(X), G \geq F} G(f)$. Let now $F, F' \in \mathbf{P}^{\triangle}_1(X)$. For short, we let $f$ range implicitly over $\langle X \to \mathbb{R}^+ \rangle_1$, $G$ and $G'$ over $\mathbf{P}^{\triangle}_1(X)$. Note that $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{Q}}(CCoeur_1(F), CCoeur_1(F')) = \max_{G' \geq F'} \min_{G \geq F} \sup_f \max(G(f) - G'(f), 0)$. If $F \leq F'$, then $d_{\mathrm{H}}(F, F') = 0$ (using Theorem 3 and the easy fact that the specialization ordering of the weak topology is $\leq$), while every $G' \geq F'$ is also above $F$, so $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{Q}}(CCoeur_1(F), CCoeur_1(F')) = 0$. Otherwise, $d_{\mathrm{H}}^{\bigtriangledown \mathbf{P}_1(X)}(F, F') = \sup_f (F(f) - F'(f)) = \sup_f (\min_{G \geq F} G(f) - \min_{G' \geq F'} G'(f)) = \sup_{G' \geq F'} \sup_f \min_{G \geq F} (G(f) - G'(f)) = \sup_{G' \geq F'} \min_{G \geq F} \sup_f (G(f) - G'(f))$ (by Theorem 6) $= (d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{Q}}(CCoeur_1(F), CCoeur_1(F'))$. To check that Theorem 6 applies, we first realize that $\mathbf{P}^{\triangle}_1(X)$ equipped with the Hutchinson hemi-metric has the weak topology (Theorem 3), which is (stably) compact because $X$ is stably compact (see Proof of Proposition 4, Appendix, in [12, Long version]), and $\lambda G, f \cdot G(f) - G'(f)$ is continuous in $G$ for each $f$ (since the inverse image of $(t, +\infty)$ is the weak open $[f > G'(f) + t]$), convex in $\langle X \to \mathbb{R}^+ \rangle$ (i.e., in $f$; this is because $G$ and $G'$ are in fact linear), and is concave in $\mathbf{P}^{\triangle}_1(X)$ (i.e., in $G$; this is in fact again linear). The argument is similar for $d_{\mathrm{H}}^{\bigtriangleup \mathbf{P}_1(X)}$ and $(d_{\mathrm{H}}^{\mathbf{P}^{\triangle}_1(X)})_{\mathcal{H}}$, where by [12, Concluding remarks], for any continuous normalized upper previsions $F$ and $F'$, when $F \not\leq F'$,

$d_{\mathrm{H}}^{\Delta\,\mathbf{P}_1(X)}(F,F') = \sup_f(F(f)-F'(f)) = \sup_f(\sup_{G\leq F} G(f) - \sup_{G'\leq F'} G'(f)) = \sup_{G\leq F}\sup_f \inf_{G'\leq F'}(G(f)-G'(f))$, and $(d_{\mathrm{H}}^{\mathbf{P}_1^{\triangle}(X)})_{\mathcal{H}}(CPeau_1(F), CPeau_1(F'))$ $= \sup_{G\leq F}\inf_{G'\leq F'}\sup_f \max(G(f)-G'(f),0)$ and we again conclude by Theorem 6. Finally, the $d_{\mathrm{H}}^{\mathbf{F}_1(X)}$ and $(d_{\mathrm{H}}^{\mathbf{P}_1^{\triangle}(X)})_{\mathcal{P}\ell}$ case is a consequence of the first two cases, and of Proposition 3. □

## 7 Conclusion

We contend that our Hutchinson hemi-metric $d_{\mathrm{H}}$ is not only a natural way of building simulation hemi-metrics through a fixpoint construction à la Ferns *et al.* [8], allowing one to bound the error in payoffs when starting from different states; but also that $d_{\mathrm{H}}$ has an elegant mathematical theory: it generalizes all variants of the Hausdorff hemi-metric on spaces of demonic, angelic, and chaotic non-determinism, and admits a (directed) Kantorovich-Rubinstein theorem in the probability case. Theorem 7 shows that $d_{\mathrm{H}}$ is actually a composition of a Hausdorff-like hemi-metric with a Kantorovich-like hemi-metric $d_{\mathrm{H}}^{\mathbf{P}_1^{\triangle}(X)}$, as in Ferns *et al.* [8]. Finally, Theorem 7 exhibits $d_{\mathrm{H}}^{\nabla\,\mathbf{P}_1(X)}$ as the $\sup\inf\sup$ formula $d_{\mathrm{H}}^{\nabla\,\mathbf{P}_1(X)}(F,F') = \sup_{G'\geq F'}\min_{G\geq F}\sup_f \mathsf{d}_{\mathbb{R}}(G(f),G'(f'))$. This is akin to the *a posteriori* directed metric on the finite-state games of de Alfaro *et al.* [4, Equation (5)] in the turn-based case. On the other hand, $d_{\mathrm{H}}^{\Delta\,\mathbf{P}_1(X)}$ is defined as the $\sup\sup\inf$ formula $\sup_f\sup_{G\leq F}\inf_{G'\leq F'}\mathsf{d}_{\mathbb{R}}(G(f),G'(f'))$, which matches exactly the definition of the *a priori* directed metric of de Alfaro *et al.* [4, Equation (6)]. So the difference between the two seems to be one between demonic and angelic choice. Further work should identify whether the corresponding simulation hemi-metrics can be characterized by logics similar to that of op.cit., in the topological setting of PrTSes.

**Acknowledgments.** Thanks to the anonymous referees for their useful comments.

## References

1. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, OUP, vol. 3, pp. 1–168 (1994)
2. Baddeley, A.J.: Hausdorff metric for capacities. Technical Report BS-R9127, CWI, Dept. of Operations Research, Statistics, and System Theory (1991)
3. Bonsangue, M.M., van Breugel, F., Rutten, J.: Generalized metric spaces: Completion, topology, and powerdomains via the Yoneda embedding. Th. Comp. Sci. 193, 1–51 (1998)
4. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game relations and metrics. In: 22nd IEEE Symp. Logic in Comp. Sci (LICS 2007), pp. 99–108 (2007)
5. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labelled Markov processes. Information and Computation 179(2), 163–193 (2002)
6. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: The metric analogue of weak bisimulation for probabilistic processes. In: 17th IEEE Symp. Logic in Comp. Sci. (LICS 2002), pp. 413–422 (2002)

7. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled Markov processes. Theoretical Computer Science 318, 323–354 (2004)
8. Ferns, N., Panangaden, P., Precup, D.: Metrics for Markov decision processes with infinite state spaces. In: 21st Annual Conf. on Uncertainty in Artificial Intelligence, Arlington, VA, pp. 201–208. AUAI Press (2005)
9. Frenk, J., Kassay, G.: Minimax results and finite dimensional separation. E.I. Report 9845/A, Econometric Institute, Erasmus University Rotterdam (October 1998)
10. Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M., Scott, D.S.: Continuous lattices and domains. In: Encycl. Mathematics and its Applications, vol. 93, CUP (2003)
11. Goubault-Larrecq, J.: Continuous capacities on continuous state spaces. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 764–776. Springer, Heidelberg (2007)
12. Goubault-Larrecq, J.: Continuous previsions. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 542–557. Springer, Heidelberg (2007)
13. Goubault-Larrecq, J.: Simulation hemi-metrics between infinite-state stochastic games. Research Report LSV-07-34, Laboratoire Spécification et Vérification, ENS Cachan, France, 47 pages (October 2007)
14. Goubault-Larrecq, J.: Une introduction aux capacités, aux jeux et aux prévisions (June 2007), http://www.lsv.ens-cachan.fr/~goubault/ProNobis/pp_1_8.pdf
15. Goubault-Larrecq, J.: Prevision domains and convex powercones. In: FoSSACS 2008. LNCS, vol. 4962, pp. 318–333. Springer, Heidelberg (2008)
16. Jones, C.: Probabilistic Non-Determinism. PhD thesis, University of Edinburgh, Technical Report ECS-LFCS-90-105 (1990)
17. Jung, A.: Stably compact spaces and the probabilistic powerspace construction. In: Desharnais, J., Panangaden, P. (eds.) Domain-theoretic Methods in Probabilistic Processes. Electronic Notes in Theoretical Computer Science, vol. 87, Elsevier, Amsterdam (2004)
18. Kantorovich, L.V.: On the translocation of masses. Comptes Rendus (Doklady) de l'Acad. Sci. URSS 37, 199–201 (1942); Reprinted in Management Science, vol. 5, pp. 1–4 (1958)
19. Keimel, K., Lawson, J.: Measure extension theorems for $T_0$-spaces. Topology and its Applications 149(1–3), 57–83 (2005)
20. Kopperman, R.: Asymmetry and duality in topology. Topology and its Applications 66, 1–39 (1995)
21. Künzi, H.P., Schellekens, M.P.: On the Yoneda completion of a quasi-metric space. Theoretical Computer Science 278(1-2), 159–194 (2002)
22. Lawvere, F.W.: Metric spaces, generalized logic, and closed categories. Reprints in Theory and Applications of Categories (1), 1–27 (originally published in 1973, 2002)
23. Mislove, M.: Topology, domain theory and theoretical computer science. Topology and Its Applications 89, 3–59 (1998)
24. Mislove, M., Ouaknine, J., Worrell, J.: Axioms for probability and nondeterminism. El. Notes Th. Comp. Sci. 91(3), 7–28 (2003); 10th Intl. Workshop on Expressiveness in Concurrency (EXPRESS 2003) (2003)
25. Nachbin, L.: Topology and Order. Van Nostrand Reinhold, New York (1965)
26. Roth, W.: Hahn-Banach type theorems for locally convex cones. J. Australian Math. Soc. 68(1), 104–125 (2000)
27. Schrijver, A.: On the history of combinatorial optimization (till 1960). In: Aardal, K., Nemhauser, G.L., Weismantel, R. (eds.) Handbook of Discrete Optimization, pp. 1–68. Elsevier, Amsterdam (2005)
28. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. MIT Press, Cambridge (1996)

29. Tix, R.: Stetige Bewertungen auf topologischen Räumen. Diplom, T.H. Darmstadt (June 1995)
30. Tix, R., Keimel, K., Plotkin, G.: Semantic domains for combining probability and non-determinism. Electronic Notes in Theor. Comp. Sci. 129, 1–104 (2005)
31. van Breugel, F., Worrell, J.: An algorithm for quantitative verification of probabilistic transition systems. In: CONCUR 2001. LNCS, vol. 2976, pp. 421–432. Springer, Heidelberg (2001)
32. Walley, P.: Statistical Reasoning with Imprecise Probabilities. Chapman and Hall, London (1991)

# Beyond Rank 1: Algebraic Semantics and Finite Models for Coalgebraic Logics

Dirk Pattinson[1] and Lutz Schröder[2]

[1] Department of Computing, Imperial College London
[2] DFKI-Lab Bremen and Department of Computer Science, Universität Bremen

**Abstract.** Coalgebras provide a uniform framework for the semantics of a large class of (mostly non-normal) modal logics, including e.g. monotone modal logic, probabilistic and graded modal logic, and coalition logic, as well as the usual Kripke semantics of modal logic. In earlier work, the finite model property for coalgebraic logics has been established w.r.t. the class of *all* structures appropriate for a given logic at hand; the corresponding modal logics are characterised by being axiomatised in rank 1, i.e. without nested modalities. Here, we extend the range of coalgebraic techniques to cover logics that impose global properties on their models, formulated as frame conditions with possibly nested modalities on the logical side (in generalisation of frame conditions such as symmetry or transitivity in the context of Kripke frames). We show that the finite model property for such logics follows from the finite algebra property of the associated class of complex algebras, and then investigate sufficient conditions for the finite algebra property to hold. Example applications include extensions of coalition logic and logics of uncertainty and knowledge.

## 1 Introduction

The coalgebraic semantics of modal logic has proved to be useful to establish results that apply uniformly to a large class of modal logics. For example, [17] provides a finite model construction and decidability results and [18] derives uniform PSPACE bounds for coalgebraic modal logics. The class of logics covered by the coalgebraic approach includes e.g. monotone modal logic and the standard logic $K$ but also less well-studied specimen such as Pauly's coalition logic [16], probabilistic modal logic [13,9], and graded modal logic [8]. Moreover, the coalgebraic approach allows combining logics modularly [4] while preserving completeness [5] and complexity bounds [19].

However, the range of the coalgebraic techniques is hitherto limited to logics axiomatised in rank 1, i.e. with nesting depth of modal operator uniformly equal to 1, thus excluding standard logics such as $K4$ and $S5$. The reason for this limitation is that in previous work, only such modal logics have been considered that are interpreted over the class of *all* structures of an appropriate type. Indeed it is shown in [17] that the class of all structures of a given type is always axiomatisable in rank 1. By analogy, rank 1 axioms play the role of the $K$-axioms for Kripke frames: they ensure completeness w.r.t. the class of *all* frames.

However, it is often desirable to have completeness for a subclass of all structures that satisfy additional properties like transitivity or reflexivity in a relational context. These

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 66–80, 2008.

additional properties are captured as frame conditions on the logical side; e.g. the (4) axiom $\Box a \to \Box\Box a$ ensures transitivity for Kripke frames. This is our starting point: we extend a given complete rank-1 axiomatisation of a class of structures (that we formalise as coalgebras for an endofunctor) by additional frame conditions and establish the finite model property (and hence completeness) with respect to the class of all structures that satisfy the additional axioms.

In view of our interest in finite model results, our main technical tool is finite Stone duality, i.e. the dual equivalence between finite sets and finite boolean algebras. Accordingly, the finite model property is established in two steps: the first step shows that the finite model property follows from the finite algebra property of an associated algebraic theory by transporting finite algebraic models to the coalgebraic side via Stone duality (the converse implication, i.e. that the finite model property implies the finite algebra property, is trivial). In the second step, we use algebraic filtrations in the style of Lemmon [14], adapted to a non-normal setting, to obtain the finite algebra property. In view of the duality between modal algebras and neighbourhood frames [6] this latter step is equivalent to establishing the finite model property with respect to neighbourhood semantics, albeit at the expense of losing the correspondence with the algebraic semantics.

The versatility of our approach is demonstrated by two extended examples. For logics combining uncertainty and knowledge as described in [7], we show that the finite model property can be derived purely synthetically. In particular, we derive the finite model property in the presence of axioms that stipulate interaction between belief and uncertainty, and our results are modular in the axiomatisation of agent belief. The second example uses our techniques to establish the finite model property for various extensions of Pauly's coalition logic [16].

## 2 Preliminaries and Notation

The category of sets and functions is denoted by $\mathsf{Set}$, and we write $\mathsf{BA}$ for the category of boolean algebras. We use $\mathsf{Fin}$ to denote the category of finite sets, and $\mathsf{FBA}$ is the category of finite boolean algebras. Stone duality [10] restricts to a dual equivalence between $\mathsf{Fin}$ and $\mathsf{FBA}$ given by the contravariant powerset functor $2 : \mathsf{Fin} \to \mathsf{FBA}$ and the functor $\mathsf{Uf} : \mathsf{FBA} \to \mathsf{Fin}$ that maps a finite boolean algebra to the set of its ultrafilters. If $A \in \mathsf{FBA}$, we write $\iota_A : 2 \circ \mathsf{Uf}(A) \to A$ for the canonical isomorphism.

Given an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$, a *$T$-coalgebra* is a pair $(C, \gamma)$ consisting of a carrier $C \in \mathsf{Set}$ and a transition function $\gamma : C \to TC$. A *coalgebra morphism* $f : (C, \gamma) \to (D, \delta)$ is a function $f : C \to D$ for which $\delta \circ f = Tf \circ \gamma$. We denote the category of $T$-coalgebras by $\mathsf{Coalg}(T)$ and write $\mathsf{Coalg}(T)_f$ for the full subcategory of $\mathsf{Coalg}(T)$ consisting of all those $(C, \gamma) \in \mathsf{C}(T)$ for which the carrier $C$ is finite. Dually, if $L : \mathsf{BA} \to \mathsf{BA}$ is a functor, we write $\mathsf{Alg}(L)$ for the category of $L$-algebras, that is pairs $(A, \alpha)$ where $A \in \mathsf{BA}$ and $\alpha : LA \to A$ is a morphism of boolean algebras. As for coalgebras, $\mathsf{Alg}(L)_f$ denotes the full subcategory of those $(A, \alpha) \in \mathsf{Alg}(L)$ whose carrier $A \in \mathsf{FBA}$ is finite. Throughout, we fix a denumerable set $V$ of propositional variables. The set of propositional formulas over a set $X$ is denoted by $\mathsf{Prop}(X)$ and the set of clauses over $X$ by $\mathsf{Cl}(X)$.

## 3 Rank-1 Logics

We start by introducing rank-1 logics that we take as extensions of propositional logic with unary modal operators. The restriction to unary modalities is purely for convenience; all of our results generalise to polyadic modalities in a straightforward way. Rank-1 logics are the basic building blocks of our theory, as they provide a sound and complete axiomatisation of the class $\mathsf{Coalg}(T)$ of all $T$-coalgebras that we extend with frame conditions to effect specific properties later.

**Definition 1 (Modal signatures, formulas).** A *modal signature* or *modal similarity type* is a set $\Lambda$ consisting of (unary) modal operators. For a set $S$, we write $\Lambda(S) = \{M(s) \mid M \in \Lambda, s \in S\}$ for the set of formulas that arise by prefixing elements of $S$ by precisely one modality in $\Lambda$. The set $\mathcal{F}(\Lambda)$ of *$\Lambda$-formulas* is inductively given by

$$\mathcal{F}(\Lambda) \ni \phi, \psi ::= p \mid \bot \mid \phi \to \psi \mid M(\phi)$$

where $p \in V$ is a propositional variable and $M \in \Lambda$.

We interpret modal logics over $T$-coalgebras, where $T$ is an endofunctor on $\mathsf{Set}$. Modal operators are interpreted using predicate liftings [15].

**Definition 2 (Structures, Coalgebraic Semantics).** If $\Lambda$ is a modal signature, a *$\Lambda$-structure* consists of an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$ and a predicate lifting (a natural transformation)$[\![M]\!] : 2 \to 2 \circ T$ for every $M \in \Lambda$. A *morphism* between two $\Lambda$-structures $S$ and $T$ is a natural transformation $\mu : S \to T$ such that $[\![M]\!]_S = \mu^{-1} \circ [\![M]\!]_T$ for all $M \in \Lambda$. The *coalgebraic semantics* of $\phi \in \mathcal{F}(\Lambda)$ w.r.t. a $T$-coalgebra $\mathbb{C} = (C, \gamma)$ and a valuation $\pi : V \to \mathcal{P}(C)$ is inductively defined by

$$[\![p]\!]^\pi_{\mathbb{C}} = \pi(p) \qquad [\![M\phi]\!]^\pi_{\mathbb{C}} = \gamma^{-1} \circ [\![M]\!]_C([\![\phi]\!]^\pi_{\mathbb{C}})$$

and the standard clauses for propositional connectives. If $\Theta \subseteq \mathcal{F}(\Lambda)$ and $\mathbb{C} \in \mathsf{Coalg}(T)$ we write $\mathbb{C} \models \Theta$ if $[\![\phi]\!]^\pi_{\mathbb{C}} = \top$ for all $\pi : V \to \mathcal{P}(C)$ and all $\phi \in \Theta$ and denote the full subcategory of all $\mathbb{C} \in \mathsf{Coalg}(T)$ that satisfy every formula in $\Theta$ by $\mathsf{Coalg}(T, \Theta)$. Finally, a formula $\phi$ is *(finitely) satisfiable* in $\mathsf{Coalg}(T, \Theta)$ if there exists a (finite) coalgebra $\mathbb{C} \in \mathsf{Coalg}(T, \Theta)$ and a valuation $\pi : V \to \mathcal{P}(C)$ with $[\![\phi]\!]_\pi \neq \emptyset$.

The axiomatisation of the modal logics considered here consists of two parts: a set of rank-1 axioms that is (sound and) complete for the class of all $T$-coalgebras (and thus accounts for the structure of $\mathsf{Coalg}(T)$) and a set of frame conditions that specify additional properties. The logic of $\mathsf{Coalg}(T)$ can always be axiomatised by rank-1 axioms [17].

**Definition 3.** Suppose $\Lambda$ is a modal similarity type. A *rank-1 axiom* over $\Lambda$ is a propositional formula over $\Lambda(\mathsf{Prop}(V))$; propositional formulas over $\Lambda(\mathsf{Prop}(V)) \cup V$ are called *rank-0/1*. A *rank-1 logic* is a pair $\mathcal{L} = (\Lambda, \mathcal{A})$ where $\mathcal{A}$ is a set of rank-1 axioms over $\Lambda$. An *extended rank-1 logic* is a triple $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ where $(\Lambda, \mathcal{A})$ is a rank-1 logic and $\Theta \subseteq \mathcal{F}(\Lambda)$ is a set of additional axioms. The logic $\mathcal{L}$ is *rank-0/1* if every $\phi \in \Theta$ is rank-0/1.

Deduction over (extended) rank-1 logics is standard:

**Definition 4 (Derivability).** Suppose $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ is an extended rank-1 logic. We write $\mathcal{L} \vdash \phi$ if $\phi$ is contained in the least set closed under the rules

$$\frac{\phi \in \mathcal{A} \cup \Theta}{\mathcal{L} \vdash \phi\sigma} \qquad \frac{\mathcal{L} \vdash \phi \to \psi \quad \mathcal{L} \vdash \phi}{\mathcal{L} \vdash \psi} \qquad \frac{\phi \in \mathsf{Taut}(V)}{\mathcal{L} \vdash \phi\sigma} \qquad \frac{\mathcal{L} \vdash \phi \leftrightarrow \psi}{\mathcal{L} \vdash M\phi \leftrightarrow M\psi}$$

where $\mathsf{Taut}(V)$ is the set of propositional tautologies over the set $V$ of (propositional) variables and $\sigma : V \to \mathcal{F}(\Lambda)$ ranges over $\mathcal{F}(\Lambda)$-substitutions. This is extended to sets of formulas, and we write $\Psi \vdash_{\mathcal{L}} \phi$ if there exist $\psi_1, \ldots, \psi_n \in \Psi$ such that $\mathcal{L} \vdash \psi_1 \wedge \cdots \wedge \psi_n \to \phi$.

To prove completeness in the presence of frame conditions, we require that the rank-1 axioms are one-step sound and one-step complete; both notions are as in [15,5]. For the sake of brevity, we (ab)use subsets of a set $X$ as propositional variables with the obvious interpretation in the boolean algebra $\mathcal{P}(X)$.

**Definition 5 (One-step Soundness and Completeness).** Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, and let $T$ be a $\Lambda$-structure. The *one-step semantics* $[\![\phi]\!]_{TX} \subseteq TX$ of $\phi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ at a set $X$ is defined inductively by the standard clauses for propositional connectives and $[\![MA]\!]_{TX} = [\![M]\!]_X(A)$ for $M \in \Lambda$, $A \subseteq X$. We write $TX \models \phi$ if $[\![\phi]\!]_{TX} = TX$. The relation $\mathcal{A}, X \vdash \phi$ of *one-step derivability* at a set $X$ is generated by

$$\frac{\phi \in \mathsf{Taut}(\Lambda(\mathcal{P}(X)))}{\mathcal{A}, X \vdash \phi} \qquad \frac{\mathcal{A}, X \vdash \phi \to \psi \quad \mathcal{A}, X \vdash \phi}{\mathcal{A}, X \vdash \psi} \qquad \frac{\phi \in \mathcal{A}}{\mathcal{A}, X \vdash \phi\sigma},$$

where $\sigma : V \to \mathcal{P}(X)$ is a $\mathcal{P}(X)$-valuation in the last rule.

The logic $\mathcal{L}$ is *one-step sound* w.r.t. a structure $T$ if, for all sets $X$ and all $\phi \in \mathcal{F}(\Lambda)$, $\mathcal{A}, X \vdash \phi$ implies $TX \models \phi$; $\mathcal{L}$ is *one-step complete* if the converse implication holds.

One-step soundness guarantees soundness in the standard sense:

**Proposition 6.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be an extended rank-1 logic which is one-step sound w.r.t. a structure $T$. Then $\mathcal{L}$ is sound w.r.t.* $\mathsf{Coalg}(T, \Theta)$*, i.e. if $\mathcal{L} \vdash \phi$ for $\phi \in \mathcal{F}(\Lambda)$, then* $\mathsf{Coalg}(T, \Theta) \models \phi$.

We prove the converse, and simultaneously establish the finite model property, by constructing finite algebraic models that we then transport to the coalgebraic side.

## 4 Algebraic Semantics

We recall the concept of the functorial presentation of a rank-1 logic, due to Kurz and collaborators [12,1]. Again, this concept is most conveniently introduced by using elements of a boolean algebra as variables with the obvious interpretation.

**Definition 7.** Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, let $A$ be a boolean algebra, and let $\phi, \psi \in \mathsf{Prop}(\Lambda(A))$. We write $\phi =_A \psi$ if $\phi = \psi$ can be derived equationally from the axioms of boolean algebra augmented with the set $\{\psi\sigma = \top \mid \psi \in \mathcal{A}, \sigma : V \to A\}$. The functor $L : \mathsf{BA} \to \mathsf{BA}$ defined by $L(A) = \mathsf{Prop}(\Lambda(A))/=_A$ is called the *functorial presentation* of $\mathcal{L}$.

The functorial presentation of a logic allows us to view formulas as terms that are interpreted over $L$-algebras.

**Definition 8.** Let $L$ be the functorial presentation of a rank-1 logic $\mathcal{L} = (\Lambda, \mathcal{A})$, and let $\mathbb{A} = (A, \alpha) \in \mathsf{Alg}(L)$. The *algebraic semantics* of $\phi \in \mathcal{F}(\Lambda)$ w.r.t. $\mathbb{A}$ and a valuation $\pi : V \to A$ is defined inductively by the clauses

$$[\![p]\!]^\pi_{\mathbb{A}} = \pi(p) \qquad [\![M(\phi)]\!]^\pi_{\mathbb{A}} = \alpha \circ q(M[\![\phi]\!]^\pi_{\mathbb{A}})$$

where $p \in V$ and $q : \mathsf{Prop}(\Lambda(A)) \to \mathsf{Prop}(\Lambda(A))/{=_A}$ is the quotient mapping, with propositional connectives interpreted via the boolean algebra structure of $\mathbb{A}$. If $\mathbb{A} = (A, \alpha) \in \mathsf{Alg}(L)$ and $\Theta \subseteq \mathcal{F}(\Lambda)$, we write $\mathbb{A} \models \Theta$ if $[\![\phi]\!]^\pi_{\mathbb{A}} = \top$ for all $\phi \in \Theta$ and all $\pi : V \to A$. We denote the full subcategory of all $\mathbb{A} \in \mathsf{Alg}(L)$ with $\mathbb{A} \models \Theta$ by $\mathsf{Alg}(L, \Theta)$. Finally, a formula is (finitely) satisfiable in $\mathsf{Alg}(L, \Theta)$ if there exists a (finite) $\mathbb{A} = (A, \alpha) \in \mathsf{Alg}(L, \Theta)$ such that $[\![\phi]\!]^\pi_{\mathbb{A}} \neq \bot$ for some valuation $\pi : V \to A$.

To capitalise on the duality between $\mathsf{Fin}$ and $\mathsf{FBA}$ we need to insist that $L$ restricts to an endofunctor on $\mathsf{FBA}$ which is the case if the set of modalities is finite. Despite its simplicity, we state this fact as a lemma as this will be a recurrent theme later.

**Lemma 9.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic with $\Lambda$ finite, and let $L$ be the functorial presentation of $\mathcal{L}$. Then $LA$ is finite for all $A \in \mathsf{FBA}$.*

The functorial presentation of a logic with a finite number of modalities gives rise to a (dual) functor that we denote by $L^\star = \mathsf{Uf} \circ L \circ 2 : \mathsf{Set} \to \mathsf{Set}$. The equivalence between $\mathsf{FBA}$ and $\mathsf{Fin}$ now extends to a dual equivalence between *finite* $L^\star$-coalgebras and *finite* $L$-algebras:

**Lemma 10.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic over a finite similarity type $\Lambda$, and let $L$ be the functorial presentation of $\mathcal{L}$. Then the functors $\mathsf{A} : \mathsf{Coalg}(L^\star)_f \to \mathsf{Alg}(L)_f$ and $\mathsf{C} : \mathsf{Alg}(L)_f \to \mathsf{Coalg}(L^\star)_f$ defined by $\mathsf{A}(C, \gamma) = (\mathcal{P}(A), \gamma^{-1} \circ \iota_{L2C})$ and $\mathsf{C}(A, \alpha) = (\mathsf{Uf}(A), \mathsf{Uf} \circ L\iota_A \circ \mathsf{Uf}\alpha)$ define a dual equivalence between finite $L^\star$-coalgebras and finite $L$-algebras.*

Under this equivalence, the carrier of the $L$-algebra associated with $(C, \gamma) \in \mathsf{Coalg}(L^\star)_f$ is the powerset $\mathcal{P}(C)$ of $C$, so that the interpretation of a formula $\phi$ in the algebra $\mathsf{A}(C, \gamma)$ is in fact a subset of $C$. This allows us to relate the algebraic and the coalgebraic semantics conveniently as follows:

**Lemma 11.** *Let $(C, \gamma) \in \mathsf{Coalg}(L^\star)$, and let $(A, \alpha) = \mathsf{A}(C, \gamma)$. Then $[\![\phi]\!]^\pi_{\mathbb{A}} = [\![\phi]\!]^\pi_{\mathbb{C}}$ for all $\phi \in \mathcal{F}(\Lambda)$ and all $\pi : V \to \mathcal{P}(C)$.*

In particular, we can reduce satisfiability in $\mathsf{Coalg}(L^\star)$ to satisfiability on $\mathsf{Alg}(L)$ also in the presence of extralogical axioms.

**Corollary 12.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be a rank-1 logic over a finite similarity type $\Lambda$. Then the following are equivalent for $\phi \in F(\Lambda)$:*

1. *$\phi$ is finitely satisfiable in $\mathsf{Alg}(L, \Theta)$*
2. *$\phi$ is finitely satisfiable in $\mathsf{Coalg}(L^\star, \Theta)$.*

We now describe the functor $L^\star$ to the extent that that is needed for purposes of this work. A more complete description can be found in [20].

**Definition 13.** Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, and let $X$ be a set. A subset $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is *one-step $\mathcal{L}$-inconsistent at $X$* iff there are finitely many $\phi_1, \ldots, \phi_n \in \Phi$ such that $\mathcal{A}, X \vdash \phi_1 \wedge \cdots \wedge \phi_n \to \bot$, and *one-step $\mathcal{L}$-consistent at $X$* otherwise. $\Phi$ is *one-step $\mathcal{L}$-maximally consistent at $X$* if it is maximal (w.r.t. $\subseteq$) among the one-step $\mathcal{L}$-consistent sets at $X$.

**Lemma 14.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic with $\Lambda$ finite. Then $L^\star X \cong \{\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X))) \mid \Phi$ one-step $\mathcal{L}$-maximally consistent$\}$.*

Thus, one can go back and forth between $L^\star$ and its equivalent description by introducing and dissolving equivalence classes. In the sequel, we will silently use the description of $L^\star$ that is most convenient for our purposes. We note that $L^\star$ gives rise to a canonical structure for a rank-1 logic $\mathcal{L}$.

**Lemma and Definition 15.** *Let $L$ be the functorial presentation of a rank-1 logic $\mathcal{L} = (\Lambda, \mathcal{A})$. Then the natural transformations defined by $[\![M]\!]_X(A) = \{\Phi \in L^\star(X) \mid MA \in \Phi\}$ for $M \in \Lambda$, $A \subseteq X$ define a $\Lambda$-structure for $L^\star$; we call this structure the* canonical $\Lambda$*-structure.*

It can be shown that the canonical $\Lambda$-structure is final in the category of all $\Lambda$-structures [20]. For our purposes, the following suffices:

**Lemma 16.** *Let $T$ be a $\mathcal{L}$-structure, and let $L$ be the functorial presentation of $\mathcal{L}$. Then the family of maps*

$$\mu(X) : TX \to L^\star X, t \mapsto \{\phi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X))) \mid t \in [\![\phi]\!]_{TX}\}$$

*defines a morphism between $T$ and the canonical $L^\star$-structure.*

As a consequence, the semantics of modal formulas is preserved if we move between an arbitrary $\Lambda$-structure and the canonical such.

**Lemma 17.** *Let $\mathbb{C} = (C, \gamma) \in \mathsf{Coalg}(T)$, and let $\mathbb{D} = (C, \mu(C) \circ \gamma)$. Then $[\![\phi]\!]^\pi_{\mathbb{C}} = [\![\phi]\!]^\pi_{\mathbb{D}}$ for all $\phi \in \mathcal{F}(\Lambda)$.*

In order to exploit the duality between finite sets and finite boolean algebras also in the presence of infinitely many modalities, we restrict attention to those modalities that occur either in the frame conditions or in the particular formula that we seek to satisfy. This cutting out of modalities is effected formally as follows:

**Definition 18.** Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, and let $\Gamma \subseteq \Lambda$. The *$\Gamma$-reduct* of $\mathcal{L}$ is the rank-1 logic $\mathcal{L}_\Gamma = (\Gamma, \mathcal{A}_\Gamma)$ where $\mathcal{A}_\Gamma = \{\phi \in \mathsf{Prop}(\Gamma(\mathsf{Prop}(V))) \mid \mathcal{L} \vdash \phi\}$.

$\Gamma$-reducts of complete rank-1 logics remain complete:

**Lemma 19.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic which is one-step sound and one-step sound complete w.r.t. a structure $T$. Then $\mathcal{L}_\Gamma$ is one-step sound and one-step complete for the structure given by $T$ together with the predicate liftings $[\![M]\!]$ for $M \in \Gamma$.*

We have already seen in Lemma 17 that semantics is preserved when moving from an arbitrary $\Lambda$-structure to the canonical structure. The next lemma is the key result as it provides for a passage in the other direction and thus allows us to reduce satisfiability over $\mathsf{Coalg}(T)$ to satisfiability over $\mathsf{Coalg}(L^\star)$.

**Lemma 20.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, let $\Gamma \subseteq \Lambda$ be finite, and let $L$ be the functorial presentation of $\mathcal{L}_\Gamma$. If $X \in \mathsf{Fin}$ is a finite set, then $\mu_X : TX \to L^\star X$, with $\mu$ as in Lemma 16, is surjective.*

The reduction of satisfiability over $\mathsf{Coalg}(T)$ to satisfiability over $\mathsf{Coalg}(L^\star)$ can now be achieved by picking a one-sided inverse of $\mu$.

**Lemma 21.** *Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic, let $\Gamma \subseteq \Lambda$ be finite, and let $L$ be the functorial presentation of $\mathcal{L}_\Gamma$. Then for every $\mathbb{C} = (C, \gamma) \in \mathsf{Coalg}(L^\star)$ with $C$ finite, there exists $\mathbb{D} = (C, \delta) \in \mathsf{Coalg}(T)$ with the same carrier such that $[\![\phi]\!]^\pi_{\mathbb{C}} = [\![\phi]\!]^\pi_{\mathbb{D}}$ for all $\phi \in \mathcal{F}(\Gamma)$, $\pi : V \to \mathcal{P}(C)$ and $c \in C$.*

As the passage from $\mathsf{Coalg}(L^\star)$ to $\mathsf{Coalg}(T)$ provided by Lemmas 17 and 21 in particular preserves validity of additional frame conditions, we can summarise as follows:

**Corollary 22.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be an extended rank-1 logic, and let $L$ be the functorial presentation of $(\Lambda, \mathcal{A})$. If $\Gamma \subseteq \Lambda$ is finite and $\Theta \subseteq \mathcal{F}(\Gamma)$, then the following are equivalent for $\phi \in \mathcal{F}(\Gamma)$:*

1. *$\phi$ is finitely satisfiable in $\mathsf{Coalg}(T, \Theta)$*
2. *$\phi$ is finitely satisfiable in $\mathsf{Coalg}(L^\star, \Theta)$*

Together with Corollary 12 this means that we can reduce satisfiability over $\mathsf{Coalg}(T)$ to satisfiability over $\mathsf{Alg}(L)$; we pursue this theme in the subsequent sections.

## 5 The Finite Model Property

We will now exploit Corollary 22 to reduce the question of $\mathcal{L}$ having the finite model property to the question of an associated equational logic having the finite algebra property. We begin by associating an algebraic theory to every (extended) rank-1 logic $\mathcal{L}$.

**Definition 23.** Let $\mathcal{L} = (\Lambda, \mathcal{A})$ be a rank-1 logic. The algebraic theory *associated with* $\mathcal{L}$ is the pair $(\Sigma, E)$, where

- $\Sigma = \Lambda \cup \{\bot, \to\}$ is the signature of boolean algebra augmented with a unary operator for every modality of $\Lambda$ (silently assuming $\to, \bot \notin \Lambda$)
- $E = E_{\mathsf{BA}} \cup \{\phi = \top \mid \phi \in \mathcal{A}\}$ consists of an equational axiomatisation $E_{\mathsf{BA}}$ of boolean algebra, together with the (equational form of) the axioms of $\mathcal{L}$.

If $\mathcal{T} = (\Sigma, E)$ is the algebraic theory associated with a rank-1 logic, we write $\mathsf{Alg}(\mathcal{T})$ for the category of $(\Sigma, E)$-algebras in the sense of universal algebra [21] and adopt the standard interpretation of terms $\phi \in F(\Lambda)$ given a valuation of the (propositional) variables in $V$. As every $(\Sigma, E)$-algebra in particular carries a boolean algebra structure, we present $(\Sigma, E)$-algebras as $(A, (f_M)_{M \in \Lambda})$ where $A \in \mathsf{BA}$ and $f_M : A \to A$ for all $M \in \Lambda$.

If $\phi \in \mathcal{F}(\Lambda)$ and $\mathbb{A} \in \mathsf{Alg}(\mathcal{T})$, we abbreviate $\mathbb{A} \models \phi$ iff $\mathbb{A} \models \phi = \top$, i.e. $[\![\phi]\!]^{\pi}_{\mathbb{A}} = \top$ for all valuations $\pi : V \to A$. As previously, if $\Theta \subseteq \mathcal{F}(\Lambda)$, we write $\mathsf{Alg}(\mathcal{T}, \Theta)$ for the full subcategory of all $\mathcal{T}$-algebras $\mathbb{A}$ for which $\mathbb{A} \models \phi$ for all $\phi \in \Theta$; note that $\mathsf{Alg}(\mathcal{T}, \Theta) = \mathsf{Alg}(\Sigma, E \cup \Theta)$. If $\phi, \psi \in \mathcal{F}(\Lambda)$, we write $\mathcal{T}, \Theta \vdash \phi = \psi$ if $\phi = \psi$ can be derived equationally from $E \cup \Theta$. A formula $\phi \in \mathcal{F}(\Lambda)$ is $(\mathcal{T}, \Theta)$-*inconsistent*, if $\mathcal{T}, \Theta \vdash \phi = \bot$ and $\phi$ is $(\mathcal{T}, \Theta)$-*consistent*, otherwise.

Finally, a formula $\phi \in \mathcal{F}(\Lambda)$ is *(finitely) satisfiable* in $\mathsf{Alg}(\mathcal{T}, \Theta)$ if there exists a (finite) $\mathbb{A} \in \mathsf{Alg}(\mathcal{T}, \Theta)$ such that $\mathbb{A} \not\models \phi = \bot$.

In essence, if $\mathcal{T}$ is the algebraic theory associated with a rank-1 logic, $\mathcal{T}$-algebras are boolean algebras with operators in the sense of Jónnson and Tarski [11] but without the requirement that the operators preserve either joins or meets. We now relate the categories $\mathsf{Alg}(L)$ and $\mathsf{Alg}(\mathcal{T})$; this is in essence Theorem 15 of [2].

**Lemma 24.** *Let $\mathcal{L}$ be a rank-1 logic with functorial presentation $L$ and algebraic theory $\mathcal{T}$. Then there exists a concrete isomorphism $C : \mathsf{Alg}(L) \to \mathsf{Alg}(\mathcal{T})$ that commutes with the respective forgetful functors, i.e. $U_L \circ C = U_{\mathcal{T}}$ where $U_L : \mathsf{Alg}(L) \to \mathsf{Set}$ and $U_{\mathcal{T}} : \mathsf{Alg}(\mathcal{T}) \to \mathsf{Set}$. Moreover, this isomorphism is compatible with the (algebraic) semantics of modal formulas: If $\phi \in \mathcal{F}(\Lambda)$, $\mathbb{A} = (A, \alpha) \in \mathsf{Alg}(L)$ and $\pi : V \to A$ is a valuation, then $[\![\phi]\!]^{\pi}_{A} = [\![\phi]\!]^{\pi}_{C(\mathbb{A})}$.*

Together with Corollary 22, we obtain:

**Corollary 25.** *Let $\Gamma \subseteq \Lambda$ be finite, and let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be an extended rank-1 logic with $\Theta \subseteq \mathcal{F}(\Gamma)$. Then the following are equivalent for $\phi \in \mathcal{F}(\Gamma)$:*

1. *$\phi$ is finitely satisfiable in $\mathsf{Coalg}(T, \Theta)$*
2. *$\phi$ is finitely satisfiable in $\mathsf{Alg}(\mathcal{T}, \Theta)$.*

We are now in the position to relate the finite model property of a modal logic with the finite algebra property of the associated algebraic theory.

**Definition 26.** Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be an extended rank-1 logic, and let $\mathcal{T}$ be the algebraic theory associated with $(\Lambda, \mathcal{A})$. If $T$ is a structure for $\mathcal{L}$, we say that $\mathcal{L}$ has the finite model property w.r.t. $\mathsf{Coalg}(T, \Theta)$ if every $\mathcal{L}$-consistent formula is finitely satisfiable in $\mathsf{Coalg}(T, \Theta)$. Dually, $\mathcal{T}$ has the finite algebra property w.r.t. $\mathsf{Alg}(\mathcal{T}, \Theta)$ if every $(\mathcal{T}, \Theta)$-consistent formula $\phi \in \mathcal{F}(\Lambda)$ is finitely satisfiable in $\mathsf{Alg}(\mathcal{T}, \Theta)$.

It is easy to see that the above definition of the finite algebra property is equivalent to the standard definition (validity over $\mathsf{Alg}(\mathcal{T}, \Theta)_f$ implies validity over $\mathsf{Alg}(\mathcal{T}, \Theta)$) as we are dealing with extensions of boolean algebras.

The only ingredient that is missing for our first main theorem is the following (standard) lemma that relates equational and modal deduction.

**Lemma 27.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be an extended rank-1 logic, and let $\mathcal{T}$ be the algebraic theory associated with $(\Lambda, \mathcal{A})$. Then*

$$\mathcal{T}, \Theta \vdash \phi = \psi \text{ iff } \mathcal{L} \vdash \phi \leftrightarrow \psi \text{ and } \mathcal{L} \vdash \phi \text{ iff } \mathcal{T}, \Theta \vdash \phi = \top$$

*for all $\phi, \psi \in \mathcal{F}(\Lambda)$.*

Our main conceptual contribution can now be formulated as follows:

**Theorem 28.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be a rank-1 logic, let $\Gamma \subseteq \Lambda$ be finite, and let $\Theta \subseteq \mathcal{F}(\Gamma)$. For $\Gamma \subseteq \Delta \subseteq \Lambda$, let $\mathcal{T}_\Delta$ denote the algebraic theory associated with the $\Delta$-reduct of $(\Lambda, \mathcal{A})$. Then $\mathcal{L}$ has the finite model property w.r.t. $\mathsf{Coalg}(T, \Theta)$ if all $\mathcal{T}_\Delta$, for $\Delta$ finite with $\Gamma \subseteq \Delta \subseteq \Lambda$, have the finite algebra property w.r.t $\mathsf{Alg}(\mathcal{T}_\Delta, \Theta)$. In this case $\mathcal{L}$ is moreover complete w.r.t. $\mathsf{Coalg}(T, \Theta)$, i.e. $\mathcal{L} \vdash \phi$ if $\mathsf{Coalg}(T, \Theta) \models \phi$.*

Some remarks are in order: First we can only allow frame conditions $\Theta$ over finitely many modalities, as our techniques rely on the duality between $\mathsf{FBA}$ and $\mathsf{Fin}$, cf Lemma 9. Second, to establish the finite model property for $\mathcal{L}$, we need to establish the finite algebra property for all $\mathcal{T}_\Delta$ in the terminology of the previous theorem where $\Delta \subseteq \Gamma$ contains the modalities occurring in the frame conditions $\Theta$ together with those occurring in a particular consistent formula that we seek to satisfy. This restriction is however not problematic as we will see in the following two sections.

## 6 The Finite Algebra Property

Theorem 28 leaves an important question unanswered: which algebraic theories enjoy the finite algebra property? This question is partially answered in the present section, where we describe a general mechanism for constructing finite algebras that covers a large variety of cases and generalises [14] to a non-normal setting and – modulo the duality between neighbourhood frames and modal algebras [6] – also [3] to combinations of rank 0/1 formulas and the (4) axiom.

For the whole section, suppose that $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ is an extended rank-1 logic and $\mathcal{T} = (\Sigma, E)$ is the algebraic theory associated with $(\Lambda, \mathcal{A})$. Our goal in this section is to find conditions that ensure that $\mathcal{T}$ has the finite algebra property w.r.t $\mathsf{Alg}(\mathcal{T}, \Theta)$, which we pursue by constructing finite algebras over maximally consistent subsets of closed sets of formulas.

**Definition 29.** The *normalised negation* $\sim \phi$ of a formula is the formula $\neg\phi$, if $\phi$ is not of the form $\neg\psi$, and $\sim \phi = \psi$ if $\phi = \neg\psi$. A set $\Delta \subseteq \mathcal{F}(\Lambda)$ is *closed*, if $\Delta$ contains $\psi$ whenever $\psi$ is a subformula of some $\phi \in \Delta$ and $\Delta$ contains $\sim \phi$ if $\phi \in \Delta$. We write $\mathbf{cl}(\phi)$ for closure (the smallest closed set containing $\phi$) of a single formula $\phi \in \mathcal{F}(\Lambda)$. If $\Delta$ is a closed set and $\Gamma \subseteq \Lambda$ is a set of modal operators, the *$\Gamma$-extension* of $\Delta$ is the set

$$\Delta_\Gamma = \{M\phi \mid \phi \in \Delta, M \in \Gamma\} \cup \{\neg M\phi \mid \phi \in \Delta, M \in \Gamma\} \cup \Delta$$

We write $\mathcal{M}_\Delta$ for the set of maximally $\mathcal{L}$-consistent subsets of a closed set $\Delta$ and drop the subscript if there is no danger of confusion.

Note that $\Delta_\Gamma = \Delta$ if $\Gamma = \emptyset$. The construction of a satisfying model for a consistent formula $\phi$ that we describe will be based on the closure $\mathbf{cl}(\phi)$ or its $\Gamma$-extension for a finite set $\Gamma$ of modalities. The construction uses the map $\sigma$ introduced below.

**Lemma 30.** *Let $\Delta$ be closed and finite. The assignment*

$$\sigma : \mathcal{P}(\mathcal{M}_\Delta) \to \mathcal{F}(\Lambda), A \mapsto \bigvee_{\Phi \in A} \bigwedge \Phi$$

*satisfies $\mathcal{T}, \Theta \vdash \sigma(A \cup B) = \sigma(A) \vee \sigma(B)$ and $\mathcal{T}, \Theta \vdash \sigma(\neg A) = \neg\sigma(A)$.*

Conceptually speaking, $\sigma$ induces a morphism of boolean algebras $\mathcal{P}(\mathcal{M}) \to \mathcal{F}(\Lambda)/\sim$ where $\sim$ is logical equivalence. We now introduce (algebraic) filtrations that we employ to witness the finite algebra property.

**Definition 31.** Let $\Gamma \subseteq \Lambda$, and let $\Delta$ be a closed and finite subset of $\mathcal{F}(\Lambda)$. Let $\mathcal{M}$ denote the collection of maximally consistent subsets of $\Delta_\Gamma$. The *natural valuation* induced by $\Delta$ and $\Gamma$ is the mapping $\tau : V \to \mathcal{P}(\mathcal{M})$, $\tau(p) = \{\Phi \in \mathcal{M} \mid p \in \Phi\}$. A $\Sigma$-algebra $\mathbb{F} = (\mathcal{P}(\mathcal{M}), (f_M)_{M\in\Lambda})$ is a $\Delta(\Gamma)$-*filtration* if it satisfies

$$f_M(\{\Phi \in \mathcal{M} \mid \phi \in \Phi\}) = \{\Phi \in \mathcal{M} \mid M\phi \in \Phi\}$$

for all $M\phi \in \Delta$. If $\Gamma = \emptyset$ we will simply speak of a $\Delta$-filtration. We call a $\Delta(\Gamma)$-filtration *safe* for a set $\Phi \subseteq \mathcal{F}(\Lambda)$ of axioms if $\mathbb{F} \models \phi = \top$ for all $\phi \in \Phi$.

Informally, a $\Delta(\Gamma)$-filtration puts a $\Sigma$-algebra structure on the set $\mathcal{P}(\mathcal{M})$ of sets of maximally consistent subsets of $\Delta_\Gamma$ such that the truth lemma is satisfied for all $\phi \in \Delta$. This will be applied to the case where $\Delta = \mathbf{cl}(\phi)$ is the closure of a single formula and we will usually choose $\Gamma = \emptyset$; however in some cases (notably in presence of the (5) axiom) we need to rely on the additional structure provided by formulas $\psi \in \Delta_\Gamma \setminus \Delta$ to prove the truth lemma for formulas of $\Delta$. The structure present in $\Delta(\Gamma)$-filtrations clearly suffices to prove the (algebraic) truth lemma and the finite algebra property follows if all axioms are safe.

**Proposition 32.** *Let $\Delta \subseteq \mathcal{F}(\Lambda)$ be closed, let $\Gamma \subseteq \Lambda$, and let $\mathbb{F}$ be a $\Delta(\Gamma)$ filtration. Then*

$$[\![\phi]\!]^\tau_{\mathbb{F}} = \{\Phi \in \mathcal{M}_{\Delta_\Gamma} \mid \phi \in \Phi\}$$

*for all $\phi \in \Delta$. If moreover every closed and finite set $\Delta$ admits a $\Delta(\Gamma)$-filtration for a finite subset $\Gamma \subseteq \Lambda$ that is safe for $\mathcal{A} \cup \Theta$, then $\mathcal{T}$ has the finite algebra property w.r.t.* $\mathsf{Alg}(\mathcal{T}, \Theta)$.

The remainder of this section is devoted to showing that every modal logic $\mathcal{L}{=}(\Lambda, \mathcal{A}, \Theta)$ where $\Theta$ is rank-0/1 admits a safe $\Delta$-filtration for every closed set $\Delta \subseteq \mathcal{F}(\Lambda)$; we refer the reader to Definition 3 for the notions of rank-0/1 axioms and logics.

**Definition 33.** Let $\Delta$ be closed and finite. A *sieve* over $\Delta$ is a mapping $\nu : \mathcal{M} \to \mathcal{P}(\mathcal{F}(\Lambda))$ such that $\Phi \subseteq \nu(\Phi)$ for all $\Phi \in \mathcal{M}$ and $\nu(\Phi)$ is maximally $\mathcal{L}$-consistent. The $\Sigma$-algebra $(\mathcal{P}(\mathcal{M}_\Delta), (f_M)_{M\in\Lambda})$ where

$$f_M : \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M}), \quad A \mapsto \{\Phi \in \mathcal{M} \mid \nu(\Phi) \vdash M(\sigma(A))\}$$

is called the *standard $\Delta$-filtration* defined by the sieve $\nu$.

We usually omit the explicit mention of the operators and just use $\mathcal{P}(\mathcal{M})$ to refer to the standard filtration. Note that the standard filtration implicitly depends on a choice of maximally consistent extension $\nu(\Phi)$ of $\Phi \in \mathcal{M}$, but the choice of $\nu(\Phi)$ is immaterial.

**Lemma 34.** *Let $\mathcal{P}(\mathcal{M})$ be the standard $\Delta$-filtration for a closed and finite set $\Delta \subseteq \mathcal{F}(\Lambda)$ given by a sieve $\nu$. Then $f_M(\{\Phi \in \mathcal{M} \mid \phi \in \Phi\}) = \{\Phi \in \mathcal{M} \mid M\phi \in \Phi\}$ for all $M \in \Lambda$.*

We are now in the position to prove that the algebraic theory associated to a rank-0/1 logic has the finite algebra property if we can show that the standard filtration is safe for all rank-0/1 axioms.

**Proposition 35.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be rank-0/1 and assume that $\Delta$ is closed and finite. Then the standard $\Delta$-filtration defined by any sieve is safe for $\mathcal{A} \cup \Theta$.*

Consequently, we have the finite algebra property for all rank 0/1 logics.

**Theorem 36.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be rank-0/1, and let $\mathcal{T}$ be the algebraic theory associated with $(\Lambda, \mathcal{A})$. Then $\mathcal{T}$ has the finite algebra property w.r.t. $\mathsf{Alg}(\mathcal{T}, \Theta)$.*

The finite model property for $\mathcal{L}$ now follows at once from Theorem 28.

**Corollary 37.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ be rank-0/1, and let $\Theta \subseteq \mathcal{F}(\Gamma)$ for a finite subset $\Gamma \subseteq \Lambda$. Then $\mathcal{L}$ has the finite model property w.r.t. $\mathsf{Coalg}(T, \Theta)$; in particular $\mathcal{L}$ is complete w.r.t. $\mathsf{Coalg}(T, \Theta)$.*

## 7 Frame Conditions beyond Rank 0/1

We now extend the ideas developed in the previous section to logics beyond rank 0/1. As the standard filtration does not necessarily satisfy the frame conditions imposed by axioms beyond rank 0/1, we have to adapt the standard filtration accordingly. We treat three instances of frame conditions beyond rank-1 in detail: instances of the (4) axiom, generalised to not necessarily normal operators, and instances of the (B) and (5) axioms for normal operators, the latter being primarily of interest in the context of logics that additionally feature non-normal operators.

**Definition 38.** Let $\nu$ be a sieve over a finite closed set $\Delta$. For $M \in \Lambda$ we write $M^\star$ for $M$'s dual $\neg M \neg$. If $\Phi, \Psi \in \mathcal{M}_\Delta$ we say that $\Phi$ *evolves* to $\Psi$ along $M$ ($\Phi \rightsquigarrow_M \Psi$) iff $\nu(\Phi) \vdash M^\star \bigwedge \Psi$. Similarly $\Phi$ *zigzags* to $\Psi$ along $M$ ($\Phi \leftrightsquigarrow_M \Psi$) iff there are $\Omega_0, \ldots, \Omega_n$ such that

- $\Omega_0 \rightsquigarrow_M \Phi$ and $\Omega_n \rightsquigarrow_M \Psi$
- $\Omega_i \rightsquigarrow_M \Omega_{i-1}$ or $\Omega_{i-1} \rightsquigarrow_M \Omega_i$ for all $i = 1, \ldots, n$.

We consider the following operators of type $\mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ (see Definition 33 for the definition of $f_M$)

$$f^B_M(A) = \{\Psi \in f_M(A) \mid \forall \Phi \in \mathcal{M}(\Phi \rightsquigarrow_M \Psi \Rightarrow \Psi \in A)\}$$
$$f^5_M(A) = \{\Psi \in f_M(A) \mid \forall \Phi \in \mathcal{M}(\Phi \leftrightsquigarrow_M \Psi \Rightarrow \Phi \in A)\}$$
$$f^{\bullet 4}_M(A) = \mathrm{gfp}(X \mapsto f^\bullet_M(A) \cap f^\bullet_M(X))$$

where gfp denotes the greatest fixpoint and in the last line, $\bullet$ stands for nothing or one of B, 5.

As we will see later, each of the above operators can be used to force the validity of the corresponding axioms in a filtration so that e.g. $f^{45}$ forces the validity of the axioms $(4)$ and $(5)$ if the corresponding axioms are derivable in $\mathcal{L}$.

**Definition 39.** Every modality $M \in \Lambda$ is of type $E$. The operator $M$ is normal, if $\mathcal{L} \vdash M(p \to q) \to Mp \to Mq$ and $\mathcal{L} \vdash M\top$. It is *monotone* if $\mathcal{L} \vdash M(p \wedge q) \to Mp$. Moreover $M$ is of type

- 4, if $M$ is monotone and $\mathcal{L} \vdash Mp \to MMp$
- $B$, if $M$ is normal and $\mathcal{L} \vdash p \to MM^\star p$
- 5 if $M$ is normal and $\mathcal{L} \vdash M^\star p \vdash MM^\star p$

where again $M^\star = \neg M \neg$ denotes $M$'s dual. We say that an operator is of type $4B$ (of type 45) if it is both of type 4 and of type $B$ (of type 5).

We now show that the operators defined above give rise to a $\Delta(\Gamma)$ filtration provided the types of modalities match the associated operators.

**Proposition 40.** *Let $t(M) \in \{E, 4, 5, B, 4B, 45\}$, and let $M$ be of type $t(M)$ for all $M \in \Lambda$. If $\Delta \subseteq \mathcal{F}(\Lambda)$ is closed then the $\Sigma$-algebra $(\mathcal{P}(\mathcal{M}_{\Delta_\Gamma}), (f_M^{t(M)})_{M\in\Lambda})$ is a $\Delta(\Gamma)$-filtration if $\Gamma \supseteq \{M \in \Lambda \mid t(M) = 5$ or $t(M) = 45\}$.*

This last proposition needs the extra generality provided by a $\Delta(\Gamma)$ filtration for $\Gamma \neq \emptyset$ to prove the truth lemma for modalities of type $5, 45$ which is not needed to show safety of axioms:

**Proposition 41.** *Let $t(M) \in \{E, 4, 5, B, 4B, 45\}$, and let $M$ be of type $t(M)$ for all $M \in \Lambda$. If $\Delta \subseteq \mathcal{F}(\Lambda)$ is closed then the $\Sigma$-algebra $(\mathcal{P}(\mathcal{M}_\Delta), (f_M^{t(M)})_{M\in\Lambda})$ satisfies*

- *$Mp \to MMp$ if $t(M)$ is one of 4, 4B, 45*
- *$M^\star p \to MM^\star p$ if $t(M) = 5$ or $t(M) = 45$*
- *$p \to MM^\star p$ if $t(M) = B$ or $t(M) = 4B$.*

*where again $M^\star$ is the dual of $M$.*

The next proposition collects additional safe axioms:

**Proposition 42.** *Let $t(M) \in \{E, B, 5, 4, 4B, 45\}$, and let $M$ be of type $t(M)$ for all $M \in \Lambda$. Then the following axioms are valid in the filtration $(\mathcal{P}(\mathcal{M}), (f_M^{t(M)})_{M\in\Lambda})$*

1. *any clause whose right side only mentions operators of type $E$, that is all clauses $\bigwedge_i M_i\phi_i \wedge \bigwedge_j p_j \to \bigvee_j M_k\phi_k \vee \bigvee_l p_l$ where $t(M_k) = E$ for all $k$*
2. *truth and falsity preservation, i.e. the axioms $M\top$ and $\neg M\bot$ for all $M \in \Lambda$*
3. *preservation of conjunctions, i.e. the axiom $M(p \wedge q) = Mp \wedge Mq$*
4. *the $(S)$-axiom $p \to Mp$*

As $T$-coalgebras are just Kripke frames for $T = \mathcal{P}$, the preceding proposition immediately gives the finite algebra property (and hence the finite model property) for a large number of well-studied systems, including $K, KT, KB, KB4, S4, S5$. We discuss two examples that apply our techniques outside the realm of normal logics in the next section.

## 8 Reasoning about Uncertainty and Knowledge

This section shows how the theory developed in the preceding sections can be instantiated to obtain synthetic proofs of completeness and the finite model property for logics of uncertainty and belief discussed in [7]. We fix a finite set $N$ of agents and the modal signature $\Lambda = \{\mathsf{K}_n \mid n \in N\} \cup \{L^i_p \mid p \in [0,1] \cap \mathbb{Q}, i \in N\}$. We read the formula $\mathsf{K}_i\phi$ as "agent $i$ knows phi" and $L^i_p\phi$ as "according to agent $i$, the formula $\phi$ hods with probability at least $p$". Formulas of $\mathcal{F}(\Lambda)$ are interpreted over $T$-coalgebras where $TX = \prod_{i\in N} \mathcal{D}(X) \times \mathcal{P}(X)$ and $\mathcal{D}(X)$ is the probability distributions functor $\mathcal{D}(X) = \{\mu : X \to [0,1] \mid \{x \in X \mid \mu(x) > 0\}$ is finite and $\sum_{x\in X} \mu(x) = 1\}$. That is, a $T$-coalgebra consists of a carrier set $C$ and additionally, for each agent $i \in N$, a relation $R_i \subseteq C \times C$ and a function $f_i : C \to \mathcal{D}(C)$ that assigns probabilities to events in $C$. We turn $T$ into a structure for $\Lambda$ by virtue of the predicate liftings

$$[\![\mathsf{K}_i]\!]_X(A) = \{(\mu_i, S_i)_{i\in N} \mid S_i \subseteq A\}$$
$$[\![L^i_p]\!]_X(A) = \{(\mu_i, S_i)_{i\in N} \mid \textstyle\sum_{x\in A} \mu_i(x) \geq p\}.$$

The semantics of $\mathcal{F}(\Lambda)$ differs slightly from the semantics discussed in [7] as probability distributions are supposed to have finite support; however this difference is immaterial in the light of the finite model property.

It follows from [5] that a one-step complete axiomatisation of each $\mathsf{K}_i$ together with a one-step complete axiomatisation of the $L^i_p$ is one-step complete for $T$; we denote the full set of axioms by $\mathcal{A}$. We refer to [5] for the precise form of the axioms and note that the one-step rule in *loc.cit.* can be equivalently presented as a rank-1 axiom by virtue of Proposition 15 in [17]. However note that any such one-step complete axiomatisation only prescribes the $K$-axioms $\mathsf{K}_i\top = \top$ and $\mathsf{K}_i(a \wedge b) \leftrightarrow \mathsf{K}_i a \wedge \mathsf{K}_i b$ for the knowledge operators $K_i$, and no interaction of knowledge and quantitative uncertainty on the logical level. On top of the basic (one-step complete) axiomatisation $\mathcal{A}$, we consider the axioms ($\mathsf{K}^\star_i = \neg\mathsf{K}_i\neg$)

| | | | |
|---|---|---|---|
| (4) | $\mathsf{K}_i\mathsf{K}_i p \to \mathsf{K}_i p$ | $(C)$ | $\mathsf{K}_i p \to L^i_1 p$ |
| (5) | $\mathsf{K}^\star_i p \to \mathsf{K}_i\mathsf{K}^\star_i p$ | $(U)$ | $p \to L^i_1 p$ |
| $(B)$ | $p \to \mathsf{K}_i\mathsf{K}^\star_i p$ | $(S)$ | $p \to \mathsf{K}_i p$ |

where the names of the axioms on the right abbreviate the corresponding frame properties of [7]. Theorems 28 together with Propositions 40 and 42 now show that any extension of $\mathcal{A}$ with one or more of the above axioms has the finite model property. We can even equip different agents with different reasoning facilities, i.e. allow positive introspection for some but not for others. This generalises the corresponding result in [7] in that we establish the finite model property for all logics $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$, where $\Theta$ is a sub-collection of the above frame conditions. By understanding every axiom $(A)$ as the collection of instances of $(A)$ for all agents $i \in N$ we can restrict the validity of axioms to specific sets of agents in the next theorem.

**Theorem 43.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ and suppose that $\Theta \subseteq \{(4), (5), (B), (C), (U), (S)\}$ not containing both $(B)$ and $(5)$ for the same agent. Then $\mathcal{L}$ has the finite model property w.r.t.* $\mathsf{Coalg}(T, \Theta)$.

## 9 A Logic for Coalitions and Filibusters

As a second example, we apply our techniques to prove completeness and the finite model property for an extension of Pauly's coalition logic [16]. We consider a fixed set $N = \{1, \dots, n\}$ of *agents*. Subsets of $N$ are called *coalitions*. The signature $\Lambda$ of coalition logic consists of modal operators $[C]$, where $C$ ranges over coalitions, read 'coalition $C$ has a collaborative strategy to ensure that ...'. A coalgebraic semantics for coalition logic is based on the class-valued signature functor $T$ defined by

$$TX = \{(S_1, \dots, S_n, f) \mid \emptyset \neq S_i \in \mathsf{Set}, f : \textstyle\prod_{i \in N} S_i \to X\}.$$

The elements of $TX$ are understood as *strategic games* with set $X$ of states, i.e. tuples consisting of nonempty sets $S_i$ of *strategies* for all agents $i$, and an *outcome function* $(\prod S_i) \to X$. A $T$-coalgebra is a *game frame* [16]. We denote the set $\prod_{i \in C} S_i$ by $S_C$, and for $\sigma_C \in S_C, \sigma_{\bar{C}} \in S_{\bar{C}}$, where $\bar{C} = N - C$, $(\sigma_C, \sigma_{\bar{C}})$ denotes the obvious element of $\prod_{i \in N} S_i$. A $\Lambda_{\mathsf{C}}$-structure over $T$ is defined by

$$[\![[C]]\!]_X(A) = \{(S_1, \dots, S_n, f) \in TX \mid \exists \sigma_C \in S_C. \forall \sigma_{\bar{C}} \in S_{\bar{C}}. f(\sigma_C, \sigma_{\bar{C}}) \in A\}.$$

A one-step complete axiomatisation $\mathcal{A}$ of coalition logic consists of the axioms $(\top)$, $(\bot)$, $(N)$ and $(S)$ below

$$\begin{array}{lll} (\bot) & \neg[C]\bot & \\ (\top) & [C]\top & \\ (N) & [\emptyset]\phi \vee [N]\neg\phi & \\ (S) & [C_1]\phi \wedge [C_2]\phi \to [C_1 \cup C_2](\phi \wedge \psi) & \end{array} \qquad \begin{array}{lll} (C) & [C]\phi \to \phi & \\ (F) & [C][C]\phi \to [C]\phi & \\ (P) & \phi \to [C]\phi & \end{array}$$

where $C_1 \cap C_2 = \emptyset$ in the superadditivity axiom $(S)$. One-step completeness of $\mathcal{A}$ is proved in [18] using the rule format of the axioms above. Additionally, we consider a subset $\Theta \subseteq \{(C), (F), (P)\}$, again with axioms in $\Theta$ possibly restricted to specific coalitions.

The $(C)$ axiom expresses that the power of a coalition is limited to forcing things that are already valid; we think of such coalitions as conservative. In a similar vein, if $\phi$ expresses the act of blocking a motion and a coalition (of senators) has the power to achieve $\phi$, then the $(F)$-axiom (together with monotonicity) expresses that $C$ can block this motion indefinitely. Accordingly we refer to $(F)$ as the filibuster axiom. Finally, by virtue of axiom $(P)$, a coalition can perpetuate properties of a strategic game. Using the same convention as in the previous example, we understand each axiom $(A)$ as the collection of instances of $(A)$ for all coalitions. Hence a subset of $\{(C), (F), (P)\}$ in general only contains instances of each axiom for a specific set of coalitions. Again, the combination of Theorem 28 and Propositions 40 and 42 shows:

**Theorem 44.** *Let $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ where $\Theta \subseteq \{(C), (F), (P)\}$. Then $\mathcal{L}$ has the finite model property w.r.t* $\mathsf{Coalg}(T, \Theta)$*; in particular $\mathcal{L}$ is complete w.r.t.* $\mathsf{Coalg}(T, \Theta)$.

*Acknowledgements.* The first author would like to thank Tomasz Kowalski for discussions on algebraic filtrations during the conference TANCL 2007.

## References


1. Bonsangue, M., Kurz, A.: Duality logics for transition systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 455–469. Springer, Heidelberg (2005)
2. Bonsangue, M., Kurz, A.: Presenting functors by operations and equations. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 172–186. Springer, Heidelberg (2006)
3. Chellas, B.: Modal Logic, Cambridge (1980)
4. Cîrstea, C.: A compositional approach to defining logics for coalgebras. Theoret. Comput. Sci. 327, 45–69 (2004)
5. Cîrstea, C., Pattinson, D.: Modular construction of modal logics. Theoret. Comput. Sci. (to appear). In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 258–275. Springer, Heidelberg (2004)
6. Dŏsen, K.: Duality between modal algebras and neighbourhood frames. Studia Logica 48(2), 219–234 (1989)
7. Fagin, R., Halpern, J.: Reasoning about knowledge and probability. J. ACM 41, 340–367 (1994)
8. Fine, K.: In so many possible worlds. Notre Dame J. Formal Logic 13, 516–520 (1972)
9. Heifetz, A., Mongin, P.: Probabilistic logic for type spaces. Games and Economic Behavior 35, 31–53 (2001)
10. Johnstone, P.: Stone spaces. Cambridge Studies in Advanced Mathematics, vol. 3. Cambridge University Press, Cambridge (1993)
11. Jónnson, B., Tarski, A.: Boolean algebras with operators I. Amer. J. Math. 73, 891–939 (1951)
12. Kupke, C., Kurz, A., Venema, Y.: Stone coalgebras. Theor. Comput. Sci. 327(1–2), 109–134 (2004)
13. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inform. Comput. 94(1), 1–28 (1991)
14. Lemmon, E.: Algebraic semantics for modal logics I. Journal of Symbolic Logic 31(1), 46–65 (1966)
15. Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. Theoret. Comput. Sci. 309, 177–193 (2003)
16. Pauly, M.: A modal logic for coalitional power in games. J. Logic Comput. 12(1), 149–166 (2002)
17. Schröder, L.: A finite model construction for coalgebraic modal logic. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 157–171. Springer, Heidelberg (2006)
18. Schröder, L., Pattinson, D.: PSPACE reasoning for rank-1 modal logics. In: Logic in Computer Science, LICS 2006, pp. 231–240. IEEE, Los Alamitos (2006)
19. Schröder, L., Pattinson, D.: Compositional algorithms for heterogeneous modal logics. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, Springer, Heidelberg (2007)
20. Schröder, L., Pattinson, D.: Rank-1 modal logics are coalgebraic. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 573–585. Springer, Heidelberg (2007)
21. Wechler, W.: Universal Algebra for Computer Scientists. EATCS Monographs on Theoretical Computer Science, vol. 25. Springer, Heidelberg (1992)

# A Linear-non-Linear Model for a Computational Call-by-Value Lambda Calculus (Extended Abstract)

Peter Selinger[1] and Benoît Valiron[2]

[1] Dalhousie University
selinger@mathstat.dal.ca
[2] University of Ottawa
bvali087@uottawa.ca

**Abstract.** We give a categorical semantics for a call-by-value linear lambda calculus. Such a lambda calculus was used by Selinger and Valiron as the backbone of a functional programming language for quantum computation. One feature of this lambda calculus is its linear type system, which includes a duplicability operator "!" as in linear logic. Another main feature is its call-by-value reduction strategy, together with a side-effect to model probabilistic measurements. The "!" operator gives rise to a comonad, as in the linear logic models of Seely, Bierman, and Benton. The side-effects give rise to a monad, as in Moggi's computational lambda calculus. It is this combination of a monad and a comonad that makes the present paper interesting. We show that our categorical semantics is sound and complete.

## 1 Introduction

In the last few years, there has been some interest in the semantics of quantum programming languages. [18] gave a denotational semantics for a flow-chart language, but this language did not include higher-order types. Several authors defined quantum lambda calculi [21,19] as well as quantum process algebras [11,12], which had higher-order features and a well-defined operational semantics, but lacked denotational semantics. [20] gave a categorical model for a higher-order quantum lambda calculus, but omitted all the non-linear features (i.e., classical data). Meanwhile, Abramsky and Coecke [2,9] developed categorical axiomatics for Hilbert spaces, but there is no particular language associated with these models.

In this paper, we give the first categorical semantics of an unabridged quantum lambda calculus, which is a version of the language studied in [19].

For the purposes of the present paper, an understanding of the precise mechanics of quantum computation is not required. We will focus primarily on the type system and language, and not on the structure of the actual "built-in" quantum operations (such as unitary operators and measurements). In this sense, this paper is about the semantics of a generic call-by-value linear lambda calculus, which is parametric on some primitive operations that are not further

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 81–96, 2008.

explained. It should be understood, however, that the need to support primitive quantum operations motivates particular features of the type system, which we briefly explain now.

The first important language feature is linearity. This arises from the well-known *no-cloning* property of quantum computation, which asserts that quantum data cannot be duplicated [23]. So if $x : qbit$ is a variable representing a quantum bit, and $y : bit$ is a variable representing a classical bit, then it is legal to write $f(y, y)$, but not $g(x, x)$. In order to keep track of duplicability at higher-order types we use a type system based on linear logic. We use the duplicability operator "!" to mark classical types. In the categorical semantics, this operator gives rise to a comonad as in the work of [16] and [6]. Another account of mixing copyable and non-copyable data is [10], where the copyability is internal to objects.

A second feature of quantum computation is its probabilistic nature. Quantum physics has an operation called measurement, which converts quantum data to classical data, and whose outcome is inherently probabilistic. Given a quantum state $\alpha|0\rangle + \beta|1\rangle$, a measurement will yield output 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. To model this probabilistic effect in our call-by-value setting, our semantics requires a computational monad in the sense of [14]. The coexistence of the computational monad $T$ and the duplicability comonad ! in the same category is what makes our semantics interesting and novel. It differs from the work of [7], who considered a monad and a comonad one two different categories, arising from a single adjunction.

The computational aspects of linear logic have been extensively explored by many authors, including [8,6,5,1,22]. However, these works contain explicit lambda terms to witness the structural rules of linear logic, for example, $x : !A \rhd \mathrm{derelict}(x) : A$. By contrast, in our language, structural rules are implicit at the term level, so that $!A$ is regarded as a subtype of $A$ and one writes $x : !A \rhd x : A$. As we have shown in [19], linearity information can automatically be inferred by the type checker. This allows the programmer to program as in a non-linear language.

This use of subtyping is the main technical complication in our proof of well-definedness of the semantics. This is because one has to show that the denotation is independent of the choice of a potentially large number of possible derivations of a given typing judgment. We are forced to introduce a Church-style typing system, and to prove that the semantics finally does not depend on the additional type annotations.

Another technical choice we made in our language concerns the relation between the exponential ! and the pairing operation. Linear logic only requires $!A \otimes !B \rhd !(A \otimes B)$ and not the opposite implication. However, in our programming language setting, we find it natural to identify a classical pair of values with a pair of classical values, and therefore we will have an isomorphism $!A \otimes !B \cong !(A \otimes B)$.

The plan of the paper is the following. First, we describe the lambda calculus and equational axioms we wish to consider. Then, we develop a categorical

model, called linear category for duplication, which is inspired by [8] and [14]. We then show that the language is an internal language for the category, thus obtaining soundness and completeness.

## 2 The Language

We will describe a linear typed lambda calculus with higher-order functions and pairs. The language is designed to manipulate both classical data, which is duplicable, and quantum data, which is non-duplicable. For simplicity, we assume the language is strictly linear, and not affine linear as in [19]. This means duplicable values are both copyable and discardable, whereas non-duplicable values must be used once, and only once.

### 2.1 The Type System

The set of types is given as follows: *Type* $A, B ::= \alpha \mid (A \multimap B) \mid (A \otimes B) \mid \top \mid !A$.

Here $\alpha$ ranges over type constants. While the remainder of this paper does not depend on the choice of type constants, in our main application [19] this is intended to include a type *qbit* of quantum bits, and a type *bit* of classical bits. $A \multimap B$ stands for functions from $A$ to $B$, $A \otimes B$ for pairs, $\top$ for the unit type, and $!A$ for duplicable objects of types $A$. We denote $!!\cdots!A$ with $n$ !'s by $!^nA$.

The intuitive definition of $!A$ is the key to the spirit in which we want the language to be understood: The ! on $!A$ is understood as specifying a property, rather than additional structure, on the elements of $A$. Therefore, we will have $!A \cong !!A$. Whether or not a given value of type $A$ is also of type $!A$ should be something that is inferred, rather than specified in the code.

Since a term of type $!A$ can always be seen as a term of type $A$, we equip the type system with a subtyping relation as follows: Provided that $(m = 0) \vee (n \geqslant 1)$,

$$\frac{}{!^n\alpha <: !^m\alpha}\ (ax), \qquad \frac{}{!^n\top <: !^m\top}\ (\top),$$

$$\frac{A <: A' \quad B <: B'}{!^n(A' \multimap B) <: !^m(A \multimap B')}\ (\multimap), \qquad \frac{A <: A' \quad B <: B'}{!^n(A \otimes B) <: !^m(A' \otimes B').}\ (\otimes).$$

This relation encapsulates the main properties terms should satisfy with respect to duplicability.

### 2.2 Terms

The language consists of the following typed terms, divided into *values* on the one hand, and general terms, or *computations*, on the other. Both share a subset of the values, the *core values*.

$$\textit{CoreValue } U ::= x^A \mid c^A \mid *^n \mid \lambda^n x^A.M,$$

$$\textit{Value } V, W \ ::= U \mid \langle V, W\rangle \mid \textit{ let } x^A = V \textit{ in } W \mid \textit{ let } \langle x^A, y^B\rangle^n = V \textit{ in } W \mid \textit{ let } * = V \textit{ in } W,$$

$$\textit{Term } M, N \ ::= U \mid \langle M, N\rangle \mid (MN) \mid \textit{ let } \langle x^A, y^B\rangle^n = M \textit{ in } N \mid \textit{ let } * = M \textit{ in } N,$$

where $n$ is an integer, $c$ ranges over a set of constant terms, $x$ over a set of term variables and $\alpha$ over a set of constant types. We abbreviate $(\lambda^0 x^A.M)N$ by $let\ x^A = N\ in\ M$, $\lambda^n x^{!^m\top}.\ let * = x^\top\ in\ M$ by $\lambda^n *^m.M$ and we omit numerical indexes when they are null.

Note that the above terms carry Church-style type annotations, as well as integer superscripts; we call these terms *indexed terms*. We also define a notion of *untyped terms* as terms with no index:

$$\begin{aligned} \mathit{PureTerm}\ M, N ::= {} & x \mid c \mid * \mid \lambda x.M \mid (MN) \mid \langle M, N\rangle \mid \\ & let\langle x, y\rangle = M\ in\ N \mid let * = M\ in\ N\,. \end{aligned}$$

The erasure operation $\mathit{Erase} : \mathit{Term} \to \mathit{PureTerm}$ is defined as the operation of removing the types and integers attached to a given indexed term. If $M = \mathit{Erase}(\bar{M})$, we say that $\bar{M}$ is an *indexation of* $M$.

Finally, we define an $\alpha$-equivalence on terms, denoted by $=_\alpha$, in the usual way (see for example [3]).

### 2.3 Duplicable Pairs and Pairs of Duplicable Elements

Before we formally present the type system, let us informally motivate our choice of typing rules. One non-obvious choice we had to make is for the interaction of pairs and duplicability. Unlike previous works with comonads [8,5], we want to think of the type $!(A \otimes B)$ as a type of pairs of elements of type $A$ and $B$: we want to use the same operation to access the components as one would use for a pair of type $A \otimes B$, without having to use a dereliction operation.

This immediately raises a concern: consider a pair of elements $\langle x, y\rangle$ of type $!(A \otimes B)$. Are $x$ and $y$ duplicable? In the usual linear logic interpretation, they are not. Having a infinite supply of pair of shoes does not mean one has an infinite supply of right shoes: we cannot discard the left shoes. On the other hand, in our interpretation of "classical" data as residing in "classical" memory and therefore being duplicable, if the string $\langle x, y\rangle$ is duplicable, then so should be the elements $x$ and $y$. In other words, we want the duplication to "permeate" the pairing.

The choice of such a "permeable" pairing is more or less forced on us by our desire to have no explicit term syntax for structural rules. Consider the following untyped terms, which can be typed if $t$ is of type $!(A \otimes !(B \otimes C))$:

$$let\langle x, u\rangle = t\ in\ let\langle y, z\rangle = u\ in\ \langle\langle z, y\rangle, x\rangle, \tag{1}$$

$$let\langle x, u\rangle = t\ in\ \langle let\langle y, z\rangle = u\ in\ \langle z, y\rangle, x\rangle. \tag{2}$$

First, we expect these two terms to be axiomatically equal. Term (2) should be of type $!(!(C \otimes B) \otimes A)$, regardless of the permeability of the pairing: if $\langle y, z\rangle$ is duplicable, so should be $\langle z, y\rangle$. Now, consider the term (1) with a non-permeable pairing. In the naive type system, $u$ ends up being of type $B \otimes C$, and the variables $y$ and $z$ in the final recombination end up being respectively of type $B$ and $C$. It is not possible to make $\langle z, y\rangle$ of the duplicable type $!(C \otimes B)$.

We therefore choose a permeable pairing, which will be reflected, albeit subtly, in the typing rule $(\otimes.I)$ and $(\otimes.E)$ in the following section.

**Table 1.** Typing rules

$$\frac{A <: B}{!\Delta, x : A \rhd x^B : B}\ (ax_1) \qquad \frac{A_c <: B}{!\Delta \rhd c^B : B}\ (ax_2) \qquad \frac{\Gamma_1, !\Delta \rhd M : A \multimap B \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd MN : B}\ (app)$$

$$\frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda^0 x^A.M : A \multimap B}\ (\lambda_1) \qquad \frac{!\Delta, x : A \rhd M : B}{!\Delta \rhd \lambda^{n+1} x^A.M : !^{n+1}(A \multimap B)}\ (\lambda_2) \qquad \frac{}{!\Delta \rhd *^n : !^n \top}\ (\top.I)$$

$$\frac{!\Delta, \Gamma_1 \rhd M_1 : !^n A_1 \quad !\Delta, \Gamma_2 \rhd M_2 : !^n A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle^n : !^n (A_1 \otimes A_2)}\ (\otimes.I) \qquad \frac{!\Delta, \Gamma_1 \rhd M : \top \quad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd let\ * = M\ in\ N : A}\ (\top.E)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n (A_1 \otimes A_2) \quad !\Delta, \Gamma_2, x_1 : !^n A_1, x_2 : !^n A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd let \langle x_1^{A_1}, x_2^{A_2} \rangle^n = M\ in\ N : A}\ (\otimes.E)$$

### 2.4 Typing Judgments

A typing judgment is a tuple $\Delta \rhd M : A$, where $M$ is an indexed term, $A$ is a type, and $\Delta$ is a typing context. To each constant term $c$ we assign a type $!A_c$. A *valid typing judgment* is a typing judgment that can be derived from the typing rules in Table 1. We use the notation $!\Delta$ for a context where all variables have a type of the form $!A$. Finally, when we write a context $\Gamma, \Delta$, we assume the contexts $\Gamma$ and $\Delta$ to be disjoint.

The following lemmas are proved by structural induction on terms or type derivations as appropriate.

**Lemma 1.** *If $V$ is a value such that $\Delta \rhd V : !A$ is a valid typing judgment, then $\Delta = !\Delta'$ for some context $\Delta'$.* □

**Lemma 2.** *Consider the following valid typing judgment: $\Delta, x : A \rhd M : B$. Then for every free instance $x^{A'}$ in $M$, $A <: A'$.* □

**Definition 1.** In a typing judgment $\Delta \rhd M : A$, a term variable $x \in |\Delta|$ is called *dummy* if $x \notin FV(M)$.

**Lemma 3.** *Any dummy variable $x$ in $\Delta \rhd M : B$ satisfies $\Delta(x) = !A$, for some $A$. Conversely, if $\Delta \rhd M : B$ is valid and if $x \notin FV(M)$, then for all types $A$ the typing judgment $\Delta, x : !A \rhd M : B$ is valid.* □

Typing derivations are not unique *per se*. However for a given valid typing judgement $\Delta \rhd M : A$ two typing derivations will only differ with respect to the placement of *dummy variables*, namely the unused variables in context.

### 2.5 Type Casting and Substitution Lemma

**Lemma 4.** *Suppose $\Delta \rhd M : A$ is a valid typing judgment, and suppose $\Delta' <: \Delta$ and $A <: A'$. Then there exists a canonical valid typing judgment $\Delta' \rhd M' : A'$ such that $Erase(M) = Erase(M')$. Moreover, if $M$ is a value, so is $M'$.*

*Proof.* By induction on $M$. □

We will denote this $M'$ with $\{\Delta' <: \Delta \rhd M : A <: A'\}$. If $\Delta' = \Delta$ or $A' = A$, we omit them for clarity.

**Table 2.** Axiomatic equivalence axioms

$$
\begin{array}{lll}
(\beta_\lambda)\ \Delta \rhd let\ x = V\ in\ M & \approx_{ax} M[V/x] & : A \\
(\beta_\otimes)\ \Delta \rhd let \langle x, y\rangle^n = \langle V, W\rangle^n\ in\ M & \approx_{ax} M[V/x, W/y] & : A \\
(\beta_*)\ \Delta \rhd let * = *\ in\ M & \approx_{ax} M & : A \\
(\eta_\lambda)\ \Delta \rhd \lambda^n x^A.\{V : !^n(A \multimap B) \lessdot A \multimap B\} x^A & \approx_{ax} V & : !^n(A \multimap B). \\
(\beta^2_\lambda)\ \Delta \rhd let\ x^A = N\ in\ x^A & \approx_{ax} N & : A. \\
(\eta_\otimes)\ \Delta \rhd let \langle x^A, y^B\rangle^n = N\ in\ \langle x^{!^n A}, y^{!^n B}\rangle^n & \approx_{ax} N & : !^n(A \otimes B). \\
(\eta_*)\ \Delta \rhd let * = N\ in\ *^n & \approx_{ax} N & : !^n \top.
\end{array}
$$

$$
\begin{array}{lll}
(let_1)\ \Delta \rhd let\ -_1 = (let\ -_2 = M\ in\ N)\ \ in\ P & \approx_{ax} let\ -_2 = M\ in\ let\ -_1 = N\ in\ P & : A \\
(let_2)\ \Delta \rhd let\ -_1 = V\ in\ let\ -_2 = W\ in\ M & \approx_{ax} let\ -_2 = W\ in\ let\ -_1 = V\ in\ M & : A \\
(let^{app})\ \Delta \rhd let\ x^{A\multimap B} = M\ in\ let\ y^A = N\ in\ xy & \approx_{ax} MN & : B \\
(let^{\lambda})\ \Delta \rhd let\ x^D = V\ in\ \lambda^n y^A.M & \approx_{ax} \lambda^n y^A.\ let\ \ x^D = V\ in\ M & : !^n(A \multimap B) \\
(let^{\otimes})\ \Delta \rhd let\ x^A = M\ in\ let\ y^B = N\ in\ \langle x^A, y^B\rangle^n & \approx_{ax} \langle M, N\rangle^n & : !^n(A \otimes B)
\end{array}
$$

$$
\begin{array}{ll}
(app_{\lessdot})\ \{M : !^n(A \multimap D) \lessdot B \multimap D'\}\{N : C \lessdot B\} & \\
\qquad \approx_{ax} \{\{M : !^n(A \multimap D) \lessdot A \multimap D\}\{N : C \lessdot A\} : D \lessdot D'\} & \\
(let^{\otimes}_{\lessdot})\ let \langle x^{A'}, y^{B'}\rangle^{n'} = \{M : !^n(A \otimes B) \lessdot !^{n'}(A' \otimes B')\}\ in\ N & \\
\qquad \approx_{ax} let\ \langle x^A, y^B\rangle^n = M\ in\ \{\Delta, x : !^n A, y : !^n B \lessdot \Delta, x : !^{n'} A', y : !^{n'} B' \rhd N\} & \\
(let^{x}_{\lessdot})\ let\ x^{A'} = \{M : A \lessdot A'\}\ in\ N \approx_{ax} let\ x^A = M\ in\ \{\Delta, x : A \lessdot \Delta, x : A' \rhd N\} & \\
(let^{*}_{\lessdot})\ let * = \{M : !^m \top \lessdot !^n \top\}\ in\ N \approx_{ax} let * = M\ in\ N &
\end{array}
$$

**Table 3.** Axiomatic equivalence: derived rules

$$
\begin{array}{lll}
(\alpha_{let}) & \Delta, x : A \rhd let\ y^A = x^A\ in\ M : B & \approx_{ax} \Delta, y : A \rhd M & : B \\
(let^{!\lambda}) & !\Delta \rhd let\ x^{!C} = V\ in\ \lambda y.M & \approx_{ax} \lambda^{n+1} y.\ let\ \ x^{!C} = V\ in\ M & : !^{n+1}(A \multimap B) \\
(let^{\otimes}_1) & \Delta \rhd \langle V, let\ -= M\ in\ N\rangle & \approx_{ax} let\ -= M\ in\ \langle V, N\rangle & : !^n(A \otimes B) \\
(let^{\otimes}_2) & \Delta \rhd \langle let\ -= M\ in\ N, V\rangle & \approx_{ax} let\ -= M\ in\ \langle N, V\rangle & : !^n(A \otimes B) \\
(let^{app}_1) & \Delta \rhd V(let\ -= M\ in\ N) & \approx_{ax} let\ -= M\ in\ VN & : B \\
(let^{app}_2) & \Delta \rhd (let\ -= M\ in\ N)V & \approx_{ax} let\ -= M\ in\ NV & : B
\end{array}
$$

**Definition 2.** Given two valid typing judgments $!\Delta, \Gamma_1 \rhd V : A$ and $!\Delta, \Gamma_2, x : A \rhd M : B$ where $V$ is a value, we define the *substitution* (with capture avoiding) $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ as follows: we replace each free instance $x^{A'}$ (where $A \lessdot A'$ from Lemma 2) in $M$ by $\{\Delta \rhd V : A \lessdot A'\}$.

**Lemma 5 (Substitution Lemma).** *In Definition 2, $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ is well-typed. Also, if $M$ is a value, so is $M[V/x]$.*

*Proof.* Proof by structural induction on $M$, using Lemmas 2 and 4. □

### 2.6 Axiomatic Equivalence

We define an equivalence relation on (indexed) typing judgments. We write $\Delta \rhd M \approx_{ax} M' : A$, or simply $M \approx_{ax} M'$, to indicate that $\Delta \rhd M : A$ and $\Delta \rhd M' : A$ are equivalent. Axiomatic equivalent is defined as the reflexive, symmetric, transitive, and congruence closure of the rules from Tables 2, so long as both sides of the equivalences are well-typed. The symbol "$-$" is a place holder for $x$, $*$, or $\langle x, y\rangle$.

**Lemma 6.** *The equivalences of Table 3 are derivable.* □

The following result stipulates that all the indexations of a given erasure live in the same axiomatic class. In other words, the axiomatic equivalence class of a term is independent of its indexation.

**Theorem 1.** *If* $Erase(M) = Erase(M')$ *and if* $\Delta \rhd M, M' : A$ *are valid typing judgments, then* $M \approx_{ax} M'$.

*Proof (Sketch).* The actual proof is long and technical, and is omitted here for space reasons. We proceed by first defining a special subset of terms, called *neutral* terms, for which the Theorem is obvious. We then prove that every term is axiomatically equivalent to a neutral term via a series of rewrite systems. □

## 3 Linear Category for Duplication

As it was advertised, the structure of the categorical semantics will closely follow the one proposed by Bierman [8], but with the added twist of a computational monad à la Moggi [14]. Indeed, since one has tensor product and a tensor unit, one can expect the categorical model to be symmetric monoidal. Since one can construct candidate maps for building a comonad, a comonoid structure for each $!A$ and coherence maps for the comonad, we have a linear category. Finally, the computational aspect will be taken care by Moggi's computational monad.

### 3.1 Linear Exponential Comonads

In his Ph.D. thesis, Bierman [8] gives the definition of a *linear category*. We prefer here the terminology given in [15], and use the concept of *linear exponential comonad*.

**Definition 3.** Let $(\mathcal{C}, \otimes, \top)$ be a symmetric monoidal category [13], where $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, $\lambda_A : \top \otimes A \to A$, $\rho_A : A \otimes \top \to A$ and $\sigma_{A,B} : A \otimes B \to B \otimes A$ are the usual associativity, left unit, right unit and symmetry morphisms. Let $(L, \delta, \epsilon, d^L, d^L)$ be a monoidal comonad [8], where $\epsilon_A : LA \to A$, $\delta_A : LA \to LLA$, $d^L_{A,B} : LA \otimes LB \to L(A \otimes B)$ and $d^L_\top : \top \to L\top$. We say that $L$ is a *linear exponential comonad* [15] provided that

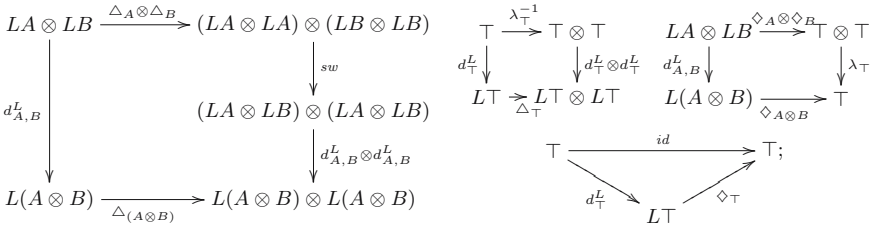
1. each object in $\mathcal{C}$ of the form $LA$ is equipped with a commutative comonoid $(LA, \triangle_A, \diamondsuit_A)$, where $\triangle_A : LA \to LA \otimes LA$ and $\diamondsuit_A : LA \to \top$;
2. $\triangle_A$ and $\diamondsuit_A$ are monoidal natural transformations;
3. $\triangle_A : (LA, \delta_A) \to (LA \otimes LA, (\delta_A \otimes \delta_A); d_A)$ and $\diamondsuit_A : (LA, \delta_A) \to (\top, d^L_\top)$ are $L$-coalgebra morphisms;
4. Every map $\delta_A$ is a comonoid morphism $(LA, \diamondsuit_A, \triangle_A) \to (L^2A, \diamondsuit_{LA}, \triangle_{LA})$.

The equations for 2–4 are to be found in Table 4.

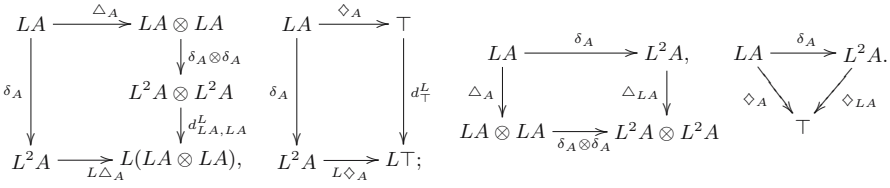
### 3.2 Strong Monad and $T$-Exponentials

To capture the computational effect of the probabilistic measurement, we use the notion of *strong monad*, as in [14]. Recall that a *monad* over a category $\mathcal{C}$

**Table 4.** Equations for a linear exponential comonad



$\triangle_A$ and $\diamondsuit_A$ are monoidal natural transformations.



$\triangle_A$ and $\diamondsuit_A$ are $L$-coalgebra maps. $\delta_A$ us a comonoid morphism.

is a triple $(T, \eta, \mu)$ where $T : \mathcal{C} \to \mathcal{C}$ is a functor, $\eta : id \dot{\to} T$ and $\mu : T^2 \dot{\to} T$ are natural transformations and such that $T\mu_A; \mu_A = \mu_{TA}; \mu_A$ and $\eta_{TA}; \mu_A = id_{TA} = T\eta_A; \mu_A$. Given a map $f : A \to TB$, we define the map $f^* : TA \to TB$ by $Tf; \mu_B$.

**Definition 4.** A *strong monad* over a monoidal category $\mathcal{C}$ is a monad $(T, \eta, \mu)$ together with a natural transformation $t_{A,B} : A \otimes TB \to T(A \otimes B)$, called the *tensorial strength*, subject to a number of coherence conditions.

*Remark 1.* If the category $\mathcal{C}$ is symmetric, the tensorial strength $t$ induces two natural transformations $TA \otimes TB \to T(A \otimes B)$, namely

$$\Psi_1 : TA \otimes TB \xrightarrow{\sigma_{TA,TB}} TB \otimes TA \xrightarrow{t_{TB,A}} T(TB \otimes A) \xrightarrow{(\sigma_{TB,A}; t_{A,B})^*} T(A \otimes B),$$
$$\Psi_2 : TA \otimes TB \xrightarrow{t_{TA,B}} T(TA \otimes B) \xrightarrow{(\sigma_{TA,B}; t_{B,A})^*} T(B \otimes A) \xrightarrow{T\sigma_{B,A}} T(A \otimes B).$$

Note that $\Psi_1$ and $\Psi_2$ might not be equal: the map $\Psi_1$ "evaluates" the first variable and then the second one. The map $\Psi_2$ does the opposite. The strength is called *commutative* if $\Psi_1 = \Psi_2$.

**Lemma 7.** *If $(T, \eta, \mu, t)$ is a strong monad on a symmetric monoidal category $\mathcal{C}$, then $(T, \eta, \mu, \Psi_1)$ and $(T, \eta, \mu, \Psi_2)$ are monoidal monad.* □

**Definition 5.** A symmetric monoidal category $(\mathcal{C}, \otimes, \top)$ together with a strong monad $(T, \eta, \mu)$ is said to have *$T$-exponentials* [14], or *Kleisli exponentials*, if it is equipped with a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, and a natural isomorphism

$$\Phi : \mathcal{C}(A, B \multimap C) \xrightarrow{\cong} \mathcal{C}(A \otimes B, TC).$$

**Lemma 8.** *The map $\Phi$ induces a natural transformation $\varepsilon_{A,B} : (A \multimap B) \otimes A \to TB$ defined by $\Phi(id_{A \multimap B})$.* □

### 3.3 Idempotent, Strong Monoidal Comonad

A comonad $(L, \epsilon, \delta)$ on some category is said to be *idempotent* if $\delta : L \dot{\to} LL$ is an isomorphism. A monoidal comonad $(L, \delta, \epsilon, d^L, d^L)$ is *strong monoidal* if $d^L_\top$ and $d^L_{A,B}$ are isomorphisms.

**Definition 6.** Given a monoidal category $(\mathcal{C}, \otimes, \top)$ with an idempotent, strong monoidal comonad $(L, \epsilon, \delta)$, a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, we define a *canonical arrow for* $\mathcal{C}$ *with respect to duplication* by induction: For all objects $A$, the arrows $id_A$, $\epsilon_A$, $\delta_A$, $d^L_\top$ and $d^L_{A,B}$ are canonical. All *expansions of canonical arrows with respect to duplication* are also canonical. An expansion of an arrow $f : A \to B$ is defined to be either $f$ or any of $Lg$,$X \otimes g$, $g \otimes X$, $X \multimap g$, $g \multimap X$, where $g$ is an expansion of $f$ and $X$ ranges over the objects of the category. Finally, compositions of canonical arrows are also canonical.

**Theorem 2 (Coherence for idempotent comonads).** *Given a category $\mathcal{C}$ with the structure in Definition 6, if $f, g : A \to B$ are two canonical arrows with respect to duplication, then they are equal.* □

### 3.4 Linear Category for Duplication

We now have enough background to define a candidate for the categorical model of the language we describe in Section 2.

**Definition 7.** A *linear category for duplication* is a category $\mathcal{C}$ with the following structure:

- a symmetric monoidal structure $(\otimes, \top, \alpha, \lambda, \rho, \sigma)$;
- an idempotent, strongly monoidal, linear exponential comonad $(L, \delta, \epsilon, d^L, d^L, \Diamond, \triangle)$;
- a strong monad $(T, \mu, \eta)$;
- a Kleisli exponential $\multimap$.

The *computational linear category* is defined to be the Kleisli category $\mathcal{C}_T$, as defined in [14].

*Remark 2.* A linear category for duplication gives rise to a double adjunction $\mathcal{C}_L \underset{F^L}{\overset{U^L}{\rightleftarrows}} \mathcal{C} \underset{F^T}{\overset{U^T}{\rightleftarrows}} \mathcal{C}_T$,. Here the left adjunction arises from the co-Kleisli category $\mathcal{C}_L$ of the comonad $L$. It is as in the linear-non-linear models of [4], and $\mathcal{C}_L$ is a category of classical (non-quantum) values. The right adjunction arises from the Kleisli category $\mathcal{C}_T$ of the computational monad $T$, as in [14]. Here $\mathcal{C}_T$ is a category of (effectful) quantum computations.

**Table 5.** Definitions of maps and operations on maps in $\mathcal{C}_\lambda$

$$
\begin{array}{lll}
\alpha_{A,B,C} = & x : A\otimes(B\otimes C) \rhd let\langle y,z\rangle = x \ in \ let\langle t,u\rangle = z \ in \ \langle\langle y,t\rangle,u\rangle & : (A\otimes B)\otimes C \\
\lambda_A = & x : \top \otimes A \rhd let\langle y,z\rangle = x \ in \ let * = y \ in \ z & : A \\
\rho_A = & x : A \otimes \top \rhd let\langle y,z\rangle = x \ in \ let * = z \ in \ y & : A \\
\sigma_{A,B} = & x : A \otimes B \rhd let\langle y,z\rangle = x \ in \ \langle z,y\rangle & : B \otimes A \\
\eta_A = & x : A \rhd \lambda *.x & : \top \multimap A \\
\mu_A = & x : \top \multimap (\top \multimap A) \rhd \lambda *.(x*)* & : \top \multimap A \\
t_{A,B} = & z : A \otimes (\top \multimap B) \rhd let\langle x,y\rangle = z \ in \ \lambda *.\langle x,y*\rangle & : \top \multimap (A \otimes B) \\
\epsilon_A = & x : !A \rhd x^A & : A \\
\delta_A = & x : !A \rhd x^{!^2 A} & : !^2 A \\
d^!_{A,B} = & z : !A \otimes !B \rhd let\langle x,y\rangle = z \ in \ \langle x,y\rangle & : !(A \otimes B) \\
d^!_\top = & z : \top \rhd let * = z \ in * & : !\top \\
\triangle_A = & x : !A \rhd \langle x,x\rangle & : !A \otimes !A \\
\diamondsuit_A = & x : !A \rhd * & : \top
\end{array}
$$

$$
\begin{array}{lll}
(x : A \rhd V : B) \otimes (y : C \rhd W : D) & = & z : A \otimes B \rhd let\langle x,y\rangle = z \ in \ \langle V,W\rangle : C \otimes D \\
(x : A \rhd V : B) \multimap (y : C \rhd W : D) & = & z : B \multimap C \rhd \lambda x.(let \ y = zV \ in \ W) : A \multimap D \\
(x : A \rhd V : \top \multimap B)^* & = & y : \top \multimap A \rhd \lambda *. \ let \ \ x = (y*) \ in \ (V*) : \top \multimap B \\
\Phi_{A,B,C}\,(x : A \rhd V : B \multimap C) & = & t : A \otimes B \rhd \lambda *. \ let \ \langle x,y\rangle = t \ in \ Vy : \top \multimap C
\end{array}
$$

### 3.5 The Category $\mathcal{C}_\lambda$

**Definition 8.** We can define a category $\mathcal{C}_\lambda$ as follows: Objects are types, and arrows $A \to B$ are axiomatic classes of valid typing judgments of the form $x : A \rhd V : B$, where $V$ is a value. We define the composition of arrows $x : A \rhd V : B$ and $y : B \rhd W : C$ to be $x : A \rhd let \ y = V \ in \ W : C$. The identity on $A$ is set to be the arrow $x : A \rhd x : A$.

**Lemma 9.** *The category $\mathcal{C}_\lambda$ is well-defined.*

*Proof.* The composition of two arrows yields an arrow axiomatically equivalent to a value due to Axiom $(\beta_\lambda)$ and Lemma 5. Composition is associative due to Axiom $(let_1)$. The arrow $x : A \rhd x : A$ is indeed the identity on $A$ due to axioms $(\alpha_{let})$ and $(\beta^2_\lambda)$. □

**Lemma 10.** *Given a valid typing judgment $\Delta \rhd V : A$ where $V$ is a value, there exists a canonical value $V'$ such that $Erase(V') = Erase(V)$ and such that $!\Delta \rhd V' : !A$. We denote this $V'$ by $\{!\Delta \lessdot \Delta \rhd V : A \gtrdot A'\}$.*

*Proof.* By induction on $V$. □

**Lemma 11.** *If $\Delta \rhd V \approx_{ax} W : A$, and if $V' = \{!\Delta \lessdot \Delta \rhd V : A \gtrdot A'\}$ and $W' = \{!\Delta \lessdot \Delta \rhd W : A \gtrdot A'\}$, then $V' \approx_{ax} W'$.*

*Proof.* By induction on $V \approx_{ax} W$. □

**Theorem 3.** *If we define $T(A) := \top \multimap A$ and $L(A) = !A$, together with the maps and the operations on maps defined in Table 5, $\mathcal{C}_\lambda$ is a linear category for duplication.*

*Proof.* The proof is mainly a long list of verifications. It uses Theorem 1, Lemmas 9, 10 and 11. □

## 4 Denotational Semantics

### 4.1 Interpretation of the Language

The lambda-calculus defined in Section 2 is thought as a *computational* lambda-calculus. Using Moggi's technique, we split the interpretation of the language into the interpretation of the *values* in a linear category for duplication $\mathcal{C}$ and the interpretation of the *computations*, i.e. general terms, in its Kleisli category $\mathcal{C}_T$. Without loss of generality, for notation purposes, we assume the category to be strictly monoidal.

We define an *interpretation of the type system* to be a map $\Theta$ that assigns to each constant type $\alpha$ an object $\Theta(\alpha)$. Each type $A$ is interpreted as an object of $\mathcal{C}$: $[\![\alpha]\!]_\Theta = \Theta(\alpha)$, $[\![\top]\!]_\Theta = \top$, $[\![!A]\!]_\Theta = L[\![A]\!]_\Theta$, $[\![A \otimes B]\!]_\Theta = [\![A]\!]_\Theta \otimes [\![B]\!]_\Theta$ and $[\![A \multimap B]\!]_\Theta = [\![A]\!]_\Theta \multimap [\![B]\!]_\Theta$.

Given a valid subtyping $A <: B$, there exists a canonical arrow $[\![A]\!]_\Theta \to [\![B]\!]_\Theta$ in $\mathcal{C}$ with respect to duplication, as defined in Definition 6. Moreover, this arrow is unique by Theorem 2. We extend the map $\Theta$ to interpret $A <: B$ as this unique arrow and we denote it by $I_{A,B}$.

We use the following straightforward shortcut definitions, where $A, A', B, B'$ are types and $\Delta$, $\Gamma$ and $\Gamma'$ are typing contexts:

- $Split_{!\Delta,\Gamma,\Gamma'} : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![\Gamma']\!] \to [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma']\!]$.
- Given $f : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \to [\![A]\!]$ and $g : [\![!\Delta]\!] \otimes [\![\Gamma']\!] \to [\![B]\!]$, we define the map $f \otimes_{!\Delta} g : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![\Gamma']\!] \to A \otimes B$.
- Given a natural transformation $n_A : FA \to GA$, if $\Delta = \{x_1 : A_1 \dots x_n : A_n\}$ we define $n_\Delta = n_{[\![A_1]\!]} \otimes \dots \otimes n_{[\![A_n]\!]}$.

**Definition 9.** The map $\Theta$ is said to be an *interpretation of the language* if moreover it assigns to each constant term $c : A_c$ an arrow $\Theta(c) : \top \to [\![A_c]\!]$ in $\mathcal{C}$.

Given a linear category for duplication $\mathcal{C}$, it is possible to interpret the typing derivation of a well-typed value as a map in $\mathcal{C}$ and the typing derivation of a valid computation as a map in the Kleisli category $\mathcal{C}_T$. We define them inductively.

- If $x_1 : A_1, \dots x_n : A_n \rhd V : B$ is a value with typing derivation $\pi$, its *value interpretation* $[\![\pi]\!]^v_\Theta$ is an arrow $[\![A_1]\!] \otimes \dots \otimes [\![A_n]\!] \to_{\mathcal{C}} [\![B]\!]$;
- if $x_1 : A_1, \dots x_n : A_n \rhd M : A$ is a term with typing derivation $\pi$, its *computational interpretation* $[\![\pi]\!]^c_\Theta$ is an arrow $[\![A_1]\!] \otimes \dots \otimes [\![A_n]\!] \to_{\mathcal{C}} T([\![B]\!])$.

Table 6 formulates the definition in the simple case where the contexts $\Delta$, $\Gamma_1$ and $\Gamma_2$ contain only one variable. One can easily extend this to the general setting.

As we already noted in Section 2.4, a valid typing judgment does not have a unique typing tree *per se*. However the following result holds:

**Table 6.** Interpretation of values and computations

Interpretation of core values:

$$\begin{aligned}[\![!\Delta, x : A \rhd x : B]\!]^v_\Theta &= [\![!\Delta]\!] \otimes [\![A]\!] \xrightarrow{\Diamond_\Delta \otimes I_{A,B}} [\![B]\!] \\ [\![!\Delta \rhd c : B]\!]^v_\Theta &= [\![!\Delta]\!] \xrightarrow{\Diamond_\Delta} \top \xrightarrow{\Theta(c)} [\![A_c]\!] \xrightarrow{I_{A_c,B}} [\![B]\!] \\ [\![!\Delta \rhd * : !^n\top]\!]^v_\Theta &= [\![!\Delta]\!] \xrightarrow{\Diamond_\Delta} \top \xrightarrow{d^L_\top} L\top \xrightarrow{I_{!\top,!^n\top}} L^n\top\end{aligned}$$

$$\frac{[\![!\Delta, x : A \rhd M : B]\!]^c_\Theta = [\![\Delta]\!] \otimes [\![A]\!] \xrightarrow{f} T([\![B]\!])}{[\![!\Delta \rhd \lambda x.M : A \multimap B]\!]^v_\Theta = [\![\Delta]\!] \xrightarrow{\Phi^{-1}(f)} [\![A]\!] \multimap [\![B]\!]}$$

$$\frac{[\![!\Delta, x : A \rhd M : B]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![A]\!] \xrightarrow{f} T([\![B]\!])}{[\![!\Delta \rhd \lambda x.M : !^{n+1}(A \multimap B)]\!]^v_\Theta = [\![!\Delta]\!] \xrightarrow{L(\Phi^{-1}f);I_{!(A\multimap B),!^{n+1}(A\multimap B)}} L^{n+1}([\![A]\!] \multimap [\![B]\!])}$$

Interpretation of extended values:

$$\frac{[\![!\Delta, \Gamma_1 \rhd V : A]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} [\![A]\!] \qquad [\![!\Delta, \Gamma_2, x : A \rhd W : B]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A]\!] \xrightarrow{g} [\![B]\!]}{[\![!\Delta, \Gamma_2, \Gamma_1 \rhd let\ x = V\ in\ W : B]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{id \otimes !_\Delta f} [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A]\!] \xrightarrow{g} [\![B]\!]}$$

$$\frac{\begin{array}{l}[\![!\Delta, \Gamma_1, \rhd V : !^n(A_1 \otimes A_2)]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} L^n([\![A_1]\!] \otimes [\![A_2]\!]) \\ [\![!\Delta, \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd W : C]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!] \xrightarrow{g} [\![C]\!]\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_2, \Gamma_1 \rhd let\langle x, y\rangle^n = V\ in\ W : C]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{id \otimes !_\Delta f} [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes L^n([\![A_1]\!] \otimes [\![A_2]\!]) \\ \qquad \xrightarrow{id \otimes \left(d^{L^n}_{[\![A_1]\!],[\![A_2]\!]}\right)^{-1}} [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!] \xrightarrow{g} [\![C]\!]\end{array}}$$

$$\frac{[\![!\Delta, \Gamma_2 \rhd V : \top]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f} \top \qquad [\![!\Delta, \Gamma_1 \rhd W : C]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{g} [\![C]\!]}{[\![!\Delta, \Gamma_1, \Gamma_2 \rhd let * = V\ in\ W : C]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{id \otimes !_\Delta f} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{g} [\![C]\!]}$$

$$\frac{[\![!\Delta, \Gamma_1 \rhd V : !^n A]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} L^n[\![A]\!] \qquad [\![!\Delta, \Gamma_2 \rhd W : !^n B]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{g} L^n[\![B]\!]}{[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \langle V, W\rangle^n : !^n(A \otimes B)]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f \otimes !_\Delta g} L^n[\![A]\!] \otimes L^n[\![B]\!] \xrightarrow{d^{L^n}_{A,B}} L^n([\![A]\!] \otimes [\![B]\!])}$$

Interpretation of computations: First, if $U$ is a core value, $[\![\Delta \rhd U : A]\!]^c_\Theta = [\![\Delta \rhd U : A]\!]^v_\Theta ; \eta_A$.

$$\frac{[\![!\Delta, \Gamma_1 \rhd M : A \multimap B]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} T([\![A]\!] \multimap [\![B]\!]) \qquad [\![!\Delta, \Gamma_2 \rhd N : A]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{g} T([\![A]\!])}{[\![!\Delta, \Gamma_1, \Gamma_2 \rhd MN : B]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f \otimes !_\Delta g} T([\![A]\!] \multimap [\![B]\!]) \otimes T([\![A]\!]) \xrightarrow{\Psi_1} T(([\![A]\!] \multimap [\![B]\!]) \otimes [\![A]\!]) \xrightarrow{\varepsilon^\star_{A,B}} T([\![B]\!])}$$

$$\frac{\begin{array}{l}[\![!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2)]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} TL^n([\![A_1]\!] \otimes [\![A_2]\!]) \\ [\![!\Delta, \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd N : C]\!]^v_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!] \xrightarrow{g} T([\![C]\!])\end{array}}{\begin{array}{l}[\![!\Delta, \Gamma_2, \Gamma_1 \rhd let\langle x, y\rangle^n = M\ in\ N : !^n C]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{id \otimes !_\Delta f} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes TL^n([\![A_1]\!] \otimes [\![A_2]\!]) \\ \qquad \xrightarrow{t;T\left(id \otimes \left(d^{L^n}\right)^{-1}\right)} T([\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes L^n[\![A_1]\!] \otimes L^n[\![A_2]\!]) \xrightarrow{g^*} T[\![C]\!]\end{array}}$$

$$\frac{[\![!\Delta, \Gamma_2 \rhd M : \top]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f} T(\top) \qquad [\![!\Delta, \Gamma_1 \rhd N : C]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{g} T([\![C]\!])}{[\![!\Delta, \Gamma_1, \Gamma_2 \rhd let * = M\ in\ N : C]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{id \otimes !_\Delta f} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes T(\top) \xrightarrow{t;g^*} T([\![C]\!])}$$

$$\frac{[\![!\Delta, \Gamma_1 \rhd M : !^n A]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \xrightarrow{f} TL^n[\![A]\!] \qquad [\![!\Delta, \Gamma_2 \rhd N : !^n B]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{g} TL^n[\![B]\!]}{[\![!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M, N\rangle^n : !^n(A \otimes B)]\!]^c_\Theta = [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \xrightarrow{f \otimes !_\Delta g} TL^n[\![A]\!] \otimes TL^n[\![B]\!] \xrightarrow{\Psi_1; Td^{L^n}_{A,B}} TL^n([\![A]\!] \otimes [\![B]\!])}$$

**Theorem 4.** *Given a valid typing judgment with two typing derivations $\pi$ and $\pi'$, for any interpretation $\Theta$ we have $[\![\pi]\!]^c_\Theta = [\![\pi']\!]^c_\Theta$ (and $[\![\pi]\!]^v_\Theta = [\![\pi']\!]^v_\Theta$ if the typing judgment is a value).*

*Proof.* The proof is done by showing that given any typing judgment $\Delta \rhd M : A$ with denotation $f$ one can factor $f$ as $\Diamond_{!\Gamma} \otimes \bar{f}$, where $\bar{f}$ is the denotation of $\Delta' \rhd M : A$, where $\Delta', !\Gamma = \Delta$ and $|\Gamma|$ is the set of dummy variables. □

**Definition 10.** Given a interpretation $\Theta$ of the language in a category $\mathcal{C}$, we define the denotation of a valid typing judgment $\Delta \rhd M : A$ with typing derivation $\pi$ to be $[\![\Delta \rhd M : A]\!]^c_\Theta = [\![\pi]\!]^c_\Theta$ and $[\![\Delta \rhd M : A]\!]^v_\Theta = [\![\pi]\!]^v_\Theta$ if $M$ is a value.

**Lemma 12.** *Suppose that $\Delta \rhd V : A$ is a valid typing judgment where $V$ is a value. Then $[\![\Delta \rhd V : A]\!]^c = [\![\Delta]\!] \xrightarrow{[\![\Delta \rhd V:A]\!]^v} [\![A]\!] \xrightarrow{\eta_{[\![A]\!]}} T([\![A]\!])$.*

*Proof.* Proof by induction on $V$, using Lemma 7, the bifunctoriality of $\otimes_{LA}$ and the equations for strong monadicity in Definition 4. □

### 4.2 Soundness of the Denotation

The axiomatic equivalence and the categorical semantics are two faces of the same coin. Indeed, as we will prove in this section, two terms in the same axiomatic equivalence class have the same denotation. A corollary is that the indexation of terms does not influence the denotation. This proves semantically the fact that it is safe to work with untyped terms. An alternate justification of this fact is of course the operational semantics, which was given in [19].

**Lemma 13.** *Suppose* $M' = \{\Delta' <: \Delta \rhd M : A <: A'\}$. *Then* $[\![\Delta' \rhd M' : A']\!]^c = I_{\Delta',\Delta}; [\![\Delta \rhd M : A]\!]^c; T(I_{A,A'})$. *If* $M = V$ *is a value, from Lemma 4,* $M' = V'$ *is a value. Then* $[\![\Delta' \rhd V' : A']\!]^v = I_{\Delta',\Delta}; [\![\Delta \rhd V : A]\!]^v; I_{A,A'}$.

*Proof.* Proof by structural induction on $M$. □

**Lemma 14 (Substitution).** *Given two valid typing judgments* $!\Delta, \Gamma_1, x : A \rhd M : B$ *and* $!\Delta, \Gamma_2 \rhd V : A$, *the typing judgment* $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ *is valid. Let* $h$ *be* $[\![!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B]\!]^c$ *and* $h'$ *be* $[\![!\Delta, \Gamma_1, \Gamma_2 \rhd W[V/x] : B]\!]^v$, *in the case where* $M = W$ *is a value. Then they are defined by*

$$\begin{array}{ccc} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] & \overset{h}{\dashrightarrow} & T([\![B]\!]) \\ \downarrow Split_{!\Delta,\Gamma_1,\Gamma_2} & & \uparrow [\![!\Delta,\Gamma_1,x:A \rhd M:B]\!]^c \\ [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] & \xrightarrow{id \otimes [\![!\Delta,\Gamma_2 \rhd V:A]\!]^v} & [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![A]\!], \end{array} \qquad \begin{array}{ccc} [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] & \overset{h'}{\dashrightarrow} & [\![B]\!] \\ \downarrow Split_{!\Delta,\Gamma_1,\Gamma_2} & & \uparrow [\![!\Delta,\Gamma_1,x:A \rhd W:B]\!]^v \\ [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma_2]\!] & \xrightarrow{id \otimes [\![!\Delta,\Gamma_2 \rhd V:A]\!]^v} & [\![!\Delta]\!] \otimes [\![\Gamma_1]\!] \otimes [\![A]\!]. \end{array}$$

*Proof.* Proof by induction on $M$, using Lemma 11, Lemma 12 and the naturality of $\Phi$. □

**Theorem 5.** *If* $\Delta \rhd M \approx_{ax} M' : A$ *then* $[\![\Delta \rhd M : A]\!]^c_\Theta = [\![\Delta \rhd M' : A]\!]^c_\Theta$ *(and* $[\![\Delta \rhd M : A]\!]^v_\Theta = [\![\Delta \rhd M' : A]\!]^v_\Theta$ *if* $M$ *is a value) for every interpretation* $\Theta$.

*Proof.* Proof by induction on $M \approx_{ax} M'$, using Lemmas 13 and 14. □

**Corollary 1.** *If* $Erase(M) = Erase(M')$ *and if* $\Delta \rhd M, M' : A$ *are valid typing judgments, then* $[\![M]\!]^c = [\![M']\!]^c$ *(and* $[\![M]\!]^v = [\![M']\!]^v$ *if* $M$ *is a value).*

*Proof.* Corollary of Theorems 1 and 5. □

### 4.3 Completeness

The category $\mathcal{C}_\lambda$ being a linear category for duplication, one can interpret the language in it. This section states that the defined lambda-calculus is an *internal language* of linear categories for duplication.

Since the category $\mathcal{C}_\lambda$ is a monoidal category, one can w.l.o.g. generalize the notion of pairing to finite tensor products of terms. Then the following results are true:

**Lemma 15.** *In $\mathcal{C}_\lambda$, a valid typing judgment $x_1 : A_1, \ldots x_n : A_n \rhd M : B$ has for computational denotation $(t : A_1 \otimes \cdots \otimes A_n \rhd let\langle x_1, \ldots x_n\rangle = t \text{ in } \lambda*.M : \top \multimap B)$. If $M = V$ is a value, the value interpretation is $(t : A_1 \otimes \cdots \otimes A_n \rhd let\langle x_1, \ldots x_n\rangle = t \text{ in } V : B)$.*

*Proof.* Proof by structural induction on $M$ and $V$. □

**Theorem 6.** *In $\mathcal{C}_\lambda$, $\Theta$ being the identity, one has $[\![x : A \rhd M : B]\!]^c_\Theta \approx_{ax} (x : A \rhd \lambda*.M : \top \multimap B)$ and $[\![x : A \rhd V : B]\!]^v_\Theta \approx_{ax} (x : A \rhd V : B)$.*

*Proof.* Corollary of Lemma 15. □

## 5 Towards a Denotational Model of Quantum Lambda Calculus

As noted in the introduction, this paper is mostly concerned with the categorical requirements for modeling a generic call-by-value linear lambda calculus, i.e., its type system (which includes subtyping) and equational laws. We have not yet specialized the language to a particular set of built-in operators, for example, those that are required for quantum computation.

However, since the quantum lambda calculus [19] is the main motivation behind our work, we will comment very briefly on what additional properties would be required to interpret its primitives. The quantum lambda calculus is obtained by instantiating and extending the call-by-value language of this paper with the following primitive types, constants, and operations:

Types: $bit$, $qbit$

Constants: $0 : !bit$, $1 : !bit$

$new : !(bit \multimap qbit)$, $U : !(qbit^n \multimap qbit^n)$, $meas : !(qbit \multimap !bit)$

Operations: $$\frac{\Gamma_1, !\Delta \rhd P : bit \quad \Gamma_2, !\Delta \rhd M : A \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd \mathit{if}\ P\ \mathit{then}\ M\ \mathit{else}\ N : A}\ (\mathit{if})$$

Here, $U$ ranges over a set of built-in unitary operations. In the intended semantics, $!bit \cong bit$, while $!qbit$ is empty. *new* creates a new qubit, and *meas* measures a qubit.

The denotational semantics of these operations is already well-understood in the absence of higher-order types. They can all be interpreted in the category **Q** of superoperators from [18]. The part that is not yet well-understood is how these features interact with higher-order types.

In light of our present work, we can conclude that a model of the quantum lambda calculus consists of a linear category for duplication $(\mathcal{C}, L, T, \multimap)$, such that the associated category of computations $\mathcal{C}_T$ contains the category **Q** of [18] as a full monoidal subcategory. To construct an actual instance of such a model is still an open problem.

## 6 Conclusion

We have developed a call-by-value, computational lambda-calculus for manipulating duplicable and non-duplicable data, together with an axiomatic equivalence relation on typed terms. We use a subtyping relation in order to have implicit promotion, dereliction, copying and discarding. Then we developed categorical model for the language, inspired by the work of [8] and [14]. We finally showed that the model is sound and complete with respect to the axiomatic equivalence.

## References


1. Abramsky, S.: Computational interpretations of linear logic. Theoretical Computer Science 111, 3–57 (1993)
2. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Proceedings of LICS 2004, pp. 415–425 (2004)
3. Barendregt, H.P.: The Lambda-Calculus, its Syntax and Semantics. North Holland, Amsterdam (1984)
4. Benton, N.: A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 121–135. Springer, Heidelberg (1995)
5. Benton, N., Bierman, G., de Paiva, V.C.V., Hyland, M.: A term calculus for intuitionistic linear logic. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 75–90. Springer, Heidelberg (1993)
6. Benton, N., Bierman, G., Hyland, M., de Paiva, V.C.V.: Linear lambda-calculus and categorical models revisited. In: Martini, S., Börger, E., Kleine Büning, H., Jäger, G., Richter, M.M. (eds.) CSL 1992. LNCS, vol. 702, Springer, Heidelberg (1993)
7. Benton, N., Wadler, P.: Linear logic, monads and the lambda calculus. In: Proceedings of LICS 1996, pp. 420–431 (1996)
8. Bierman, G.: On Intuitionistic Linear Logic. PhD thesis, Computer Science Department, Cambridge University, Cambridge (1993)
9. Coecke, B.: Quantum information-flow, concretely, abstractly. In: Selinger, P., (ed.) Proceedings of QPL 2004. TUCS General Publication No. 33, Turku Centre for Computer Science pp. 57–73 (2004)
10. Coecke, B., Pavlovic, D.: Quantum measurements without sums. In: Chen, G., Kauffman, L., Lomonaco, S.J. (eds.) Mathematics of Quantum Computation and Technology, pp. 559–596. Chapman & Hall, Boca Raton (2007)
11. Gay, S.J., Nagarajan, R.: Communicating quantum processes. In: Proceedings of POPL 2005, ACM Press, New York (2005)
12. Lalire, M., Jorrand, P.: A process algebraic approach to concurrent and distributed computation: Operational semantics. In: Selinger, P. (ed.) Proceedings of QPL 2004. TUCS General Publication No. 33, Turku Centre for Computer Science, pp. 109–126 (2004)
13. Mac Lane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1998)
14. Moggi, E.: Notions of computation and monads. Information and Computation 93, 55–92 (1991)

15. Schalk, A.: What is a model for linear logic. Manuscript (2004)
16. Seely, R.A.G.: *-autonomous categories and cofree coalgebras. Contemporary Mathematics 92 (1989)
17. Selinger, P. (ed.): Proceedings of QPL 2004. TUCS General Publication No. 33, Turku Centre for Computer Science (2004)
18. Selinger, P.: Towards a quantum programming language. Mathematical Structures in Computer Science 14, 527–586 (2004)
19. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Mathematical Structures in Computer Science 16, 527–552 (2006)
20. Selinger, P., Valiron, B.: On a fully abstract model for a quantum linear functional language. In: Preliminary proceedings of QPL 2006, pp. 103–115 (2006)
21. van Tonder, A.: A lambda calculus for quantum computation. SIAM Journal of Computing 33, 1109–1135 (2004)
22. Wadler, P.: There's no substitute for linear logic. Manuscript, presented at MFPS 1992 (1992)
23. Wootters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature 299, 802–803 (1982)

# The ω-Regular Post Embedding Problem*

P. Chambart and Ph. Schnoebelen

LSV, ENS Cachan, CNRS
61, av. Pdt. Wilson, F-94230 Cachan, France
{chambart,phs}@lsv.ens-cachan.fr

**Abstract.** Post's Embedding Problem is a new variant of Post's Correspondence Problem where words are compared with embedding rather than equality. It has been shown recently that adding regular constraints on the form of admissible solutions makes the problem highly non-trivial, and relevant to the study of lossy channel systems. Here we consider the infinitary version and its application to recurrent reachability in lossy channel systems.

## 1 Introduction

*Post's correspondence problem,* or shortly PCP, can be stated as the question whether two morphisms $u, v : \Sigma^* \to \Gamma^*$ agree non-trivially on some input, i.e., whether $u(\sigma) = v(\sigma)$ for some non-empty $\sigma \in \Sigma^+$. This undecidable problem plays a central role in computer science because it is very often easier and more natural to prove undecidability by reduction from PCP than from, say, the halting problem for Turing machines.

In a recent paper, we introduced PEP, the *Post Embedding Problem*, a variant of PCP where one asks whether $u(\sigma)$ is a (scattered) *subword* of $v(\sigma)$ for some $\sigma$ [CS07]. The subword relation, also called embedding, is denoted "$\sqsubseteq$": $w \sqsubseteq w' \stackrel{\text{def}}{\Leftrightarrow} w$ can be obtained from $w'$ by erasing some letters, possibly all of them, possibly none. We also introduced $\mathsf{PEP}^{\text{reg}}$, an extension of PEP where one adds the requirement that a solution $\sigma$ belongs to a regular language $R \subseteq \Sigma^*$.

PEP is a trivial, hence not very interesting, problem. However, and quite surprisingly, $\mathsf{PEP}^{\text{reg}}$ behaves very differently. $\mathsf{PEP}^{\text{reg}}$ is decidable but it is not primitive recursive. In fact it is (non-trivially) equivalent to the reachability problem for lossy channel systems. Thus $\mathsf{PEP}^{\text{reg}}$ is a new representative of the strange computational niche that hosts lossy channel systems and other problems in timed automata and logics [LW05, ADOW05, OW06, OW07], concurrency models [AM02, Del07, LNO+07], temporal and modal logic [DL06, GKWZ06, KWZ05, Kur06], and other areas [JL07]. We could also use $\mathsf{PEP}^{\text{reg}}$ to solve open problems on unidirectional channel systems combining one reliable and one lossy channel. These unidirectional systems, introduced in [CS07], are currently under our active scrutiny because of their fundamental role in the classification of channel systems that mix reliable and unreliable channels along arbitrary network topologies [Cha07].

* Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 97–111, 2008.

*The ω-regular Post Embedding Problem.* In this paper we consider infinitary extensions of $\mathsf{PEP}^{\text{reg}}$[1], most prominently $\mathsf{PEP}^{\omega\text{-reg}}$, where one asks for an *infinite* $\sigma \in \Sigma^\omega$ such that $u(\sigma) \sqsubseteq v(\sigma)$, and where an ω-regular constraint can further be imposed upon σ. Our motivation is twofold. Firstly, we aim at deepening our understanding of PEP and $\mathsf{PEP}^{\text{reg}}$, two exciting new problems. Secondly, and based on the existing results for the finitary case, we expect that connections can be established between $\mathsf{PEP}^{\omega\text{-reg}}$ and recurrent reachability questions on channel systems.

*Our contribution.* In this paper, we show the equivalence between $\mathsf{PEP}^{\omega\text{-reg}}$ and recurrent reachability questions for unidirectional channel systems. This equivalence is shown using the *2-dimensional correspondence+embedding problem*, or 2PCEP, a new intermediary problem that leads to a clearer, more abstract and more modular approach. The approach handles both the finitary and the infinitary cases in a single way.

We also show that $\mathsf{PEP}^{\omega\text{-reg}}$ can be reduced to $\mathsf{PEP}^{\text{reg}}$, so that the two problems are equivalent. Hence $\mathsf{PEP}^{\omega\text{-reg}}$ is decidable. This has the surprising consequence that recurrent reachability for unidirectional channel systems is decidable. It further shows that the links we established between unidirectional channel systems and lossy channel systems (in [CS07]) do not carry over from reachability to recurrent reachability.

Finally, we show that recurrent reachability for lossy channel systems can be reduced to $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$, the variant of $\mathsf{PEP}^{\omega\text{-reg}}$ where we look for *direct* solutions (informally, solutions where $v(\sigma)$ must be ahead of $u(\sigma)$ at all times when σ grows from ε to its final value). Hence $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ is undecidable (while $\mathsf{PEP}^{\omega\text{-reg}}_{\text{codir}}$ is decidable). Again, this contrasts with the finitary case, where $\mathsf{PEP}^{\text{reg}}$, $\mathsf{PEP}^{\text{reg}}_{\text{dir}}$ and $\mathsf{PEP}^{\text{reg}}_{\text{codir}}$ are equivalent.

*Outline of the paper.* Section 2 recalls the necessary definitions and notations on embeddings between finite or infinite words. Section 3 states the ω-regular Post embedding problem, solves it in the unconstrained case, and shows that restricting to short morphisms is no loss of generality. Section 4 shows the equivalence between $\mathsf{PEP}^{\text{reg}}$ and $\mathsf{PEP}^{\omega\text{-reg}}$, before Section 5 links $\mathsf{PEP}^{\omega\text{-reg}}$ and $\mathsf{PEP}^{\text{reg}}$ with reachability and recurrent reachability questions for unidirectional channel systems. Finally, section 6 solves the remaining case, $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$, by linking it to recurrent reachability for lossy channel systems.

## 2 Notations and Definitions

*Words.* We write $u, v, w, t, \sigma, \rho, \alpha, \beta, \ldots$ for words, i.e., finite or infinite (i.e., ω-length) sequences of letters such as $a, b, i, j, \ldots$ from alphabets $\Sigma, \Gamma, \ldots$. The *length* of $u \in \Sigma^* \cup \Sigma^\omega$ is written $|u|$, the set $alph(u)$ is the set of letters (a subset of Σ) that occur in $u$. We denote with $u.v$, or $uv$, the concatenation of $u$ and $v$, with $uv = u$ when $u$ has ω-length.

A *morphism* from $\Sigma^*$ to $\Gamma^*$ is a map $h : \Sigma^* \to \Gamma^*$ that respects the monoidal structure, i.e., with $h(\varepsilon) = \varepsilon$ and $h(\sigma.\rho) = h(\sigma).h(\rho)$. Its extension over $\Sigma^\omega$ is defined in the obvious way: note that, in general, it takes values in $\Gamma^* \cup \Gamma^\omega$ since $h(u) = \varepsilon$ for $u \neq \varepsilon$ is allowed. A morphism $h$ is completely defined by its image $h(1), h(2), \ldots,$

[1] Recall that the classic PCP problem is undecidable but r.e., while the infinitary extension, denoted $\mathsf{PCP}^\omega$, is $\Sigma^1_1$-complete.

on $\Sigma = \{1,2,\ldots\}$. We often simply write $h_1, h_2, \ldots$, and $h_\sigma$, instead of $h(1), h(2), \ldots$, and $h(\sigma)$.

*Embeddings.* Given two words $u = a_1 \ldots a_n$ and $v = b_1 \ldots b_m$, we write $u \sqsubseteq v$ when $u$ is a *subword* of $v$, i.e., when $u$ can be obtained by erasing some letters (possibly none) from $v$. For example, *abba* $\sqsubseteq$ *abracadabra*. Equivalently, $u \sqsubseteq v$ when $u$ can be embedded in $v$, i.e., when there exists an order-preserving injective map (called an "*embedding*") $h : \{1,\ldots,n\} \to \{1,\ldots,m\}$ such that $a_i = b_{h(i)}$ for all $i = 1,\ldots,n$. Embeddings between ω-words are defined similarly, with a strictly increasing $h : \mathbb{N} \smallsetminus 0 \to \mathbb{N} \smallsetminus 0$. We explicitly allow the embedding of finite words into infinite ones.

It is well-known that the subword relation is a partial ordering on finite words. Observe that, between ω-words, embedding is only a (partial) *quasi*-ordering: $u \sqsubseteq v$ and $v \sqsubseteq u$ together do not imply $u = v$. For example, $(ab)^\omega \sqsubseteq (bba)^\omega \sqsubseteq (ab)^\omega$. We write $u \equiv v$ when $u \sqsubseteq v$ and $v \sqsubseteq u$.

*Halving* ω*-words.* For some $u \in \Sigma^\omega$, let $inf(u) \subseteq \Sigma$ denote the set of letters that occur infinitely many times in $u$. The word $u$ can be decomposed under the form $u'.u''$ where $u'$ is a finite prefix and the corresponding suffix $u'' \in \Sigma^\omega$, only contains letters from $inf(u)$. Such a decomposition is called a *halving* of $u$. There exists several (in fact, infinitely many) halvings of any $u \in \Sigma^\omega$: the *canonical halving* is obtained by selecting the shortest possible prefix $u'$.

The following lemma is a classic tool when considering embeddings between ω-words (see, e.g., [Fin85]).

**Lemma 2.1.** *Let $u, v \in \Sigma^\omega$ be two ω-words with $u'.u''$ and $v'.v''$ two arbitrary halvings of $u$ and $v$. Then*

$$u \sqsubseteq v \text{ iff } \begin{cases} alph(u'') \subseteq alph(v''), \text{ and} \\ \text{there exists } x \in alph(v'')^* \text{ such that } u' \sqsubseteq v'x. \end{cases}$$

Furthermore, when $u \sqsubseteq v$, then $x$ can be chosen with $|x| \leq |u'|$, and for any halving $u = u'.u''$ there exists a halving $v = v'.v''$ such that $u' \sqsubseteq v'$.

**Corollary 2.2.** *Let $u_1, u_2$ be two ω-words such that $inf(u_1) = alph(u_1) = alph(u_2) = inf(u_2)$. Then $u.u_1 \equiv u.u_2$ for all $u \in \Sigma^*$.*

## 3 Post Embedding Problems

Post embedding problems are variants of Post correspondence problems where correspondence (equality between words) is replaced by embedding, and where an additional regular constraint may be imposed over the solution.

Formally, given two morphisms $u, v : \Sigma^* \to \Gamma^*$ we say that $\sigma \in \Sigma^*$ is a *(finite) solution* to Post's embedding problem if $u_\sigma \sqsubseteq v_\sigma$. If $\sigma \in \Sigma^\omega$ and $u_\sigma \sqsubseteq v_\sigma$, then $\sigma$ is an *infinite solution* (also called, an ω-solution).

We say that $\sigma$ is a *direct* solution if $u_\rho \sqsubseteq v_\rho$ for every prefix $\rho$ of $\sigma$. It is a *codirect solution* if $u_\rho \sqsubseteq v_\rho$ for every suffix $\rho$ of $\sigma$. When considering finite solutions [CS07],

there is a symmetry between the notions of direct and codirect solutions, since a direct solution for some $u, v$ is a codirect solution for the mirror instance $\tilde{u}, \tilde{v}$. This symmetry does not carry over to infinite solutions because the mirror of an ω-word is not an ω-word. Also, observe that the prefixes of a direct ω-solution are finite (direct) solutions, and that the suffixes of a codirect ω-solution are other infinite (codirect) solutions.

The Post embedding problems we considered in [CS07] are $\mathsf{PEP}^{\mathrm{reg}}$, $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ that ask, given two morphisms $u, v$ and a regular $R \subseteq \Sigma^*$, whether $R$ contains a solution (respectively, a direct solution, a codirect solution). The infinitary extensions of these problems are $\mathsf{PEP}^{\omega\text{-reg}}$, $\mathsf{PEP}^{\omega\text{-reg}}_{\mathrm{dir}}$ and $\mathsf{PEP}^{\omega\text{-reg}}_{\mathrm{codir}}$, that ask, given $u, v$ and an ω-regular $R \subseteq \Sigma^\omega$, whether there exists an ω-solution $\sigma \in R$ (resp., a direct ω-solution, a codirect ω-solution).

In the above definition, the regular constraint applies to $\sigma$ but this is inessential and our results still hold when the constraint applies to $u_\sigma$, or $v_\sigma$, or both (see [CS07]).

For complexity issues, we assume that the constraint $R$ is given as a nondeterministic automaton $\mathcal{A}_R$, that can be a FSA or a Büchi automaton depending on whether $R$ is finitary or not. By a *reduction* between two decision problems, we mean a logspace many-one reduction, except when specified otherwise (as in Section 4). We say two problems are *equivalent* when they are inter-reducible.

### 3.1 General Embedding for Direct Solutions

We now state a technical lemma that shows that the above definition of a direct solution, "$u_\rho \sqsubseteq v_\rho$ for all prefixes $\rho$ of $\sigma$", can be replaced by a stronger requirement: that there exists an embedding of $u_\sigma$ into $v_\sigma$ that embeds any $u_\rho$ into the corresponding $v_\rho$.

Let a $\mathsf{PEP}^\omega$ instance be given by two morphisms $u, v$, and consider an infinite $\sigma \in \Sigma^\omega$, of the form $\sigma = i_1.i_2.i_3\ldots$

For $k = 0, 1, 2, \ldots$, we let $l_k$ and $l'_k$ denote respectively, the lengths $|u_{i_1 i_2 \ldots i_k}|$ and $|v_{i_1 i_2 \ldots i_k}|$.

**Lemma 3.1.** *The following are equivalent:*

(a). $\sigma$ *is a* direct *solution,*
(b). *For all* $k \in \mathbb{N}$*, there exists an embedding* $h_k : \{1, 2, \ldots, l_k\} \to \{1, 2, \ldots, l'_k\}$ *that witnesses* $u_{i_1 i_2 \ldots i_k} \sqsubseteq v_{i_1 i_2 \ldots i_k}$,
(c). *There exists a general embedding* $h : \mathbb{N} \to \mathbb{N}$ *that witnesses* $u_\sigma \sqsubseteq v_\sigma$ *and such that its restriction to* $\{1, 2, \ldots, l_k\}$ *witnesses* $u_{i_1 i_2 \ldots i_k} \sqsubseteq v_{i_1 i_2 \ldots i_k}$.

*Proof (Sketch).* (a) and (b) are equivalent by definition of being a direct solution. (c) obviously implies (b). We prove (c) from (b) by defining $h(i) \stackrel{\text{def}}{=} \min_{k=1,2,\ldots} h_k(i)$. □

### 3.2 The Unrestricted Problems

$\mathsf{PEP}$ and $\mathsf{PEP}^\omega$ are the special case of $\mathsf{PEP}^{\mathrm{reg}}$ and $\mathsf{PEP}^{\omega\text{-reg}}$ where $R = \Sigma^+$ (respectively, $R = \Sigma^\omega$), i.e., where there are no regularity constraints over the form of a solution. The remark that $\mathsf{PEP}$ is trivial extends to $\mathsf{PEP}^\omega$, $\mathsf{PEP}^\omega_{\mathrm{dir}}$ and $\mathsf{PEP}^\omega_{\mathrm{codir}}$:

**Proposition 3.2.** *Given two morphisms* $u, v : \Sigma^* \to \Gamma^*$ *defining a Post embedding problem:*

*1. There is a solution in* $\Sigma^+$ *if and only if there is a direct* ω*-solution in* $\Sigma^\omega$ *if and only if there is some* $i \in \Sigma$ *such that* $u_i \sqsubseteq v_i$.
*2. There is an* ω*-solution in* $\Sigma^\omega$ *if and only if there is there is a codirect* ω*-solution if and only if there exists a non-empty subset* $\Sigma'$ *of* $\Sigma$ *s.t.* $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$.

*Proof.* 1. Obviously, if $u_i \sqsubseteq v_i$ then $i \in \Sigma$ is a solution in $\Sigma^+$, and $i^\omega$ is a direct ω-solution. Conversely, if there is a direct solution $\sigma = i_1 i_2 i_3 \dots$ in $\Sigma^\omega$, then $u_{i_1} \sqsubseteq v_{i_1}$ by definition of directness. If there is a finite solution $\sigma = i_1 i_2 i_3 \dots i_m$ in $\Sigma^+$, then either $u_{i_1} \sqsubseteq v_{i_1}$ and we are done, or $i_2 i_3 \dots i_m$ is a shorter finite solution, and we'll eventually encounter some $u_i \sqsubseteq v_i$.
2. Obviously, if $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$ for some non-empty $\Sigma' = \{i_1, \dots, i_m\}$, then $(i_1 \dots i_m)^\omega$ is an ω-solution, and even a codirect one. Conversely, given an ω-solution σ, Lemma 2.1 entails that, letting $\Sigma' \stackrel{\text{def}}{=} inf(\sigma)$, one has $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$. □

The corollary is:

**Theorem 3.3.** $\mathsf{PEP}^\omega$ *and* $\mathsf{PEP}^\omega_{codir}$ *coincide, and are* PTime*-complete.* $\mathsf{PEP}^\omega_{dir}$ *coincides with the finitary problems* PEP, $\mathsf{PEP}_{dir}$ *and* $\mathsf{PEP}_{codir}$*, and these problems are in* LogSpace.

*Proof (Sketch).* There exists a simple polynomial-time decision procedure for $\mathsf{PEP}^\omega$. It computes the largest $\Sigma'$ satisfying $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$ and then checks that this $\Sigma'$ is not empty. This largest $\Sigma'$ is obtained by starting with $\Sigma' := \Sigma$ and then removing from $\Sigma'$ every $i$ for which $alph(u_i)$ is not included in the current $\Sigma'$, until eventual stabilization (PTime-hardness is proved in the full version of this paper). Regarding $\mathsf{PEP}^\omega_{\text{dir}}$, one only needs deterministic logarithmic space to find whether $u_i \sqsubseteq v_i$ for some $i$. □

### 3.3 Short Morphisms

$\mathsf{PEP}^{\text{reg}}_{\leq 1}$ (respectively $\mathsf{PEP}^{\omega\text{-reg}}_{\leq 1}$) is $\mathsf{PEP}^{\text{reg}}$ (respectively $\mathsf{PEP}^{\omega\text{-reg}}$) with the constraint that all images $u_i$'s and $v_i$'s have length $\leq 1$, i.e., the morphisms can be seen as maps $u, v : \Sigma \to \Gamma \cup \{\varepsilon\}$.

**Proposition 3.4**
*1.* $\mathsf{PEP}^{reg}$ *and* $\mathsf{PEP}^{reg}_{\leq 1}$ *are equivalent (inter-reducible).*
*2.* $\mathsf{PEP}^{\omega\text{-}reg}$ *and* $\mathsf{PEP}^{\omega\text{-}reg}_{\leq 1}$ *are equivalent (inter-reducible).*

*Proof.* It is enough to show that PEP reduces to $\mathsf{PEP}_{\leq 1}$. For this, let $u, v : \Sigma^* \to \Gamma^*$ be a PEP instance. Let $k > 0$ be large enough so that, for all $i \in \Sigma$, $u_i$ and $v_i$ have at most $k$ letters. Then we can write each $u_i$ under the form $u_i^1 \dots u_i^k$ with $u_i^j \in \Gamma \cup \{\varepsilon\}$, i.e., $|u_i^j| \leq 1$. Similarly, we write every $v_i$ as some $v_i^1 \dots v_i^k$ with $|v_i^j| \leq 1$. We now define $\Sigma' \stackrel{\text{def}}{=} \Sigma \times \{1, \dots, k\}$ and two morphisms $u', v' : \Sigma'^* \to \Gamma^*$ with $u'(i,j) \stackrel{\text{def}}{=} u_i^j$ and $v'(i,j) \stackrel{\text{def}}{=} v_i^j$. Observe that $u', v'$ defines a $\mathsf{PEP}_{\leq 1}$ instance. Now, with $R \subseteq \Sigma^*$ (or $R \subseteq \Sigma^\omega$) one associates a constraint $R' \subseteq \Sigma'^*$ (resp., $R' \subseteq \Sigma'^\omega$) by $R' \stackrel{\text{def}}{=} h(R)$ with $h : \Sigma^* \to \Sigma'^*$ given by $h(i) = (i,1)(i,2)\dots(i,k)$. $R'$ is regular (resp., ω-regular) since $R$ is, and $u', v'$ admits a solution in $R'$ iff $u, v$ has one in $R$. □

## 4 Reducing PEP$^{\omega\text{-reg}}$ to PEP$^{\text{reg}}$

**Theorem 4.1 (Main result).** PEP$^{\omega\text{-}reg}$ *and* PEP$^{reg}$ *are equivalent (modulo elementary reductions).*

**Corollary 4.2.** PEP$^{\omega\text{-}reg}$ *is decidable (but not primitive-recursive).*

One direction of Theorem 4.1 is obvious: any PEP$^{\text{reg}}$ instance $u, v, R$ can be seen as a PEP$^{\omega\text{-reg}}$ instance by adding an extra symbol $\bot$ to $\Sigma$ and $\Gamma$, replacing $R$ with $R.\bot^\omega$, and letting $u(\bot) = v(\bot) = \bot$.

For the other direction, we consider a PEP$^{\omega\text{-reg}}$ instance given by two morphisms $u, v : \Sigma^* \to \Gamma^*$ and an $\omega$-regular $R \subseteq \Sigma^\omega$.

**Lemma 4.3.** *There exists $\sigma \in R$ such that $u_\sigma \sqsubseteq v_\sigma$ if and only if there exists two finite words $\rho_1$ and $\rho_2$ in $\Sigma^*$ such that*

(a) $\rho_1.\rho_2^\omega \in R$,
(b) $u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2}$, *and*
(c) $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$.

*Proof.* The "$\Leftarrow$" direction is easy since taking $\sigma = \rho_1.\rho_2^\omega$ is sufficient. For the "$\Rightarrow$" direction, we assume that $\sigma = a_1 a_2 a_3 \ldots \in R$ satisfies $u_\sigma \sqsubseteq v_\sigma$ and show how to build $\rho_1$ and $\rho_2$.

Let $\mathcal{A}_R = (Q, \Sigma, q_0, F, \delta)$ be a Büchi automaton for $R$, and $\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots$ be an accepting run of $\mathcal{A}_R$ over $\sigma$. This run is an $\omega$-sequence of transitions "$q_{i-1} \xrightarrow{a_i} q_i$", so that $\pi \in \delta^\omega$ can be halved under the form $\pi = \pi'.\pi''$. This gives rise to two halvings $u'.u''$ and $v'.v''$ of, respectively, $u_\sigma$ and $v_\sigma$.

Let us pick a finite prefix $\theta$ of $\pi''$ that uses every transition from $inf(\pi)$ at least once, and that ends on the starting state of $\pi''$. Hence $\theta$ is some $q_n \xrightarrow{a_{n+1}} q_{n+1} \xrightarrow{a_{n+2}} \cdots \xrightarrow{a_{n+k}} q_{n+k}$ with $n = |\pi'|$, $q_n = q_{n+k}$, and $inf(\sigma) = \{a_{n+1}, a_{n+2}, \ldots, a_{n+k}\}$. Let now $\rho_1 \stackrel{\text{def}}{=} a_1 a_2 \ldots a_n$ and $\rho \stackrel{\text{def}}{=} a_{n+1} a_{n+2} \ldots a_{n+k}$. Clearly $\rho_1.\rho^\omega \in R$ as witnessed by the ultimately periodic run $\pi'.\theta^\omega$. Furthermore, from $u' = u_{\rho_1}$ and $inf(u'') = alph(u'') = alph(u_\rho)$, we deduce $u_\sigma = u'.u'' \equiv u_{\rho_1.\rho^\omega}$ using Corollary 2.2. Similarly, $v_\sigma \equiv v_{\rho_1.\rho^\omega}$. Hence $u_\sigma \sqsubseteq v_\sigma$ entails $u_{\rho_1.\rho^\omega} \sqsubseteq v_{\rho_1.\rho^\omega}$. Using Lemma 2.1, we conclude that $u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2}$ can be obtained by picking for $\rho_2$ a large enough power $\rho_2 \stackrel{\text{def}}{=} \rho.\rho \ldots \rho$ of $\rho$. Such a $\rho_2$ further ensures $\rho_2^\omega = \rho^\omega$, so that requirements (a) and (c) are inherited from $\rho$. □

For the next step, we show how to state the existence of two finite $\rho_1$ and $\rho_2$ as in Lemma 4.3 under the form of a PEP$^{\text{reg}}$ problem.

Let $\mathcal{A}_R = (Q, \Sigma, q_0, F, \delta)$ be the Büchi automaton defining $R$. As is standard, for $q, q' \in Q$, we let $L_{q,q'} \subseteq \Sigma^*$ denote the (regular) language accepted by starting $\mathcal{A}_R$ in $q$ and stopping in $q'$.

Let $\Sigma' = \{1', 2', \ldots\}$ be a copy of $\Sigma = \{1, 2, \ldots\}$ where letters have been primed: for $x \in \Sigma^*$ and $L \subseteq \Sigma^*$, we let $x' \in \Sigma'^*$ and $L' \subseteq \Sigma'^*$ denote primed versions of $x$ and $L$.

We can now express condition (a) as a regularity constraint on $\rho_1.\rho_2'$: by definition, $\rho_1.\rho_2^\omega$ belongs to $R$ iff for some $q \in Q$, $\rho_1 \in L_{q_0,q}$ and $\rho_2 \in (L_{q,q} \setminus \varepsilon)$. That is, if and only if $\rho_1.\rho_2' \in R_1$ with

$$R_1 \stackrel{\text{def}}{=} \bigcup_{q \in Q} L_{q_0,q}.(L'_{q,q} \smallsetminus \varepsilon).$$

Condition (b) can be stated as an embedding property on $\rho_1.\rho'_2$: let $u', v' : (\Sigma \cup \Sigma')^* \to \Gamma^*$ be the extensions of $u$ and $v$ given by $u'_{i'} \stackrel{\text{def}}{=} \varepsilon$ and $v'_{i'} \stackrel{\text{def}}{=} v_i$. Then

$$u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2} \text{ if and only if } u'_{\rho_1.\rho'_2} \sqsubseteq v'_{\rho_1.\rho'_2}.$$

Finally, condition (c) can be expressed as another regularity constraint. Indeed, for $X \subseteq \Gamma$, $alph(u_{\rho_2}) \subseteq X$ and $alph(v_{\rho_2}) \subseteq X$ require $\rho_2 \in u^{-1}(X^*)$ and, respectively, $\rho_2 \in v^{-1}(X^*)$. These are regular conditions on $\rho_2$ since inverse morphisms preserve regularity. Let now

$$R_2 \stackrel{\text{def}}{=} \bigcup_{X \subseteq \Gamma} \Big( u^{-1}(X^*) \cap v^{-1}(X^*) \cap \bigcap_{a \in X} \overbrace{\Sigma^* \{ i \in \Sigma \mid a \in alph(v_i) \} \Sigma^*}^{a \in alph(v_{\rho_2})} \Big).$$

Clearly, $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$ if and only if $\rho_2 \in R_2$. Hence $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$ if, and only if, $\rho_1.\rho'_2 \in \Sigma^*.(R_2)'$ where we observe that $R_2$, hence $\Sigma^*.(R_2)'$ too, are regular.

Finally, $u, v$ has an ω-solution in $R$ iff $u', v'$ has a finite solution in $R_1 \cap (R_2)'$, which provides the reduction from $\mathsf{PEP}^{\omega\text{-reg}}$ to $\mathsf{PEP}^{\text{reg}}$.

*Remark 4.4.* The automaton for $R_1$ has size linear in $|\mathcal{A}_R|$. The automaton for $R_2$ has size exponential in $|\Sigma|$: this is because we consider all subsets $X \subseteq \Sigma$. Hence the reduction from $\mathsf{PEP}^{\omega\text{-reg}}$ to $\mathsf{PEP}^{\text{reg}}$ is not logspace when the constraint $R$ is given by a non-deterministic FSA. It is polynomial-space, which is certainly fine enough to state "equivalence" by inter-reducibility between problems that are not primitive-recursive.

There exists other possible choices for the precise finitary way with which $R$ is supposed to be provided in a PEP instance: for many of these choices, from various logical formalisms (e.g., MSO) to various automata-based framework (e.g., alternating automata), logspace reductions from $\mathsf{PEP}^{\omega\text{-reg}}$ to $\mathsf{PEP}^{\text{reg}}$ exist. □

We conclude this section with the following observation:

**Theorem 4.5.** $\mathsf{PEP}^{\omega\text{-}reg}_{codir}$ *and* $\mathsf{PEP}^{reg}_{codir}$ *are equivalent (inter-reducible).*

This can be proved using the same techniques we used in this section, in particular one can state a version of Lemma 4.3 that accounts for codirect solutions (while this is not possible for direct solutions). Then a *codirect* infinite solution $\sigma$ induces the existence of a *codirect* $\rho_1.\rho_2^\omega$, and the existence of such an infinite $\rho_1.\rho_2^\omega$ can be witnessed by a finite $\rho_1.\rho'_2$ that solves a derived $\mathsf{PEP}^{\text{reg}}_{\text{codir}}$ instance.

## 5 Unidirectional Channel Systems

Unidirectional channel systems, shortly UCS, are systems composed of two finite-state machines that communicate *unidirectionally* via one reliable and one lossy channel,

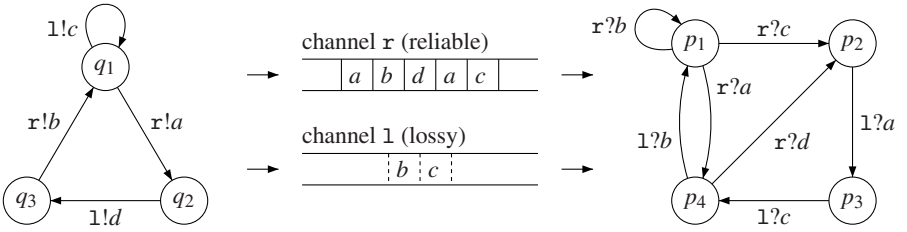



**Fig. 1.** A unidirectional channel system with one r eliable and one l ossy channel

as illustrated in Fig. 1. No feedback communication from the receiver to the sender is possible. UCS's are a key ingredient in the complete classification of mixed channel systems according to their network topologies [Cha07].

Formally, a UCS has the form $S = (Q_1, Q_2, \mathtt{M}, \{\mathtt{r},\mathtt{l}\}, \Delta_1, \Delta_2)$, where $Q_1$ and $\Delta_1$ (respectively, $Q_2$ and $\Delta_2$) are the finite set of states and set of rules of the sender (respectively, the receiver), M is the finite message alphabet, r and l are the names of, respectively, the reliable and the lossy channel. The sender's rules, $\Delta_1$, is a subset of $Q_1 \times \{\mathtt{r},\mathtt{l}\} \times \{!\} \times \mathtt{M}^* \times Q_1$, i.e., it contains rules of the form $q \xrightarrow{\mathtt{r}!u} q'$ or $q \xrightarrow{\mathtt{l}!u} q'$. The receiver's rules have the form $q \xrightarrow{\mathtt{r}?u} q'$ or $q \xrightarrow{\mathtt{l}?u} q'$ with $q, q' \in Q_2$.

A configuration of $S$ is a tuple $\langle q_1, q_2, v_1, v_2\rangle$ with control states $q_1$ and $q_2$ for the components, contents $v_1$ for channel r, and $v_2$ for l. The operational semantics is as expected. A rule $q \xrightarrow{\mathtt{r}!u} q'$ (resp. $q \xrightarrow{\mathtt{l}!u} q'$) from $\Delta_1$ gives rise to all transitions $\langle q, q_2, v_1, v_2\rangle \to \langle q', q_2, v_1 u, v_2\rangle$ (resp. all $\langle q, q_2, v_1, v_2\rangle \to \langle q', q_2, v_1, v_2 u'\rangle$ for $u' \sqsubseteq u$). A rule $q \xrightarrow{\mathtt{r}?u} q'$ (resp. $q \xrightarrow{\mathtt{l}?u} q'$) from $\Delta_2$ gives rise to all transitions $\langle q_1, q, uv_1, v_2\rangle \to \langle q_1, q', v_1, v_2\rangle$ (resp. all $\langle q_1, q, v_1, uv_2\rangle \to \langle q_1, q', v_1, v_2\rangle$). Observe that message losses only occur when writing to channel l. A run $\pi$ is a sequence

$$\pi : \langle q_1^0, q_2^0, v_1^0, v_2^0\rangle \to \langle q_1^1, q_2^1, v_1^1, v_2^1\rangle \to \langle q_1^2, q_2^2, v_1^2, v_2^2\rangle \to \cdots$$

of configurations linked by valid transitions.

We consider reachability and recurrent reachability problems for UCS's. Formally, given a UCS $S$, two initial states $q^1_{\text{init}} \in Q_1$ and $q^2_{\text{init}} \in Q_2$, two sets $F_1 \subseteq Q_1$ and $F_2 \subseteq Q_2$ of final states, the *reachability problem*, denoted ReachUcs, asks whether there exists a run that starts from configuration $\langle q^1_{\text{init}}, q^2_{\text{init}}, \varepsilon, \varepsilon\rangle$ and ends in some configuration $\langle q^1_{\text{final}}, q^2_{\text{final}}, \varepsilon, \varepsilon\rangle$ with $(q^1_{\text{final}}, q^2_{\text{final}}) \in F_1 \times F_2$. The *recurrent reachability problem*, denoted RecReachUcs, asks whether there exists an infinite run starting from $\langle q^1_{\text{init}}, q^2_{\text{init}}, \varepsilon, \varepsilon\rangle$ and visiting infinitely many configurations $\langle q_1^i, q_2^i, v_1^i, v_2^i\rangle$ with $(q_1^i, q_2^i) \in F_1 \times F_2$.

*Remark 5.1.* As explained in [CS07], requiring that our reachability questions have empty channels in the initial and the target configurations is just a technical simplification. More general reachability questions, including *control-state reachability*, where the channels contents in the target configuration are existentially quantified upon, reduce easily to ReachUcs. □

**Theorem 5.2 (Equivalence between UCS and Post Embedding)**
*1.* PEP$^{reg}$ *and* ReachUcs *are equivalent (inter-reducible).*
*2.* PEP$^{\omega\text{-}reg}$ *and* RecReachUcs *are equivalent (inter-reducible).*

The finitary case was first stated and proved in [CS07]. In the rest of this section, we develop a new and more modular proof that also applies to the ω-regular case.

We first introduce an abstract version of the UCS problems that is closer to PEP:

**Definition 5.3** (2PCEP)
a. *The* 2-dimensional correspondence plus embedding problem *asks, given two pairs of morphisms* $f_1, g_1 : \Sigma_1^* \to \Gamma^*$ *and* $f_2, g_2 : \Sigma_2^* \to \Gamma^*$, *to find words* $\sigma_1$ *and* $\sigma_2$ *s.t.* $f_1(\sigma_1) = f_2(\sigma_2)$ *(correspondence) and* $g_1(\sigma_1) \sqsubseteq g_2(\sigma_2)$ *(embedding).*
b. 2PCEP$^{reg}$ *is the decision problem, where given* $f_1, g_1, f_2, g_2$ *and two regular languages* $R_1 \subseteq \Sigma_1^*$ *and* $R_2 \subseteq \Sigma_2^*$, *one asks whether there is a solution with* $\sigma_1 \in R_1$ *and* $\sigma_2 \in R_2$.
c. 2PCEP$^{\omega\text{-}reg}$ *is the infinitary version of* 2PCEP$^{reg}$, *where now* $R_1 \subseteq \Sigma_1^\omega$ *and* $R_2 \subseteq \Sigma_2^\omega$ *are two given* ω*-regular languages, and where one looks for* ω*-solutions with* $\sigma_1 \in R_1$ *and* $\sigma_2 \in R_2$.

**Lemma 5.4 (See Appendix)**
*1.* ReachUcs *and* 2PCEP$^{reg}$ *are equivalent.*
*2.* RecReachUcs *and* 2PCEP$^{\omega\text{-}reg}$ *are equivalent.*

We now reduce 2-dim correspondence+embedding to Post embedding:

**Lemma 5.5 (See Appendix)**
*1.* 2PCEP$^{reg}$ *reduces to* PEP$^{reg}$.
*2.* 2PCEP$^{\omega\text{-}reg}$ *reduces to* PEP$^{\omega\text{-}reg}$.

We can now conclude the proof of Theorem 5.2: since PEP$^{\text{reg}}$ can be seen as a special case of 2PCEP$^{\text{reg}}$ (let $f_1 = f_2 = Id$, $g_1 = u$, $g_2 = v$) and, similarly, PEP$^{\omega\text{-reg}}$ as a special case of 2PCEP$^{\omega\text{-reg}}$, Lemmas 5.4 and 5.5 entail the equivalence of PEP$^{\text{reg}}$ and ReachUcs on the one hand, of PEP$^{\omega\text{-reg}}$ and RecReachUcs on the other hand.

## 6 Lossy Channel Systems

Systems composed of several finite-state components communicating via several channels (all of them lossy) can be simulated by systems with a single channel and a single component (see, e.g., [Sch02, Section 5]). Hence we define here a lossy channel system (a LCS) as a tuple $S = (Q, \mathtt{M}, \{\mathtt{c}\}, \Delta)$ as illustrated in Fig. 2. Rules read from, or write to, the single channel c. Configurations of $S$ are pairs $\langle q, v\rangle \in Q \times \mathtt{M}^*$ of a state and a channel contents. Transitions between configurations are obtained from the rules as expected, in the write-lossy spirit we just used for UCS's (see [CS07] for a formal definition).

ReachLcs, the *reachability problem for LCS's*, is the question, given a LCS $S$, an initial state $q_{\text{init}} \in Q$ and a set $F \subseteq Q$ of final states, whether $S$ has a run that goes from $\langle q_{\text{init}}, \varepsilon\rangle$ to $\langle q, \varepsilon\rangle$ for some $q \in F$. RecReachLcs, the *recurrent reachability problem for LCS's*, is the question whether $S$ has an infinite run $\langle q_{\text{init}}, \varepsilon\rangle \to \langle q_1, v_1\rangle \to$

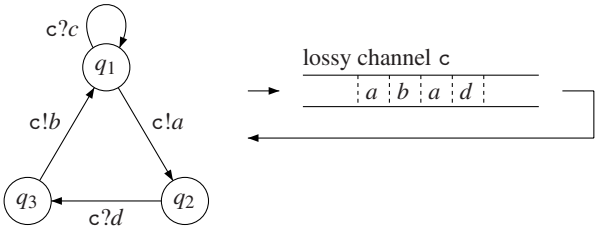



**Fig. 2.** A single-component system with a single lossy channel

$\langle q_2, v_2\rangle \to \cdots$ with $q_k \in F$ for infinitely many $k \in \mathbb{N}$. Recall that ReachLcs is decidable [Pac87, AJ96b, BBS06] (albeit not primitive-recursive [Sch02]) while RecReachLcs is undecidable [AJ96a] (albeit r.e.).[2] Furthermore, ReachUcs and ReachLcs (and $\mathsf{PEP}^{\mathrm{reg}}$) are inter-reducible [CS07].

In the rest of this section we prove the following theorem.

**Theorem 6.1.** $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$ *and* RecReachLcs *are equivalent (inter-reducible).*

**Corollary 6.2.** $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$ *is (r.e. but) undecidable.*

The two directions of Theorem 6.1 are given by Lemmas 6.3 and 6.4.

**Lemma 6.3.** $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$ *reduces to* RecReachLcs.

*Proof.* The reduction from $\mathsf{PEP}^{\omega\text{-reg}}_{\mathrm{dir}}$ to RecReachLcs is illustrated in Fig. 3, where the "rules" of the form $q \xrightarrow{\mathtt{c}!x\,\mathtt{c}?y} q'$ are just a shorthand description for two consecutive rules $q \xrightarrow{\mathtt{c}!x} q_?$ and $q_? \xrightarrow{\mathtt{c}?y} q'$ that traverse an anonymous intermediary state $q_?$. Simply put, the LCS $S_{u,v,R}$ mimics the Büchi automaton $\mathcal{A}_R$ that defines the constraint $R \subseteq \Sigma^\omega$. A run of the LCS that visits $F$ infinitely often will performs steps $1,2,3,\ldots$, writing to the channel some $v'_1, v'_2, v'_3, \ldots$, that are subwords (because of message losses) of $v_{i_1}, v_{i_2}, v_{i_3}, \ldots$ (the writes prescribed by the rules). During these same steps, it reads $u_{i_1}, u_{i_2}, u_{i_3}, \ldots$, from the channel. These read letters must have been written earlier, hence for $k = 1,2,3,\ldots$, $u_{i_1}\ldots u_{i_k}$ is a prefix of $v'_1\ldots v'_k$, hence a subword of $v_{i_1}\ldots v_{i_k}$. Finally, $\sigma \stackrel{\text{def}}{=} i_1.i_2.i_3\ldots$ is a direct solution.

Reciprocally, given a direct solution $\sigma = i_1.i_2.i_3\ldots$, it is possible (using the general embedding provided by Lemma 3.1) to find subwords $v'_1, v'_2, v'_3, \ldots$ of $v_{i_1}, v_{i_2}, v_{i_3}, \ldots$ s.t., for all $k = 1,2,\ldots$, $u_{i_1}\ldots u_{i_k}$ is a prefix of $v'_1\ldots v'_k$. Using these $v'_k$, one easily obtains an infinite run of the LCS that shows the associated RecReachLcs is positive. □

**Lemma 6.4.** RecReachLcs *reduces to* $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$.

*Proof.* Consider a RecReachLcs instance $S = (Q, \mathtt{M}, \{\mathtt{c}\}, \Delta)$ with given $q_{\mathrm{init}}$ and $F$. With it, we associate a $\mathsf{PEP}^{\omega\text{-reg}}_{\mathrm{dir}}$ instance where $\Sigma = \Delta$ and where $R \subseteq \Sigma^\omega$ is given by the Büchi automaton that is exactly like $S$, with the difference that any rule $\delta$ between some

[2] For Turing machines, the reachability problem is undecidable albeit r.e., while the recurrent reachability problem is $\Sigma^1_1$-complete.

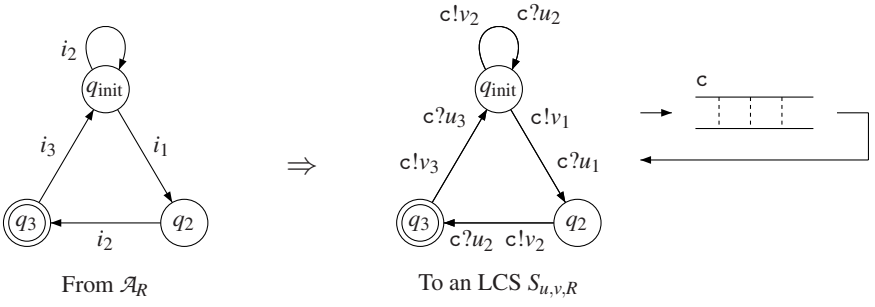



**Fig. 3.** Reductions between $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ and RecReachLcs

states $q$ and $q'$ is now a transition $q \xrightarrow{\delta} q'$ in $\mathcal{A}_R$. The morphisms $u, v$ are defined by $u(\delta) \stackrel{\text{def}}{=}$ "what rule δ reads in channel c", $v(\delta) \stackrel{\text{def}}{=}$ "what δ writes in c". Since $u(\delta) = \varepsilon$ or $v(\delta) = \varepsilon$ for every rule (LCS's rules either read or write to c, not both), $S$ (essentially) coincides with $S_{u,v,R}$ (Fig. 3). Hence the proof of Lemma 6.3 shows that $u, v, R$ is a positive $\mathsf{PEP}^{\omega\text{-reg}}$ instance iff the original RecReachUcs instance is positive. □

## 7 Concluding Remarks

We introduced infinitary versions of $\mathsf{PEP}^{\text{reg}}$, a new and exciting variant of Post Correspondence Problem based on embedding rather than equality, which also is an abstract representative of the LCS complexity niche.

Our main result is that two such infinitary versions, $\mathsf{PEP}^{\omega\text{-reg}}$ and $\mathsf{PEP}^{\omega\text{-reg}}_{\text{codir}}$, are equivalent to the finitary $\mathsf{PEP}^{\text{reg}}$. Hence they are decidable albeit not in primitive-recursive time. Since one can link $\mathsf{PEP}^{\omega\text{-reg}}$ and RecReachUcs, the recurrent reachability problem for unidirectional channel systems, we obtain the decidability of RecReachUcs. In fact, and quite surprisingly, RecReachUcs and PEP or ReachLcs are equivalent. The last version, $\mathsf{PEP}^{\omega\text{-reg}}_{\text{codir}}$, is equivalent to RecReachLcs, the recurrent reachability problem for lossy channel systems, which is undecidable albeit r.e. Finally, the PTime-complete unconstrained $\mathsf{PEP}^{\omega}$ is harder that the unconstrained PEP that can be solved in logspace.

## References

[ADOW05] Abdulla, P.A., Deneux, J., Ouaknine, J., Worrell, J.: Decidability and complexity results for timed automata via channel machines. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1089–1101. Springer, Heidelberg (2005)

[AJ96a] Abdulla, P.A., Jonsson, B.: Undecidable verification problems for programs with unreliable channels. Information and Computation 130(1), 71–90 (1996)

[AJ96b] Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)

[AM02] Amadio, R., Meyssonnier, C.: On decidability of the control reachability problem in the asynchronous π-calculus. Nordic Journal of Computing 9(2), 70–101 (2002)

[BBS06] 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)

[Cha07] Chambart, P.: Canaux fiables et non-fiables: frontières de la décidabilité. Rapport de Master, Master Parisien de Recherche en Informatique, Paris, France (September 2007)

[CS07] Chambart, P., Schnoebelen, P.: Post embedding problem is not primitive recursive, with applications to channel systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)

[Del07] Delzanno, G.: Constraint-based automatic verification of abstract models of multithreaded programs. Theory and Practice of Logic Programming 7(1–2), 67–91 (2007)

[DL06] Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. In: Proc. LICS 2006, pp. 17–26. IEEE Comp. Soc. Press, Los Alamitos (2006)

[Fin85] Finkel, A.: Une généralisation des théorèmes de Higman et de Simon aux mots infinis. Theoretical Computer Science 38(1), 137–142 (1985)

[GKWZ06] Gabelaia, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Non-primitive recursive decidability of products of modal logics with expanding domains. Annals of Pure and Applied Logic 142(1–3), 245–268 (2006)

[JL07] Jurdziński, M., Lazić, R.: Alternation-free modal mu-calculus for data trees. In: Proc. LICS 2007, pp. 131–140. IEEE Comp. Soc. Press, Los Alamitos (2007)

[KWZ05] Konev, B., Wolter, F., Zakharyaschev, M.: Temporal logics over transitive states. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 182–203. Springer, Heidelberg (2005)

[Kur06] Kurucz, A.: Combining modal logics. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logics, vol. 3, ch. 15, pp. 869–926. Elsevier, Amsterdam (2006)

[LW05] Lasota, S., Walukiewicz, I.: Alternating timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 250–265. Springer, Heidelberg (2005)

[LNO+07] Lazić, R., Newcomb, T., Ouaknine, J., Roscoe, A.W., Worrell, J.: Nets with tokens which carry data. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 301–320. Springer, Heidelberg (2007)

[OW06] Ouaknine, J., Worrell, J.: On metric temporal logic and faulty Turing machines. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 217–230. Springer, Heidelberg (2006)

[OW07] Ouaknine, J., Worrell, J.: On the decidability and complexity of Metric Temporal Logic over finite words. Logical Methods in Comp. Science 3(1), 1–27 (2007)

[Pac87] Pachl, J.K.: Protocol description and analysis based on a state transition model with channel expressions. In: Proc. PSTV 1987, pp. 207–219. North-Holland, Amsterdam (1987)

[Sch02] Schnoebelen, P.: Verifying lossy channel systems has nonprimitive recursive complexity. Information Processing Letters 83(5), 251–261 (2002)

# A Proofs for Section 5

## A.1 Commuting UCS Steps

We first state a trivial but important property about runs of unidirectional systems. Let $S = (Q_1, Q_2, \mathtt{M}, \{\mathtt{r}, \mathtt{l}\}, \Delta_1, \Delta_2)$ be some UCS, and $\langle q_1, q_2, x, y\rangle \xrightarrow{\delta_2} \langle q_1, q_2', x', y'\rangle \xrightarrow{\delta_1} \langle q_1', q_2', x'', y''\rangle$ be two consecutive steps with $\delta_1 \in \Delta_1$ and $\delta_2 \in \Delta_2$, i.e., where the receiver performs the first step, and the sender the second step. Then it is possible to fire $\delta_1$ before $\delta_2$ and reach the same configuration. More precisely, there exists $x'''$ and $y'''$ with $\langle q_1, q_2, x, y\rangle \xrightarrow{\delta_1} \langle q_1', q_2, x''', y'''\rangle \xrightarrow{\delta_1} \langle q_1', q_2', x'', y''\rangle$.

The corollaries are

**Lemma A.1.** *If $S$ has a run $\langle q_1, q_2, x, y\rangle \xrightarrow{\Delta_1 \cup \Delta_2}{}^* \langle q_1', q_2', x', y'\rangle$ then it has one such run of the form*

$$\langle q_1, q_2, x, y\rangle \xrightarrow{\Delta_1}{}^* \langle q_1', q_2, x'', y''\rangle \xrightarrow{\Delta_2}{}^* \langle q_1', q_2', x', y'\rangle.$$

**Lemma A.2.** *If $S$ has an infinite run from $\langle q_0^1, q_0^2, x_0, y_0\rangle$ of the form*

$$\langle q_0^1, q_0^2, x_0, y_0\rangle \to \langle q_1^1, q_1^2, x_1, y_1\rangle \to \langle q_2^1, q_2^2, x_2, y_2\rangle \to \cdots$$

*with $q^1 = q_i^1$ for infinitely many $i$'s, and $q^2 = q_i^2$ for infinitely many $i$'s (not necessarily the same), then it has one such run with $(q^1, q^2) = (q_i^1, q_i^2)$ for infinitely many $i$'s.*

## A.2 Proof of Lemma 5.4

2PCEP$^{\text{reg}}$ reduces to ReachUcs, and 2PCEP$^{\omega\text{-reg}}$ to RecReachUcs

For this, consider a 2PCEP$^{\text{reg}}$ instance $f_1, g_1, f_2, g_2, R_1, R_2$ as in Definition 5.3.b. Further assume that, for $i = 1, 2$, $R_i$ is given by some FSA $\mathcal{A}_i = (Q_i, \Sigma_i, q_{\text{init}}^i, F_i, \delta_i)$.

With this instance, we associate an UCS where the the sender is obtained from $\mathcal{A}_2$ by replacing transitions $q \xrightarrow{i} q' \in \delta_2$ with rules $q \xrightarrow{\mathtt{r}!f_2(i)\,\mathtt{l}!g_2(i)} q'$, and the receiver is obtained from $\mathcal{A}_1$ by replacing transitions $q \xrightarrow{i} q' \in \delta_1$ with rules $q \xrightarrow{\mathtt{r}?f_1(i)\,\mathtt{l}?g_1(i)} q'$.

If the 2PCEP$^{\text{reg}}$ instance is positive, then a solution $\sigma_1, \sigma_2$ can be used in a straightforward way to build, out of $\sigma_2$, a run in the UCS that will start from $\langle q_{\text{init}}^2, q_{\text{init}}^1, \varepsilon, \varepsilon\rangle$, will reach some $\langle q_{\text{final}}^2, q_{\text{init}}^1, f_2(\sigma_2), x\rangle$ for some $q_{\text{final}}^2 \in F_2$, and where, using message losses, we can choose to reach any $x \sqsubseteq g_2(\sigma_2)$. By picking $x = g_1(\sigma_1)$, we can now continue the run, using $\sigma_1$, and reach $\langle q_{\text{final}}^1, q_{\text{final}}^2, \varepsilon, \varepsilon\rangle$ for some $q_{\text{final}}^1 \in F_1$.

Reciprocally, using Lemma A.1, a run from $\langle q_{\text{init}}^2, q_{\text{init}}^1, \varepsilon, \varepsilon\rangle$ to some $\langle q_{\text{final}}^1, q_{\text{final}}^2, \varepsilon, \varepsilon\rangle$ can be reordered into some

$$\langle q_{\text{init}}^2, q_{\text{init}}^1, \varepsilon, \varepsilon\rangle \underbrace{\xrightarrow{r_1} \xrightarrow{r_2} \cdots \xrightarrow{r_n}}_{\text{rules from } \Delta_1} \langle q_{\text{final}}^2, q_{\text{init}}^1, x, y\rangle \underbrace{\xrightarrow{r_1'} \xrightarrow{r_2'} \cdots \xrightarrow{r_m'}}_{\text{rules from } \Delta_2} \langle q_{\text{final}}^1, q_{\text{final}}^2, \varepsilon, \varepsilon\rangle$$

where all sender's steps occur first, followed by the receiver steps. This translates into a path $q_{\text{init}}^2 \xrightarrow{\sigma_2} q_{\text{final}}^2$ in $\mathcal{A}_2$, and $q_{\text{init}}^1 \xrightarrow{\sigma_1} q_{\text{final}}^1$ in $\mathcal{A}_1$ where $f_2(\sigma_2) = x = f_1(\sigma_1)$, and where $g_2(\sigma_2) \sqsupseteq y = g_1(\sigma_1)$, solving the 2PCEP$^{\text{reg}}$ instance.

Finally, the 2PCEP$^{\text{reg}}$ instance is positive iff the associated ReachUcs instance is. Hence 2PCEP$^{\text{reg}}$ reduces to ReachUcs.

The same association of an UCS with $f_1, g_1, f_2, g_2, \mathcal{A}_1, \mathcal{A}_2$ shows that 2PCEP$^{\omega\text{-reg}}$ reduces to RecReachUcs.

Indeed, an infinite solution $\sigma_1, \sigma_2$ in some ω-regular languages $R_1$ and $R_2$, can be used to build an infinite run of the UCS that visit infinitely many configurations $\langle q^2_{\text{final}}, q^1_i, x_i, y_i\rangle$ with some $q^2_{\text{final}} \in F_2$, and infinitely many configurations $\langle q^2_i, q^1_{\text{final}}, x'_i, y'_i\rangle$ with some $q^1_{\text{final}} \in F_1$. Using Lemma A.2, this run can be reordered into a run visiting infinitely many configurations $\langle q^2_{\text{final}}, q^1_{\text{final}}, x''_i, y''_i\rangle$, showing the RecReachUcs instance is positive.

Reciprocally, from an infinite run of the UCS that visits infinitely many configurations of the form $\langle q^2_{\text{final}}, q^1_{\text{final}}, x''_i, y''_i\rangle$, one extracts two solutions $\sigma_1, \sigma_2$ that show that the 2PCEP$^{\omega\text{-reg}}$ instance is positive.

ReachUcs reduces to 2PCEP$^{\text{reg}}$, and RecReachUcs to 2PCEP$^{\omega\text{-reg}}$

Consider an ReachUcs instance with some UCS $S = (Q_1, Q_2, \mathtt{M}, \{\mathtt{r}, \mathtt{l}\}, \Delta_1, \Delta_2)$, some initial states $q^1_{\text{init}}, q^2_{\text{init}}$, and some sets of final states $F_1, F_2$.

With this instance, we associate a 2PCEP$^{\text{reg}}$ instance where $\Sigma_1 \stackrel{\text{def}}{=} \Delta_2$ and $\Sigma_2 \stackrel{\text{def}}{=} \Delta_1$ are the set of rules. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ for $R_1$ and $R_2$ are obtained from the control graph of the receiver (resp., the sender) in the obvious way. (Note that we extract FSA's from an ReachUcs instance, and Büchi automata from an RecReachUcs instance.) The morphisms are defined in the obvious way:

$$f_1(\delta) \stackrel{\text{def}}{=} x \text{ and } g_1(\delta) \stackrel{\text{def}}{=} y \text{ for } \delta = q \xrightarrow{\mathtt{r}?x\,\mathtt{l}?y} r \text{ in } \Delta_2,$$
$$f_2(\delta) \stackrel{\text{def}}{=} x \text{ and } g_2(\delta) \stackrel{\text{def}}{=} y \text{ for } \delta = q \xrightarrow{\mathtt{r}!x\,\mathtt{l}!y} r \text{ in } \Delta_1.$$

### A.3 Proof of Lemma 5.5

We consider a 2PCEP instance $f_1, g_1, f_2, g_2$ where we assume that the morphisms are short, i.e., $f_i$ and $g_i$ can be seen as having type $(\Sigma_i \cup \{\varepsilon\}) \to (\Gamma_\cup \{\varepsilon\})$. For 2PCEP$^{\text{reg}}$ and 2PCEP$^{\omega\text{-reg}}$, and thanks to the possibility offered by the regular constraints, this assumption is no loss of generality, as can be easily proved using the techniques from section 3.3.

Let $\Sigma \stackrel{\text{def}}{=} (\Sigma_1 \cup \{\varepsilon\}) \times (\Sigma_2 \cup \{\varepsilon\})$ and define $X \subseteq \Sigma$ by

$$(i, j) \in X \text{ if and only if } f_1(i) = f_2(j).$$

Then $(i_1, j_1).(i_2, j_2)\ldots(i_n, j_n) \in X^*$ implies that $f_1(i_1.i_2\ldots i_n) = f_2(j_1.j_2\ldots j_n)$. Reciprocally, if $f_1(\sigma_1) = f_2(\sigma_2)$, then $\sigma_1$ and $\sigma_2$ can be decomposed under the form $\sigma_1 = i_1.i_2\ldots i_n$ and $\sigma_2 = j_1.j_2\ldots j_n$ such that $(i_k, j_k) \in X$ for $k = 1, \ldots, n$. Observe that in this decomposition, $n \geq |\sigma_i|$ is possible since $i_k = \varepsilon$ or $j_k = \varepsilon$ (or both) is allowed.

Now define projection morphisms $h_1 : \Sigma^* \to \Sigma_1^*$ and $h_2 : \Sigma^* \to \Sigma_2^*$ in the obvious way, and let $u, v : \Sigma^* \to \Gamma^*$ be two morphisms given by $u \stackrel{\text{def}}{=} g_1 \circ h_1$ and $v \stackrel{\text{def}}{=} g_2 \circ h_2$. Then $u_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)} \sqsubseteq v_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)}$ if and only if $g_1(i_1.i_2\ldots i_n) \sqsubseteq g_2(j_1.j_2\ldots j_n)$.

Finally, the $\mathsf{2PCEP}^{\mathrm{reg}}$ instance with regular constraints $R_1, R_2$ translates into an equivalent $\mathsf{PEP}^{\mathrm{reg}}$ instance, with morphisms $u$ and $v$ as above, and with constraint

$$R \stackrel{\mathrm{def}}{=} X^* \cap {h_1}^{-1}(R_1) \cap {h_2}^{-1}(R_2),$$

which is regular. Similarly, the $\mathsf{2PCEP}^{\omega\text{-reg}}$ instance with ω-regular constraints $R_1, R_2$ translates into an equivalent $\mathsf{PEP}^{\omega\text{-reg}}$ instance, with same morphisms $u$ and $v$, and with constraint

$$R \stackrel{\mathrm{def}}{=} X^\omega \cap {h_1}^{-1}(R_1) \cap {h_2}^{-1}(R_2),$$

which is ω-regular.

# Complexity of Decision Problems for Mixed and Modal Specifications*

Adam Antonik[1], Michael Huth[1],
Kim G. Larsen[2], Ulrik Nyman[2], and Andrzej Wąsowski[2,3]

[1] Department of Computing, Imperial College London, United Kingdom
{aa1001,mrh}@doc.imperial.ac.uk
[2] Department of Computer Science, Aalborg University, Denmark
{kgl,ulrik}@cs.aau.dk
[3] IT University of Copenhagen, Denmark
wasowski@itu.dk

**Abstract.** We consider decision problems for modal and mixed transition systems used as specifications: the *common implementation* problem (whether a set of specifications has a common implementation), the *consistency* problem (whether a single specification has an implementation), and the *thorough refinement* problem (whether all implementations of one specification are also implementations of another one). *Common implementation* and *thorough refinement* are shown to be PSPACE-hard for modal, and so also for mixed, specifications. *Consistency* is PSPACE-hard for mixed, while trivial for modal specifications. We also supply upper bounds suggesting strong links between these problems.

## 1 Introduction

*Bisimulation equivalence* [1,2] is widely accepted as a correctness criterion for realizations of abstract specifications. Bisimulation is, however, a rather strong relation that severely, and often unnecessarily, limits the choices of designers in how specifications should be realized. At the same time, the main alternative, bisimulation's sister preorder *simulation* [1], is often too weak to use in this context as it only limits faulty behaviours, without enforcing any correct ones.

In order to address these shortcomings, Larsen and Thomsen [3] have proposed *modal transition systems* and the accompanying *modal refinement*, in this paper referred to simply as refinement. Modal transition systems feature required and allowed transitions able to simultaneously describe an under- and over-approximation of behavior within a single specification. Modal refinement generalizes both simulation and bisimulation, letting the specifier choose the required level of strictness in the spectrum between the two. In [4] Larsen argued that any sufficiently expressive specification language necessarily must accommodate

* Partially supported by the UK EPSRC projects *Efficient Specification Pattern Library for Model Validation (EP/D50595X/1)* and *Complete and Efficient Checks for Branching-Time Abstractions (EP/E028985/1).*

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 112–126, 2008.

inconsistent specifications, akin to inconsistent logical formulæ, and thus lifted the consistency requirement. The same type of systems were independently reintroduced by Dams as *mixed transition systems* [5,6].

Here we establish complexities of several decision procedures for this family of specification languages, addressing several long outstanding open problems:

CI. Deciding whether $k > 1$ modal transition systems have a *common implementation* is PSPACE-hard in the sum of the sizes of these $k$ systems.
C. Deciding whether a mixed transition system is *consistent*, i.e. whether it has an implementation, is PSPACE-hard in the size of that system.
TR. Deciding whether one modal transition system *thoroughly refines* another modal transition system is PSPACE-hard in the size of these systems.

We show quite strong links between these problems. In particular we efficiently reduce problems of type CI to problems of type C, and problems of type C to problems of type TR—though *mixed*, not necessarily modal, transition systems are the targets of that latter reduction. All three problems C, CI, and TR are shown to be in EXPTIME.

We begin with discussing the related work in Section 2 and introducing the basic concepts in Section 3. The hardness results and the aforementioned problem reductions for common implementation, consistency, and thorough refinement are the subject of Sections 4, 5, and 6 respectively. A general discussion, including the provision of upper bounds, is given in Section 7. We conclude in Section 8.

## 2 Related Work

Our terminology differs from that used in [7]: what we call "modal transition systems" and "mixed transition systems" are called respectively "syntactically consistent modal transition systems" and "modal transition systems" therein.

In [8] a superpolynomial algorithm was given for deciding CI for $k > 1$ modal specifications. The algorithm is exponential in $k$, but polynomial if $k$ is fixed. In particular, it computes a common implementation if there is one. These upper bounds also follow easily from the polynomial algorithm for consistency checking of a conjunction of disjunctive modal transition systems, as studied in [9].

Larsen et al. [7] show that TR is coNP-hard, while C is NP-hard. We strengthen both of these bounds here. They also hint at exponential upper bounds for both problems, without arguing how these can be achieved. We elaborate on how to attain these bounds, by giving precise reductions in Section 7.

Hussain and Huth [10] present an example of two modal specifications that have a common implementation but no greatest common implementation.

Fischbein et al. [11] use modal specifications for behavioral conformance checking of products with specifications of product families. They propose a new thorough refinement whose implementations are defined through a refinement notion that generalizes branching bisimulation. The thorough refinement obtained in this manner is finer than weak refinement, and argued to be more suitable for conformance checking. In the light of the present work it is very likely that this refinement can be shown to be PSPACE-hard in the size of the specifications.

## 3 Background

Let us begin with defining the basic objects of interest in our study [12,5,13]:

**Definition 1.** *For an action alphabet $\Sigma$, a* mixed transition system *$M$ is a triple $(S, R^{\Box}, R^{\Diamond})$, where $S$ is a set of states and $R^{\Box}, R^{\Diamond} \subseteq S \times \Sigma \times S$ are must- and may-transitions relations respectively. A* modal transition system *is a mixed transition system satisfying $R^{\Box} \subseteq R^{\Diamond}$; all its must-transitions are also may-transitions. A* pointed *mixed (respectively modal) transition system $(M, s)$ is a mixed (modal) transition system $M$ with a designated initial state $s \in S$. The size $|M|$ of a mixed (modal) transition system $M$ is defined as $|S| + |R^{\Box} \cup R^{\Diamond}|$. All transition systems considered here are finite, i.e. $\Sigma$ and $S$ are always finite sets.*

Throughout this paper we refer to pointed modal (mixed) transition systems as modal (mixed) *specifications*. Throughout figures, solid arrows denote $R^{\Box}$-transitions, dashed arrows denote $R^{\Diamond}$-transitions. Arrows without labels have an implicit $\star$-label, where $\star \in \Sigma$ is an action with context-dependent meaning. Two examples of modal specifications are depicted in Fig. 1, while a mixed specification that is not a modal specification can be seen in Fig. 5.

Modal refinement [12,5,13] is a refinement relationship for mixed specifications that allows verifying that one such specification is more abstract than another. It generalizes bisimulation [14] to underspecified models:

**Definition 2.** *A mixed specifications $(N, t_0) = ((S_N, R^{\Box}_N, R^{\Diamond}_N), t_0)$* refines *another mixed specification $(M, s_0) = ((S_M, R^{\Box}_M, R^{\Diamond}_M), s_0)$ over the same alphabet, written $(M, s_0) \prec (N, t_0)$, iff there is a relation $Q \subseteq S_M \times S_N$ containing $(s_0, t_0)$ and whenever $(s, t) \in Q$ then*

1. *for all $(s, a, s') \in R^{\Box}_M$ there exists some $(t, a, t') \in R^{\Box}_N$ with $(s', t') \in Q$.*
2. *for all $(t, a, t') \in R^{\Diamond}_N$ there exists some $(s, a, s') \in R^{\Diamond}_M$ with $(s', t') \in Q$.*

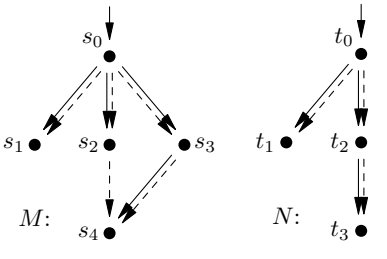



**Fig. 1.** Specifications $(M, s_0)$, $(N, t_0)$ with $I(M, s_0) = I(N, t_0)$ (so $I(M, s_0) \subseteq I(N, t_0)$), but not $(N, t_0) \prec (M, s_0)$

Deciding whether one finite-state mixed specification refines another one is in P. *Labeled transition systems* over an alphabet $\Sigma$ are pairs $(S, R)$ where $S$ is a set of states and $R \subseteq S \times \Sigma \times S$ is a transition relation. We identify labeled transition systems $(S, R)$ with modal transition systems $(S, R, R)$. The set of implementations $I(M, s)$ of a mixed specification $(M, s)$ are all pointed labeled transition systems $(T, t)$ refining $(M, s)$. Note that $I(M, s)$ may be empty in general, but is guaranteed to be non-empty if $M$ is a modal transition system.

*Example.* (Due to Harald Fecher) Figure 1 shows modal specifications $(M, s_0)$ and $(N, t_0)$ over alphabet $\{\star\}$. Relation $Q = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_2), (s_4, t_3)\}$ witnesses that $(N, t_0)$ refines $(M, s_0)$, but $(M, s_0)$ does not refine $(N, t_0)$.

As in [7] we define the *thorough refinement* $(M,s) \prec_{th} (N,t)$ to be the predicate $I(N,t) \subseteq I(M,s)$. Transitivity of refinement ensures that refinement soundly characterizes thorough refinement: $(M,s) \prec (N,t)$ implies $(M,s) \prec_{th} (N,t)$. But the converse does not hold: completeness of refinement for thorough refinement is known to be false [15,16,17]; Figure 1 provides a counterexample.

We shall now formally define the problems that we study, and briefly discuss their significance.

*Common implementation* (CI): given $k > 1$ mixed specifications $(M_i, s_i)$, is the set $\bigcap_{i=1}^{k} I(M_i, s_i)$ non-empty? For example, $(M_1, s_1)$ could be our system model and all other $(M_i, s_i)$ could be definitions of faulty behavior (respectively features). Common implementations are then possible implementations of our model that can exhibit all $k-1$ faults (features).

*Consistency* (C): Is $I(M,s)$ non-empty for a mixed specification $(M,s)$? Specification formalisms need the ability to express inconsistencies so that conflicts in systems or their design are detectable. Equally, inconsistent specifications may well result from the composition of consistent specifications.

*Thorough refinement* (TR): Does a mixed specification $(N,t)$ thoroughly refine a mixed specification $(M,s)$, i.e., do we have $I(N,t) \subseteq I(M,s)$? As refinement is only sound but not complete for thorough refinement, the question arises of whether thorough refinement has an efficient, e.g. co-inductive, definition that can be integrated in refinement tools.

We assume that specifications are finite-state, given their abstract nature. But implementations may (have to) be infinite-state as we otherwise cannot express important features, e.g. unbounded ranges of data types. For the three decision problems studied in this paper, it turns out that they won't change if we restrict implementations to finite-state ones. For example, a mixed specification $(M,s)$ is consistent in the infinite sense iff its characteristic modal mu-calculus formula $\Psi_{(M,s)}$ [18] is satisfiable. Appealing to the small model theorem for that logic, $\Psi_{(M,s)}$ is satisfiable iff it is satisfiable over finite-state implementations. We can reason in a similar manner about common implementation, through the formula $\bigwedge_i \Psi_{(M_i,s_i)}$. Finally, $(M,s) \prec_{th} (N,t)$ is false iff $\Psi_{(N,t)} \wedge \neg\Psi_{(M,s)}$ is satisfiable. This justifies that we consider only finite-state specifications and implementations.

Throughout this paper we work with Karp reductions, many-one reductions computable by deterministic Turing machines in polynomial time. This choice is justified since we reduce problems that are PSPACE-complete.

## 4 Common Implementation

We show that the CI problem is PSPACE-hard for modal specifications, which then automatically renders the same hardness result for mixed specifications.

**Theorem 3.** *Let* $\{(M_i, s_i) \mid 1 \leq i \leq k\}$ *with* $k > 1$ *be a finite family of modal specifications over the same action alphabet* $\Sigma$*. Deciding emptiness of the set* $\bigcap_{i=1}^{k} I(M_i, s_i)$ *is PSPACE-hard in* $\sum_{i=1}^{k} |M_i|$.

We argue for this by reduction from the Generalized Geography game [19,20].

**Definition 4.** *A rooted, directed graph is a structure* $G = (V, E, v_0)$*, where* $V$ *is a finite set of vertices,* $E \subseteq V \times V$ *is a set of edges and* $v_0 \in V$ *is the root. For an edge* $e = (u, v) \in E$ *we write* $\mathbf{tgt}\, e$ *for* $v$ *and* $\mathbf{src}\, e$ *for* $u$*, and we define* $Follow(e) := \{f \in E \mid \mathbf{tgt}\, e = \mathbf{src}\, f\}$ *and* $Init := \{e \in E \mid \mathbf{src}\, e = v_0\}$.

For $G = (V, E, v_0)$ the two-player *Generalized Geography* game on $G$ is played according to the following rules:

> "The two players alternate choosing a new edge from $E$. The first edge chosen (by player 1) must have its source at $v_0$ and each subsequently chosen edge must have its source at the vertex that was the target of the previous edge and must not have been previously chosen in the game. The first player unable to choose such a new edge loses." [19, p. 254]

The generalized geography problem (GenGeo) is whether given a rooted directed graph $G$ does there exist a winning strategy for player 1 in the *Generalized Geography* game played on $G$? GenGeo is PSPACE-complete [19].

*Proof (of Theorem 3).* We reduce GenGeo to checking CI of $k$ modal specifications $\{(M_i, s_i)\}$, where both $k$ and each $|M_i|$ are at most polynomial in the size of $G$. The reduction should be such that a common implementation of all $(M_i, s_i)$, if it exists, will explicitly give the winning strategy for Player 1.

We will create a set of modal specifications for each kind of conditions imposed by the game. All specifications will share an alphabet $\Sigma = E \cup \{\star\}$, where $\star$ is a fresh name such that $\star \notin E$. Choosing an edge in the game corresponds to taking a transition in these specifications.

Let us begin with modal specifications $(P_1, s_1)$ and $(P_2, s_2)$ presented in Figure 2, which ensure that Player 1 can always continue – a necessary condition for obtaining a winning strategy. Transitions with labels $X \subseteq \Sigma$ denote sets of transitions, one for each $e \in X$. We keep track of whose turn it is in the game by distinguishing Player 1 states from Player 2 states, labeling states with Player numbers for the sake of clarity. Observe that both $P_1$ and $P_2$ oscillate between Player 1 and Player 2 decisions. Each Player 2 move is modeled directly by a single transition, while a Player 1 move is modeled by exactly two transitions; a $\star$-transition followed by a regular edge transition. As will be seen later, disjunctive choices will only occur in Player 1 mode, so $\star$-transitions used to encode disjunctions are there only for Player 1 states. Specification $P_1$ limits choices of Player 1 to a disjunction of all legal actions, while $P_2$ enforces that at least one of these choices is indeed taken.

Let us continue with the remaining GenGeo game rules. We can enforce that an edge $e$ is played at most once using a modal specification $(M_e, s_e)$ shown in the left part of Figure 3. This specification models a flag that disallows any further $e$-transitions once $e$ has been used. Similarly, for each edge $e$ create a modal specification $(N_e, t_e)$, as shown in the right part of Figure 3, to constrain the moves following an $e$ move to edges directly following it. $N_e$ has a $\star$-labeled

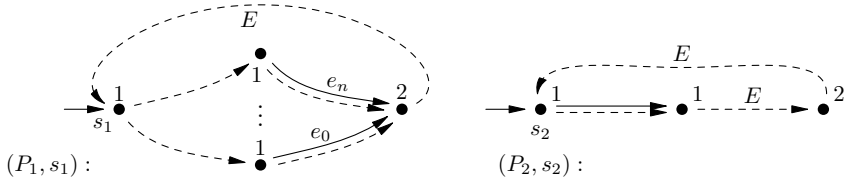



**Fig. 2.** Modal specifications $(P_1, s_1)$ and $(P_2, s_2)$ together ensuring that Player 1 can always continue playing. Assume $E = \{e_0, \dots e_n\}$.

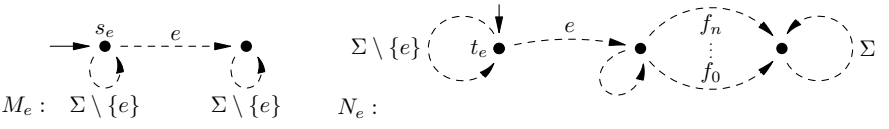



**Fig. 3.** Specifications $M_e$, $N_e$ instantiated for each $e \in E$ and $\{f_0, \dots, f_n\} = Follow(e)$

loop on its middle state to account for both Player 1 and Player 2 moves. Recall that if $e$ was played by Player 2, then in our encoding it will be first followed by a $\star$ before Player 1 plays any subsequent edge. The requirement that Player 1 should choose one of the transitions leaving the root as the first move is enforced by $(P_0, s_0)$ as shown in the left part of Figure 4.

We are left with the last and the most complex game rule, namely that whenever Player 1 makes a choice then Player 2 has to be able to respond with any so far unused edge $f$ following that choice. Our implementation, which directly represents the strategy, should thus have all transitions representing possible choices in such a state. We model this by creating a specification $(M_{ef}, s_{ef})$ for every pair of edges $e$ and $f$ such that $f \in Follow(e) \setminus \{e\}$. The idea is that each modal transition system $M_{ef}$ enforces an $f$ transition after an $e$ transition has been chosen by Player 1, unless $f$ has already been used (either by Player 1 or Player 2), or $e$ has been used by Player 2. See the right part of Figure 4.

The answer to GenGeo$(V, E, v_0)$ is yes iff the answer to CI is yes for

$$\left( \bigcup_{i=0..2} \{(P_i, s_i)\} \right) \cup \bigcup_{e \in E} \left( \{(M_e, s_e), (N_e, t_e)\} \cup \bigcup_{f \in Follow(e) \setminus \{e\}} \{(M_{ef}, s_{ef})\} \right). \quad (1)$$

The size of each of these $O(|E|^2)$ specifications is $O(|E|)$. □

**Corollary 5.** *The common implementation problem for $k > 1$ mixed specifications is PSPACE-hard in the size of these specifications.*

*Proof.* This follows from Theorem 3 and the fact that the set of mixed specifications is a superset of the set of modal specifications. □

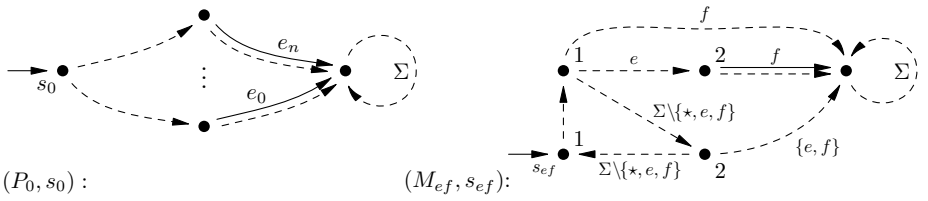



**Fig. 4.** Specifications $(P_0, s_0)$ and $(M_{ef}, s_{ef})$ assuming that $Init = \{e_0, \ldots, e_n\}$

## 5 Consistency

Let us now show that consistency of a single mixed specification is PSPACE-hard in its size. We achieve this by appealing to Theorem 3, and reducing CI for several modal specifications to the C for a single mixed specification.

**Theorem 6.** *Consistency of a mixed specification is PSPACE-hard.*

*Proof.* By Theorem 3, it suffices to show how $k > 1$ mixed specifications $(M_i, s_i)$ can be conjoined into one mixed specification $(M, c_k)$ with $|M|$ being polynomial in $\sum_i |M_i|$ such that $(M, c_k)$ has an implementation iff all $(M_i, s_i)$ have a common implementation.

Figure 5 illustrates the construction, which originates in [7], by showing a conjunction of states $s_1$, $s_2$, $s_3$ up to $s_k$. In order to conjoin two states $s_1$ and $s_2$, two new $\star$-transitions are added from a fresh state $c_2$ to each of $s_1$, $s_2$. One of the $\star$-transitions is a may $\star$-transition and the other is a must $\star$-transition. Only two states can be conjoined directly in this way, but the process can be iterated as many times as needed, as seen in the figure, by adding a corresponding number of $\star$-transitions to the newly conjoined systems. Observe that the resulting specification is properly mixed (not modal). Its size is linear in $\sum_i |M_i|$ and quadratic in $k$, which itself is $O(\sum_i |M_i|)$.

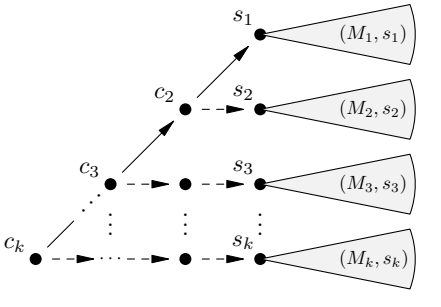



**Fig. 5.** Conjunction of $k$ mixed specifications into one mixed specification

If the specifications that are being conjoined have a common implementation, then the new specification will also have an implementation which is the same implementation prefixed with a sequence of $k-1$ $\star$-transitions. Conversely if the new mixed specification has an implementation, then this implementation will contain at least a sequence of $k-1$ $\star$-transitions, followed by an implementation that must individually satisfy all the systems that have been conjoined. □

## 6 Thorough Refinement

We show PSPACE-hardness of TR for mixed specifications by appeal to Theorem 6 and a reduction of consistency checks to thorough refinement checks.

**Theorem 7.** *Thorough refinement of mixed specifications is PSPACE-hard in the size of these specifications.*

*Proof.* By Theorem 6 deciding C for a mixed specification is PSPACE-hard. Therefore it suffices to reduce C to TR. Let $(M, s)$ be a mixed specification over $\Sigma$. Consider a modal specification $(N, t)$ over $\Sigma \cup \{\star\}$ with $N = (\{t\}, \{\}, \{\})$, which only has a single state and no transitions. From $(M, s)$ construct the mixed specification $(M', s')$ over $\Sigma \cup \{\star\}$ by prefixing $s$ with a new state $s'$ and a single transition $(s', \star, s) \in R^{\Diamond}_{M'} \setminus R^{\Box}_{M'}$. Then $(M', s')$ is a mixed specification that has $(N, t)$ as an implementation, where $Q = \{(s', t)\}$ is the witnessing refinement relation. We show that $(M, s)$ is consistent iff not $(N, t) \prec_{th} (M', s')$.

1° If $(M, s)$ is consistent, then it has an implementation $(L, l)$, from which we get an implementation $(L', l')$ of $(M', s')$ by creating a new state $l'$ with a transition $(l', \star, l)$. But then $(M', s')$ has an implementation that is not allowed by $(N, t)$ and so $I(M', s') \not\subseteq I(N, t)$.

2° Conversely, if $I(M', s') \not\subseteq I(N, t)$ then there exists an implementation $(L, l')$ of $(M', s')$, which is not an implementation of $(N, t)$ – and so $(L, l')$ has a transition $(l', \star, l)$. Moreover $(L, l)$ refines $(M, s)$ since $(L, l')$ refines $(M', s')$ and $s$ is the unique successor of $s'$ in $M'$. Thus $(M, s)$ is consistent.

Remark: Observe that the first argument above would also work for refinement instead of thorough refinement. However we would not be able to get the second implication for refinement, due to its incompleteness. □

Let us now strengthen Theorem 7 to the subclass of *modal* specifications, by a polynomial reduction from the PSPACE-complete decision problem QUANTIFIED 3SAT [19, pp. 171-2] of computing the truth value of closed quantified Boolean formulæ in 3CNF. These formulæ are of the form $Qx_1 \ldots Qx_n . \chi$, where each $Q$ is $\exists$ or $\forall$ and $\chi$ is a propositional formula over $x_1, \ldots, x_n$ in 3CNF. We refer to them as QCNF formulæ in here. We can assume without loss of generality that our formulæ do not contain any clauses with duplicate literals, nor vacuously true clauses. We use $\forall x \, \exists y \, (\neg x \vee y) \wedge (\neg y \vee x)$ as a running example.

We present the semantics of QCNF formulæ in a style that will facilitate our proof. Each formula $\varphi$ can be rewritten into a set of *valuation trees*. The nondeterministic rewrite system for this is depicted in Figure 6. Universal quantification rewrites into branching, existential quantification into a choice, and the 3CNF kernel $\chi$ into the set of variables selected to be true on the path from the tree root to that kernel node. The terminals of this rewrite system for term $(\varphi, \emptyset)$ are valuation trees of $\varphi$. One such valuation tree for the formula $\forall x \, \exists y \, (\neg x \vee y) \wedge (\neg y \vee x)$ can be seen in Figure 7. Each leaf of a valuation tree $T$ contains all those $x_i$ that are true in the respective model for the propositional

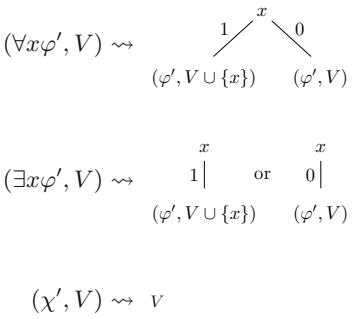



**Fig. 6.** Semantics of QCNF as a non-deterministic rewrite system

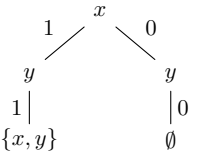



**Fig. 7.** Valuation tree witnessing the truth of $\forall x\, \exists y\, (\neg x \vee y) \wedge (\neg y \vee x)$

kernel formula $\chi$. We define $T \models \varphi$ to mean that all models of leaves of $T$ satisfy the kernel $\chi$ of $\varphi$. Finally, $\varphi$ is defined to be true iff there is a valuation tree $T$ for $\varphi$ such that $T \models \varphi$. For example, $T \models \varphi$ for the valuation tree $T$ in Fig. 7, as the CNF kernel $(\neg x \vee y) \wedge (\neg y \vee x)$ is true in both models $\{x, y\}$ ($x$ and $y$ are true) and $\emptyset$ ($x$ and $y$ are false). Thus, $\varphi$ is true.

In Figure 8 we present a second non-deterministic rewrite system whose terminals are *potential valuation trees*. In this new system there is no path context, existential quantification has two more rewrite rules, and the CNF kernel may rewrite into any subset of its variables. The terminals of this rewrite system are *potential* valuation trees of $\varphi$. By construction, every valuation tree is a potential valuation tree. A potential valuation tree that is not a valuation tree is called a *flawed valuation tree*. Figure 9 shows a valuation tree for our running example with three kinds of flaws: the leftmost $y$ node has no successor, the rightmost $y$ node has two successors, and the leaf set $\{x, y\}$ is inconsistent with the 0 label for $x$ on its path.

Our reduction constructs for any $\varphi$ of QCNF two modal specifications $(N_\varphi, t_\varphi)$ and $(M_\varphi, s_\varphi)$ such that

$$I(N_\varphi, t_\varphi) \subseteq I(M_\varphi, s_\varphi) \qquad \text{iff} \qquad \varphi \text{ is false.} \tag{2}$$

The intuition behind the construction is that $(N_\varphi, t_\varphi)$ models potential valuation trees and $(M_\varphi, s_\varphi)$ models flawed, and only flawed, valuation trees of $\varphi$.

More precisely, these modal specifications are such that any valuation tree $T$ with $T \models \varphi$ can be transformed into an implementation of $(N_\varphi, t_\varphi)$ that is not an implementation of $(M_\varphi, s_\varphi)$ and, conversely, that any element of $I(N_\varphi, t_\varphi) \setminus I(M_\varphi, s_\varphi)$ can be transformed into such a valuation tree $T$ with $T \models \varphi$.

Both models are defined over the following alphabet

$$\Sigma_\varphi = \{\star\} \cup \{v_{x_i}, v_{\neg x_i} \mid 1 \leq i \leq n\} \tag{3}$$

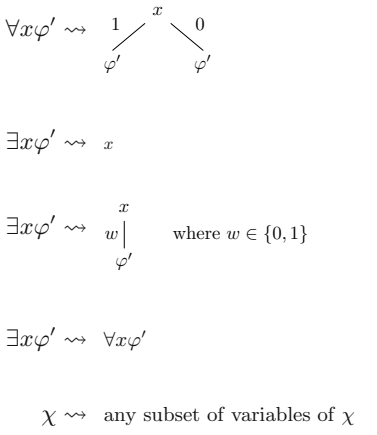



**Fig. 8.** Non-deterministic rewrite system for QCNF deriving *potential* valuation trees

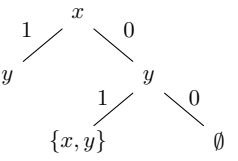



**Fig. 9.** Flawed valuation tree for formula $\forall x\, \exists y\, (\neg x \vee y) \wedge (\neg y \vee x)$

where $x_1, \ldots, x_n$ is the set of variables of $\varphi$.[1] Specification $(N_\varphi, t_\varphi)$ is defined by structural induction on $\varphi$ according to the rules presented in Figure 11.

The initial state $t_\varphi$ has a must $\star$-transition to the continuation of the compilation of $N_\varphi$. Each quantifier $Qx_i$ gets translated into a diamond shaped model of $\star$-transitions, where the upper half consists of must and may transitions for quantifiers $\forall$ and $\exists$ (respectively). The corners of diamonds have "spikes", transitions labeled with a "truth value" $v_{x_i}$ or $v_{\neg x_i}$, for quantifier variable $x_i$, to a dead-end state. After all quantifiers have been compiled in this manner, conjunction is compiled as a fork of two must $\star$-transitions, disjunction as a fork of two may $\star$-transitions, and literals compiled as spikes of truth values. See the result of this compilation for our running example in Figure 12.

Refinement, as defined for modal specifications, does not guarantee that a fork of may $\star$-transitions (present in the compilation of $\exists x_i$ and $\vee$) will implement at least one of these may $\star$-transitions. Also, an implementation may be inconsistent as to its choice of truth values $v_{x_i}$ or $v_{\neg x_i}$. Each path through a sequence of diamonds corresponds to a choice of such truth values, recorded in the respective spike transition. When such a path reaches the compilation of a propositional literal, that literal may well be inconsistent with the spike for that literal encountered en route. In total, these are then the static criteria for corresponding to a flawed valuation tree, and hence drive the construction of specification $(M_\varphi, s_\varphi)$, whose architecture

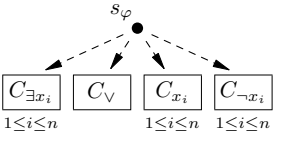



**Fig. 10.** Structure of modal specification $(M_\varphi, s_\varphi)$: $\star$-transitions lead from $s_\varphi$ to components that detect possible flaws in potential valuation trees of $\varphi$

[1] A stronger, albeit more complicated, reduction is possible to TR of specifications over a singleton alphabet. We show the simpler variant here for the sake of clarity.

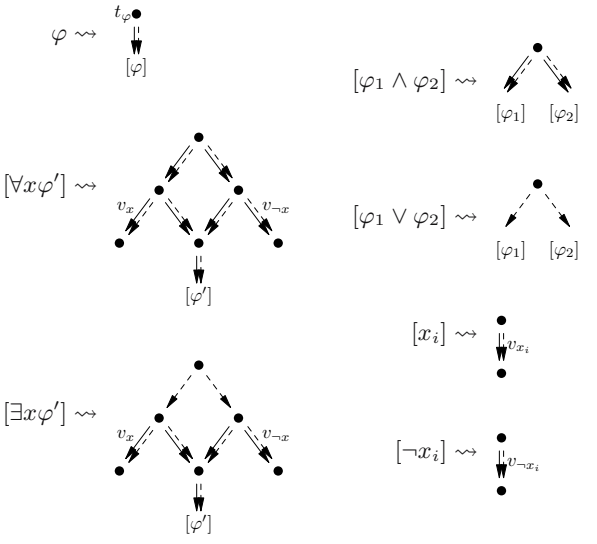



**Fig. 11.** Deterministic rules rewriting a QCNF formula $\varphi$ into a specification $(N_\varphi, t_\varphi)$

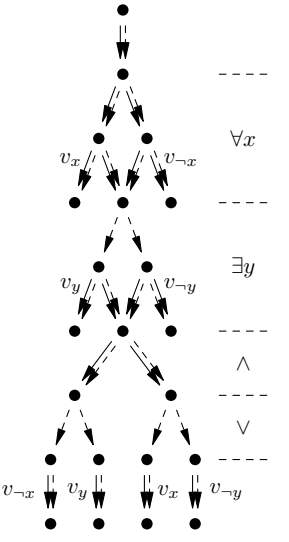



**Fig. 12.** Modal specification $(N_\varphi, t_\varphi)$ for $\varphi = \forall x \, \exists y \, (\neg x \vee y) \wedge (\neg y \vee x)$

is depicted in Fig. 10. Initial state $s_\varphi$ has may $\star$-transitions to modal specifications, components that each encode a potential flaw for a valuation tree. For each variable $x_i$ of $\varphi$ we have a component

- $C_{\exists x_i}$, whose $M_\varphi$-implementations have no "witness" for $\exists x_i$, i.e., no may transitions on the top of the diamond encoding the quantifier
- $C_{x_i}$, whose $M_\varphi$-implementations have a path on which there is some $v_{x_i}$ spike but where, on that same path, a $v_{\neg x_i}$-transition occurs subsequently
- $C_{\neg x_i}$, whose $M_\varphi$-implementations have a path on which there is some $v_{\neg x_i}$ spike but where, on that same path, a $v_{x_i}$-transition occurs subsequently.

Finally there is a component $C_\vee$ whose $M_\varphi$-implementations all have a path of 3n $\star$-transitions to a dead-end state, and so no such implementation can encode all disjunctions of $\varphi$ correctly.

Based on the constructions we can present the following theorem.

**Theorem 8.** *Thorough refinement between modal specifications is PSPACE-hard in the size of these specifications.*

Since the modal transition systems $N_\varphi$ and $M_\varphi$ can be constructed in polynomial time in the size of $\varphi$, it suffices to show that (3) holds.

Note that, by construction, $((\{s_\varphi\}, \emptyset, \emptyset), s_\varphi)$ is an implementation of $(M_\varphi, s_\varphi)$ but not of $(N_\varphi, t_\varphi)$. So the result also applies to strict thorough refinement.

**Corollary 9.** *Strict thorough refinement, whether $I(N,t) \subset I(M,s)$, is PSPACE-hard in $|M|$ and $|N|$ for modal and thus also for mixed specifications.*

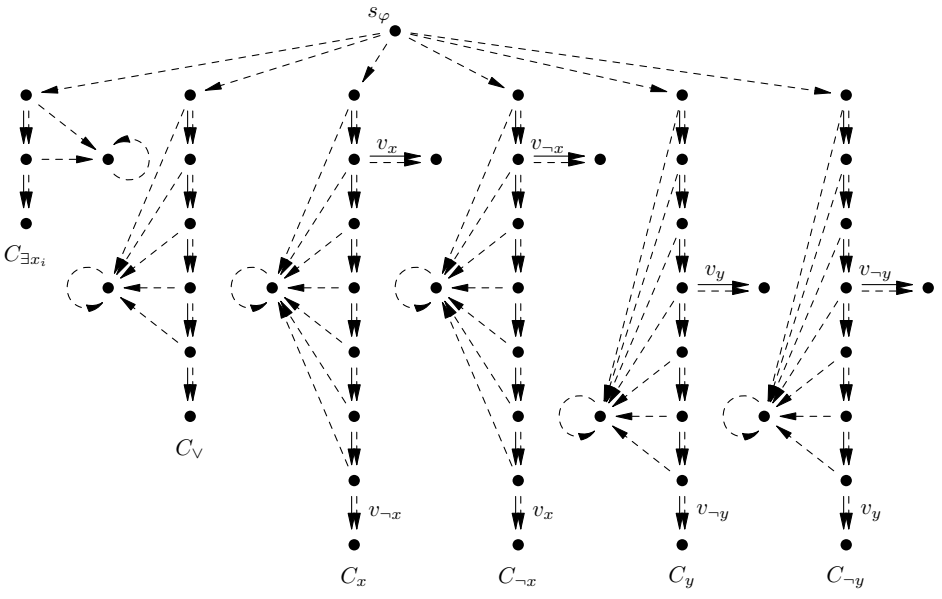



**Fig. 13.** Modal specification $(M_\varphi, s_\varphi)$ for $\varphi = \forall x \exists y\, (\neg x \vee y) \wedge (\neg y \vee x)$. All incoming and outgoing transitions of all loop states are labeled with $\Sigma_\varphi$ (omitted for clarity).

## 7 Discussion

First, we relate our results to the complexity of related problems. Second, we discuss and derive our upper bounds.

In [21] efficient translations are given between various classes of 3-valued models such that these translations preserve and reflect the respective refinement notions. These classes of models are all consistent and one of them subsumes modal transition systems. Therefore our complexity results for common refinement and thorough refinement for modal transition systems transfer to these model classes if we define our three concepts in the same manner for each respective notion of refinement. In particular, our complexity results apply to partial Kripke structures and Kripke modal transition systems.

It is likely that our results extend to "weak" refinement notions that generalize weak bisimulation. This, however, requires a further study. Such refinement notions were systematically studied in [7].

The "conjunction" gadget used in reducing the common implementation problem for modal transition systems to consistency of a mixed transition system (Section 5) is able to identify states uniquely based on the may/must pattern of transitions encountered en route from the initial state. Nominals, used in hybrid logic [22], are a well known mechanism for identifying states uniquely. One can show NP-hardness of the common implementation problem for *two* modal transition systems already if such systems are enriched with nominals [23].

If specifications are "closed under negation" in that $\neg(M,s)$ has the complement of $I(M,s)$ as set of implementations, then thorough refinement reduces to common implementation: $(M,s) \prec_{th} (N,t)$ is false iff $(M,s)$ and $\neg(N,t)$ have a common implementation. From the results in [18] it follows easily that modal transition systems do not have such a negation. Support of negation for specifications should require more structure than that found in mixed transition systems. Another open problem is whether non-empty languages $I(M,s)$ accepted by mixed specifications $(M,s)$ can also be accepted by modal specifications; in other words—if a mixed specification is consistent, is it refinement-equivalent to a modal specification?

Generalized model checking [24] considers judgments $\mathsf{GMC}(M,s,\varphi)$ which are true iff there is an implementation of $(M,s)$ that satisfies $\varphi$. For pointed modal specifications $(M,s)$ and Hennessy-Milner formulae $\varphi$ this is PSPACE-complete in the size of $\varphi$ [24,21]. For each such $\varphi$ there are $1 \leq m < \infty$ pointed modal specifications $(M_i,s_i)$ such that $\mathsf{GMC}(M,s,\varphi)$ is false iff $I(M,s) \subseteq \bigcup_{i=1}^{m} I(M_i,s_i)$ [18]. Intuitively, the union on the right-hand side is the set of implementations that satisfy $\neg\varphi$. In general, $m > 1$ so there seems to be no natural and direct reduction of generalized model checking to thorough refinement. For $\varphi$ in CTL, $\mathsf{GMC}(M,s,\varphi)$ is EXPTIME-complete [24,21] but $1 < m$ or $m = \infty$ may hold.

We finally discuss what upper bounds we can provide for the decision problems presented in this paper. Mixed and modal specifications $(M,s)$ have characteristic formulæ $\Psi_{(M,s)}$ [18] in the modal $\mu$-calculus such that pointed labeled transition systems $(L,l)$ are implementations of $(M,s)$ iff $(L,l)$ satisfies $\Psi_{(M,s)}$. The common implementation and consistency problem reduce to satisfiability checks of $\bigwedge_i \Psi_{(M_i,s_i)}$ and $\Psi_{(M,s)}$, respectively. The thorough refinement problem of whether $(M,s) \prec_{th} (N,t)$ reduces to a validity check of $\neg\Psi_{(N,t)} \vee \Psi_{(M,s)}$.

Validity checking of such vectorized modal $\mu$-calculus formulæ is in EXPTIME (an unpublished popular wisdom, for which we give a formal argument here). One way in which this membership in EXPTIME can be seen is by translating the problem into alternating tree automata. It is well known that formulæ $\Psi_{(M,s)}$ can be efficiently translated [25] into alternating tree automata $\mathsf{A}_{(M,s)}$ (with parity acceptance condition) that accept exactly those pointed labeled transition systems that satisfy $\Psi_{(M,s)}$. Since non-emptiness, intersection, and complementation of languages is in EXPTIME for alternating tree automata, we get our EXPTIME upper bounds if these automata have size polynomial in $|M|$. Since the size of $\Psi_{(M,s)}$ may be exponential in $|M|$ we require a direct translation from $(M,s)$ into a version of $\mathsf{A}_{(M,s)}$. The formulæ $\Psi_{(M,s)}$ can be written as a system of recursive equations [4] $X_s = body_s$ for each state $s$ of $M$. We can therefore construct all $\mathsf{A}_{(M,s)}$ in a compositional manner: whenever $X_s$ refers in its $body_s$ to some $X_t$, then $\mathsf{A}_{(M,s)}$ has a transition to the initial state of $\mathsf{A}_{(M,t)}$ at that point. This $\mathsf{A}_{(M,s)}$ generates the same language as the one constructed from $\Psi_{(M,s)}$, by appeal to the existence of memoryless winning strategies in parity games. The system of equations is polynomial in $|M|$, and so the compositional version of $\mathsf{A}_{(M,s)}$ is polynomial in the size of that system of equations. We summarize:

**Table 1.** Tabular summary of the results provided in this paper

| | Modal specifications | Mixed specifications |
|---|---|---|
| Common implementation | PSPACE-hard, EXPTIME | PSPACE-hard, EXPTIME |
| Consistency | trivial | PSPACE-hard, EXPTIME |
| Thorough refinement | PSPACE-hard, EXPTIME | PSPACE-hard, EXPTIME |

**Theorem 10.** *The common implementation, consistency and thorough refinement problems are all in EXPTIME for modal and mixed specifications.*

## 8 Conclusion

We studied modal and mixed specifications and their fundamental decision problems: consistency (a form of realizability), common implementations (a conjunctive form of consistency), and thorough refinement (a form of implication) of specifications. We established that all these decision problems are in EXPTIME and PSPACE-hard for mixed as well as for modal specifications – keeping in mind that all modal specifications are consistent by construction. These results showed that some of these decision problems are at least as hard as others studied here. This raises the question of whether they in fact have the same complexity.

**Acknowledgments.** Harald Fecher made us aware of the counterexample for incompleteness of refinement used in this paper. This then led to the rediscovery of a history of such counterexamples. Nir Piterman helped in improving the presentation of the proof for Theorem 8. We thank Igor Walukiewicz, Wolfgang Thomas and Dietmar Berwanger for independently confirming that validity of vectorized $\mu$-calculus formulæ is in EXPTIME. The referees' comments helped with improving the presentation of this paper.

## References

1. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
2. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, Springer, Heidelberg (1981)
3. Larsen, K.G., Thomsen, B.: A modal process logic. In: Third Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 203–210. IEEE Computer Society, Los Alamitos (1988)
4. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
5. Dams, D.: Abstract Interpretation and Partition Refinement for Model Checking. PhD thesis, Eindhoven University of Technology (July 1996)

6. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Trans. Program. Lang. Syst. 19(2), 253–291 (1997)
7. Larsen, K.G., Nyman, U., Wąsowski, A.: On modal refinement and consistency. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 105–119. Springer, Heidelberg (2007)
8. Hussain, A., Huth, M.: On model checking multiple hybrid views. Technical report, Department of Computer Science, University of Cyprus, TR-2004-6 (2004)
9. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: Fifth Annual IEEE Symposium on Logics in Computer Science (LICS), Philadelphia, PA, USA, June 4–7, 1990, pp. 108–117 (1990)
10. Hussain, A., Huth, M.: Automata games for multiple-model checking. Electr. Notes Theor. Comput. Sci. 155, 401–421 (2006)
11. Fischbein, D., Uchitel, S., Braberman, V.: A foundation for behavioural conformance in software product line architectures. In: ROSATEA 2006 Proceedings, pp. 39–48. ACM Press, New York (2006)
12. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
13. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
14. Park, D.: Concurrency and automata on infinite sequences. In: Proceedings of the 5th GI-Conference on Theoretical Computer Science, pp. 167–183. Springer, London, UK (1981)
15. Hüttel, H.: Operational and denotational properties of modal process logic. Master's thesis, Computer Science Department. Aalborg University (1988)
16. Xinxin, L.: Specification and Decomposition in Concurrency. PhD thesis, Department of Mathematics and Computer Science, Aalborg University (April 1992)
17. Schmidt, H., Fecher, H.: Comparing disjunctive modal transition systems with a one-selecting variant. Submitted for publication to JLAP (2007)
18. Huth, M.: Labelled transition systems as a Stone space. Logical Methods in Computer Science 1(1), 1–28 (2005)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
20. Jonsson, B., Larsen, K.G.: On the complexity of equation solving in process algebra. In: Abramsky, S., Maibaum, T.S.E. (eds.) TAPSOFT 1991. LNCS, vol. 493, pp. 381–396. Springer, Heidelberg (1991)
21. Godefroid, P., Jagadeesan, R.: On the expressiveness of 3-valued models. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) VMCAI 2003. LNCS, vol. 2575, pp. 206–222. Springer, Heidelberg (2002)
22. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). J. Applied Logic 4(3), 279–304 (2006)
23. Antonik, A.: MPhil/PhD transfer report. Imperial College London, United Kingdom (January 2007)
24. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 168–182. Springer, Heidelberg (2000)
25. Wilke, T.: Alternating tree automata, parity games, and modal $\mu$-calculus. Bull. Soc. Math. Belg. 8(2) (May 2001)

# Classes of Tree Homomorphisms with Decidable Preservation of Regularity*

Guillem Godoy[1], Sebastian Maneth[2], and Sophie Tison[3]

[1] Universitat Politècnica de Catalunya (UPC)
Jordi Girona 1, Barcelona, Spain
ggodoy@lsi.upc.edu
[2] NICTA and UNSW, Sydney, Australia
sebastian.maneth@nicta.com.au
[3] Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq Cedex, France
sophie.tison@lifl.fr

**Abstract.** Decidability of regularity preservation by a homomorphism is a well known open problem for regular tree languages. Two interesting subclasses of this problem are considered: first, it is proved that regularity preservation is decidable in polynomial time when the domain language is constructed over a monadic signature, i.e., over a signature where all symbols have arity 0 or 1. Second, decidability is proved for the case where non-linearity of the homomorphism is restricted to the root node (or nodes of bounded depth) of any input term. The latter result is obtained by proving decidability of this problem: Given a set of terms with regular constraints on the variables, is its set of ground instances regular? This extends previous results where regular constraints where not considered.

## 1 Introduction

Representations of sets of terms are used in many areas of computer science. The choice of formalism depends on the expressiveness, but also on the properties from a computational point of view. Tree automata [3,7] are a well studied formalism for representing term languages. They are the natural extension of standard finite automata over words to tree/term languages. For example, the tree automaton

$$\begin{array}{rl} a \to q_a & \\ g(q_a) \to q_g & \qquad g(q_g) \to q_g \\ f(q_g, q_f) \to q_{accept} & \qquad f(q_a, q_a) \to q_f \end{array}$$

recognizes the language $f(g^+(a), f(a, a))$. The languages recognized by tree automata are also called regular. They are a classical concept which has been used in many contexts: for instance, they adequately describe the parse trees of a context-free grammar or the well-formed terms over a sorted signature, and they naturally capture type formalisms for tree-structured (XML) data [14,9]. Similar as in the case of regular sets of

* The first author was supported by Spanish Min. of Educ. and Science by the LogicTools project (TIN2004-03382), and by the FORMALISM project (TIN2007-66523).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 127–141, 2008.

words, the class of regular term languages has many convenient properties such as closure under boolean operations (intersection, union, negation), decidable properties such as inclusion and equivalence, and they are characterized by many different formalisms such as regular grammars, regular term expressions, congruence classes of finite index, deterministic bottom-up finite tree automata, nondeterministic top-down finite tree automata, sentences of monadic second-order logic, etc, cf. [3,7]. Deterministic tree automata, for instance, can effectively be minimized and give rise to efficient parsing procedures.

Nevertheless, the expressiveness of regular tree languages is considerably limited: simple languages like the recursively defined set $T_{\mathtt{bin}} = \{a\} \cup \{f(t,t) \mid t \in T_{\mathtt{bin}}\}$, i.e., the set of complete trees over $\{f, a\}$ where $f$ is a binary symbol and $a$ is a constant, is not regular. Accordingly, it is often convenient to use a more general formalism to describe a set of terms, loosing then some of the nice computational properties. In this setting, it makes sense to study the decidability of the regularity of a given set of terms represented by a concrete (and more expressive) formalism. Practically speaking, we want to allow the specification of term languages in a more general formalism than regular term languages, but want to be able to detect if a specific given language is in fact regular. Deciding a class of languages inside a larger class is a very hard problem; for instance, given a context-free (word) language it follows from Greibach's Theorem that is impossible to decide whether or not the language is in fact regular (see, e.g., [8]).

In this paper we study the decidability of regularity for two particular and related representations: when the set of terms is described as the image of a regular set by a tree homomorphism, and when it is described as the ground instances of a finite set of terms with regular constraints on the variables.

Tree homomorphisms are the natural extension of the usual word homomorphisms to trees. A homomorphism $H$ is usually defined by associating to each symbol $g$ in the input signature a term $H(g)$ with variables $x_1, x_2, \ldots, x_k$, where $k$ is the arity of $g$. This definition is then homomorphically extended to any term of the signature. For example, the homomorphism $\{H(g) = f(x_1, x_1), H(a) = a\}$ applied to the language $g^*(a) = \{a, g(a), g(g(a)), \ldots\}$ generates as image the language $T_{\mathtt{bin}}$ of above. This example shows that the image of a regular language by a homomorphism is in general *not* regular. In fact, it is a long-standing open problem whether or not it is decidable if the homomorphic image of a regular set of terms is regular ("the HOM-problem"); cf., e.g., Conclusions of [6]. Some particular cases are known to be decidable, for instance when the homomorphism is linear, see [5], or when in the terms $H(g)$, for all input symbols $g$, multiple occurrences of the same variable all have the same parent node (i.e., are siblings of each other) [2]. If slightly more expressive tree translation devices than homomorphisms are considered, then regularity preservation quickly becomes undecidable: for a deterministic top-down tree transducer with only two states, it is undecidable whether its image of a regular tree language is again regular [6] (homomorphisms naturally correspond to the 1-state case of top-down or bottom-up tree transducers). Let us also note that the problem of regularity of a set of ground normal forms of a term rewriting system is known to be decidable but with rather intricate proofs [11], while it can be reduced to a particular subclass of the HOM-problem.

We solve the HOM-problem for two new interesting subcases: first, we show decidability in polynomial time for the case where the domain language is constructed over a monadic signature, i.e., over a signature where all symbols have arity 0 or 1. This is obtained by characterizing regularity using a pumping argument, and then reducing the regularity check to certain finiteness tests on images of the homomorphism. Second, decidability is proved for the case where non-linearity of the homomorphism is restricted to the root node of any input term (or, more precisely, to input nodes of bounded depth). The latter result is obtained by proving decidability of the following problem: Given a set of terms with regular constraints on the variables, is its set of ground instances regular? This extends previous results where regular constraints were not considered, see, e.g., [10]. For this case, we give a sufficient condition for non-regularity by the existence of certain infinite solutions of a formula. The formula is of first-order logic with equality predicates and membership constraints in regular languages, altogether interpreted over a term algebra. Our algorithm for this case makes iterated use of the decidability for this sufficient condition, which follows from a result by Comon and Delor [4].

## 2 Preliminaries

**Convention.** In this article we use the words "term" and "tree" interchangeably, and the same holds for "position" and "node".

A *signature* consists of an alphabet $\Sigma$, i.e., a finite set, together with a mapping that assigns to each symbol in $\Sigma$ a natural number, its *arity*. We write $\Sigma^{(k)}$ to denote the subset of symbols in $\Sigma$ that are of arity $k$, and we write $\sigma^{(k)}$ to denote that $\sigma$ is a symbol of arity $k$. The *set of all terms over* $\Sigma$ is denoted $T_\Sigma$ and is inductively defined as the smallest set $T$ such that for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \ldots, t_k \in T$, the term $\sigma(t_1, \ldots, t_k)$ is in $T$. For a term of the form $a()$ we simply write $a$. For instance, if $\Sigma = \{f^{(2)}, a^{(0)}\}$ then $T_\Sigma$ is the set of terms that represent all binary trees with internal nodes labeled $f$ and leaves labeled $a$. We fix the set $X = \{x_1, x_2, \ldots\}$ of variables. The set of trees over $\Sigma$ with variables in $X$, denoted $T_\Sigma(X)$, is the set of terms over $\Sigma \cup X$ where every symbol in $X$ has arity zero. Given a tree $\sigma(t_1, \ldots, t_k) \in T_\Sigma$, its set of positions $\mathrm{Pos}(t)$ equals $\{\varepsilon\} \cup_{1 \leq i \leq k} \{i.p \mid p \in \mathrm{Pos}(t_i)\}$. Thus, $\varepsilon$ denotes the root node, and $p.i$ denotes the $i$th child of position $p$. The subtree of $t$ at position $p$ is denoted by $t/p$, and the symbol of $t$ at position $p$ is denoted by $t[p]$. For instance, for $s = \sigma(f(a, b), c)$, $s/1$ equals $f(a, b)$ and $s[1]$ equals $f$. For a signature $\Gamma$, we use $\mathrm{Pos}_\Gamma(t)$ to denote the set of positions of $t$ that are labeled by symbols in $\Gamma$. E.g., for $s$ of above, $\mathrm{Pos}_{\{c\}}(s) = \{2\}$ and $\mathrm{Pos}_X(s) = \varnothing$. For terms $s, t$ and $p \in \mathtt{Pos}(s)$, we denote by $s[p \leftarrow t]$ the result of replacing the subtree at position $p$ in $s$ by the term $t$. For instance, $f(f(a, a), a)[1 \leftarrow a] = f(a, a)$.

A (deterministic) bottom-up tree automaton (for short, DTA) is a tuple $A = (Q, Q_a, \Sigma, \delta)$ where $Q$ is a finite set of states, $Q_a \subseteq Q$ is the set of accepting states, $\Sigma$ is a signature, and $\delta = (\delta_\sigma)_{\sigma \in \Sigma}$ is a collection of transition functions such that for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $\delta_\sigma$ is a function from $Q^k$ to $Q$. The language $L(A)$ recognized by $A$ is the set $\{t \in T_\Sigma \mid A(t) \in Q_a\}$ where $A$ is the extension of $\delta_\sigma$ to trees in $T_\Sigma$, recursively defined as: $A(\sigma(t_1, \ldots, t_k)) = \delta_\sigma(A(t_1), \ldots, A(t_k))$ for $\sigma \in \Sigma^{(k)}$, $k \geq 0$,

and $t_1, \ldots, t_k \in T_\Sigma$. We also define, for a state $q$, the set $L(A, q) = \{t \in T_\Sigma \mid A(t) = q\}$ of trees for which $A$ arrives in state $q$. A term language $L \subseteq T_\Sigma$ is *regular* if there exists a DTA $A$ such that $L = L(A)$.

**Tree Homomorphisms.** Let $\Sigma$ and $\Delta$ be signatures. A *homomorphism (from $\Sigma$ to $\Delta$)* is a mapping $H$ that associates to every symbol $\sigma \in \Sigma$ of arity $k$ a tree in $T_\Delta(\{x_1, \ldots, x_k\})$. The homomorphism $H$ is extended to trees over $\Sigma$ by defining $H(\sigma(s_1, \ldots, s_k)) = H(\sigma)[x_1 \leftarrow H(s_1), \ldots x_k \leftarrow H(s_k)]$. The term $t[x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k]$ denotes the *substitution* of every occurrence of $x_i$ in $t$ by the tree $t_i$. The homomorphism $H$ is *linear* if $H(\sigma)$ is linear in $\{x_1, \ldots, x_k\}$ for each $\sigma \in \Sigma$ of arity $k$, i.e., if each $x_i$, $1 \leq i \leq k$ occurs at most once in $H(\sigma)$. We will later make use of the following known result, cf. Corollary 3.10 of [5].

**Proposition 1.** *Let $H$ be a linear homomorphism and $L$ a regular tree language. Then $H(L)$ is effectively regular.*

**Sets of Terms with Regular Constraints.** Let $V = \{x_1, \ldots, x_n\}$. A regular *constraint* for variables $x_1, \ldots, x_n$ maps each $x_i$ in $V$ to a regular tree language. Let $\Gamma$ be a signature. A *solution* of $C$ (over $\Gamma$) is a ground substitution $\varphi : V \to T_\Gamma$ such that $\varphi(x_i) \in C(x_i)$ for all $i \in \{1, \ldots, n\}$. A set of terms with regular constraints is a pair $\langle S, C\rangle$ where $S$ is a set of terms (with variables) and $C$ is a constraint defined at least for the variables occurring in $S$. The language of $\langle S, C\rangle$, denoted as $L(\langle S, C\rangle)$, is defined as $\{t \mid \exists \varphi, s : (t = \varphi(s) \wedge s \in S \wedge \varphi \text{ is a solution of } C)\}$. The following result is due to [12], cf. also [10].

**Proposition 2.** *Let $\mathcal{C}$ be the trivial constraint mapping every variable to the set of all terms. Then, regularity of $L(\langle S, \mathcal{C}\rangle)$ is decidable for given $S$.*

## 3 The Monadic Case

We consider the particular case where the regular input term language is constructed over a *monadic* signature with only unary symbols and constants and prove that regularity of the image by a homomorphism is decidable in polynomial time.

Let $\Sigma$ be a monadic signature, i.e., $\Sigma = \Sigma^{(1)} \cup \Sigma^{(0)}$. Let $L$ be a regular term language over $\Sigma$, and $H$ be a homomorphism. For simplicity we assume that the signature for the domain language $L$ contains just one constant $c$, and moreover, $H(c) = c$. If this was not the case we can easily transform $L$ and $H$ into new $L'$ and $H'$ such that $L'$ satisfies this requirement and $H'(L') = H(L)$: we simply introduce a new constant $c$ not present in $\Sigma$ and define $L'$ as $H_{\mathtt{lin}}(L)$ where $H_{\mathtt{lin}}$ is the linear homomorphism with $H_{\mathtt{lin}}(f) = f(x_1)$ for all unary functions $f$, and $H_{\mathtt{lin}}(e) = e(c)$ for constants $e$. We define $H'$ as $H'(f) = H(f)$ for unary function symbols $f$, $H'(e) = H(e)$ for constants $e$ of the original signature of $L$, and $H'(c) = c$. Note that constants $e$ of the original signature of $L$ are now unary function symbols of the signature of $L'$.

For the current case the language $L$ is essentially a language on words. We use then expressions like $abbaa(c)$ or just $abbaa$ for denoting $a(b(b(a(a(c)))))$. Note that $\Sigma = \Sigma^{(1)} \cup \{c^{(0)}\}$.

A unary symbol $a$ is called *erasing* if $H(a(x)) = x$, and by $\lfloor w \rfloor$ we denote the number of positions of $w$ with non-erasing symbols. Intuitively, we are counting the number of symbols that generate at least one output node in the image; this means that the size of the term $H(w)$ is larger than or equal to $\lfloor w \rfloor$.

**Definition 3.** *Let $\Sigma$ be a monadic signature, $H$ a homomorphism over $\Sigma$, and $w \in T_\Sigma$. The* level *of a position $p$ with respect to $w$ and $H$, denoted by level$(p, w, H)$, is $k+1$ if there exists a factoring of the form $w = w_1 a w_2$ with $\lfloor w_1 \rfloor = k$ such that $p \notin \mathtt{Pos}(H(w_1))$ but $p \in \mathtt{Pos}(H(w_1 a))$.*

Intuitively, $\mathtt{level}(p, w, H) = k$ means that $p$ in $H(w)$ was generated by the $k$-th non-erasing symbol in the word $w$. For instance, if $H(g) = f(x_1, x_1)$ and $H(c) = c$ then $\mathtt{level}(\varepsilon, ggg(c), H) = 1$ because $ggg = w_1 g w_2$ with $w_1$ the empty word, $\varepsilon \notin \mathtt{Pos}(H(w_1)) = \emptyset$, and $\varepsilon \in \mathtt{Pos}(H(g))$. If $H$ is also defined as $H(d) = x_1$, then $\mathtt{level}(1.2, gdgg(c), H) = 3$ because the node 1.2 in the term $H(gdgg(c)) = f(f(f(c,c), f(c,c)), f(f(c,c), f(c,c)))$ was generated by the third non-erasing symbol of $gdgg(c)$, i.e., by the last $g$. The following lemma is straightforward from this definition.

**Lemma 4.** *Let $p, q$ be positions such that $q$ is a prefix of $p$. If $\mathtt{level}(p, w, H) \geq k + \mathtt{level}(q, w, H)$, then $|p| \geq k + |q|$.*

**Definition 5.** *A position $p$ is* live *in $\langle w, H \rangle$ if there exists $p'$ such that $\mathtt{level}(p.p', w, H) > \mathtt{level}(p, w, H)$.*

We define $h := \mathtt{Max}_{a \in \Sigma^{(1)}}(\mathtt{height}(H(a)))$. The following two lemmas are straightforward from the previous definitions.

**Lemma 6.** *Let $p$ be a position and $w$ a word such that $\mathtt{height}(H(w)/p) > h$. Then, $p$ is live in $\langle w, H \rangle$.*

**Lemma 7.** *Let $p$ and $q$ be positions and $w$ a word such that, $p$ and $q$ are live in $\langle w, H \rangle$, and $\mathtt{level}(p, w, H) = k + \mathtt{level}(q, w, H)$. Then, $|\mathtt{height}(H(w)/p) - \mathtt{height}(H(w)/q)| \leq h \cdot (k+1)$.*

The following lemma is essential for doing a pumping argument in the lemma after, which characterizes the regularity of the image of a homomorphism for the monadic case.

**Lemma 8.** *Let $t$ be a term in $H(L)$, let $p, p.q_1, p.q_2$ be positions in $\mathtt{Pos}(t)$ such that $|q_1|, |q_2| \leq h$ and $\mathtt{height}(t/p.q_1), \mathtt{height}(t/p.q_2) > h$. Then $|\mathtt{height}(t/p.q_1) - \mathtt{height}\ (t/p.q_2)| \leq h(h+1)$.*

*Proof.* By our assumptions, $|p.q_1| - |p| \leq h$ and $|p.q_2| - |p| \leq h$. By Lemma 4, $\mathtt{level}\ (p.q_1, w, H) = k_1 + \mathtt{level}(p, w, H)$ and $\mathtt{level}(p.q_2, w, H) = k_2 + \mathtt{level}(p, w, H)$ for some $k_1, k_2 \leq h$. Therefore, either $\mathtt{level}(p.q_1, w, H) = k + \mathtt{level}(p.q_2, w, H)$ or $\mathtt{level}(p.q_2, w, H) = k + \mathtt{level}(p.q_1, w, H)$ for some $k \leq h$. By Lemma 6, $p.q_1$ and $p.q_2$ are live in $\langle w, H \rangle$. Finally, by Lemma 7, $|\mathtt{height}(H(w)/p.q_1) - \mathtt{height}\ (H(w)/p.q_2)| \leq h(k+1) \leq h(h+1)$. □

**Definition 9.** *We say that a symbol $a \in \Sigma^{(1)}$ is* deleting *in $H$, or that $H$ is deleting on $a$, if $H(a)$ does not contain $x_1$. A word $w$ is deleting (or "$H$ is deleting on $w$") if it contains a deleting symbol. Let $a \in \Sigma^{(1)}$. We define $L_{a\_} := \{w \mid \exists u : (uaw \in L \wedge u \text{ is non-deleting})\}$.*

We are now ready to prove the main result of this section. It characterizes regularity of the homomorphic image of a monadic tree language. A symbol $a \in \Sigma^{(1)}$ is *copying in* $H$, or $H$ is *copying on* $a$, if $H(a)$ contains at least two occurrences $x_1$. A word $w$ is *copying* (or "$H$ is copying on $w$") if it contains a copying symbol.

**Lemma 10.** *$H(L)$ is not regular iff there exists $a \in \Sigma^{(1)}$ such that $H$ is copying on $a$ and $H(L_{a\_})$ is infinite.*

*Proof.* We first prove the *if*-direction. Let $a \in \Sigma^{(1)}$ such that $H$ is copying on $a$ and $H(L_{a\_})$ is infinite. We assume that $H(L)$ is regular in order to reach a contradiction. Then, there exists a bottom-up tree automaton $A$ recognizing $H(L)$. Recall from the Preliminaries that $A(t)$ denotes the state in which $A$ arrives after processing the term $t$. From $L_{a\_}$ we choose two words $w_1$ and $w_2$ such that $\mathtt{height}(H(w_1)), \mathtt{height}(H(w_2)) > h$, $|\mathtt{height}(H(w_1)) - \mathtt{height}(H(w_2))| > h(h+1)$, and $A(H(w_1)) = A(H(w_2))$; this is possible by non-finiteness of $H(L_{a\_})$. Now, since $w_1$ belongs to $L_{a\_}$, there exists a word of the form $uaw_1$ in $L$ where $u$ is non-deleting. Therefore, since $a$ is copying, $H(uaw_1)$ is of the form $s[p \leftarrow t'[q_1 \leftarrow H(w_1)][q_2 \leftarrow H(w_1)]]$ for positions $p$, $q_1$, $q_2$ satisfying $|q_1|, |q_2| \leq h$ and $q_1$ is "disjoint" from $q_2$ ($q_1$ is not a prefix of $q_2$, and $q_2$ is not a prefix of $q_1$). Recall from the Preliminaries that $u[q \leftarrow v]$ denotes the result of replacing position $q$ in the term $u$ by the term $v$. The term $t = s[p \leftarrow t'[q_1 \leftarrow H(w_1)][q_2 \leftarrow H(w_2)]]$ is also in $H(L)$ because $A(H(w_1)) = A(H(w_2))$. By our previous conditions we have $\mathtt{height}(t/p.q_1), \mathtt{height}(t/p.q_2) > h$, and $|\mathtt{height}(t/p.q_1) - \mathtt{height}(t/p.q_2)| > h(h+1)$. This is a contradiction to Lemma 8.

Next, we prove the *only-if*-direction. Assume that for all $a \in \Sigma^{(1)}$ such that $H$ is copying on $a$, $H(L_{a\_})$ is finite. We define $L_a = aL_{a\_}$, i.e., $L_a = \{aw \mid \exists u : (uaw \in L \text{ and } u \text{ is non-deleting})\}$, for every $a \in \Sigma^{(1)}$, and also define $T = \{t \mid \exists a \in \Sigma^{(1)} : (H \text{ is copying on } a \wedge t \in H(L_a))\}$, which is a finite set. Further, we define a new set of constants $C = \{c_t \mid t \in T\}$.

Our goal is to define an alternative language $L'$ and a homomorphism $H'$ satisfying $H'(L') = H(L)$, and for which it is easy to see that $H'(L')$ is regular. We first define some particular subsets of $L'$.

$$
\begin{aligned}
L_{\mathtt{cop}} &= \{uc_t \mid H \text{ is neither copying nor deleting on } u \wedge \\
&\qquad \exists a, w : ((H \text{ is copying on } a) \wedge uaw \in L \ \wedge \ H(aw) = t)\} \\
L_{\mathtt{del}} &= \{ua \mid H \text{ is neither copying nor deleting on } u \wedge \\
&\qquad H \text{ is deleting on } a \ \wedge \ \exists w : (uaw \in L)\} \\
L_{\mathtt{oth}} &= \{u \mid H \text{ is neither copying nor deleting on } u \ \wedge \ u \in L\}
\end{aligned}
$$

Finally we define $L' = L_{\mathtt{cop}} \cup L_{\mathtt{del}} \cup L_{\mathtt{oth}}$, and $H'$ is defined like $H$ for the symbols $a \in \Sigma$ that are not copying, $H'$ is undefined for the $a \in \Sigma^{(1)}$ that are copying, and $H'(c_t) = t$ for every $t \in T$. Note that $L'$ does not have words containing symbols such that $H$ is copying on them.

We show first that $H'(L') = H(L)$. The inclusion $\subseteq$ is straightforward from the definition of $L'$ and $H'$. For $\supseteq$ let $v$ be a word of $L$. Either $v$ does not contain any copying or deleting symbol, or it is of the form $uaw$ where $u$ does not contain any copying or deleting symbols, and $a$ is either copying or deleting. Depending on the case, it is easily seen that either $v \in H'(L_{\mathtt{oth}})$ or $v \in H'(L_{\mathtt{cop}})$ or $v \in H'(L_{\mathtt{del}})$.

Second, we see that $L'$ is regular. From an automaton $A$ recognizing $L$ we can easily define an automata for $L_{\mathtt{cop}}$, $L_{\mathtt{del}}$ and $L_{\mathtt{oth}}$, respectively. Without lost of generality we assume that all states of $A$ can reach an accepting state. For the case of $L_{\mathtt{oth}}$ it suffices to remove from $A$ the transitions with copying and deleting symbols. For the case of $L_{\mathtt{del}}$ it suffices to remove the transitions with copying symbols and to redirect all the transitions with deleting symbols to a new accepting state. For the case of $L_{\mathtt{cop}}$ we have to remove all transitions of deleting symbols and all transitions of constant symbols. We also need to consider the transitions with a copying symbol $a$ from a state $q$ to a state $q'$, such that $q$ is reachable and $q'$ can reach an accepting state through non-deleting and non-copying symbols. Every of these transitions has to be replaced by several new transitions to the state $q'$ one for every constant $c_t$ such that $t \in H(a\, L(A, q'))$. Finally, from the regularity of $L'$ and the fact that $H'$ is linear we conclude by Proposition 1 that $H'(L') = H(L)$ is regular. □

Before we use Lemma 10 to show decidability of regularity of $H(L)$ for monadic $L$, let us give an example application of the only-if direction of the previous lemma.

*Example 11.* Let $\Sigma = \{a^{(1)}, b^{(1)}, d^{(1)}, c^{(0)}\}$ and, for $n \geq 0$, $L_n = \{d(a|b)^k(c) \mid k \leq n\}$. The language $L_n$ is recognized by the automaton $A_n = (\{q_0, q_1, \ldots, q_n, q_f\}, \{q_f\}, \Sigma, \delta)$ where $\delta_c() = q_0$, $\delta_a(q_0) = \delta_b(q_0) = q_1$, $\delta_a(q_1) = \delta_b(q_1) = q_2$, …, $\delta_a(q_{n-1}) = \delta_b(q_{n-1}) = q_n$, and $\delta_d(q_i) = q_f$ for all $0 \leq i \leq n$. The tree homomorphism $H$ is defined by $H(a) = a(x_1)$, $H(b) = b(x_1)$, $H(d) = f(x_1, x_1)$, and $H(c) = c$. Thus, $H(L_n) = \{f(w, w) \mid w \in (a|b)^k c, k \leq n\}$. Since $H(L_n)$ is finite, it is obviously regular. Let us now follow the only-if direction of the proof of Lemma 10 in order to construct a tree automaton $B_n$ which recognizes $H(L_n)$. Note that $L_{d\_} = \{w \in (a|b)^k c \mid k \leq n\}$. The set $T$ defined in the proof equals $H(\{dw \mid w \in L_{d\_}\}) = \{f(w, w) \mid w \in L_{d\_}\}$. Thus, for every possible $w \in \{a, b\}^*$ of length at most $n$, the set $T$ contains the tree $t = f(w, w)$, and accordingly, $C$ contains the constant $c_t$. For instance, if $n = 2$, then $C = \{c_{f(c,c)}, c_{f(a(c),a(c))}, c_{f(b(c),b(c))}, c_{f(aa(c),aa(c))}, c_{f(ab(c),ab(c))}, c_{f(ba(c),ba(c))}, c_{f(bb(c),bb(c))}\}$.

Now, to define $L_{\mathtt{oth}}$ we simply remove from $A_n$ all $d$-transitions because $d$ is copying. Since the resulting automaton has no transitions to accepting states, its language is empty, i.e., $L_{\mathtt{oth}} = \varnothing$. Also $L_{\mathtt{del}} = \varnothing$ because $d$-transition are removed from $A_n$ and no new transitions are added because there are no deleting symbols. The automaton for $L_{\mathtt{cop}}$ is obtained from $A_n$ by deleting the transition $\delta_c() = q_0$ and by replacing each transition $\delta_d(q_i) = q_f$, $0 \leq i \leq n$ by the new transitions $\delta_{c_t}() = q_f$ for each $c_t \in C$. Thus, the resulting automaton $B_n$ recognizes precisely the set $C$. Since for every $t \in H(L_n)$ the constant $c_t$ is in $C$, and the new homomorphism $H'$ has $H'(c_t) = t$ for all $c_t \in C$, we obtain that $H'(L(B_n)) = H'(C) = H(L_n)$. □

It is interesting to observe that a tree automaton that recognizes the language $H(L_n)$ of Example 11 is bound to be of size at least $O(2^n)$. This is easy to see, because for

every possible subtree $s$ of $f(s,s) \in H(L_n)$ we need one extra state. In fact, even if we consider *nondeterministic* bottom-up tree automata, which are obtained by generalizing the transition functions $\delta_\sigma : Q^k \to Q$ to functions $\delta_\sigma : Q^k \to \mathcal{P}(Q)$, then any smallest automaton for $H(L_n)$ is still of size $O(2^n)$. Thus, through copying of a finite set of terms (recognized by a DTA $A_n$ with $O(n)$ states) we obtain a finite set of terms for which any tree automaton needs exponentially more states than $A_n$. In other words, the description of $H(L_n)$ through the homomorphism $H$ and the automaton $A_n$ is *exponentially more succinct* than the description through any (even nondeterministic) tree automaton for $H(L_n)$. We obtain the following proposition. Let the size of a homomorphism be defined as the sum of sizes of the trees $H(g)$ for which $H$ is defined, and the size of a term $t$ equals the cardinality of $\mathtt{Pos}(t)$.

**Proposition 12.** *For every $n$, there exists a homomorphism $H$ and a DTA $A_n$ such that (1) $H$ is of size $O(n)$ (2) $A_n$ has $O(n)$-many states, and (3) $H(L(A_n))$ cannot be recognized by any nondeterministic tree automaton with less than $2^n$ states.*

Note that the language $H(L_n)$ of Example 11 cannot be recognized by a deterministic top-down tree automaton. We are ready to state the main result of this section.

**Theorem 13.** *Regularity of the homomorphic image of a regular term language $L$ is decidable in polynomial time if $L$ is defined over a unary alphabet.*

*Proof.* By Lemma 10 it suffices to prove that the infiniteness of $H(L_{a\_})$ can be checked in polynomial time, for a given tree automaton $A$ recognizing the regular language $L$, a homomorphism $H$, and copy symbol $a$. To this end, we just have to look for a transitions $\delta_a(q_1) = q_2$ and $\delta_b(q_3) = q_4$ such that $q_1$ is reachable from the initial state using no deleting symbols, $b$ is not deleting nor erasing, $q_3$ is reachable from $q_2$ without deleting symbols, and $q_3$ is reachable from $q_4$ without deleting symbols. The statement of the theorem follows from the fact that all these checks can be done in polynomial time with usual algorithms for finding paths. □

## 4 Sets of Terms with Regular Constraints and Bounded-Depth Copying Homomorphisms

In this section we prove decidability of regularity for the set of ground instances of a given finite set of terms. This is done in three steps. In Section 4.1 we give a sufficient condition for non-regularity by the existence of a certain infinite set of instances. In Section 4.2 we prove decidability of such existence as a direct consequence of a result from [4] and provide an algorithm for this problem. We also define in Section 4.3 a class of pairs $(L, H)$, where $L$ is a language and $H$ is a homomorphism, called *bounded-depth copying*, and prove that regularity of $H(L)$ is decidable.

### 4.1 A Sufficient Condition for Non-regularity

Some technical definitions are needed in order to prove our condition in Lemma 14. We say that a term $s$ is *determined* on a position $p$ if either $p$ is a position of $s$ and

is labeled by a non-variable symbol, i.e., $s[p] \notin X$, or there exists a prefix $p'$ of $p$ such that $s/p'$ is a non-variable symbol, i.e., $s/p' = s[p'] \notin X$. Equivalently, $s$ is determined on $p$ if for any two substitutions $\varphi_1, \varphi_2$, either $p$ is not defined in both $\varphi_1(s)$ and $\varphi_2(s)$, or $p$ is defined in both $\varphi_1(s)$ and $\varphi_2(s)$ and $\varphi_1(s)[p] = \varphi_2(s)[p]$. We say that $s$ is determined on a set of positions $P$ if $s$ is determined on all positions $p$ in $P$. Recall from the Preliminaries that a set of terms with regular constraints is a pair $\langle S, C\rangle$ where $C$ maps each variable occurring in $S$ to a regular term language, and that $L(\langle S, C\rangle) = \{\varphi(s) \mid s \in S, \varphi \text{ is a solution of } C\}$.

**Lemma 14.** *Let $s$ be a term, $S$ a set of terms, and $C$ a constraint for the variables occurring in $s$ and $S$. Let $x$ be a variable that occurs twice in $s$. Let $\{\varphi_1, \varphi_2, \ldots\}$ be an infinite set of solutions of $C$ satisfying $\varphi_i(s) \notin L(\langle S, C\rangle)$ for all $i > 0$, and $\varphi_i(x) \neq \varphi_j(x)$ for all $1 \leq i < j$. Then $L(\langle\{s\} \cup S, C\rangle)$ is not regular.*

*Proof.* Without lost of generality we may suppose that the set of variables occurring in $s$ is disjoint from the set of variables occurring in $S$; if it is not the case we simply introduce a new variable for each variable that occurs in $s$ and $S$. Let $S = \{s_1, \ldots, s_n\}$. We may make the following assumption. For a term $t$, let $\mathtt{Pos}_{\mathtt{nv}}(t)$ denote the positions $p$ in $\mathtt{Pos}(t)$ that are not labeled by a variable, i.e., for which $t[p] \notin X$.

**Assumption 1:** $s$ is determined on $\bigcup_{i\in\{1,\ldots,n\}} \mathtt{Pos}_{\mathtt{nv}}(s_i)$.

If the assumption is not satisfied, we proceed as follows. Since $s$ is not determined on $\bigcup_{i\in\{1,\ldots,n\}} \mathtt{Pos}_{\mathtt{nv}}(s_i)$, there exists a position $p \in \mathtt{Pos}(s)$ such that $s/p$ is a certain variable $y$ and $p \in \bigcup_{i\in\{1,\ldots,n\}} \mathtt{Pos}_{\mathtt{nv}}(s_i)$. Let $A$ be a deterministic bottom-up tree automaton that recognizes the regular term language $C(y)$. The automaton $A$ has a finite set of transitions of the form $\delta_g(q_1, \ldots, q_m) \to q$ for states $q_1, \ldots, q_m, q$ of $A$ where $q$ is an accepting state. For every of these rules $\delta_g(q_1, \ldots, q_m) \to q$ we construct a substitution $\gamma_{g,q_1,\ldots,q_m,q}$ that is the identity for all variables except for $y$, for which it is defined as $\gamma_{g,q_1,\ldots,q_m,q}(y) = g(z_1, \ldots, z_m)$, for new variables $z_1, \ldots, z_m$. We also extend the constraint $C$ to a new constraint $C'$ by defining $C'(z_i) = L(A, q_i)$. We do this for all transitions, choosing new variables for every transition. Let $s'_1, \ldots, s'_l$ be the terms obtained by applying all the substitutions $\gamma_{g,q_1,\ldots,q_m,q}$ to $s$. Note that $L(\langle\{s'_1, \ldots, s'_l\} \cup S, C'\rangle)$ coincides with $L(\langle\{s\}\cup S, C\rangle)$, and that the languages $L(\langle s'_1, C'\rangle), \ldots, L(\langle s'_l, C'\rangle)$ are disjoint, since $A$ is deterministic. By finiteness, for one of the terms $s'_1, \ldots, s'_l$, say $s'_1$, there exists an infinite subset $\{\varphi_{i_1}, \varphi_{i_2}, \ldots\}$ of $\{\varphi_1, \varphi_2, \ldots\}$ such that the run of $A$ on each $\varphi_{i_j}(y)$ applies the same transition $\delta_g(q_1, \ldots, q_m) = q$ at the root node. Let $k$ be the arity of this $g$, and note that in the case where $y$ equals $x$, $k$ is greater than 0 due to the condition $\varphi_i(x) \neq \varphi_j(x)$ for all $1 \leq i < j$. There exist substitutions $\theta_{i_1}, \theta_{i_2}, \ldots$ such that every $\varphi_{i_j}$ coincides with the corresponding $\theta_{i_j} \circ \gamma_{g,q_1,\ldots,q_m,q}$. Moreover, in the case where $y$ equals $x$, there exists $k'$ between 1 and $k$ and an infinite subset $\Theta$ of $\{\theta_{i_1}, \theta_{i_2}, \ldots\}$ such that for all different $\theta, \theta' \in \Theta$, $\theta(z_{k'}) \neq \theta'(z_{k'})$, where $z_1, \ldots, z_k$ are the new variables in $\gamma_{q,q_1,\ldots,q_m,q}(y)$.

The term $s'_1$, the set of terms $\{s'_2, \ldots, s'_l\} \cup S$, the set of substitutions $\Theta$ or $\{\theta_{i_1}, \theta_{i_2}, \ldots\}$, deppending on whether $x$ equals $y$ or not, the constraint $C'$, and the variable $z_{k'}$ if $y$ was $x$, or the variable $x$ otherwise, satisfy the conditions of the lemma, but also, $p$ is determined on $s'_1$. Hence, since $L(\langle\{s'_1, \ldots, s'_l\} \cup S, C'\rangle)$ coincides with $L(\langle\{s\} \cup$

$S, C\rangle)$, the statement of the lemma remains unchanged when replacing the elements by these new ones, for which there is strictly one less position that is not determined. We can repeat this process if there are still positions in $\bigcup_{i\in\{1,\ldots,n\}} \mathtt{Pos}_{\mathtt{nv}}(s_i)$ that $s'_1$ is not determined on them, until Assumption 1 on determined positions is satisfied.

Now, we come back to the principal proof with this additional assumption. We prove that $L(\langle\{s\} \cup S, C\rangle)$ is not regular by contradiction. Hence, suppose that $B$ is a deterministic tree automaton recognizing this language. We also assume an implicit election of an automaton $A_{C(y)}$ recognizing $C(y)$, for every variable $y \in \mathtt{Dom}(C)$. At this point we need an additional assumption.

**Assumption 2:** For every position $p \in \mathtt{Pos}(s)$, the run of $B$ gives the same state on all terms $\varphi_i(s)/p$. Moreover, for every of such $p \in \mathtt{Pos}(s)$ and $A_{C(y)}$, the run of $A_{C(y)}$ gives the same state on all terms $\varphi_i(s)/p$.

Similar as before, Assumption 2 can easily be enforced by choosing a certain infinite subset $\Gamma$ of the substitutions $\varphi_1, \varphi_2, \ldots$

Now, note that the fact that a certain term $\varphi_i(s)$ is not in $L(\langle\{s_j\}, C\rangle)$ for any $s_j \in S$ can be due to several different causes:

(i) There is a position $p \in \mathtt{Pos}_{\mathtt{nv}}(s_j)$ such that either $p$ is not defined in $s$ or it is defined in $s$ but then $s[p] \neq s_j[p]$.
(ii) Item (i) is not satisfied, but there exists a position $p$ such that $s_j/p$ is a variable and $\varphi_i(s)/p \notin C(s_j/p)$.
(iii) Items (i) and (ii) are not satisfied, but there exist positions $p_1, p_2$ such that $s_j/p_1$ and $s_j/p_2$ are the same variable, but $\varphi_i(s)/p_1 \neq \varphi_i(s)/p_2$.

We say that either (i), or (ii), or (iii) with the corresponding chosen $p_1$ and $p_2$, are *the cause* for $\varphi_i \in \Gamma$ and $s_j \in S$, depending on the case.

Finally, we fix a position $q$ such that $s/q = x$ and a substitution $\varphi \in \Gamma$.

Now, consider $\varphi$ and any other substitution $\varphi' \in \Gamma$ such that $\varphi'(x)$ is not a subterm of $\varphi(s)$. Note that such a $\varphi'$ exists thanks to the fact that all the $\varphi_i(x)$ are different. The term $\varphi(s)[q \leftarrow \varphi'(x)]$ is accepted by $B$ due to Assumption 2. Hence, to reach a contradiction it suffices to see that it is not in $L(\langle\{s\} \cup S, C\rangle)$, i.e. the language accepted by $B$. Of course it is not in $L(\langle\{s\}, C\rangle)$, since $x$ appears multiple times in $s$, and in particular at position $q$ and in some other position $q'$ satisfying $\varphi(s)[q \leftarrow \varphi'(x)]/q \neq \varphi(s)[q \leftarrow \varphi'(x)]/q'$. But also, $\varphi(s)[q \leftarrow \varphi'(x)]$ is not in any $L(\langle\{s_j\}, C\rangle)$ for $s_j \in S$, and we prove it deppending on which is the cause for $\varphi$ and $s_j$. If the cause for $\varphi$ and $s_j$ is (i), this is trivial. If the cause for $\varphi$ and $s_j$ is (ii), this follows directly from Assumption 2. If the cause for $\varphi$ and $s_j$ is (iii) with positions $p_1$ and $p_2$, then, either $q$ is disjoint with $p_1$ and $p_2$ and this is trivial, or otherwise, $q$ has to be a suffix of either $p_1$ or $p_2$, due to Assumption 1. In the latter case, due to the election of $\varphi'$, $\varphi(s)[q \leftarrow \varphi'(x)]/p_1 \neq \varphi(s)[q \leftarrow \varphi'(x)]/p_2$, and hence, $\varphi(s)[q \leftarrow \varphi'(x)]$ is not in $L(\langle\{s_j\}, C\rangle)$ again. □

### 4.2 Computing the Existence of Infinite Instances

Due to our sufficient condition, and the use of it that is done later by the algorithm for deciding regularity, we need to prove that the following problem is computable:

**Infinite-Instances Problem**
Input: $x, s, \{s_1, ..., s_n\}, C$.
Output: If there exists an infinite set $\varphi_1, \varphi_2, \ldots$ of solutions of $C$ such that $\varphi_i(x) \neq \varphi_j(x)$, and $\varphi_i(s)$ is not in $L(\langle\{s_1, \ldots, s_n\}, C\rangle)$, then give output "yes". Otherwise, give as output the finite set $\{t_1, \ldots, t_k\}$ of ground terms $t_i$ satisfying: $t_i \neq t_j$ for $1 \leq i < j \leq k$, $t_i$ is ground for $1 \leq i \leq k$, and for any solution $\varphi$ such that $\varphi(s)$ is not in $L(\langle\{s_1, \ldots, s_n\}, C\rangle)$, it holds that $\varphi(x) = t_i$ for some $i$ in $\{1, \ldots, k\}$.

The decidability of this problem is *directly deduced* from the work of Comon and Delor [4] on equational formulae with membership constraints[1]. An *equational formula with membership constraints* is a first order formula whose atoms are equations $t = u$, membership constraints $t \in S$ ($S$ is called a sort symbol) or $\bot$. Formulae are interpreted w.r.t. to a mapping $\mathcal{U}$ which associates with each sort symbol $S$ a subset of $T_\Sigma$. A ground substitution $\sigma$ is a solution of the equation $s = t$ if its domain contains the set of free variables of $\{s, t\}$ and $t\sigma \equiv s\sigma$, where $\equiv$ denotes the syntactic equality in terms. This is a solution of $t \in S$ if $t\sigma$ belongs $\mathcal{U}(S)$. The connectives are interpreted as usual and with a formula we associate the set of solutions. Two formulae are said equivalent w.r.t. $\mathcal{U}$ if they have the same solutions w.r.t. $\mathcal{U}$. From now on, we will suppose that the interpretation of the sort symbols is fixed and that sorts are interpreted as recognizable languages. In [4], the authors provide a complete, correct and terminating set of rules, which reduces a formula to an equivalent solved form which is either $\bot$, or a finite disjunction of formulae (called definition with constraints) of the form:
$\exists \boldsymbol{w} : x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge z_1 \neq u_1 \wedge z_m \neq u_m \wedge y_1 \in S_1 ... \wedge y_k \in S_k$ where:

- $x_1, \ldots, x_n$ are free variables that occur only once in the formula
- $z_1, \ldots, z_m$ are variables, s.t. for all $i$, $z_i \notin Var(u_i)$
- $y_1, \ldots, y_k$ are distinct variables
- for all $i$, $S_i$ is interpreted as an infinite (recognizable) language.
- $t_1, \ldots, t_n, u_1, \ldots, u_m$ are -non necessarly ground- terms.

Furthermore, Comon and Delor prove that every definition with constraints has at least one solution (Lemma 2 of [4]). Using those results we can even get:

**Lemma 15.** *Let $D = \exists \boldsymbol{w} : x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge z_1 \neq u_1 \wedge \ldots \wedge z_m \neq u_m \wedge y_1 \in S_1 \wedge \ldots \wedge y_k \in S_k$ be a definition with constraints satisfying the above conditions. Let $x$ a free variable in $D$. $D$ has an infinite set of solutions satisfying $\sigma_i(x) \neq \sigma_j(x)$ for all $j > i > 0$ iff it does not contain any equation $x = t$ with $t$ ground.*

*Proof.* Of course, if $D$ contains an equation $x = t$ with $t$ ground, $D$ has no infinite set of solutions satisfying $\sigma_i(x) \neq \sigma_j(x)$.

Conversely, let us suppose that $D$ does not contain any equation $x = t$ with $t$ ground. We can suppose without lost of generality that $\{y_1, \ldots, y_k\}$ contains all the variables except the $x_i$'s (we can add $v \in \top$ if necessary). By Lemma 2 of [4], D has one solution $\sigma_1$. Now consider a new formula $D'$ obtained by replacing every $S_i$ by $S_i - \{\sigma_1(y_i)\}$. This formula is again a definition with constraints (the new sets are regular and infinite).

[1] Thanks to Hubert Comon who made us aware of this fact.

For $D'$, Lemma 2 of [4] applies again and we have a solution $\sigma_2$ for $D'$, which in fact is also a solution of $D$. Moreover, $\sigma_1(x)$ differs from $\sigma_2(x)$ due to the definition of $\sigma_2$, and the fact that either $x$ is some $y_i$, or $x$ is some $x_i$ and the corresponding $t_i$ is not ground and with all its variables in $\{y_1, \ldots, y_k\}$. This process can be repeated again by replacing every $S_i$ by $S_i - \{\sigma_1(y_i), \sigma_2(y_i)\}$, and so on, thus obtaining the desired sequence. □

Now we come back to the Infinite-Instances problem. Let $x, s, S = \{s_1, ..., s_n\}, C$ be an input of the problem. For this input we define $\Phi$ as the following equational formula with membership constraints:

$$(\wedge_{y \in Var(s)} y \in C(y)) \wedge$$
$$(\wedge_{1 \leq i \leq n, \{x_1, \ldots, x_k\} = Var(s_i)} (\forall x_1, \ldots, x_k : (x_1 \in C(x_1) \wedge \ldots \wedge x_k \in C(x_k) \Rightarrow s_i \neq s)))$$

Let $Sol(\Phi)$ the set of solutions of $\Phi$. There exists an infinite set $\varphi_1, \varphi_2, \ldots$ such that $\varphi_i(x) \neq \varphi_j(x)$, and $\varphi_i(s)$ is not in $L(\langle\{s_1, \ldots, s_n\}, C\rangle)$, iff there exists an infinite set $\varphi_1, ..., \varphi_j ..$ in $Sol(\Phi)$ such that $\varphi_i(x) \neq \varphi_j(x), 0 < i < j$.

But now, as said before, we can transform $\Phi$ into an equivalent finite disjunction of definitions with constraints $\vee_{1 \leq i \leq k} D_i$. For each of these definitions $D_i$, we check whether it contains an equation $x = t_i$, with $t_i$ ground. If the answer is no for at least one element of the disjunction, there exists an infinite set $\{\varphi_1, \varphi_2, \ldots\}$ included in $Sol(\Phi)$ satisfying that $\varphi_i(x) \neq \varphi_j(x)$ for all $j > i > 0$ and we output yes. If the answer is yes for each definition $D_i$, we output "no" and the set $\{t_1, \ldots, t_k\}$ fulfills the required conditions.

**Lemma 16.** *The Infinite-Instances problem is computable.*

**Deciding Regularity of the Instances of a Set of Terms with Regular Constraints** Using Lemmas 14 and 16 we obtain an algorithm for deciding whether $L(\langle S, C\rangle)$ is regular for given $S$ and $C$. It checks repeatedly the previous sufficient condition looking for non-regularity. After a finite number of steps it terminates with an equivalent linear set, thus concluding regularity.

1 Input: A finite set of terms $S$ and a constraint $C$.
2 If all terms in $S$ are linear then stop with answer "yes".
3 Otherwise, chose a term $s \in S$ with a certain variable $x$ occurring at least twice in $s$, and ask for the existence of an infinite set $\{\varphi_1, \varphi_2, \ldots\}$ of solutions of $C$ such that $\varphi_i(x) \neq \varphi_j(x)$, and $\varphi_i(s)$ is not in $L(\langle\{S - \{s\}, C\rangle)$.
4 If the answer is "yes" then stop with answer "no".
5 Otherwise consider the set of ground terms $\{t_1, \ldots, t_k\}$ satisfying $t_i \neq t_j$ for $1 \leq i < j \leq k$, $t_i$ is ground for $1 \leq i \leq k$, and for any solution $\varphi$ of $C$ such that $\varphi(s)$ is not in $L(\langle\{S - \{s\}, C\rangle)$, it holds that $\varphi(x) = t_i$ for some $i$ in $\{1, \ldots, k\}$, and do: $S := (S - \{s\}) \cup \{\{x \mapsto t_1\}(s), \ldots, \{x \mapsto t_k\}(s)\}$.
6 Go to 2.

**Theorem 17.** *It is decidable whether $L(\langle S, C\rangle)$ is regular for given set of terms $S$ and constraint $C$.*

### 4.3 Deciding the HOM-Problem for Bounded-Depth Copying Homomorphisms

We now come back to the problem of deciding regularity of homomorphic images of regular tree languages, i.e., deciding whether $H(L)$ is regular for a given tree homomorphism $H$ and regular tree language $L$. Let $L$ be an arbitrary regular tree language. We show that if $H$ is "bounded-depth copying", then we can construct a set of terms with constraints $\langle S, C\rangle$ such that $L(\langle S, C\rangle) = H(L)$. Since for such sets we can decide regularity by Theorem 17, we obtain decidability of regularity of $H(L)$. Roughly speaking, "bounded-depth copying" means that symbols $\sigma$ for which $H(\sigma)$ is non-linear ("copying"), occur at bounded depth. Let us first fix some notations. For a homomorphism $H$, denote by $\mathtt{Copy}(H)$ the set of symbols $\sigma$ for which $H(\sigma)$ is non-linear, i.e., for which a variable $x_i$ occurs more than once in $H(\sigma)$. A symbol $\sigma$ is called *erasing* if $H(\sigma)$ is a variable, i.e., $H(\sigma) \in X$. A position $p$ of $t$ is *deleted in* $t$ *by* $H$ if there exists $q, r$ and an $i \geq 1$ such that $p = q.i.r$ and $t[q]$ is a symbol $b$ such that $H(b)$ does not contain the variable $x_i$.

**Definition 18.** *Let $L$ be a set of trees, $H$ a tree homomorphism, and $k$ a natural number. The pair $(L, H)$ is* depth-k copying *if for any $t$ in $L$ and position $p$ in $t$ such that $t[p]$ belongs to $\mathtt{Copy}(H)$, either $p$ is deleted in $t$ by $H$ or the path from the root of $t$ to $p$ contains at most $k$ occurrences of non-erasing symbols. The pair $(L, H)$ is* bounded-depth copying *if there exists a $k$ such that $(L, H)$ is depth-$k$ copying.*

Note that, given $(L, H)$, it is decidable whether or not $(L, H)$ is bounded-depth copying: we can construct a finite state word automaton $A$ that recognizes the labeled paths of trees in $L$ which lead to a (non-deleted) copy symbol in $\mathtt{Copy}(H)$. Then $(L, H)$ is bounded-depth copying if and only if we can bound the number of non-erasing symbols occurring in words in $L(A)$, which is decidable. Note that this is similar to computing the inverse image of a regular tree language under a homomorphism, used at the end of the proof of Lemma 20; cf. also the discussion in the Conclusions. Clearly, if symbols of $\mathtt{Copy}(H)$ only occur at deleted positions in trees of $L$, then $H(L)$ is regular. The reason for this is that only linear rules of $H$ are applied when translating trees in $L$.

**Lemma 19.** *Let $L$ recognizable and $H$ a homomorphism such that any position in a tree $t \in L$ labeled by a symbol in $\mathtt{Copy}(H)$ is deleted in $t$ by $H$. Then $H(R)$ is regular.*

**Lemma 20.** *Let $L$ be recognizable and $H$ a tree homomorphism such that $(L, H)$ is bounded-depth copying. There effectively exists a finite set of terms with regular constraints $\langle S, C\rangle$ such that $L(\langle S, C\rangle) = H(L)$.*

*Proof.* Let $A = (Q, Q_a, \Sigma, \delta)$ be a deterministic tree automaton with $L(A) = L$ and let $k \geq 0$ such that $(L, H)$ is depth-$k$ copying. The proof is by induction on $k$.

If $k = 0$ then for any $t \in L$ and $p \in \mathtt{Pos}(t)$ such that $t[p] \in \mathtt{Copy}(H)$, the path from the root of $t$ to $p$ does not contain non-erasing symbols. Let $a$ be a non-erasing symbol. Let $Set_a = \{(q_1, \ldots, q_n) \mid \exists t(x), \delta(t(a(q_1, \ldots, q_n)) \in Q_f \wedge H(t(x)) = x\}$. Then, by assumption, for any $(q_1, \ldots, q_n)$ in $Set_a$, the languages $L(A, q_1), \ldots, L(A, q_n)$ do not contain any occurrence of symbols in $\mathtt{Copy}(H)$ at non-deleted positions. Hence, by Lemma 19, their images by $H$ are recognizable languages, let us say $R_{q_1}, \ldots, R_{q_n}$. Now, any term $u$ can be decomposed into $u_1(u_2)$ where $H(u_1(x)) = x$ and the root of

$u_2$ is labeled by a non-erasing symbol. So, $H(L)$ is the union of the $L(\langle\{H(a)\}, C\rangle)$ for $a$ non-erasing, where $C(x_i) = R_{q_i}$. Thus, by introducing new variables for each such $a$ and extending $C$ appropriately, we obtain a set of terms with constraints $\langle S, C\rangle$ which equals $H(L)$.

*Induction step:* Let us suppose the lemma holds for pairs of regular tree language and homomorphism which are depth-$k$ copying. Let $(L, H)$ be depth-$(k+1)$ copying. This means that for any position $p$ in $t$ such that $t[p]$ belongs to $\mathtt{Copy}(H)$, the path from the root of $t$ to $p$ contains at most $k+1$ non-erasing symbols. Once more, let $a$ be a non-erasing symbol and $Set_a$ defined as before. Then $H(L)$ is the union of the $L(\langle H(a), \{(x_i, H(L(A, q_i)) \mid 1 \leq i \leq n\}\rangle)$ for non-erasing $a$ of arity $n$ and $(q_1, \ldots, q_n)$ in $Set_a$. The $H(L(A, q_i))$ are not necessarily regular but $(H, L(A, q_i))$ is depth-$k$ copying. Thus, by induction hypothesis, for any $L(A, q_i)$ there exists some $\langle S_i, C_i\rangle$ such that $H(L(A, q_i)) = L(\langle S_i, C_i\rangle)$. By renaming of variables we can compose these sets $\langle S_i, C_i\rangle$ to obtain a set $\langle S, C\rangle$ such that $L(\langle S, C\rangle) = H(L)$. This concludes our inductive proof.

In order to show that the above proof is constructive, it suffices to show that $Set_a$ is computable. Thus, we need to show how to compute the set of trees $t(x)$ for which $H(t(x)) = x$. To this end, let $\perp$ be a new constant and let $G$ be the extension of $H$ by $G(\perp) = \perp$. Then $G^{-1}(\perp)$ is effectively regular because inverses of tree transducers effectively preserve the regular tree languages (see, e.g., Proposition 20.1 of [7] where this is shown for the class of top-down tree transducers which includes the class of tree homomorphisms). Let $L_\perp = \{t[p \leftarrow \perp] \mid t \in L, p \in \mathtt{Pos}(t)\}$ be the regular tree language of all trees $t$ in $L$ in which exactly one node has been replaced by $\perp$. Then $G^{-1}(\perp) \cap L_\perp$ is the required set of trees $t(x)$ for which $H(t(x)) = x$. □

Using Lemma 20 and Theorem 17 we obtain our main result about pairs of regular tree languages $L$ and homomorphisms $H$ with decidable regularity of $H(L)$.

**Theorem 21.** *Let $L$ be a regular tree language and $H$ a tree homomorphism such that $(L, H)$ is bounded-depth copying. It is decidable whether or not $H(L)$ is regular.*

**Conclusions and Future Work.** For a given regular tree language $L$ and a tree homomorphism $H$ we have shown that regularity of $H(L)$ is decidable in the following cases: (1) if $L$ is monadic, i.e., only uses symbols of arity 1 or 0, (2) if non-linear rules of $H$ are only applied at positions of bounded depth in the trees of $L$. For case (1) we have given a decision procedure that runs in polynomial time; furthermore, we have shown that, even in the monadic case, there are pairs of $L$, $H$ that generate regular languages for which any smallest (nondeterministic) tree automaton is exponentially larger than the representation of $L$ and $H$. It remains open what the precise complexity of the decision procedure of case (2) is: currently we apply the inverse image of a tree homomorphism in the proof, which can be expensive: for top-down tree transducers this problem is known to be EXPTIME-complete [13]. What is the complexity of this problem for tree homomorphisms? Can we drop the bounded-depth restriction of case (2) and extend decidability to homomorphisms that copy at most once? Or even to those that copy a bounded number of times? The latter is the "finite-copying" restriction [1], a well-studied topic in tree transducer theory.

## References


1. Aho, A.V., Ullman, J.D.: Translations on a context-free grammar. Information and Control 19, 439–475 (1971)
2. Bogaert, B., Seynhaeve, F., Tison, S.: The recognizability problem for tree automata with comparison between brothers. In: Thomas, W. (ed.) FOSSACS 1999. LNCS, vol. 1578, pp. 150–164. Springer, Heidelberg (1999)
3. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2002), http://www.grappa.univ-lille3.fr/tata
4. Comon, H., Delor, C.: Equational formulae with membership constraints. Inf. Comput. 112, 167–216 (1994)
5. Engelfriet, J.: Bottom-up and top-down tree transformations — A comparison. Math. Systems Theory 9, 198–231 (1975)
6. Fülöp, Z.: Undecidable properties of deterministic top-down tree transducers. Theoret. Comput. Sci. 134, 311–328 (1994)
7. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, ch. 1, Springer, Heidelberg (1997)
8. Hopcroft, J.W., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading (1979)
9. Hosoya, H.: Foundations of XML processing (2002), http://arbre.is.s.u-tokyo.ac.jp/~hahosoya/xmlbook/xmlbook.pdf
10. Kucherov, G., Rusinowitch, M.: Patterns in words versus patterns in trees: A brief survey and new results. In: Ershov Memorial Conference, pp. 283–296 (1999)
11. Kucherov, G., Tajine, M.: Decidability of regularity and related properties of ground normal form languages. Inf. Comput. 118, 91–100 (1995)
12. Lassez, J.-L., Marriott, K.: Explicit representation of terms defined by counter examples. J. Autom. Reasoning 3, 301–317 (1987)
13. Martens, W., Neven, F.: On the complexity of typechecking top-down xml transformations. Theor. Comput. Sci. 336, 153–180 (2005)
14. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of xml schema languages using formal language theory. ACM Trans. Internet Techn. 5(4), 660–704 (2005)
15. Suciu, D.: The XML typechecking problem. SIGMOD Record 31, 89–96 (2002)

# A Kleene-Schützenberger Theorem for Weighted Timed Automata

Manfred Droste and Karin Quaas

Institut für Informatik, Universität Leipzig
04009 Leipzig, Germany
{droste,quaas}@informatik.uni-leipzig.de

**Abstract.** During the last years, weighted timed automata (WTA) have received much interest in the real-time community. Weighted timed automata form an extension of timed automata and allow us to assign weights (costs) to both locations and edges. This model, introduced by Alur et al. (2001) and Behrmann et al. (2001), permits the treatment of continuous consumption of resources and has led to much research on scheduling problems, optimal reachability and model checking. Also, several authors have derived Kleene-type characterizations of (unweighted) timed automata and their accepted timed languages. The goal of this paper is to provide a characterization of the possible behaviours of WTA by rational power series. We define WTA with weights taken in an arbitrary semiring, resulting in a model that subsumes several WTA concepts of the literature. For our main result, we combine the methods of Schützenberger, a recent approach for a Kleene-type theorem for unweighted timed automata by Bouyer and Petit as well as new techniques. Our main result also implies Kleene-type theorems for several subclasses of WTA investigated before, e.g., for weighted finite automata, timed automata and timed automata with stopwatch observers.

## 1 Introduction

Since its introduction in 1994 by Alur and Dill [2], timed automata have been a thoroughly investigated model for the specification and analysis of real-time systems. In the literature, not only a variety of interesting theoretical results for timed automata and timed languages have been established (see [4] for a survey), but there has also much practical work been done such as the development of symbolic data structures and efficient algorithms, leading to model-checking tools like KRONOS, UPPAAL and HYTECH [17], [25], [22], successfully used for solving industrially relevant problems, e.g. [27], [23].

*Weighted timed automata* have been of much interest in the real-time community during the last years. The model has been introduced independently by Alur et al. [3] and Behrmann et al. [8] and allows us to assign weights to both the locations and edges of the underlying timed automaton. The weight of an edge gives the actual cost for executing it, whereas the weight of a location gives the cost for staying in this location *per time unit*. The weight for reaching a certain

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 142–156, 2008.

location $s$ (or, analogously, the weight for accepting a certain timed word) is computed by taking the minimum over the running weights of all runs ending in $s$. The running weight of a run is the sum of the costs of all participating transitions of the run. In this way, WTA have been used to model continuous consumption of resources, allowing applications in operations research, in particular *optimal* scheduling and planning [20], [28]. Consequently, a number of decision problems have been investigated, most of them concerning the reachability problem under some optimization aspect [8], [3], [11], model-checking [18], [12], [14], and weighted timed games [1].

The goal of this paper is to provide a characterization of the possible behaviours of WTA in terms of rational power series, i.e., power series constructed by the standard rational operations addition, Cauchy product and Kleene star iteration. Moreover, we give a more general definition of WTA which includes all the various definitions given in the literature so far and which gives rise to some interesting variants.

We define WTA *over a semiring* in the same manner as it is done for classical weighted finite automata [29], [24], [10]. In this way, we are not bound to a fixed set of weights, nor are we restricted to use the operations of addition and infimum for computing the weight of a word. Secondly, we do not restrict the cost functions for the locations to be linear (as previously done in [3], [8]). Instead, we consider WTA with respect to an arbitrary family of functions $\mathcal{F}$ mapping positive reals to elements in the semiring. The cost for staying in a location $s$ is defined by a cost function $C_s \in \mathcal{F}$. By introducing the notion of a family of cost functions $\mathcal{F}$ and the semiring, we hope to obtain a flexible model of WTA.

For our characterization of the behaviours of WTA by rational power series, we establish a Kleene-Schützenberger theorem for WTA. Schützenberger's theorem is the analogue to the famous Kleene theorem for the class of weighted finite automata: the set of *recognizable* power series, i.e., the set of power series that constitute the behaviour of a weighted automaton, is precisely the set of *rational* power series [30]. As is well-known, rational expressions can be used to specify properties of systems. Recently, there have also been several approaches to give a Kleene-type theorem for timed languages [5], [7], [6], [15], of which we choose the latest approach of Bouyer and Petit in 2001 [16] because of its simplicity and elegance. According to their result, the set of regular timed languages, as defined by Alur and Dill [2], coincides with the set of rational timed languages, defined over the standard rational operations $+, \cdot, ^*$ and an additional projection operation. For the proof of our main result, we combine the methods of Bouyer and Petit, Schützenberger and new techniques.

We define the semantics of WTA based on the notion of *clock words* as introduced by Bouyer and Petit [16]. Clock words, as opposed to the well-known timed words [2], bear more information concerning the actual values of clock variables than timed words, and thus enable the authors to define a concatenation operation in a natural way. Consequently, all the definitions and constructions for the Kleene theorem are given with respect to clock words. However, it can be shown that clock words can easily be mapped to timed words using a projection

function; thus the Kleene theorem for clock words can be extended to timed words. To bring weights into play, we introduce the notion of *clock series*. Clock series are a particular kind of power series which map clock words to elements in a semiring. We define addition, Cauchy product and Kleene star iteration on the set of clock series to give a formal definition for the set of rational clock series. The main objective of this work is to show that this set is equal to the set of recognizable clock series, which make up the behaviour of the class of WTA. We establish this in two steps. First, the crucial part for showing that any rational clock series is a recognizable clock series is to prove that WTA are closed under the three operations mentioned above. In our proof, for dealing with the weights assigned to locations, we need to give new methods for normalizing the automata. The proof for the other direction, i.e., any recognizable clock series is rational, is based on the solution of equations [16], [10]. Finally, we show how we can extend the theorem in such a way that it can be applied to timed semantics as well.

## 2 Preliminaries

In the following, we use $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ to denote the positive reals and natural numbers, respectively. Furthermore, we write $\mathbb{R}_{\geq 0}^{1+n}$ for $\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}^{n}$.

**Clock Constraints and Clock Valuations.** Let $X$ be a finite set of variables, called *clock variables*. We define *clock constraints* $\phi$ over $X$ to be conjunctions of formulas of the form $x \sim k$, where $k \in \mathbb{N}$, $x \in X$, and $\sim \in \{<, \leq, >, \geq\}$. Let $\Phi(X)$ be the set of all clock constraints $\phi$ over $X$. A *clock valuation* $\nu : X \to \mathbb{R}_{\geq 0}$ is a function that assigns a value to each clock variable. A clock valuation $\nu$ satisfies a clock constraint $\phi$, written $\nu \models \phi$, if $\phi$ evaluates to true according to the values given by $\nu$. Given $\delta \in \mathbb{R}_{\geq 0}$, we let $\nu + \delta$ be the clock valuation such that $(\nu+\delta)(x) = \nu(x)+\delta$ for each $x \in X$. For $\lambda \subseteq X$, we define $\nu[\lambda := 0]$ as the clock valuation that assigns 0 to each $x \in \lambda$, and agrees with $\nu$ over the remaining clock variables $x \in X \backslash \lambda$.

**Timed and Clock Words.** Let $\Sigma$ be a finite alphabet and $n \in \mathbb{N}$. A *timed word* is a finite sequence $w = (\sigma_1, t_1)(\sigma_2, t_2)...(\sigma_k, t_k) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, where the sequence $t_1 t_2 ... t_k$ is non-decreasing. Intuitively, $t_i$ gives the time of occurrence of $\sigma_i$. An *n-clock word* is a finite sequence $w = (t_0, \nu_0)(\sigma_1, t_1, \nu_1)...(\sigma_k, t_k, \nu_k)$ from the infinite set $(\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^*$, where $(\sigma_1, t_1)...(\sigma_k, t_k)$ is a timed word, and $\nu_i$ gives the values of the clock variables just after the computation of $\sigma_i$. The pair $(t_0, \nu_0)$ corresponds to the starting condition and is considered to be an empty $n$-clock word. The set of *empty n-clock words* is denoted by $\Gamma_n (= \mathbb{R}_{\geq 0}^{1+n})$. We define $(\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^+$ by $(\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^* \backslash \Gamma_n$. Let $w = (t_0, \nu_0)(\sigma_1, t_1, \nu_1)...(\sigma_k, t_k, \nu_k)$ and $w' = (t'_0, \nu'_0)(\sigma'_1, t'_1, \nu'_1)...(\sigma'_l, t'_l, \nu'_l)$ be two $n$-clock words. We say that $w$ is *compatible* with $w'$ if $(t_k, \nu_k) = (t'_0, \nu'_0)$. In this case, we define the *concatenation* $w \cdot w'$ of $w$ and $w'$ to be the $n$-clock word $(t_0, \nu_0)(\sigma_1, t_1, \nu_1)...(\sigma_k, t_k, \nu_k)(\sigma'_1, t'_1, \nu'_1)...(\sigma'_l, t'_l, \nu'_l)$. By $|w|$ we mean the length of an $n$-clock word $w$.

**Semirings and Formal Power Series.** A semiring is a tuple $\mathcal{K} = (K, \oplus, \odot, 0, 1)$ such that $(K, \oplus, 0)$ is a commutative monoid, $(K, \odot, 1)$ is a monoid, $\odot$ is both left- and right-distributive over $\oplus$, and $0 \odot x = x \odot 0 = 0$ for any $x \in K$. As examples consider the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ of natural numbers with the usual addition and multiplication, the Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$, and the tropical semiring $(\mathbb{R}_{\geq 0} \cup \{\infty\}, min, +, \infty, 0)$. Let $A$ be an arbitrary set and $\mathcal{K}$ a semiring. A function $S : A \to K$ is called a *formal power series* (fps) over $\mathcal{K}$. For historical reasons, the image of an element $w \in A$ is denoted by $(S, w)$. We write $\mathcal{K}\langle\langle A \rangle\rangle$ to mean the set of all fps $S : A \to K$.

## 3 Weighted Timed Automata

**Weighted Timed Automata.** Let $\mathcal{K}$ be a semiring, $\Sigma$ be a finite alphabet, and $X$ be a finite set of clock variables. We consider timed automata $\mathcal{A}$ augmented with cost functions that assign elements from $K$, so-called *weights* (or costs), to both the edges and the locations of $\mathcal{A}$. The weight for staying in a location depends on the amount of time we spend in this location; thus, we define a cost function from $\mathbb{R}_{\geq 0}$ to $K$ for each location. Let $\mathcal{F}$ be any family of functions from $\mathbb{R}_{\geq 0}$ to $K$. A *weighted timed automaton (WTA)* over $\mathcal{K}$, $\Sigma$, $X$ and $\mathcal{F}$ is a tuple $\mathcal{A} = (S, S_0, S_f, E, C)$, where

- $S$ is a finite set of locations (states)
- $S_0 \subseteq S$ is a set of initial locations
- $S_f \subseteq S$ is a set of final locations
- $E \subseteq S \times S \times \Sigma \times \Phi(X) \times 2^X$ is a finite set of edges. An edge $(s, s', \sigma, \phi, \lambda)$ allows a jump from location $s$ to location $s'$ if $\sigma$ is read, provided that the current values of the clock variables in location $s$ satisfy the clock constraint $\phi$. After the edge has been executed, all clock variables in $\lambda$ are reset to zero, whereas the values of all other clock variables remain unchanged.
- $C = \{C_E\} \cup \{C_s : s \in S\}$, where $C_E : E \to K$, and $C_s \in \mathcal{F}$ for any $s \in S$.

Let $\mathcal{A} = (S, S_0, S_f, E, C)$ be a WTA. The *timed semantics* of $\mathcal{A}$ is given by an infinite state transition system that corresponds to a weighted extension of the original semantics of timed automata defined by Alur and Dill [2]. However, in the following we will give an additional semantics, called *clock semantics*. This model is based on the notion of clock words rather than timed words, and allows for a natural definition of concatenation [16].

**Timed Semantics.** Let $\mathcal{S}^T_{\mathcal{A}}$ be a state-transition-system with states of the form $(s, \nu)$, where $s \in S$ and $\nu$ is a clock valuation. We define *timed transitions* to be transitions of the form $(s, \nu) \xrightarrow{\delta/c}_T (s, \nu+\delta)$ where $c = C_s(\delta)$. A *discrete transition* is of the form $(s, \nu) \xrightarrow{\sigma/c}_D (s', \nu')$ such that there is an edge $e = (s, s', \sigma, \phi, \lambda)$ in $E$ where $C_E(e) = c$, $\nu \models \phi$, and $\nu' = \nu[\lambda := 0]$. A *timed run* $r^T$ of $\mathcal{A}$ is a finite alternating sequence of timed and discrete transitions in $\mathcal{S}^T_{\mathcal{A}}$

$$r^T = \left((s_0, \nu_0) \xrightarrow{\delta_1/c_1}_T (s_1, \nu_1) \xrightarrow{\sigma_1/c'_1}_D (s'_1, \nu'_1) \xrightarrow{\delta_2/c_2}_T \ldots \xrightarrow{\sigma_k/c'_k}_D (s'_k, \nu'_k)\right)$$

where $\nu_0 = 0^{|X|}$. With $r^T$, the timed word $w = (\sigma_1, t_1)(\sigma_2, t_2)...(\sigma_k, t_k)$, such that $t_j = \sum_{i=1}^{j} \delta_j$ for each $1 \leq j \leq k$, is associated.

**Clock Semantics.** The clock semantics is very similar to the timed semantics and is given in terms of $|X|$-clock words. Consider the state-transition-system $\mathcal{S}_{\mathcal{A}}^{C}$, whose states are of the form $(s, t, \nu)$, where $s$ is a location, $t \in \mathbb{R}_{\geq 0}$, and $\nu$ is a clock valuation. The transition relation over the set of states in $\mathcal{S}_{\mathcal{A}}^{C}$ is defined in the same manner as the transition relation in $\mathcal{S}_{\mathcal{A}}^{T}$. A *clock run* $r^C$ of $\mathcal{A}$ is a finite alternating sequence of timed and discrete transitions

$$r^C = \big((s_0, t_0, \nu_0) \xrightarrow{\delta_1/c_1}_T (s_1, t_1, \nu_1) \xrightarrow{\sigma_1/c_1'}_D (s_1', t_1', \nu_1') \xrightarrow{\delta_2/c_2}_T ... \xrightarrow{\sigma_k/c_k'}_D (s_k', t_k', \nu_k')\big)$$

where $t_1 = t_0 + \delta_1$, $t_i' = t_i$ for any $1 \leq i \leq k$, and $t_i = t_{i-1}' + \delta_i$ for any $2 \leq i \leq k$. Note that in contrast to the timed semantics, $\nu_0$ can be arbitrary. The label of a canonical clock run is the $|X|$-clock word $w = (t_0, \nu_0)(\sigma_1, t_1', \nu_1')...(\sigma_k, t_k', \nu_k')$.

**Behaviour of $\mathcal{A}$.** Let $r$ be a (timed or clock) run as above. We define the *running weight* $rwt(r)$ of $r$ to be $rwt(r) = \prod_{i=1}^{k} c_i \odot c_i'$. The running weight of the *empty clock run* $(t_0, \nu_0)$ with label $(t_0, \nu_0) \in \Gamma_{|X|}$ is defined to be 1. We say that $r$ is *initialized* if $s_0 \in S_0$. It is *accepting* if $s_k' \in S_f$. If $r$ is both initialized and accepting it is called *successful*. The *timed behaviour* of the WTA $\mathcal{A}$ is the fps $\|\mathcal{A}\|^T : (\Sigma \times \mathbb{R}_{\geq 0})^* \to K$ defined by

$$(\|\mathcal{A}\|^T, w) = \sum \{rwt(r) : r \text{ is a successful timed run of } \mathcal{A} \text{ on } w\}$$

Similarly, we define the *clock behavior* of $\mathcal{A}$ to be the fps $\|\mathcal{A}\|^C : (\mathbb{R}_{\geq 0}^{1+|X|})(\Sigma \times \mathbb{R}_{\geq 0}^{1+|X|})^* \to K$ given by

$$(\|\mathcal{A}\|^C, w) = \sum \{rwt(r) : r \text{ is a successful clock run of } \mathcal{A} \text{ on } w\}$$

In the remainder of the paper, we will use the clock semantics for defining the notions of recognizability and rationality. In the last section, we show that these notions can easily be adapted to the timed semantics.

## 4 Relation to other Automata Models

Here we show that our model of WTA subsumes a number of more particular concepts of timed automata, as well as weighted (untimed) automata, which have been investigated intensively in the literature. In particular, by restricting $\mathcal{K}$ and $\mathcal{F}$, we obtain timed automata and weighted automata. This implies that our main theorem in Sect. 5 also applies to these automata classes.

**Timed Automata.** The classical (unweighted) timed automaton defined by Alur and Dill [2] can be obtained as follows. Let $\mathcal{K}$ be the Boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$, $\mathcal{F}$ be the family of constant functions 1, i.e., $C_s(\delta) = 1$ for any $s \in S$, $\delta \in \mathbb{R}_{\geq 0}$, and $C_E(e) = 1$ for any $e \in E$. Thus, Theorem 8 of Bouyer and Petit [16] is implied by our main theorem in Sect. 5.

**Other Weighted Timed Automata Models.** Weighted timed automata have been introduced independently by Alur et al. [3] and Behrmann et al. [8]. Both consider timed automata $\mathcal{A}$ augmented with a cost function that assigns a natural number to locations and edges of $\mathcal{A}$. In doing so, the increase of the cost variable is restricted to be linear. The cost of reaching a location $s$ is computed by taking the *minimum* of the costs of any run ending in $s$, where the cost of a run $r$ is the *sum* of the costs of all participating transitions in $r$. We can easily model this using the tropical semiring $(\mathbb{R}_{\geq 0} \cup \{\infty\}, min, +, \infty, 0)$, and restricting $\mathcal{F}$ to the class of linear functions.

Recently, the weighted timed automaton model has been generalized by allowing more than one cost variable. Larsen and Rasmussen introduced dual-priced timed automata [26]. The dual-priced timed automaton model can be modeled by defining a new "tropical" semiring with the underlying set $(\mathbb{R}_{\geq 0} \cup \{\infty\}) \times (\mathbb{R}_{\geq 0} \cup \{\infty\})$, and modifying the definitions of $min$ and $+$ in a suitable way, e.g. coordinate-wise. Similarly, we can extend to multi-priced models [13].

**Timed Automata with Stopwatch Observers.** A stopwatch is a clock variable that can be stopped and turned on again [21]. In other words, the rate of change of the stopwatch variable is either 0 or 1. A timed automaton augmented with a stopwatch variable that can neither be tested in a clock constraint nor be reset is called a timed automaton with a stopwatch observer. We use a WTA to model such an automaton by restricting $\mathcal{F}$ to be the constant functions 0 and 1. The edges shall not cost anything: $C_E(e) = 1$ for any $e \in E$.

**Weighted Finite Automata.** A weighted finite automaton $\mathcal{A}$ over a semiring $\mathcal{K} = (K, \oplus, \odot, 0, 1)$ is a finite automaton whose transitions are assigned costs taken from the semiring. The behavior of $\mathcal{A}$ is defined using the semiring operations $\oplus$ and $\odot$ in the same manner as it is done for WTA in Sect. 3. By restricting the family of functions $\mathcal{F}$ to the constant function 1, we yield a model which does not add any costs while staying in a location. In this way, we yield a classical weighted finite automaton. This implies that the Schützenberger theorem [30] is a special case of our main theorem in Sect. 5.

## 5 Clock Series

To describe the behaviour of a WTA $\mathcal{A}$ over $\mathcal{K}$, $\Sigma$, $X$ and $\mathcal{F}$, we want to use $\mathcal{F}$-*rational clock series*. In this section, we give a general definition of clock series, some basic properties of clock series, and the definition of rationality. Finally, we will give the main theorem of the paper. For the remainder of the paper, we fix a semiring $\mathcal{K}$, a finite alphabet $\Sigma$, a set of clock variables $X = \{x_1, ..., x_n\}$, and a family $\mathcal{F}$ of functions from $\mathbb{R}_{\geq 0}$ to $K$. If not otherwise specified, by writing $\mathcal{A}$ we mean a WTA $\mathcal{A}$ over $\mathcal{K}$, $\Sigma$, $X$ and $\mathcal{F}$.

**Clock Series.** A function $S : (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^* \to K$ is called an *n-clock series*. We denote the set of all $n$-clock series by $\mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0} \rangle\rangle$. On the set $\mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0} \rangle\rangle$,

we define the *sum* $S+T$ pointwise, i.e., $(S+T,w) = (S,w) \oplus (T,w)$. The *Cauchy product* $S \cdot T$ is defined by

$$(S \cdot T, w) = \sum_{u \cdot v = w} (S,u) \odot (T,v)$$

Furthermore, we define the clock series $\mathbb{1}$ by $(\mathbb{1}, w) = 1$ if $w \in \Gamma_n$, $(\mathbb{1}, w) = 0$ otherwise, and the clock series $\mathbb{0}$ by $(\mathbb{0}, w) = 0$ for each $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})$. The following lemma is the clock series version of the well-known fact that the set of fps over the free monoid together with sum and Cauchy product is a semiring.

**Lemma 1.** *The structure* $(\mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle, +, \cdot, \mathbb{0}, \mathbb{1})$ *is a semiring.*

For a clock series $S$, let $S^0 = \mathbb{1}$ and, inductively, $S^k = S \cdot S^{k-1}$ be the *k-th power* of $S$ for $k \geq 1$. The clock series $S \in \mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ is called *proper*, if $(S, w_0) = 0$ for any $w_0 \in \Gamma_n$. For proper clock series $S$, we define the *Kleene star iteration* $S^*$ by

$$(S^*, w) = \sum_{k \geq 0} (S^k, w)$$

Notice that from $(S, w_0) = 0$ for $w_0 \in \Gamma_n$, it follows that $(S^k, w) = 0$ for any $k > |w|$. This implies that the sum given above is finite and hence exists in $\mathcal{K}$.

**Lemma 2.** *Let* $S \in \mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ *be proper. Then* $S \cdot S^* + \mathbb{1} = S^*$.

Next, we give an explicit formula for the calculation of $S^k$. It can be proved by induction on $k$.

**Lemma 3.** *If* $S$ *is a proper* $n$*-clock series,* $k \in \mathbb{N}$*, and* $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^*$*, then* $(S^k, w)$ *has the explicit representation*

$$(S^k, w) = \sum_{w = w_1 \cdot \ldots \cdot w_k} \prod_{i=1}^{k} (S, w_i)$$

**Rational Clock Series.** For $c \in \mathcal{F}$, $k \in \mathcal{K}$, $\sigma \in \Sigma$, $\phi \in \Phi(X)$, and $\lambda \subseteq X$, we define the $\mathcal{F}$-monomial $\langle c, k, \sigma, \phi, \lambda\rangle : (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^* \to K$ as follows:

$$(\langle c, k, \sigma, \phi, \lambda\rangle, w) = \begin{cases} c(\delta) \odot k & \text{if } w = (t,\nu)(\sigma, t+\delta, \nu') \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n}) \\ & \text{s.t. } \delta \in \mathbb{R}_{\geq 0}, \nu + \delta \models \phi \text{ and } \nu' = \nu + \delta[\lambda := 0] \\ 0 & \text{otherwise} \end{cases}$$

An $n$-clock series $S$ is *$\mathcal{F}$-rational* if it can be defined starting from finitely many $\mathcal{F}$-monomials or the clock series $\mathbb{1}$ and $\mathbb{0}$, by means of a finite number of applications of $+$, $\cdot$ and $^*$, where the latter may only be applied to proper $n$-clock series. We use $\mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ to denote the set of all $\mathcal{F}$-rational $n$-clock series.

Observe that, similarly to the case of WTA, by restricting $\mathcal{K}$ and $\mathcal{F}$, we obtain rational expressions for several other (unweighted) automata classes or rational

fps for weighted automata. For instance, if $\mathcal{K}$ is the Boolean semiring and $\mathcal{F}$ is the family of constant functions 1, then rational clock series correspond to rational clock expressions defined by Bouyer and Petit [16].

*Example 1.* Consider the following specification of a real-time system with a single resource, where $\Lambda = \{a, b, c, d\}$ is a set of actions:

> The system must execute $a$ and $b$, and $b$ must be executed exactly 3 time units after $a$. Between $a$ and $b$, action $c$ (costs €3) and action $d$ (costs €2) may be executed consecutively for an arbitrary number of times, but $d$ is restricted to happen strictly between 1 and 2 time units after $c$. Being in the state after action $a$ or $d$ has been executed, costs €5 per time unit, whereas being in the state after $c$ has been executed, costs €1 per time unit.

The specification can be represented by the following rational clock series over the tropical semiring, $\Lambda$, $Y = \{x, y\}$ and $C_i(\delta) = i \cdot \delta$ for each $i, \delta \in \mathbb{R}_{\geq 0}$:

$$\langle C_0, 0, a, \top, \{x\}\rangle(\langle C_5, 3, c, \top, \{y\}\rangle\langle C_1, 2, d, 1 < y < 2, \emptyset\rangle)^*\langle C_5, 0, b, x = 3, \emptyset\rangle$$

where $\top$ means *true*. In Fig. 1, we give the corresponding WTA.

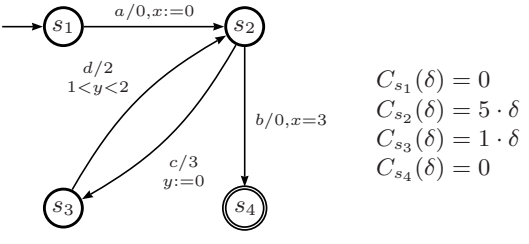



**Fig. 1.** The weighted timed automaton for Example 1

**Recognizable Clock Series.** We say that a clock series $S$ is an $\mathcal{F}$-recognizable $n$-clock series if there is a WTA $\mathcal{A} = (S, S_0, S_f, E, C)$ with $S = \|\mathcal{A}\|$. We use $\mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ to denote the set of all $\mathcal{F}$-recognizable $n$-clock series.

Now, we are ready to present the main theorem of our paper.

**Theorem 1.** *Let $\mathcal{K}$ be a semiring, $\Sigma$ be a finite alphabet, $X = \{x_1, ..., x_n\}$ be a finite set of clock variables and $\mathcal{F}$ be a family of functions from $\mathbb{R}_{\geq 0}$ to $K$. Then the set of $\mathcal{F}$-recognizable $n$-clock series is equal to the set of $\mathcal{F}$-rational $n$-clock series:*

$$\mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle = \mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$$

*Proof.* The two inclusions of this result will follow from Proposition 1 and Proposition 2, respectively, presented in the next two sections.

## 6 Rationality Implies Recognizability

In this section, we prove one inclusion of Theorem 1, namely that any rational $n$-clock series is recognizable. To this end, we will show that the basic $n$-clock series $\mathbb{1}$, $\mathbb{0}$ and $\mathcal{F}$-monomials are recognized by a WTA. Then, we will present new constructions that prove that WTA are closed under addition, Cauchy product and Kleene star iteration.

**Proposition 1.** $\mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle \subseteq \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$.

*Proof.* Follows from Lemmas 4, 5, 7 and 9 given subsequently.

**Lemma 4.** *$\mathbb{1}$, $\mathbb{0}$ and $\mathcal{F}$-monomials in $\mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ are recognizable $n$-clock series.*

*Proof.* $\mathbb{0}$ is the behaviour of the WTA $\mathcal{A}_{\mathbb{0}} = (\{s\}, \{s\}, \emptyset, \emptyset, C)$, and the WTA $\mathcal{A}_{\mathbb{1}} = (\{s\}, \{s\}, \{s\}, \emptyset, C)$ corresponds to the clock series $\mathbb{1}$. In both cases, $C_s \in \mathcal{F}$ is arbitrary. Let $S = \langle c, k, \sigma, \phi, \lambda\rangle$ be an $\mathcal{F}$-monomial in $\mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$, where $c \in \mathcal{F}$, $k \in \mathcal{K}$, $\sigma \in \Sigma$, $\phi \in \Phi(X)$, $\lambda \subseteq X$. We define the WTA $\mathcal{A}_{\langle c,k,\sigma,\phi,\lambda\rangle} = (S, S_0, S_f, E, C)$ where

- $S = \{s_1, s_2\}$
- $S_0 = \{s_1\}$
- $S_f = \{s_2\}$
- $E = \{(s_1, s_2, \sigma, \phi, \lambda)\}$
- $C = \{C_E\} \cup \{C_s : s \in S\}$

where $C_E : E \to K$ is defined by $C_E((s_1, s_2, \sigma, \phi, \lambda)) = k$. Also, let $C_{s_1} = c$, and choose any $C_{s_2} \in \mathcal{F}$. Clearly, $\|\mathcal{A}_{\langle c,k,\sigma,\phi,\lambda\rangle}\| = S$.

The proof of the next lemma can be done as in the case of traditional finite automata by taking a disjoint union of two WTA.

**Lemma 5.** *If $S, T \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$, then $S + T \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$.*

In the following, we give normalization techniques for WTA which will be essential for subsequent constructions of WTA. For showing closure of WTA under the Cauchy product, we need a final-location-normalization. We say that a WTA $\mathcal{A}$ is *final-location-normalized* if there is one single final location, and this location has no outgoing edge.

**Lemma 6.** *If $\mathcal{A}$ is a WTA, then there is a final-location-normalized WTA $\mathcal{A}'$ with $(\|\mathcal{A}\|, w) = (\|\mathcal{A}'\|, w)$ for any $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^+$.*

*Proof.* Let $\mathcal{A} = (S, S_0, S_f, E, C)$ be a WTA.
Define $\mathcal{A}' = (S', S_0', S_f', E \cup E', C \cup C')$, where

- $S' = S \cup \{s_f\}$
- $S_0' = S_0$
- $S_f' = \{s_f\}$

- $E' = \{(s, s_f, \sigma, \phi, \lambda) : \exists s' \in S_f \text{ s.t. } (s, s', \sigma, \phi, \lambda) \in E\}$
- $C' : E' \to K$ is defined by $C'((s, s_f, \sigma, \phi, \lambda)) = \sum_{\substack{s' \in S_f \\ (s, s', \sigma, \phi, \lambda) \in E}} C((s, s', \sigma, \phi, \lambda))$
- $C_{s_f} \in \mathcal{F}$

Intuitively, we redirect all edges going into a final location to the new final location. The weight of each of these new edges must be the sum of the weights of all "equivalent" edges, i.e., edges with the same label, clock constraint and reset set. Using this notion of equivalence, we can show that $(\|\mathcal{A}\|, w) = (\|\mathcal{A}'\|, w)$ for any $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^+$.

**Lemma 7.** *If $S, T \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$, then $S \cdot T \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$.*

*Proof.* We give the construction of the WTA $\mathcal{A}$ such that $\|\mathcal{A}\| = S \cdot T$. For $i = 1, 2$, let $\mathcal{A}_i = (S^i, S_0^i, S_f^i, E^i, C^i)$ such that $\|\mathcal{A}_1\| = S$ and $\|\mathcal{A}_2\| = T$. By Lemma 6, we know that there is a final-location-normalized WTA $\mathcal{A}_N = (S^N, S_0^N, \{s_f\}, E^N, C^N)$ such that $(\|\mathcal{A}_1\|, w) = (\|\mathcal{A}_N\|, w)$ for any $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^+$. Assume $C^N = \{C_{E^N}^N\} \cup \{C_s^N\}_{s \in S^N}$ and $|S_0^1 \cap S_f^1| = m$. Define $\mathcal{A}_{N,2} = (S^N \cup S^2, S_0^N, S_f^2, E^N \cup E^2 \cup E, \{C^N\} \cup \{C^2\} \cup \{C_E\})$ where

- $E = \{(s, s', \sigma, \phi, \lambda) : s' \in S_0^2, (s, s_f, \sigma, \phi, \lambda) \in E^N\}$
- $C_E : E \to K$ is defined by $C_E((s, s', \sigma, \phi, \lambda)) = C_{E^N}^N((s, s_f, \sigma, \phi, \lambda))$ if $(s, s_f, \sigma, \phi, \lambda) \in E^N$

Intuitively, we redirect all edges going into the single final location of $\mathcal{A}_N$ to the initial locations of $\mathcal{A}_2$ and preserve the cost assigned to these edges.

For $i \in \mathbb{N}$, define $\mathcal{A}_{2,i}$ to be an isomorphic copy of $\mathcal{A}_2$ such that its locations and edges are indexed by $i$. Let $\mathcal{A}$ be the disjoint union of $\mathcal{A}_{N,2}$ and $\mathcal{A}_{2,i}$ for $1 \leq i \leq m$. Then, $(\|\mathcal{A}\|, w) = S \cdot T$: $\mathcal{A}_{N,2}$ recognizes precisely the clock words $w$ that are the concatentation of two clock words $w_1$ and $w_2$ accepted by $\mathcal{A}_N$ and $\mathcal{A}_2$, respectively. Words obtained by concatentation of an empty word $w_0 \in \Gamma_n$ and a word $w_2$ recognized by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, have to be treated in a different manner. To overcome the problem that $(\|\mathcal{A}_1\|, w_0) = m \cdot 1$ (i.e., the sum of $m$ summands of $1 \in K$), whereas $(\mathcal{A}_N, w_0) = 0$ due to the construction of $\mathcal{A}_N$, we add $m$ isomorphic copies of $\mathcal{A}_2$. In this way, words of this kind are assigned the correct weight.

For showing closure of WTA under the Kleene star iteration, we need an additional normalization technique. A WTA is *initial-location-normalized* if all initial locations are sources, i.e., have no ingoing edges. Note that, in contrast to the case of classical weighted automata, we do not require to have one single initial state.

**Lemma 8.** *If $\mathcal{A}$ is a WTA, then there is an initial-location-normalized WTA $\mathcal{A}'$ with $(\|\mathcal{A}\|, w) = (\|\mathcal{A}'\|, w)$ for any $w \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^*$.*

*Proof.* Let $\mathcal{A} = (S, S_0, S_f, E, C)$ be a WTA, where $C = \{C_E\} \cup \{C_s : s \in S\}$. For each $s \in S$, let $s'$ be the copy of $s$.
Define $\mathcal{A}' = (S \cup S', S_0', S_f \cup S_f', E \cup E', C \cup C')$, where

- $S' = \{s' : s \in S_0\}$
- $S'_0 = S'$
- $S'_f = \{s' : s \in S_0 \cap S_f\}$
- $E' = \{(s', t, \sigma, \phi, \lambda) : (s, t, \sigma, \phi, \lambda) \in E\}$
- $C' = \{C'_{E'}\} \cup \{C'_{s'} : s' \in S'\}$, where $C'_{E'} : E' \to K$ is defined by $C'_{E'}((s', t, \sigma, \phi, \lambda)) = C_E((s, t, \sigma, \phi, \lambda))$, and $C'_{s'} = C_s$ for any $s \in S$.

The intuituion behind this construction is to create a new initial location $s'$ for every initial location $s \in S_0$ such that $s'$ carries only copies of the outgoing edges of $s$. In particular, no (new) initial location has any ingoing edge. The final states of $\mathcal{A}'$ consist of the final states of $\mathcal{A}$ and of those locations $s'$ whose original location $s$ is both initial and final in $\mathcal{A}$. This choice of new final states $s'$ guarantees that $\mathcal{A}'$ behaves correctly on the empty n-clock words. One can prove $\|\mathcal{A}'\| = \|\mathcal{A}\|$ by establishing a weight-preserving bijective correspondence between the successful runs of $\mathcal{A}$ and $\mathcal{A}'$.

**Corollary 1.** *Let $\mathcal{A}$ be a WTA. Then there is an initial- and final-location-normalized WTA $\mathcal{A}_N$ with $(\|\mathcal{A}_N\|, w) = (\|\mathcal{A}\|, w)$ for any $w \in (\mathbb{R}^{1+n}_{\geq 0})(\Sigma \times \mathbb{R}^{1+n}_{\geq 0})^+$. Moreover, $(\|\mathcal{A}_N\|, w_0) = 0$ for any $w_0 \in \Gamma_n$.*

**Lemma 9.** *If $S \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$ is proper, then $S^* \in \mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$.*

*Proof.* By Corollary 1 there is an initial- and final-location-normalized WTA $\mathcal{A} = (S, S_0, \{s_f\}, E, C)$ with $\|\mathcal{A}\| = S$. We define $\mathcal{A}^* = (S \cup \{s_{0f}\}, S_0 \cup \{s_{0f}\}, \{s_f, s_{0f}\}, E \cup E', C \cup C')$, where

- $s_{0f}$ is a new location (to obtain $(\|\mathcal{A}^*\|, w_0) = 1$ for $w_0 \in \Gamma_n$)
- $E' = \{(s, s', \sigma, \phi, \lambda) : s' \in S_0, (s, s_f, \sigma, \phi, \lambda) \in E\}$
- $C' : E' \to K$ is defined by $C'((s, s', \sigma, \phi, \lambda)) = C_E((s, s_f, \sigma, \phi, \lambda))$ if $(s, s_f, \sigma, \phi, \lambda) \in E$

By a careful analysis of the successful runs of $\mathcal{A}^*$ and their weights, it can be shown that $\|\mathcal{A}^*\| = \|\mathcal{A}\|^*$, which implies the result.

## 7 Recognizability Implies Rationality

In this section, we show that any $n$-clock series recognized by a WTA $\mathcal{A}$ is rational by solving a system of equations induced by $\mathcal{A}$. The solution of the system corresponds to the rational clock series. Before we present the actual result, we give some lemmas. Let $\mathcal{A} = (S, S_0, S_f, E, C)$ be a WTA. For any two locations $s, s' \in S$, we set $\mathcal{A}_{s,s'} = (S, \{s\}, \{s'\}, E, C)$. The following lemma states how we can compute $\|\mathcal{A}\|$.

**Lemma 10.** *If $\mathcal{A} = (S, S_0, S_f, E, C)$ is a WTA, then*

$$\|\mathcal{A}\| = \sum_{(s_0, s_f) \in S_0 \times S_f} \|\mathcal{A}_{s_0, s_f}\|$$

The next lemma shows that for the behaviour of any $\mathcal{A}_{s,s'}$, we can give an equivalent linear equation. This can be proved by decomposing any successful run of $\mathcal{A}_{s,s'}$ after the first discrete transition and replacing the first component by the corresponding monomial WTA, using laws of associativity and distributivity.

**Lemma 11.** *Let $\mathcal{A} = (S, S_0, S_f, E, C)$ be a WTA, and assume that $s_f \in S_f$ is fixed. Then, for any $s \in S$,*

$$\|\mathcal{A}_{s,s_f}\| = \begin{cases} \sum_{(s,s',\sigma,\phi,\lambda)\in E} \|\mathcal{A}_{\langle C_s,k,\sigma,\phi,\lambda\rangle}\| \cdot \|\mathcal{A}_{s',s_f}\| + \mathbb{1} & \text{if } s = s_f \\ \sum_{(s,s',\sigma,\phi,\lambda)\in E} \|\mathcal{A}_{\langle C_s,k,\sigma,\phi,\lambda\rangle}\| \cdot \|\mathcal{A}_{s',s_f}\| & \text{otherwise} \end{cases}$$

*where $k = C_E((s, s', \sigma, \phi, \lambda))$.*

The objective of these lemmas is to provide the basis for building a system of linear equations that represents the behaviour of a given WTA $\mathcal{A}$. The solution of this system correesponds to a rational clock series that is equivalent to the behaviour of $\mathcal{A}$. However, we need to show that it is guaranteed that there is such a solution. The next lemma supplies us with an even stronger result, namely that there is a *unique* solution.

**Lemma 12.** *Let $S, S_1, S_2 \in \mathcal{K}_n\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$, $S_1$ be proper. Then the equation $S = S_1 \cdot S + S_2$ has the unique solution $T = S_1^* \cdot S_2$.*

Finally, we present the crucial property between recognizable and rational clock series. For proving it, we use Lemmas 10, 11 and 12.

**Proposition 2.** $\mathcal{K}_n^{\mathcal{F}-rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle \subseteq \mathcal{K}_n^{\mathcal{F}-rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$.

## 8 Timed Series

As mentioned in Sect. 3, we use the clock semantics for defining a natural concatenation operation. However, research in the real-time community focuses on *timed languages* rather than clock languages. In this section, we show that a Kleene-Schützenberger theorem can be given for the corresponding class of fps, so-called *timed series*.

**Timed Series.** An fps $S : (\Sigma \times \mathbb{R}_{\geq 0})^* \to K$ is called a *timed series*. We denote the set of timed series by $\mathcal{K}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$. We say that a timed series $S \in \mathcal{K}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$ is *recognizable* if there is a WTA $\mathcal{A}$ such that $S = \|\mathcal{A}\|^T$. The set of recognizable timed series will be denoted by $\mathcal{K}^{rec}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$.

**Projection.** The use of timed semantics rather than clock semantics sacrifices some significant information concerning the values of the clock variables that precludes us from defining the notion of rationality for timed series in the same way as for clock series. Therefore, we use the approach of Bouyer and

Petit [16], and introduce a projection that maps clock series to timed series. Let $\pi : (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^* \to (\Sigma \times \mathbb{R}_{\geq 0})^*$ be the partial function defined by $\pi((t_0, \nu_0)(\sigma_1, t_1, \nu_1)...(\sigma_k, t_k, \nu_k)) = (\sigma_1, t_1)...(\sigma_k, t_k)$ if $(t_0, \nu_0) = (0, 0^n)$, undefined otherwise. We extend $\pi$ to a function $\bar{\pi} : \mathcal{K}_n^{rec}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle \to \mathcal{K}^{rec}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle : S \mapsto \bar{\pi}(S)$ where

$$(\bar{\pi}(S), w_T) = \sum_{\substack{w_C \in (\mathbb{R}_{\geq 0}^{1+n})(\Sigma \times \mathbb{R}_{\geq 0}^{1+n})^* \\ \pi(w_C) = w_T}} (S, w_C)$$

for any timed word $w_T \in (\Sigma \times \mathbb{R}_{\geq 0})^*$. Notice that the sum in the equation is finite: for any recognizable timed word, there is only a finite number of $n$-clock words $w_C$ in $\pi^{-1}(w_T)$ such that $(\|\mathcal{A}\|, w_C) \neq 0$, because there is only a finite number of runs on any clock word $w_C$.

**Rational Timed Series.** A timed series $S \in \mathcal{K}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$ is *rational* if it can be defined by a single application of $\bar{\pi}$ to a rational $n$-clock series $T \in \mathcal{K}_n^{rat}\langle\langle \Sigma, \mathbb{R}_{\geq 0}\rangle\rangle$, i.e., $S = \bar{\pi}(T)$. We use $\mathcal{K}^{rat}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$ to mean the set of rational timed series.

The following lemma gives the relation between recognizable timed series and recognizable clock series.

**Lemma 13.** *Let $\mathcal{A}$ be a WTA and $w_T \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ be a timed word. Then* $(\|\mathcal{A}\|^T, w_T) = \big(\bar{\pi}(\|\mathcal{A}\|^C), w_T\big)$.

**Corollary 2.** $\mathcal{K}^{rat}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle = \mathcal{K}^{rec}\langle\langle(\Sigma \times \mathbb{R}_{\geq 0})^*\rangle\rangle$.

*Proof.* The definition of rational timed series and Lemma 13 ensure that both rational and recognizable timed series correspond to a single application of $\bar{\pi}$ to a rational (recognizable, respectively) $n$-clock series. This and Theorem 1 imply the result.

## 9 Conclusion

We have presented a new definition of WTA for modelling consumption of resources, and we have obtained a Kleene-Schützenberger theorem. Our definition over a semiring is more general than definitions given in the literature so far and emphasizes the relation to weighted automata. The Kleene-Schützenberger theorem for WTA provides an alternative characterization of the possible behaviours of WTA. The crucial point for obtaining this result was to find new normalization techniques that allow for the construction of Cauchy product- and Kleene star-WTA. We point out that due to the cost functions assigned to the locations it is not possible to use standard normalization techniques for weighted automata.

Apart from being a fundamental theoretical result, Kleene's theorem is also of practical interest. Kleene's theorem for the set of regular languages is used for automata-based verification purposes: the rational expression is considered

to be the specification of the system, which, by the Kleene theorem, can be transformed into an equivalent finite automaton. It is a fascinating challenge to investigate whether our result can be used in the same manner.

In our paper, we have shown that WTA are closed under addition, Cauchy product and Kleene star iteration. Moreover, the corresponding constructions are effective. It is of great practical interest whether we get similar positive results for other standard properties and decidability problems. In particular, we want to investigate the emptiness problem, i.e., given a WTA $\mathcal{A}$, whether $\|\mathcal{A}\| = \mathbb{0}$. There is a good reason to hope for a positive result, as both the emptiness problem of timed automata and weighted automata is decidable, where the latter result applies to weighted automata where the semiring is a field.

Another interesting direction for future work is to consider whether there is a Büchi-type theorem for WTA, i.e., are weighted timed automata expressively equivalent to some weighted timed version of monadic second-order logic. This should combine methods of Wilke [31] and Droste and Gastin [19]. The present closure results for rational operations provide a promising starting point.

## References


1. Alur, R., Bernadsky, M., Madhusudan, P.: Optimal reachability in weighted timed games. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 122–133. Springer, Heidelberg (2004)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)
4. Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 1–24. Springer, Heidelberg (2004)
5. Asarin, E., Caspi, P., Maler, O.: A Kleene theorem for timed automata. In: LICS 1997, pp. 160–171. IEEE Computer Society Press, Los Alamitos (1997)
6. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. Journal of the ACM 49(2), 172–206 (2002)
7. Asarin, E., Dima, C.: Balanced timed regular expressions. In: Vogler, W., Larsen, K.G. (eds.) MTCS. ENTCS, vol. 68, pp. 16–33. Elsevier, Amsterdam (2002)
8. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
9. Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.): HSCC 2001. LNCS, vol. 2034. Springer, Heidelberg (2001)
10. Berstel, J., Reutenauer, C.: Rational Series and their Languages. Springer, New York, USA (1988)
11. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.-F.: On the optimal reachability problem on weighted timed automata. Formal Methods in System Design 31(2), 135–175 (2007)

12. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Inf. Process. Lett. 98(5), 188–194 (2006)
13. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design (to appear, 2007)
14. Bouyer, P., Larsen, K.G., Markey, N.: Model-checking one-clock priced timed automata. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 108–122. Springer, Heidelberg (2007)
15. Bouyer, P., Petit, A.: Decomposition and composition of timed automata. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 210–219. Springer, Heidelberg (1999)
16. Bouyer, P., Petit, A.: A Kleene/Büchi-like theorem for clock languages. J. Autom. Lang. Comb. 7(2), 167–186 (2001)
17. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
18. Brihaye, T., Bruyère, V., Raskin, J.-F.: Model-checking weighted timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 277–292. Springer, Heidelberg (2004)
19. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1–2), 69–86 (2007)
20. Fox, M., Long, D.: Modelling mixed discrete-continuous domains for planning. Journal of AI Research 27, 235–297 (2006)
21. Henzinger, T.: The theory of hybrid automata. In: LICS 1996, pp. 278–292. IEEE Computer Society Press, Los Alamitos (1996)
22. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer 1(1–2), 110–122 (1997)
23. Kristoffersen, K., Larsen, K., Pettersson, P., Weise, C.: VHS Case Study 1 - Experimental Batch Plant using UPPAAL, BRICS, University of Aalborg, Denmark (May 1999)
24. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. EATCS Monographs on Theoretical Computer Science, vol. 5. Springer, Berlin (1986)
25. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
26. Larsen, K.G., Rasmussen, J.I.: Optimal conditional reachability for multi-priced timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 234–249. Springer, Heidelberg (2005)
27. Niebert, P., Yovine, S.: Computing optimal operation schemes for chemical plants in multi-batch mode. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, Springer, Heidelberg (2000)
28. Illum Rasmussen, J., Larsen, K.G., Subramani, K.: Resource-optimal scheduling using priced timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 220–235. Springer, Heidelberg (2004)
29. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Springer, New York (1978)
30. Schützenberger, M.P.: On the definition of a family of automata. Information and Control 4, 245–270 (1961)
31. Wilke, T.: Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 694–715. Springer, Heidelberg (1994)

# Robust Analysis of Timed Automata *Via* Channel Machines*

Patricia Bouyer[1,2,**], Nicolas Markey[1], and Pierre-Alain Reynier[3,***]

[1] LSV, CNRS & ENS de Cachan, France
[2] Oxford University Computing Laboratory, UK
[3] Université Libre de Bruxelles, Belgium
{bouyer,markey}@lsv.ens-cachan.fr, reynier@ulb.ac.be

**Abstract.** Whereas formal verification of timed systems has become a very active field of research, the idealised mathematical semantics of timed automata cannot be faithfully implemented. Several works have thus focused on a modified semantics of timed automata which ensures implementability, and robust model-checking algorithms for safety, and later LTL properties have been designed. Recently, a new approach has been proposed, which reduces (standard) model-checking of timed automata to other verification problems on channel machines. Thanks to a new encoding of the modified semantics as a network of timed systems, we propose an original combination of both approaches, and prove that robust model-checking for coFlat-MTL, a large fragment of MTL, is EXPSPACE-Complete.

## 1 Introduction

*Verification of real-time systems.* In the last thirty years, formal verification of reactive systems has become a very active field of research in computer science. It aims at checking that (the model of) a system satisfies (a formula expressing) its specifications. The importance of taking real-time constraints into account in verification has quickly been understood, and the model of timed automata [2] has become one of the most established models for real-time systems, with a well-studied underlying theory, the development of mature model-checking tools (Uppaal [21], Kronos [11], ...), and numerous success stories.

*Implementation of real-time systems.* Implementing mathematical models on physical machines is an important step for applying theoretical results on practical examples. This step is well-understood for many untimed models that have been studied (*e.g.*, finite automata, pushdown automata). In the timed setting, while timed automata are widely-accepted as a framework for modelling real-time aspects of systems, it is known that they cannot be faithfully implemented

* Partly supported by project DOTS (ANR-06-SETIN-003).
** Partly supported by a Marie Curie fellowship (European Commission).
*** Partly supported by a Lavoisier fellowship (French Ministry of Foreign Affairs).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 157–171, 2008.

on finite-speed CPUs [12]. Studying the "implementability" of timed automata is thus a challenging question of obvious theoretical and practical interest.

*A semantical approach.* In [16], a new semantics for timed automata is defined that takes into account the digital aspects of the hardware on which the automaton is being executed. A timed automaton is then said to be *implementable* if, under this new semantics, the behaviours of this automaton satisfies its specifications. In order to study it efficiently, this semantics is over-approximated by the *AASAP* (for *Almost ASAP*), which consists in "enlarging" the constraints on the clocks by some parameter $\delta$. For instance, "$x \in [a, b]$" is transformed into "$x \in [a-\delta, b+\delta]$". Implementability can be ensured by establishing the existence of some positive $\delta$ for which the AASAP semantics meets the specification. The decidability of this "*robust model-checking*" problem for specifications given as LTL formula has already been solved (first basic safety properties in [14] and then full LTL [9]) using a graph algorithm on the region automaton abstraction.

*Timed temporal logics.* Until recently [23], linear-time timed temporal logics were mostly considered as undecidable, and only MITL, the fragment without punctuality of MTL [20], was recognised as really tractable and useful [3]. Very recently [7], another fragment of MTL, called coFlat-MTL, has been defined, whose model-checking is EXPSPACE-Complete. The decidability of this logic relies on a completely original method using channel machines.

*Our contribution.* Inspired by the channel machine approach of [7], we propose a new techniques to robust model-checking of linear-time timed temporal logics. It is based on the construction of a network of timed systems which captures the AASAP semantics, and which can be expressed as a channel machine. Based on this approach, we prove that the robust model-checking of coFlat-MTL is EXPSPACE-Complete, *i.e.*, not more expensive than under the classical semantics. It is worth noticing that coFlat-MTL includes LTL, our result thus encompasses the previously shown decidability results in that framework.

*Related work.* Since its definition in [16], the approach based on the AASAP semantics has received much attention, and even other kind of perturbations like the drift of clocks, have been studied [26,15,4,17]. In the case of safety properties and under some natural assumptions, this approach is equivalent to constraint enlargement and relies on similar techniques, as proved in [15]. Also, several works have proposed a symbolic, zone-based approach to the classical region-based algorithm for robustness [13,27,17]. This approach using the AASAP semantics contrasts with a modelling-based solution proposed in [1], where the behaviour of the platform is modelled as a timed automaton. Last, many other notions of "robustness" have been proposed in the literature in order to relax the mathematical idealisation of the semantics of timed automata [19,22,6,5]. Those approaches are different from ours, since they roughly consist in dropping "isolated" or "unlikely" executions. Also note that robustness issues have also been handled in the untimed case, but are even further from our approach [18].

*Outline of the paper.* We introduce the setting in Section 2. Section 3 contains our construction: we first turn the robust semantics of timed automata into networks of timed systems (Section 3.1), which are then encoded as channel automata (Section 3.2). We then explain how the resulting channel automata are used for Bounded-MTL and coFlat-MTL model-checking (Section 3.3). By lack of space, proofs are omitted and can be found in the research report [10].

## 2 Preliminaries

We present here the model of timed automata, some linear-time timed temporal logics, and the model of channel automata, which is central to our approach.

### 2.1 Timed Automata

Let $X$ be a finite set of *clock* variables. We denote by $\mathcal{G}(X)$ the set of *clock constraints* generated by the grammar $g ::= g \wedge g \mid\ x \sim k$, where $x \in X$, $k \in \mathbb{N}$, and $\sim \in \{\leq, \geq\}$. A *(clock) valuation* $v$ for $X$ is an element of $\mathbb{R}_+^X$. If $v \in \mathbb{R}_+^X$ and $t \in \mathbb{R}_+$, we write $v + t$ for the valuation assigning $v(x) + t$ to every clock $x \in X$. If $r \subseteq X$, $v[r \leftarrow 0]$ denotes the valuation assigning 0 to every clock in $r$ and $v(x)$ to every clock in $X \setminus r$.

A *timed automaton* is a tuple $\mathcal{A} = (L, \ell_0, X, I, \Sigma, T)$ where $L$ is a finite set of locations, $\ell_0 \in L$ is an initial location, $X$ is a finite set of clocks, $I\colon L \to \mathcal{G}(X)$ labels each location with its invariant, $\Sigma$ is a finite set of actions, and $T \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the set of transitions. Given a parameter value $\delta \in \mathbb{R}_{\geq 0}$, whether a valuation $v \in \mathbb{R}_+^X$ satisfies a constraint $g$ within $\delta$, written $v \models_\delta g$, is defined inductively as follows:

$$\begin{cases} v \models_\delta x \leq k & \text{iff } \ v(x) \leq k + \delta \\ v \models_\delta x \geq k & \text{iff } \ v(x) \geq k - \delta \\ v \models_\delta g_1 \wedge g_2 & \text{iff } \ v \models_\delta g_1 \text{ and } v \models_\delta g_2 \end{cases}$$

Following [16], we define a parameterised semantics for $\mathcal{A}$ as a timed transition system $[\![\mathcal{A}]\!]_\delta = \langle S, S_0, \Sigma, \to_\delta \rangle$. The set $S$ of states of $[\![\mathcal{A}]\!]_\delta$ is $\{(\ell, v) \in L \times \mathbb{R}_+^X \mid v \models_\delta I(\ell)\}$, with $S_0 = \{(\ell_0, v_0) \mid v_0(x) = 0 \text{ for all } x \in X\}$. A transition in $[\![\mathcal{A}]\!]_\delta$ is composed either of a delay move $(\ell, v) \xrightarrow{d}_\delta (\ell, v + d)$, with $d \in \mathbb{R}_+$, when both $v$ and $v+d$ satisfy the invariant $I(\ell)$ within $\delta$, or of a discrete move $(\ell, v) \xrightarrow{\sigma}_\delta (\ell', v')$ when there exists a transition $(\ell, g, \sigma, r, \ell') \in T$ with $v \models_\delta g$, $v' = v[r \leftarrow 0]$, and $v' \models_\delta I(\ell')$. The graph $[\![\mathcal{A}]\!]_\delta$ is thus an infinite transition system. Notice that, in the definitions above, the standard semantics of timed automata can be recovered by letting $\delta = 0$. In that case, we omit the subscript $\delta$.

A *run* of $[\![\mathcal{A}]\!]_\delta$ is an infinite sequence $(\ell_0, v_0) \xrightarrow{d_0}_\delta (\ell_0, v_0 + d_0) \xrightarrow{\sigma_0}_\delta (\ell_1, v_1) \xrightarrow{d_1}_\delta (\ell_1, v_1 + d_1) \ldots$ where for each $i \geq 0$, $d_i \in \mathbb{R}_+$. A *timed word* $w$ is an infinite sequence $(\sigma_i, t_i)_{i \in \mathbb{N}}$ where $\sigma_i \in \Sigma$ and $t_i \in \mathbb{R}_+$ for each $i \geq 0$, and such that the sequence $(t_i)_{i \in \mathbb{N}}$ is non-decreasing and diverges to infinity. The timed word $w$ is read on the run $(\ell_0, v_0) \xrightarrow{d_0}_\delta (\ell_0, v_0 + d_0) \xrightarrow{\sigma_0}_\delta (\ell_1, v_1) \xrightarrow{d_1}_\delta (\ell_1, v_1 + d_1) \ldots$

whenever $t_i = \sum_{j \leq i} d_j$ for every $i \in \mathbb{N}$. We write $\mathcal{L}(\llbracket \mathcal{A} \rrbracket_\delta)$ for the set of timed words that can be read on a run of $\llbracket \mathcal{A} \rrbracket_\delta$ starting in $(\ell_0, v_0)$. More generally, we write $\mathcal{L}(\llbracket \mathcal{A} \rrbracket_\delta^{(\ell,v)})$ for the set of timed words than can be read starting from $(\ell, v)$.

Since our results rely on the results of [14,9], we require that our timed automata satisfy the following requirements: *(i)* constraints in guards and invariants only involve non-strict inequalities; *(ii)* all the clocks are always bounded by some constant $M$; *(iii)* all the cycles in the region graph are *progress cycles*, *i.e.*, all the clocks are reset along those cycles. In addition, we require that the timed automata are *non-blocking*, in the sense that from every state, an action transition will eventually become firable: for every $(\ell, v) \in L \times \mathbb{R}_+^X$ such that $v \models_0 I(\ell)$, there exists $d \in \mathbb{R}_+^X$ and $\sigma \in \Sigma$ such that $(\ell, v) \xrightarrow{d}_0 (\ell, v+d) \xrightarrow{\sigma}_0 (\ell', v')$ for some $(\ell', v') \in L \times \mathbb{R}_+^X$.

### 2.2 Implementability and Robustness of Timed Automata

The parameterised semantics defined above (referred to as "enlarged semantics" in the sequel), has been defined in [16] in order to study the *implementability* of timed systems. Indeed, timed automata are governed by a mathematical, idealised semantics, which does not fit with the digital, imprecise nature of the hardware on which they will possibly be implemented. An *implementation semantics* has thus been defined in [16] in order to take the hardware into account: that semantics models a digital CPU which, every $\delta_P$ time units (at most), reads the value of the digital clock (updated every $\delta_L$ time units), computes the values of the guards, and fires one of the available transitions. We write $\llbracket \mathcal{A} \rrbracket^{\delta_P, \delta_L}$ for the resulting transition system, and $\mathcal{L}(\llbracket \mathcal{A} \rrbracket^{\delta_P, \delta_L})$ for the corresponding set of timed words. Given a (linear-time) property $\mathcal{P}$, *i.e.*, a set of accepted timed words, a timed automaton $\mathcal{A}$ is said to be implementable w.r.t. $\mathcal{P}$ iff $\mathcal{L}(\llbracket \mathcal{A} \rrbracket^{\delta_P, \delta_L}) \subseteq \mathcal{P}$ for some positive values of $\delta_P$ and $\delta_L$.

As proved in [16], the enlarged semantics simulates the implementation semantics as soon as $\delta > 4\delta_P + 3\delta_L$. As a consequence, it is sufficient to check the existence of $\delta > 0$ such that $\mathcal{L}(\llbracket \mathcal{A} \rrbracket_\delta) \subseteq \mathcal{P}$ in order to ensure implementability of $\mathcal{A}$ w.r.t. $\mathcal{P}$. We follow this idea in the sequel, and study the *robust satisfaction* relation: a timed automaton *robustly satisfies* a linear-time property $\mathcal{P}$, written $\mathcal{A} \models\!\models \mathcal{P}$, whenever $\mathcal{L}(\llbracket \mathcal{A} \rrbracket_\delta) \subseteq \mathcal{P}$ for some $\delta > 0$.

### 2.3 Some Subclasses of Metric Temporal Logic

Linear-time properties are often defined *via* temporal logic formulae. In this paper, we focus on subclasses of MTL (Metric Temporal Logic) [20].

Fix a finite, non-empty alphabet $\Sigma$. The syntax of MTL over $\Sigma$ is defined by the following grammar:

$$\mathsf{MTL} \ni \varphi ::= \sigma \mid \neg\sigma \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathbf{U}_I \varphi \mid \varphi \widetilde{\mathbf{U}}_I \varphi$$

where $\sigma \in \Sigma$, and $I$ is an interval of $\mathbb{R}^+$ with bounds in $\mathbb{N} \cup \{\infty\}$. MTL formulae are interpreted over timed words. Let $w = (\sigma_i, t_i)_{i \geq 0}$ be a timed word, and $p \in \mathbb{N}$.

The (pointwise) semantics of MTL is defined recursively as follows (we omit Boolean operations):

$$\begin{aligned} w,p \models \sigma &\Leftrightarrow \sigma_p = \sigma \\ w,p \models \varphi \,\mathbf{U}_I\, \psi &\Leftrightarrow \exists i>0 \text{ s.t. } w,p+i \models \psi,\ t_{p+i}-t_p \in I \\ &\quad \text{and } \forall 0<j<i,\ w,p+j \models \varphi \\ w,p \models \varphi \,\widetilde{\mathbf{U}}_I\, \psi &\Leftrightarrow w,p \models \neg\Big((\neg\varphi)\,\mathbf{U}_I\,(\neg\psi)\Big). \end{aligned}$$

If $w,0 \models \varphi$, we write $w \models \varphi$. Following the discussion above, we write $\mathcal{A} \models\!\models \varphi$ if, for some $\delta > 0$, we have $w \models \varphi$ for every $w \in \mathcal{L}([\![\mathcal{A}]\!]_\delta)$.

Additional operators, such as $\mathtt{tt}$ (true), $\mathtt{ff}$ (false), $\widetilde{\Rightarrow}$, $\Leftrightarrow$, $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{X}$, are defined in the usual way: $\mathbf{F}_I\,\varphi \equiv \mathtt{tt}\,\mathbf{U}_I\,\varphi$, $\mathbf{G}_I\,\varphi \equiv \mathtt{ff}\,\widetilde{\mathbf{U}}_I\,\varphi$, and $\mathbf{X}_I\,\varphi \equiv \mathtt{ff}\,\mathbf{U}_I\,\varphi$. We also use pseudo-arithmetic expressions to denote intervals. For example, '$=1$' denotes the singleton $\{1\}$.

Following [7], we identify the following syntactic fragments of MTL: LTL [25] can be considered as the fragment of MTL in which modalities are not constrained (*i.e.*, where $\mathbb{R}_+$ is the only constraining interval); Bounded-MTL is the fragment of MTL in which all interval constraints are bounded: observe that Bounded-MTL disallows unconstrained modalities, and is in particular not suitable to express invariance properties (the most basic type of temporal specifications). The fragment coFlat-MTL[1] has then been defined to remedy this deficiency:

$$\text{coFlat-MTL} \ni \varphi ::= \sigma \mid \neg\sigma \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi\,\mathbf{U}_J\,\varphi \mid \varphi\,\mathbf{U}_I\,\underline{\psi} \mid \varphi\,\widetilde{\mathbf{U}}_J\,\varphi \mid \underline{\psi}\,\widetilde{\mathbf{U}}_I\,\varphi$$

where $J$ ranges over the set of bounded intervals, $I$ over the set of all intervals, and the underlined formula $\underline{\psi}$ ranges over LTL.

One immediately sees that coFlat-MTL subsumes both LTL and Bounded-MTL, but it is not closed under negation. However, it is closed under invariance, and can then express *e.g.* bounded response-time or even richer formulae such as $\mathbf{G}\,(\mathit{request} \Rightarrow \mathbf{F}_{\leq 5}\,(\mathit{acquire} \wedge \mathbf{F}_{=1}\,\mathit{release}))$.

### 2.4 Channel Automata

Channel automata are an interesting formalism for reasoning about alternating timed automata, which has been used in [7] to prove the EXPSPACE-membership of the model-checking problem for coFlat-MTL. We only give the definition and necessary results here, and refer to [10] for some more intuition.

A *channel automaton with renaming and occurrence testing* (CAROT for short) is a tuple $\mathcal{C} = (S, s_0, \Sigma, \delta, F)$, where $S$ is a finite set of control states, $s_0 \in S$ is the initial control state, $F \subseteq S$ is a set of accepting control states, $\Sigma$ is a finite alphabet, $\delta \subseteq S \times Op \times S$ is the set of transition rules, where

$$Op = \{\sigma!, \sigma? \mid \sigma \in \Sigma\} \cup \{\mathit{zero}?(\sigma) \mid \sigma \in \Sigma\} \cup \{R \mid R \subseteq \Sigma \times \Sigma\}$$

is the set of operations. Given a rule $\tau = (s, \alpha, s') \in \delta$, we define $op(\tau) = \alpha$.

[1] We do not explain this terminology here, and refer to [7] for deeper considerations.

Intuitively, operations $\sigma!$ and $\sigma?$ are the classical write and read operations, $zero?(\sigma)$ is a guard for testing the occurrence of $\sigma$ in the channel, and $R \subseteq \Sigma \times \Sigma$ is interpreted as a global renaming. An *end-of-channel* marker can be used to count the number of times the whole channel is read: it suffices to add, from each state of the CAROT, an outgoing transition reading $\triangleright$, immediately followed by a transition writing $\triangleright$. That way, there is always a unique copy of $\triangleright$ on the channel (except when it has just been read). The number of transitions writing $\triangleright$ along a computation $\varrho$ is the number of *cycles* of $\varrho$, denoted by $cycles(\varrho)$. In the sequel, the CAROTs are assumed to contain the end-of-channel marker $\triangleright$, and to have all their states equipped with a loop reading and writing that symbol.

We consider in the sequel a restricted version of the reachability problem for CAROTs, where we impose a bound on the "time" (measured here as the number of cycles of the channel): the *cycle-bounded reachability problem* for CAROTs is defined as follows: given a CAROT $\mathcal{C}$, an input word $w \in \Sigma^*$, and a cycle bound $h$, does $\mathcal{C}$ have an accepting computation $\varrho$ on $w$ with $cycles(\varrho) \leqslant h$?

In [7], a non-deterministic procedure is presented to check the existence of an accepting cycle-bounded computation using only polynomial space in the *value* of the cycle bound and in the size of the CAROT:

**Theorem 1.** *The cycle-bounded reachability problem for CAROTs is solvable in polynomial space in the size of the channel automaton, the size of the input word, and the value of the cycle bound.*

*Remark.* Note that the algorithm could easily be adapted to cope with the cycle-bounded reachability between two global states (*i.e.*, control state and content of the channel): it suffices to first set the initial content of the channel, and to add transitions that would, from any point, run one more cycle of the channel and store its whole content in the location.

## 3 Robust Model-Checking of coFlat-MTL

In this section, we prove the main results of this paper, namely that the robust model-checking problem can be expressed using CAROTs, and then that the robust model-checking problem of coFlat-MTL is EXPSPACE-Complete. EXPSPACE-Hardness is a consequence of the EXPSPACE-Hardness of the satisfiability problem for Bounded-MTL [7]. EXPSPACE-membership is more involved and will be done in several steps:

1. We first construct a family of networks of timed systems that captures the enlarged semantics of timed automata (Subsection 3.1).
2. We then transform this family of networks into a CAROT, that will simulate the joint behaviours of those networks of timed systems with an alternating timed automaton representing the property we want to robustly verify (Subsection 3.2).
3. Finally, we use the CAROT to design a decidability algorithm for the robust model-checking problem, first for Bounded-MTL, and then for coFlat-MTL (Subsection 3.3).

For the rest of the paper, we fix a timed automaton $\mathcal{A} = (L, \ell_0, X, I, \Sigma, T)$.

### 3.1 From Robustness to Networks of Timed Systems

In this subsection, we transform the robust model-checking problem into a model-checking problem in an infinite family of timed systems. The correctness of the transformation relies on simulation relations between timed transition systems: given two timed transition systems $\mathcal{T}_i = (S_i, \Sigma, \to_i)$ for $i = 1, 2$, we say that $\mathcal{T}_2$ *simulates* $\mathcal{T}_1$ whenever there exists a non-empty relation $\mathfrak{R} \subseteq S_1 \times S_2$ such that $(s_1, s_2) \in \mathfrak{R}$ and $s_1 \xrightarrow{\alpha}_1 s_1'$ with $\alpha \in \Sigma \cup \mathbb{R}_+$ implies $s_2 \xrightarrow{\alpha}_2 s_2'$ for some $s_2' \in S_2$ with $(s_1', s_2') \in \mathfrak{R}$. We then write $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$.

Let $n$ be an integer; we denote by $\mathcal{B}^n$ the timed network composed of the following components (which are not standard timed automata because of the use of disjunction and fractional parts in the guards):

• for each $0 \leq i < n$, $\mathcal{B}_i$ is the timed automaton depicted on Fig. 1, where $x_i$ is a fresh clock not belonging to $X$, and $[x_i \leq 1]$ is an invariant forbidding the clock $x_i$ become larger than 1. That way, this automaton is forced to fire its transition when the value of $x_i$ reaches 1. We call such an automaton a $\Delta$-*automaton* in the sequel. In the following, we will take indices of clocks $x_i$ modulo $n$, and in particular $x_{i+1} = x_0$ whenever $i = n - 1$.

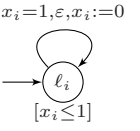



**Fig. 1.** Automaton $\mathcal{B}_i$

• the timed automaton $\mathcal{B}$, built from $\mathcal{A}$ by modifying the guards and invariants as follows: each constraint of the form $x \leq k$ is replaced with

$$(x < k+1) \ \wedge \ \Big(x > k \ \Rightarrow \bigvee_{0 \leq i < n} \{x\} \leq x_{i+1} < x_{i-1}\Big)$$

and each constraint of the form $x \geq k$ is replaced with

$$(x > k-1) \ \wedge \ \Big(x < k \ \Rightarrow \bigvee_{0 \leq i < n} \{x\} \geq x_{i-1} > x_{i+1}\Big)$$

We need to explain when a valuation $v$ satisfies these "extended" guards. Boolean operators are handled in the natural way, and $\{x\}$ is intended to denote the fractional part of the value of clock $x$.

It naturally defines a timed transition system $[\![\mathcal{B}^n]\!]$, as the synchronised product (synchronised because of the time) of all components. Denoting by $X_n = X \cup \{x_i \mid 0 \leq i < n\}$ the set of clocks of $\mathcal{B}^n$, a configuration of $[\![\mathcal{B}^n]\!]$ can be described by a pair $(\ell, v)$ where $\ell \in L$ and $v \in \mathbb{R}_+^{X_n}$ (each $\Delta$-automaton has a single location). We write $u_n$ for the valuation over $X_n$ assigning 0 to clocks in $X$ and $\frac{i}{n}$ to every clock $x_i$ for $0 \leq i < n$. The initial configuration of this timed network is the pair $(\ell_0, u_n)$. Delay and action transitions are defined naturally in the synchronised products of all the components. However in the following we will hide $\varepsilon$-moves due to the components $\mathcal{B}_i$. Thus, in $[\![\mathcal{B}^n]\!]$, we write $(\ell, v) \overset{t}{\Rightarrow} (\ell, v')$ for an

interleaving of delay transitions (which sum up to $t$) and of $\varepsilon$-moves in the $\mathcal{B}_i$'s. For uniformity, we write $\stackrel{\sigma}{\Rightarrow}$ for $\sigma$-moves in $[\![\mathcal{B}^n]\!]$. In the sequel, simulation relations assume the relation $\Rightarrow$ in $[\![\mathcal{B}^n]\!]$, and the simple transition relation $\rightarrow_\delta$ in $[\![\mathcal{A}]\!]_\delta$. In the same way, the intended language accepted by $[\![\mathcal{B}^n]\!]$ should ignore $\varepsilon$-transitions. In other words, it should also be defined using the relation $\Rightarrow$ as the transition relation:

$$\mathcal{L}([\![\mathcal{B}^n]\!]) = \{w = (\sigma_i, t_i)_{i\in\mathbb{N}} \mid \exists (\ell_0, u_n) \stackrel{d_0}{\Longrightarrow} (\ell_0, u') \stackrel{\sigma_0}{\Longrightarrow} (\ell_1, u'') \stackrel{d_1}{\Longrightarrow} \cdots \in [\![\mathcal{B}^n]\!] \text{ s.t. } \forall i \in \mathbb{N},\ t_i = \sum_{j\le i} d_j\}$$

**Lemma 2.** *For every $n \geq 3$, $[\![\mathcal{A}]\!]_{\frac{1}{n}} \sqsubseteq [\![\mathcal{B}^n]\!] \sqsubseteq [\![\mathcal{A}]\!]_{\frac{2}{n}}$.*

With the previous definition, the simulation results can be stated in terms of language inclusion (as initial states are preserved by the exhibited simulation relations): for every $n \geq 3$, $\mathcal{L}([\![\mathcal{A}]\!]_{\frac{1}{n}}) \subseteq \mathcal{L}([\![\mathcal{B}^n]\!]) \subseteq \mathcal{L}([\![\mathcal{A}]\!]_{\frac{2}{n}})$.

**Theorem 3.** *Let $\varphi \in \mathsf{MTL}$. Then, $\mathcal{A} \models \varphi \Leftrightarrow \exists n \geq 3$ s.t. $[\![\mathcal{B}^n]\!] \models \varphi$.*

### 3.2 From Networks of Timed Systems to CAROTs

Extending the approach of [7], the CAROT we build is such that it accepts joint executions of the network of timed systems we just built (and not of a single timed automaton as in [7]) and of the alternating timed automaton corresponding to the negation of the coFlat-MTL formula we want to verify. In order to handle arbitrarily many components in the network, and to deal with "extended" guards (*i.e.*, with disjunctions and fractional parts), the construction attached to the network of timed systems needs to be deeply modified. In a first step, we describe a CAROT that only encodes the behaviours of the network of timed systems.

Let $M$ be the maximal constant appearing in the automaton $\mathcal{A}$. Then $M+1$ is larger than or equal to the maximal constant of any $\mathcal{B}^n$. We assume that the clocks of $\mathcal{A}$ (and $\mathcal{B}^n$) take their values in $[0, M+1] \cup \{\bot\}$, where $\bot$ is a special value (intended to represent any value larger than $M+1$). We write $\mathsf{Reg}$ for the set $\{0, 1, \ldots, M+1, \bot\}$ and $\Lambda = \wp(L \times X \times \mathsf{Reg})$.

A configuration $(\ell, v) \in L \times \mathbb{R}_+^{X_n}$ of the network of timed systems is encoded as the element $C_{(\ell,v)} = \{(\ell, x, v(x)) \mid x \in X_n\}$, and partitioned into a sequence of disjoint subsets $C_0$, $C_1$, ..., $C_p$, $C_\bot$, obtained using standard region techniques. More precisely, $C_0$ (resp. $C_\bot$) contains elements whose fractional part is 0 (resp. whose value is $\bot$) and the others $C_i$ gather elements with the same fractional part and are sorted according to the increasing order of fractional parts. We then let $H(C_{(\ell,v)}) = \big(\mathsf{reg}(C_0),\ \mathsf{reg}(C_\bot),\ \mathsf{reg}(C_1)\mathsf{reg}(C_2)\ldots\mathsf{reg}(C_p)\big) \in \Lambda \times \Lambda \times \Lambda^*$, where we write $\mathsf{reg}(C)$ for $\{(\ell, x, \mathsf{reg}(v)) \mid (\ell, x, v) \in C\}$, with $\mathsf{reg}(v)$ the largest element of $\mathsf{Reg}$ smaller than or equal to $v$.

Using the abstraction function $H$, it is possible to define a discrete transition system $\mathcal{T}_H^n$ which abstracts away precise timing information, but which simulates

the behaviours of $\mathcal{B}^n$. The abstraction function also provides an equivalence relation $\equiv$ on configurations: $C \equiv C'$ iff $H(C) = H(C')$. Extending straightforwardly a result of [24] (regions are compatible with our extended guards), we have:

**Lemma 4.** *The equivalence relation $\equiv$ is a time-abstract*[2] *bisimulation.*

The CAROT we will build is based on the above discrete transition system. More precisely, the intended encoding of a configuration in the CAROT is the following: the integral values of the clocks of $\mathcal{A}$ are stored in the discrete state of the CAROT, as well as the sets $C_0$ and $C_\perp$, and the current location $\ell$. The other $C_i$'s are stored on the channel, from $C_1$ (recently written on the channel) to $C_p$ (the next item to be read from the channel).

In order to use the same CAROT to simulate the network of timed system with any number of $\Delta$-automata, we abstract the name of clocks $x_i$ in our encoding, representing them on the channel by a new symbol $\Delta$. Since, at any time, at most one of the $x_i$'s can have integral value, and since we only need to know the order of the $x_i$'s in order to evaluate guards of $\mathcal{B}^n$, the amount of information to be stored in the location of the CAROT does not depend on the number of $\Delta$-automata in the network.

*Example 5.* Consider for instance a configuration $C$ encoded by the word

$$H(C) = \Big(\{(\ell, x, 3), (\ell, x_2, 0)\},\ \{(\ell, z, \perp)\},\ \{(\ell, x_0, 0)\} \cdot \{(\ell, y, 1), (\ell, x_1, 0)\}\Big).$$

We assume that the maximal constant $M$ is 3. The encoding of the delay-successor of $C$ is obtained by cycling around the letters (except the last one) of the word (and increasing the values of the regions accordingly). Writing $\varnothing$ for the empty set, the first delay successor of $H(C)$ is

$$\Big(\varnothing,\ \{(\ell, z, \perp)\},\ \{(\ell, x, 3), (\ell, x_2, 0)\} \cdot \{(\ell, x_0, 0)\} \cdot \{(\ell, y, 1), (\ell, x_1, 0)\}\Big).$$

The next delay successor is

$$\Big(\{(\ell, y, 2), (\ell, x_1, 0)\},\ \{(\ell, z, \perp)\},\ \{(\ell, x, 3), (\ell, x_2, 0)\} \cdot \{(\ell, x_0, 0)\}\Big).$$

Note that, in that second delay transition, we have reset clock $x_1$ when it has reached 1, and the integral part of $y$ has increased to 2. The configuration $H(C)$ would be encoded as depicted on Fig. 2 (where we write to the left and read from the right of the channel). With respect to the channel, the first delay transition is performed through the write operation '$\langle x\Delta\rangle$!'. As in [7], it is worth noticing that a cycle of the CAROT corresponds to one time unit elapsing.

Furthermore, to simulate the extended guards used in $\mathcal{B}^n$, we need some additional information about the position of clocks of $\mathcal{A}$ w.r.t. symbols $\Delta$. As we have already seen, a clock $x$ verifies a constraint $\{x\} \leq x_{i+1} < x_{i-1}$ iff its fractional part is smaller than one of the two smallest clocks $x_j$. In our simulation, this corresponds as being "before" the second symbol $\Delta$ on the channel. And

[2] This means that precise delays of time-elapsing transitions are abstracted away.

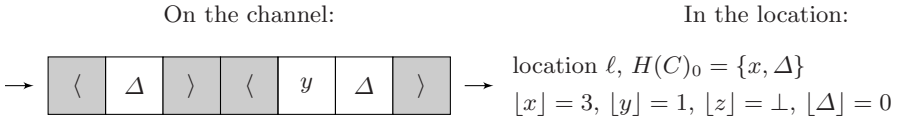



**Fig. 2.** Encoding of a configuration in a CAROT

symmetrically for constraint $\{x\} \geq x_{i-1} > x_{i+1}$. We thus have to store in the control part of the CAROT which clocks are "before" (resp. "after") the two first (resp. last) symbols $\Delta$. Whereas this can easily be done for the clocks that have been recently written on the channel, this is not possible for the clocks lying at the head of the channel (this would require to store the position of each clock w.r.t. each symbol $\Delta$). Instead, we use non-determinism to allow the CAROT make predictions about the content of the head of the channel, and then we verify when reading clocks from the channel that these predictions were correct.

For lack of space, we cannot present the formal construction of the CAROT, but report to [10]. We write $\mathcal{C}_{\mathcal{A}}$ for the resulting CAROT, $\mathcal{T}^n_{\mathcal{C}_{\mathcal{A}}}$ for the transition system associated with $\mathcal{C}_{\mathcal{A}}$ and restricted to configurations with correct predictions and $n$ occurrences of $\Delta$ on the channel (or in the location), and $\approx$ for the relation that describes which configuration $(d, c)$ of $\mathcal{T}^n_{\mathcal{C}_{\mathcal{A}}}$ (a control state together with a channel content) corresponds to a configuration of $\mathcal{T}^n_H$. The correctness of the construction relies on the following lemmas.

**Lemma 6.** *For any $n \geq 3$, the transition systems $\mathcal{T}^n_H$ and $\mathcal{T}^n_{\mathcal{C}_{\mathcal{A}}}$ are bisimilar.*

**Lemma 7.** *Let $\varrho$ be a time-divergent execution in $[\![\mathcal{B}^n]\!]$. Then any computation in $\mathcal{C}_{\mathcal{A}}$ simulating $\varrho$ has correct predictions.*[3]

Let $(d^n, c^n)$ be the configuration of $\mathcal{C}_{\mathcal{A}}$ encoding the initial configuration of $\mathcal{B}^n$, *i.e.*, such that $(d^n, c^n) \approx H(\ell_0, u_n)$. Lemmas 4, 6 and 7 then yield:

**Theorem 8.** *Let $n \geq 3$. $\mathcal{C}_{\mathcal{A}}$ has a time-divergent*[4] *computation starting in $(d^n, c^n)$ iff $\mathcal{L}([\![\mathcal{B}^n]\!]) \neq \varnothing$.*

The second part of the construction of the CAROT for encoding the robust model-checking problem consists in adding the part related to the temporal formula $\varphi$ (in MTL). This part will be handled in a very similar way as in [7], we thus simply sketch the construction. First, we build the one-clock alternating timed automaton $\mathcal{A}_{\neg\varphi}$ corresponding to $\neg\varphi$. Then, we build the product of the CAROT $\mathcal{C}_{\mathcal{A}}$ with a CAROT simulating the behaviour of $\mathcal{A}_{\neg\varphi}$. The resulting CAROT, say $\mathcal{C}_{\mathcal{A},\neg\varphi}$, running from the initial configuration $(d^{\varphi,n}, c^{\varphi,n})$ corresponding via $\approx$ to the initial configuration of $\mathcal{B}^n \times \mathcal{A}_{\neg\varphi}$, simulates joint behaviours of $\mathcal{B}^n$ and $\mathcal{A}_{\neg\varphi}$. The accepting condition for $\mathcal{C}_{\mathcal{A},\neg\varphi}$ is the Büchi condition given by 'flattening' $\mathcal{A}_{\neg\varphi}$. The results of [7] combined with our above results yield:

[3] Intuitively, this is because delay transitions force predictions checking.
[4] That is with infinitely many delay transitions.

**Theorem 9.** *Let $n \geq 3$ and $\varphi \in \mathsf{MTL}$. $\mathcal{C}_{\mathcal{A},\neg\varphi}$ has a time-divergent accepting computation starting in $(d^{\varphi,n}, c^{\varphi,n})$ iff $[\![\mathcal{B}^n]\!] \not\models \varphi$.*

*Remark.* A rough bound on the size of the CAROT $\mathcal{C}_{\mathcal{A},\neg\varphi}$ is $O\left(|\mathcal{A}|^3 \cdot 2^{O(M\cdot|\varphi|+|X|)}\right)$, where $|\varphi|$ is the number of subformulae of $\varphi$. The size of the alphabet of $\mathcal{C}_{\mathcal{A},\neg\varphi}$ is in $O(|X|)$. As proved in [7], if $\mathcal{C}_{\mathcal{A},\neg\varphi}$ has an $h$-cycle-bounded accepting execution, then there is a bound $N_0$, which depends on the size of $\mathcal{C}_{\mathcal{A},\neg\varphi}$ and $h$, such that there exists an $h$-cycle-bounded accepting execution of length no more than $N_0$. We do not give the precise value of this bound (see [10] instead), but it is exponential in $h$, which is itself exponential in the size of the input.

### 3.3 From CAROTs to Robust Model Checking

We first solve the robust model-checking problem for Bounded-MTL, and then turn to the more involved logic coFlat-MTL. Both rely on the previously proved equivalences:

$$\mathcal{A} \not\models \varphi \quad \text{iff} \quad \forall n \geq 3, \begin{cases} \mathcal{C}_{\mathcal{A},\neg\varphi} \text{ has a time-divergent accepting} \\ \text{computation starting in } (d^{\varphi,n}, c^{\varphi,n}) \end{cases}$$

**Robust model-checking for Bounded-MTL.** The algorithm to decide the robust model-checking problem for Bounded-MTL formula relies on the fact that the truth value of a Bounded-MTL formula $\varphi$ along a run $\varrho$ only depends on the first $h$ time units of $\varrho$, where $h$ is the sum of the constants appearing in $\varphi$ [7]. In $\mathcal{A}_{\neg\varphi}$, after having read a prefix of duration $h$ time units, we thus always end up in a sink state, that we report as accepting in the CAROT.

Moreover, the non-blocking assumption made on $\mathcal{A}$ implies the following property:

**Lemma 10.** *Let $\mathcal{A}$ be a timed automaton, and $\delta > 0$. Given $(\ell, v)$ a configuration of $\mathcal{A}$ such that $v \models_\delta I(\ell)$, we have that $\mathcal{L}([\![\mathcal{A}]\!]_\delta^{(\ell,v)}) \neq \varnothing$.*

Hence, robustly model-checking $\mathcal{A}$ against a Bounded-MTL property $\varphi$ will be reduced to searching, for every $n \geq 3$, for a time-bounded accepting prefix in $\mathcal{T}^n_{\mathcal{C}_{\mathcal{A},\neg\varphi}}$, and verifying that the reachable configuration is with correct predictions. Indeed, applying the previous lemma, we already know that we will be able to extend this finite prefix into a time-diverging run witnessing $\neg\varphi$ as soon as the prefix is correctly chosen (meaning it ends up in a accepting state of the CAROT).

We define the following property, for any integer $n$:

$\mathcal{P}(n)$: "$\mathcal{C}_{\mathcal{A},\neg\varphi}$ has an $h$-cycle-bounded accepting computation with correct predictions starting in $(d^{\varphi,n}, c^{\varphi,n})$"

Then our problem somehow amounts to checking that for every $n \geq 3$, property $\mathcal{P}(n)$ holds. This is some kind of "universality" checking of the CAROT, where we universally quantify on the initial number of $\Delta$'s on the channel. This is achieved using the following two lemmas:

**Lemma 11.** *Let $n, n' \in \mathbb{N}$ be such that $n' \geq 2n \geq 6$. Then $\mathcal{P}(n') \Rightarrow \mathcal{P}(n)$.*

*Proof.* We have seen that

$$[\![\mathcal{B}^{n'}]\!] \underset{\text{(Lemma 2)}}{\sqsupseteq} [\![\mathcal{A}_{\frac{2}{n'}}]\!] \underset{(\frac{2}{n'} \leq \frac{1}{n})}{\sqsupseteq} [\![\mathcal{A}_{\frac{1}{n}}]\!] \underset{\text{(Lemma 2)}}{\sqsupseteq} [\![\mathcal{B}^{n}]\!].$$

Also, for $m \geq 3$, the CAROT $\mathcal{C}_{\mathcal{A},\neg\varphi}$, when restricted to configurations with correct predictions, is time-abstract bisimilar to the product of $\mathcal{B}^m$ with $\mathcal{A}_{\neg\varphi}$. Finally, the respective initial configurations are in the relation $\approx$. □

**Lemma 12.** *Let $N \geq 2 \cdot N_0$. Then $\mathcal{P}(N) \Rightarrow \forall n \in \mathbb{N}, \exists n' \geq n$ s.t. $\mathcal{P}(n')$.*

*Proof (Sketch).* Using the notion of computation table introduced in [7], we prove a pumping lemma for CAROTs. Indeed, the height of the computation table is bounded by $h$; the number of "sliding windows" of such tables is thus bounded. Hence, once the table is large enough, it is possible to duplicate one of its fragments, building new computation tables encoding computations over larger inputs (corresponding to configurations $(d^{\varphi,n'}, c^{\varphi,n'})$ for integers $n'$ arbitrarily larger than $n$). □

Thanks to those lemmas, it suffices to only look for an $h$-cycle-bounded execution starting in one of the configurations $(d^{\varphi,N}, c^{\varphi,N})$ (for any $N \geq 2 \cdot N_0$) in order to ensure the existence of an accepting execution for any number of $\Delta$'s:

**Corollary 13.** *Let $N \geq 2 \cdot N_0$. Then $\mathcal{P}(N) \Leftrightarrow \forall n \geq 3,\ \mathcal{P}(n)$.*

**Theorem 14.** *The model-checking problem for* Bounded-MTL *is* EXPSPACE-*Complete (and* PSPACE-*Complete if constants of the formula are given in unary).*

*Proof.* The hardness parts follow from the same hardness results for Bounded-MTL satisfiability [7].

The upper bound follows from the previous study. However, since the size of the CAROT is doubly exponential, the non-deterministic algorithm of Theorem 1 has to be applied on-the-fly, without explicitly building the CAROT. Since the number $N_0$ of different sliding windows of height $h$ is also doubly-exponential in the size of the input, our non-deterministic algorithm will also have a counter, and will stop as soon as the counter reaches $N_0$. This all can be achieved within exponential space.

If the constants of the formula are unary-encoded, then $h$ is linear in the size of the input formula, and $N_0$ is simply exponential in the size of the input. The same algorithm then uses only polynomial space. □

**Robust model-checking for coFlat-MTL.** The case of coFlat-MTL is more involved than that of Bounded-MTL. The reason is that, unlike Bounded-MTL, the truth value of a coFlat-MTL formula $\varphi$ along a run $\varrho$ does not only depend on a prefix of $\varrho$ of bounded duration. Instead, we have the following decomposition lemma, which follows from [7, Theorem 12]:

**Lemma 15.** *Let $\varpi$ be an accepting run of $[\![\mathcal{A}]\!]_\delta \times [\![\mathcal{A}_{\neg\varphi}]\!]$. Then it can be decomposed as $\varpi_1 \cdot \varpi_2 \cdot \varpi_3 \cdots \varpi_{2m}$ where there is a finite automaton $\mathcal{F}_{\neg\varphi}$ [5] s.t.:*

- *(i) the duration of $\varpi_{2i-1}$ (for $1 \leq i \leq m$) is bounded by $h = (2M + 3 + W \cdot 2^{|\varphi|}) \cdot (|\varphi| \cdot 2^{|\varphi|})$,*
- *(ii) the duration of $\varpi_{2i}$ (for $1 \leq i \leq m$) is at least $2^{|\varphi|} \cdot (2W + 1)$, and along that portion, the behaviour of $\mathcal{A}_{\neg\varphi}$ is that of $\mathcal{F}_{\neg\varphi}$,*
- *(iii) the Büchi condition of $\mathcal{F}_{\neg\varphi}$ is satisfied along $\varpi_{2m}$, and*
- *(iv) $2m \leq |\varphi| \cdot 2^{|\varphi|}$,*

*where $W$ is the number of states of the region automaton of $\mathcal{A} \times \mathcal{F}_{\neg\varphi}$. Odd-numbered segments are called the* active parts*, while even-numbered ones are said* inactive*.*

This decomposition lemma inspires a decidability algorithm, where we modularly check the existence of runs not satisfying $\varphi$ by distinguishing between active and inactive parts of the runs. Indeed, given a sequence $(\varrho^\delta)_{\delta>0}$, using combinatorics arguments, it is possible to twist them so that, for $\delta > 0$ small enough (say $\delta \leq \delta_0$), all $\varrho^\delta$ look very similar (that is, roughly the junction points between active and inactive parts are close to each other and belong to the same region). Conversely, if we are given a witnessing run $\varrho^{\delta_0}$, and if we consider the junction points of that run, for each inactive (resp. active) part, it is possible for every $\delta > 0$ to build an inactive (resp. active) portion of a run joining the two junction points. The construction of an inactive portion of a run partly relies on approximation results proved in [15,8], and the construction of an active portion of a run relies on a result similar to Corollary 13. The complete proof is rather technical and gathers in an original way many results of [15,8,7].

An informal version of the decidability algorithm is the following, and can be schematised as on Figure 3:

- Guess the junction points of the active and inactive parts
- For each active part, check that the two guessed junction points are reachable in $\mathcal{C}_{\mathcal{A},\neg\varphi}$ in a cycle-bounded manner[6] (here, we need to prove a result similar to Corollary 13)
- For each inactive and bounded active part, check that the two junction points are "robustly" reachable (using results of [15,8])
- For the last unbounded active part, check that the automaton $\mathcal{A} \times \mathcal{F}_{\neg\varphi}$, from last junction point, does not robustly satisfy the acceptance condition of $\mathcal{F}_{\neg\varphi}$ interpreted as a co-Büchi condition (using the algorithm of [9]).

We can conclude with the main result of the paper:

**Theorem 16.** *The robust model-checking problem for* coFlat-MTL *is* EXPSPACE-*Complete.*

[5] Intuitively, $\mathcal{F}_{\neg\varphi}$ is obtained as the flattening of the untimed part of $\mathcal{A}_{\neg\varphi}$, see [10].

[6] The bound on the number of cycles is that of $(i)$ in the above decomposition lemma.

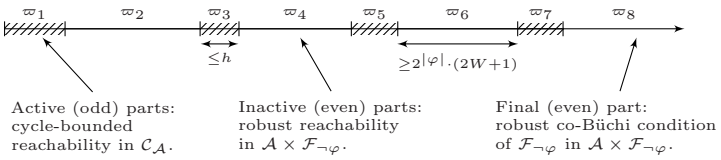



**Fig. 3.** Global view of our algorithm

## 4 Conclusion

In this paper, we have proposed a new approach to robust model-checking of timed systems based on channel machines: we construct a family of networks of timed systems such that robustly verifying a formula in a timed automaton reduces to the verification of the formula in one of the members of the family; Then we encode the behaviour of this family of timed systems using channel machines. We have applied this approach to coFlat-MTL, a rather expressive fragment of MTL, and prove that it can be decided in EXPSPACE, which is moreover optimal. The logic coFlat-MTL subsumes LTL, thus it is the more general specification language for which robust model-checking has been proved decidable.

Our correctness proofs heavily rely on technical lemmas proved in [15,8] and is unfortunately not fully CAROT-based. As future works, we plan to study robust reachability directly on the CAROT encoding the extended semantics, in order to develop a fully CAROT-based algorithm for coFlat-MTL.

## References

1. Altisen, K., Tripakis, S.: Implementation of timed automata: An issue of semantics or modeling? In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 273–288. Springer, Heidelberg (2005)
2. Alur, R., Dill, D.: A theory of timed automata. Theor. Comp. Sci. 126(2), 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
4. Alur, R., La Torre, S., Madhusudan, P.: Perturbed timed automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 70–85. Springer, Heidelberg (2005)
5. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Almost-sure model checking of infinite paths in one-clock timed automata. Research Report LSV-07-29, ENS Cachan, France (2007)
6. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and topological semantics for timed automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 179–191. Springer, Heidelberg (2007)
7. Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The cost of punctuality (109–118). In: Proc. 22nd Ann. Symp. Logic in Computer Science (LICS 2007), pp. 109–118. IEEE Comp. Soc. Press, Los Alamitos (2007)
8. Bouyer, P., Markey, N., Reynier, P.-A.: Robust model-checking of timed automata. Research Report LSV-05-06, ENS Cachan, France (2005)

9. Bouyer, P., Markey, N., Reynier, P.-A.: Robust model-checking of timed automata. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 238–249. Springer, Heidelberg (2006)
10. Bouyer, P., Markey, N., Reynier, P.-A.: Robust analysis of timed automata via channel machines. Research Report LSV-07-32, ENS Cachan, France (2007)
11. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: a model-checking tool for real-time systems. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
12. Cassez, F., Henzinger, T.A., Raskin, J.-F.: A comparison of control problems for timed and hybrid systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 134–148. Springer, Heidelberg (2002)
13. Daws, C., Kordy, P.: Symbolic robustness analysis of timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 143–155. Springer, Heidelberg (2006)
14. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robustness and implementability of timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 118–133. Springer, Heidelberg (2004)
15. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robustness and implementability of timed automata. Tech. Report 2004.30, Centre Fédéré en Vérification, Belgium (December 2005)
16. De Wulf, M., Doyen, L., Raskin, J.: Almost ASAP semantics: From timed models to timed implementations. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 296–310. Springer, Heidelberg (2004)
17. Dima, C.: Dynamical properties of timed automata revisited. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 130–146. Springer, Heidelberg (2007)
18. French, T., McCabe-Dansted, J.C., Reynolds, M.: A temporal logic of robustness. In: Konev, B., Wolter, F. (eds.) FroCos 2007. LNCS (LNAI), vol. 4720, pp. 193–205. Springer, Heidelberg (2007)
19. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
20. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
21. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. J. Software Tools for Technology Transfer 1(1–2), 134–152 (1997)
22. Ouaknine, J., Worrell, J.: Revisiting digitization, robustness and decidability for timed automata. In: Proc. 18th Ann. Symp. Logic in Computer Science (LICS 2003), IEEE Comp. Soc. Press, Los Alamitos (2003)
23. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: Proc. 19th Ann. Symp. Logic in Computer Science (LICS 2005), pp. 188–197. IEEE Comp. Soc. Press, Los Alamitos (2005)
24. Ouaknine, J., Worrell, J.: On the decidability and complexity of Metric Temporal Logic over finite words. Logical Methods in Comp. Sci. 3(1-8), 1–27 (2007)
25. Pnueli, A.: The temporal logic of programs. In: Proc. 18th Ann. Symp. Foundations of Computer Science (FOCS 1977), pp. 46–57. IEEE Comp. Soc. Press, Los Alamitos (1977)
26. Puri, A.: Dynamical properties of timed automata. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 210–227. Springer, Heidelberg (1998)
27. Swaminathan, M., Fränzle, M.: A symbolic decision procedure for robust safety of timed systems. In: Proc. 14th Intl Symp. Temporal Representation and Reasoning (TIME 2007), p. 192. IEEE Comp. Soc. Press, Los Alamitos (2007)

# The Common Fragment of ACTL and LTL

Mikołaj Bojańczyk*

Warsaw University

**Abstract.** The paper explores the relationship between tree languages definable in LTL, CTL, and ACTL, the fragment of CTL where only universal path quantification is allowed. The common fragment of LTL and ACTL is shown to be strictly smaller than the common fragment of LTL and CTL. Furthermore, an algorithm is presented for deciding if an LTL formula can be expressed in ACTL. This algorithm uses an effective characterization of level 3/2 of the concatenation hierarchy for infinite words, also a new result.

Two of the most commonly used logics in verification are LTL and CTL. The first is a linear time logic, a formula of LTL describes a property of words. To describe properties of trees, one applies universal path quantification: an LTL formula is valid in a tree if it is valid in all paths. CTL, on the other hand, is a branching time logic. A formula of CTL refers explicitly to the branching in the tree, by using both universal and existential path quantification.

What is the relationship between the two logics? Which LTL definable properties can be defined in CTL, and which CTL definable properties can be defined in LTL? In other words, what is the common fragment of CTL and LTL?

There is a well known algorithm, which given an automaton on infinite trees, determines if its language can be defined in LTL (basically, a tree constructed by mixing paths of different trees accepted by the automaton, must still be accepted by the automaton; furthermore, the appropriate word language must be aperiodic). For tree languages defined in CTL* there is also a simple characterization of Clarke and Draghilescu: a CTL* formula is equivalent to an LTL formula, if and only if it is equivalent to the one obtained by removing the path quantifiers [3]. Maidl [6] has shown that if the input is given as a CTL formula, then problem of LTL definability becomes PSPACE complete.

The converse question, however, remains an open problem: is it decidable if a given LTL formula can be equivalently written as a CTL formula? A second, more general, problem is to decide if an arbitrary regular language of infinite trees can be defined in CTL. It seems a good idea to begin with the first problem before tackling the second one.

If an LTL formula with universal path semantics can be defined by a CTL formula, then why should the CTL formula use existential modalities, such as "exists a successor with $\varphi$"? Shouldn't it be enough to consider ACTL formulas, where only universal path quantification is used? The first result of this paper is

* Author supported by Polish government grant no. N206 008 32/0810.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 172–185, 2008.

that, possibly surprisingly, this assumption is wrong. Indeed, a very simple LTL property, "all paths belong to $(ab)^*a(ab)^*c^\omega$", can be defined in CTL but not ACTL. Intuitively speaking, to catch the extra $a$ on every path, existential path quantification is needed.

Therefore, two distinct questions can be investigated: which LTL properties can be defined in CTL, and which LTL properties can be defined in ACTL. The other main result of this paper is an effective characterization of the second common fragment: one can decide if an LTL formula $\varphi$ can be expressed in ACTL. This problem has already been considered in [6], where it was shown that a necessary and sufficient condition for ACTL definability is that $\neg\varphi$, when seen as a word language, can be recognized by a certain restricted type of Büchi automaton. This condition, however, was not known to effective, i.e. there was no algorithm that decided if $\neg\varphi$ could be recognized by the restricted Büchi automaton.

The second contribution of this paper is such an algorithm. It is easy to see that the restricted Büchi automata defined in [6] are equivalent to $\omega$-regular languages on level 3/2 of the concatenation hierarchy, i.e. finite unions of expressions

$$A_0^* a_1 A_1^* a_2 \cdots A_{k-1}^* a_k A_k^\omega \ .$$

Therefore, deciding if an LTL formula can be defined in ACTL boils down to testing if an $\omega$-regular language belongs to level 3/2 of the concatenation hierarchy. This problem was known to be decidable for finite words [1,2,8]. We generalize this result to infinite words. In the process, we also present a simplified proof for finite words.

The paper is organized as follows. In Section 1, we show that the common fragment of LTL and CTL is strictly larger than the common fragment of LTL and ACTL. Section 2 gives an effective characterization of those LTL properties that can be defined in ACTL. Finally, in Section 4, we present concluding remarks. These concern mainly the common fragment of LTL and CTL, about which little is known.

## 1 The Common Fragment of CTL and LTL Needs Existential Modalities

Trees in this paper are unordered, infinite and unranked. In other words, a tree is a connected directed graph with nodes of indegree at most one, but outdegree at least one. The last condition is so that every (maximal) path in the tree is infinite. Trees are labeled.

The results in this paper would also apply to finite trees, or transition systems. Some of the results would be cleaner for finite trees, we will come back to this at the end of the paper.

LTL is a linear time temporal logic. An LTL formula specifies a property of an infinite word. (When we say a word position satisfies $\varphi$, we mean that the

suffix beginning in that position satisfies $\varphi$.) The modalities are: $\varphi \mathsf{U} \psi$ (there is a position with $\psi$, and all preceding positions satisfy $\varphi$), $\mathsf{X}\varphi$ (the second word position satisfies $\varphi$) and $\mathsf{G}\varphi$ (all positions in the word satisfy $\varphi$). Furthermore, boolean connectives and label tests (the formula $a$ describes words that begin with $a$) are allowed. Kamp's theorem [5] says that LTL has the same expressive power as first-order logic with the linear order on word positions. An LTL formula can be evaluated in a tree, it is said to be valid if all maximal paths in the tree satisfy it. In this sense, very simple tree properties, such as "some node in the tree has label $a$", cannot be defined in LTL. In the following, we will indicate whether an LTL formula is understood to define a word language, or a tree language.

CTL [4] is a temporal branching time logic, i.e. its modalities explicitly quantify over tree paths, possibly existentially. A CTL formula specifies a property of a tree. As with LTL, when we say a formula holds in a tree node (or equivalently, on a position on a path in the tree), we mean that the subtree of that node satisfies the formula. The modalities are: $\mathsf{A}\varphi \mathsf{U} \psi$ (on every path, there is a position with $\psi$, and all preceding positions on the path satisfy $\varphi$), $\mathsf{AX}\varphi$ (every successor of the root satisfies $\varphi$) and $\mathsf{AG}\varphi$ (on every path, every position satisfies $\varphi$). Furthermore, boolean connectives and label tests (the formula $a$ describes trees whose root has label $a$) are allowed. Existential quantification can be simulated using negation. In other words, CTL is obtained from LTL by adding universal path quantification next to every modality. In particular, over trees with only one path, i.e. where all nodes have exactly one successor, CTL has the same expressive power as LTL. In general, however, the two logics diverge, for instance CTL cannot express $\mathsf{FG}a$ (on every path, finitely many non $a$ labels).

ACTL is the fragment of CTL that does not allow negation, i.e. where only universal path quantification is allowed. (Here, the atomic propositions are node labels, so they are mutually exclusive; if they are are not exclusive then negation is allowed next to atomic propositions.) Clearly, ACTL is a proper fragment of CTL; for instance, the CTL property "some node has label $a$" is not definable in ACTL.

The main result in this section is:

**Theorem 1.** *The language $L$ = "all paths belong to $(ab)^*a(ab)^*c^\omega$" is definable in CTL and LTL, but not ACTL.*

This result is somewhat surprising: the language $L$ talks about all paths, while the corresponding CTL formula must quantify existentially over paths. This example shows that ideas significantly different from those in [6] are needed to understand the common fragment of LTL and CTL.

Before proceeding with the proof, we would like to remark the similarity of this "paradox" with a result for first-order logic over finite binary trees. In [9], Potthof showed that the language "all paths belong to $(aa)^*$" is definable in first-order logic over finite binary trees, even though the word language $(aa)^*$ is not definable in first-order logic over words. His technique was similar to the one invoked below, in that it used properties of "maximal" nodes.

We will now prove Theorem 1.

**Lemma 1.** *The language $L$ is definable in CTL.*

**Proof**
It is easy to show that the language "all paths belong to $(ab)^*c^\omega$" is definable in CTL. Let $\varphi_a$ be such a formula; we will use it below. Likewise we will use a formula $\varphi_b$ for the language "all paths belong to $b(ab)^*c^\omega$".

The formula for the language $L$ is a conjunction of several properties. First, we have to manage the way $c$'s are used. Formula (1) says that every path contains some $c$, and every time $c$ appears, all subsequent nodes are $c$'s; finally, only $b$ nodes can have a $c$ successor:

$$\mathsf{AF}c \ \wedge \ \mathsf{AG}(c \Rightarrow \mathsf{AG}c) \ \wedge \ \mathsf{AG}(a \Rightarrow \mathsf{AX}(a \vee b)). \tag{1}$$

Formula (2) says that the tree does not contain two consecutive $b$'s:

$$\mathsf{AG}(b \Rightarrow \mathsf{AX}(a \vee c)). \tag{2}$$

Formula (3) says that on every path, two consecutive $a$'s can be found at most once:

$$\neg\mathsf{EF}(a \wedge \mathsf{EX}(a \wedge (\mathsf{EF}(a \wedge \mathsf{EX}a)))). \tag{3}$$

So far, we have stayed within ACTL. The above three properties guarantee that every path in the tree is either in $(ab)^*c^\omega$ or in $(ab)^*a(ab)^*c^\omega$, as long as the root has label $a$. We now need to eliminate the paths of the first type. First, we enforce the root label, and say that at least one path is not in $(ab)^*c^\omega$

$$a \quad \wedge \quad \neg\varphi_a. \tag{4}$$

Note that already here, we go beyond ACTL, since $\varphi_a$ is negated. The more important property, however, says there is no node $x$ such that: some path beginning in $x$ has two consecutive $a$'s, and some successor of $x$ satisfies $\varphi_a$ or $\varphi_b$, depending on the label of $x$. This expressed by the formula:

$$\neg\mathsf{EF}(\mathsf{EF}(a \wedge \mathsf{EX}a) \wedge (a \Rightarrow \mathsf{EX}\varphi_b) \wedge (b \Rightarrow \mathsf{EX}\varphi_a)) \tag{5}$$

Here again we go beyond ACTL. We claim that a tree belongs to $L$ if and only if it satisfies the conjunction of formulas (1)-(5).

The left-to-right implication is proved as follows. Let $t$ be a tree in $L$. Clearly (1)-(4) have to be satisfied. For (5), we need to show that every node $x$ fails the property:

$$\mathsf{EF}(a \wedge \mathsf{EX}a) \wedge (a \Rightarrow \mathsf{EX}\varphi_b) \wedge (b \Rightarrow \mathsf{EX}\varphi_a).$$

We only consider the case when the node $x$ has label $a$, the other is done in a similar way. Let then $x$ be a node with label $a$ that satisfies $\mathsf{EF}(a \wedge \mathsf{EX}a)$. By (1)-(4), the path leading up to $x$ must belong to $(ab)^*$. In particular, no successor of $x$ can be the beginning of a path in $(ab)^*c^\omega$, not to mention having all paths of this form (which is what $\mathsf{EX}\varphi_a$ says).

We now take the right-to-left implication. Let then $t$ be a tree that satisfies formulas (1)-(4). We claim that if $t$ is outside $L$, then the formula (5) fails. By (1)-(4), the tree has paths of the form $(ab)^*c^\omega$, and of the form $(ab)^*a(ab)^*c^\omega$. Let $X$ be the set of nodes, which lie on the intersection of some path of the form $(ab)^*a(ab)^*c^\omega$ and some other path of the form $(ab)^*c^\omega$. By $\mathsf{AG}c$, the prefix-closed set $X$ does not contain an infinite path, therefore it has some maximal element, i.e. a node $x \in X$ without proper descendants in $X$. (Since nodes may have infinite outdegree, there may be no common bound on the depth of nodes from $X$, but this is not a problem here, since we only need one maximal node.) We claim that the maximal node $x$ witnesses the failure of (5), i. e. $x$ satisfies

$$\mathsf{EF}(a \wedge \mathsf{EX}a) \wedge (a \Rightarrow \mathsf{EX}\varphi_b) \wedge (b \Rightarrow \mathsf{EX}\varphi_a).$$

We only consider the case where $x$ has label $b$. By maximality of $x$, there must be a successor $y$ such that all paths that pass through $y$ belong to $(ab)^*c^\omega$; otherwise $y$ would belong to $X$. The node $y$ witnesses $\mathsf{EX}\varphi_a$ (note that $y$ may have label $c$, and only $c$'s in its subtree). Since the path leading to $x$ belongs to $(ab)^*$ and $x$ is on some path in $(ab)^*a(ab)^*c^\omega$, there must be two consecutive $a$'s on some path that begins in $x$. □

We now show that the language $L$ is not definable in ACTL. We will use a characterization of Maidl from [6], slightly restated:

**Theorem 2.** *Let $L \subseteq A^\omega$ be a language of infinite words. The following are equivalent:*

- *The tree language "all paths belong to $L$" is definable in ACTL;*
- *The complement of $L$ is a finite union of languages of the form*

$$A_0^* a_1 A_1^* a_2 \cdots A_{n-1}^* a_n A_n^\omega \qquad a_1, \ldots, a_n \in A,\ A_1, \ldots, A_{n+1} \subseteq A. \tag{6}$$

Languages that are finite unions of expressions as in (6) are also known as languages on *level 3/2 of the concatenation hierarchy*, see [7] for a more thorough description of this and other levels. Therefore, the second condition in the above theorem is the same as saying the complement of $L$ belongs to level 3/2.

The original statement in [6] was not in terms of level 3/2, but in terms of 1-weak Büchi automata. A 1-weak Büchi automaton is a Büchi automaton where the states are ordered, and a transition can only go down in the order, or stay in the same state. A level 3/2 expression can easily be translated into a 1-weak Büchi automaton, by using nondeterminism of the automata. The translation in the other direction is no more difficult: the subexpressions $A_i^*$ correspond to staying in the same state for some time; while $A_n^\omega$ corresponds to an infinite self-loop in an accepting state. The finite union corresponds to the finitely many possible paths in the order.

**Lemma 2.** *The language $L$ is not definable in ACTL.*

**Proof**
Towards a contradiction with Theorem 2, assume that the complement of $L$ is defined by an finite union of expressions as in (6). Let $n$ be the size of the largest of these expressions. Since $(ab)^{n+1}c^\omega$ is outside $L$, it must be captured by one of the expressions:

$$(ab)^{n+1}c^\omega \in A_0^* a_1 A_1^* a_2 \cdots A_{m-1}^* a_m A_m^\omega.$$

Since $m \leq n$, it is fairly easy to see that the word $(ab)^j a(ab)^{n+1-j}c^\omega \in L$ will also be captured by the same expression, a contradiction. □

In the next section we give an algorithm that decides if a word language can be defined on level 3/2; therefore the above proof could be replaced by just running the algorithm on $L$ (and reading the output "no").

We remark that a third equivalent condition can be added to Theorem 2: the word language $L$ is defined by a $\Pi_2$ formula, i.e. a first order formula with a quantifier prefix $\forall^*\exists^*$, where the signature allows label tests, and the linear order on word positions.

## 2 Effective Characterization of Level 3/2 for Infinite Words

In this section give show that the following problem is decidable:

Input: a regular word language $L \subseteq A^*$ (resp. $L \subseteq A^\omega$).
Question: is $L$ on level 3/2 of the concatenation hierarchy?

The case when $L$ is a language of finite words—when $L \subseteq A^*$—is treated in Section 2.2, while the infinite case—when $L \subseteq A^\omega$—is treated in Section 2.3.

The result on finite words is not new. The first proof is due to Arfi [1,2] and uses a difficult result of Hashiguchi. A different proof was presented by Pin and Weil in [8], one of the advantages being a better complexity for the algorithm. Instead of just citing these results, we present a complete proof below. There are two reasons. The first reason is that although the proof in [8] is self-contained, it does use a number of involved and general algebraic concepts. The proof below is specifically tailored to level 3/2, although the underlying ideas are similar to [8], in particular the use of Simon's factorization forests. The second reason is that we are actually interested in infinite words, for which the result has not yet been shown. Although the infinite case turns out to be a straightforward adaptation of the finite one, its proof would be difficult to understand without the finite case.

The proof will use an algebraic language, especially monoids and morphisms. Recall that a monoid is a set $S$ together with an associative concatenation operation, denoted multiplicatively. Furthermore, there is an identity element. An example of a monoid is the *free monoid over* $A$, i.e. the set $A^*$ of all words over the alphabet $A$ (the identity element is the empty word). Finite monoids are used to recognize regular languages, just as automata. In order to recognize a regular

language $L \subseteq A^*$, we use a morphism $\alpha : A^* \to S$. (A *morphism* is a function that preserves concatenation, and the identity element). A morphism is said to recognize $L$, if membership $w \in L$ depends only on the value $\alpha(w) \in S$. In other words, $L = \alpha^{-1}(\alpha(L))$. Languages recognized by morphisms into finite monoids are exactly the same ones as those recognized by finite automata, so morphisms and monoids can be used as a different way of describing regular languages. For each regular language, there is a *syntactic morphism*, whose target monoid is the smallest monoid that can be used to recognize the language. This morphism corresponds to the two-sided Myhill-Nerode equivalence of the language.

An important concept will be the non-connected subword relation. If $S$ is a monoid, and $s, t \in S$, we write $s \preceq t$ if for some $s_1, \ldots, s_n, t_0, \ldots, t_n \in S$ we have

$$s = s_1 \cdots s_n \qquad t = t_0 s_1 t_1 \cdots t_{n-1} s_n s_n.$$

This relation is transitive in the free monoid, but need not be transitive in general.

When characterizing level 3/2, we will use a result of Simon about "factorization forests". We present this result in a slightly different, but equivalent, way than the original paper [10]. As mentioned previously, the Simon result was already used in [8]; in this sense our approach to characterizing level 3/2 is not new. The Simon result is presented in the next section, while Sections 2.2 and 2.3 apply it to describing level 3/2.

### 2.1 Typed Regular Expressions

In this section, we present regular expressions that are well typed for a morphism. Before looking at the formal definition, consider first the language "even number of $a's$", over an alphabet $\{a, b\}$. This language is described by the regular expression

$$\big((b^*a)(b^*a)b^*\big)^*.$$

Consider now a morphism $\alpha : \{a, b\}^* \to \{0, 1\}$, which recognizes the language by counting the number of $a$'s modulo two. It so happens that the regular expression presented above is well-typed for this morphism, i.e. for every subexpression we can indicate the appropriate value assigned by $\alpha$:

$$b : 0 \quad b^* : 0 \quad a : 1 \quad b^*a : 1 \quad (b^*a)(b^*a) : 0 \quad \big((b^*a)(b^*a)b^*\big)^* : 0$$

Not every regular expression is well typed with respect to every morphism. However, a consequence of the factorization forest theorem of Simon says that every regular language recognized by a morphism $\alpha$ can be defined by a regular expression that is well typed with respect to $\alpha$. The rest of this section presents this result in more detail.

To simplify notation, we do not distinguish between regular expressions and the languages they describe. In particular, we will denote expressions using the same letters as languages, i.e. $L$, $K$ and $M$. Fix a morphism $\alpha : A^* \to S$. An $\alpha$-typed regular expression $L$ is like a regular expression, but each subexpression

must be typed by a value in the monoid $S$. The Kleene star is only allowed in the form $L^+$ with nonempty iterations, and furthermore $L^+$ is only allowed if $L$ is typed by an idempotent (an element $s$ with $ss = s$). The formal inductive definition follows:

- Any single word $w \in A^*$ is an $\alpha$-typed expression typed by $\alpha(s)$.
- If $L, K$ are $\alpha$-typed expressions both typed by $s \in S$, then $L \cup K$ is an $\alpha$-typed expression also typed by $s$.
- If $L, K$ are $\alpha$-typed expressions typed by $s, t \in S$ respectively, then $LK$ is an $\alpha$-typed expression typed by $st$.
- If $L$ is an $\alpha$-typed expression typed by $s \in S$, and $s$ is idempotent, then $L^+$ is an $\alpha$-typed expression also typed by $s$.

The following result is a straightforward consequence of Simon's factorization forests theorem [10]:

**Theorem 3.** *Let $\alpha : A^* \to S$ be a morphism. For any $s \in S$, the language $\alpha^{-1}(s)$ can be defined by an $\alpha$-typed expression. In particular, any language recognized by $\alpha$ can be defined by a finite union of $\alpha$-typed expressions.*

### 2.2 Characterization of Level 3/2 for Finite Words

In this section give show that the following problem is decidable:

Input: a regular word language $L \subseteq A^*$.
Question: is this language on level 3/2 of the concatenation hierarchy?

For the sake of completeness, in the theorem below we also include the equivalence between level 3/2 of the concatenation hierarchy and level $\Sigma_2$ of the first-order quantification hierarchy, a special case of a result by Thomas [11]. Recall that a sentence of $\Sigma_2$ is a first order sentence with quantifier prefix $\exists^*\forall^*$. A sentence defines the set of words where it holds, in a given word quantification is over word positions. The signature contains the linear order on word positions (but not the successor), and label tests. For instance, the following $\Sigma_2$ sentence defines the language $a^*ba^*ca^*$:

$$\exists x_1, x_2 \forall y \quad x_1 < x_2 \wedge b(x_1) \wedge c(x_2) \wedge \big(y \neq x_1 \wedge y \neq x_2 \Rightarrow a(y)\big)$$

The key point in the algorithm will be a type of pumping relation that is complete for level 3/2, i.e. all languages on level 3/2 are closed under this type of pumping and, conversely, all languages closed under this type of pumping are on level 3/2. Consider the following rewriting rule (on words in $A^*$):

$$w_1 w_2 \quad \to_\alpha \quad w_1 v w_2 \qquad \text{if } \alpha(v) \preceq \alpha(w_1) = \alpha(w_2) = \alpha(w_1 w_2). \tag{7}$$

(Recall that $\preceq$ is the non-connected subword relation.) We call this a rule, but it is more properly called a rule scheme, since there are infinitely many words $w_1, w_2, v$ with the above properties. Let $\Rightarrow_\alpha$ be the rewriting system generated by this rule, i.e. $w \Rightarrow_\alpha v$ holds if $v$ can be obtained from $w$ by applying the

rule $\rightarrow_\alpha$ to infixes a number, possibly zero, of times. We will treat this rewriting system as a pumping relation. We define $cl_\alpha(L)$ to be the set of words $w$ such that $v \Rightarrow_\alpha w$ holds for some $v \in L$.

**Theorem 4.** *For a regular language $L$ of finite words, the following are equivalent:*

1. *$L$ is definable by a sentence of $\Sigma_2$;*
2. *$L$ is on level 3/2 of the concatenation hierarchy, i.e. equivalent to a finite union of expressions $A_0^* a_1 A_1^* \cdots A_{n-1}^* a_n A_n^*$;*
3. *$L = cl_\alpha(L)$ holds for $\alpha$ the syntactic morphism of $L$.*

We do not yet show that this characterization is effective, this is the subject of Section 3.

The implication from 2 to 1 is immediate, while the implication from 1 to 3 can be shown using a standard Ehrenfeucht-Fraisse argument. We concentrate on the implication from 3 to 2.

**Lemma 3.** *For every $\alpha$-typed expression $L$, there is level 3/2 expression $K$ with $L \subseteq K \subseteq cl_\alpha(L)$.*

Before we proving this lemma, we show how it gives the implication from 3 to 2 in Theorem 4. Let $L \subseteq A^*$ be a language whose syntactic morphism is $\alpha : A^* \to S$, and which satisfies $L = cl_\alpha(L)$. By Theorem 3, $L$ can be defined as a union of $\alpha$-typed expressions $L_1 \cup \cdots \cup L_n$. Let $K_1, \ldots, K_n$ be the languages obtained by applying the lemma to $L_1, \ldots, L_n$. We claim that the union of the languages $K_i$ is the same as $L$, which concludes the proof. Indeed,

$$L = \bigcup L_i \subseteq \bigcup K_i \subseteq \bigcup_\alpha cl(L_i) = cl(L) = L.$$

**Proof**

The proof is by induction on the $\alpha$-expression. The induction base is trivial. Concatenation is a consequence of closure of level 3/2 expressions under concatenation and of

$$cl(L_1) \cdot cl(L_2) \subseteq cl(L_1 \cdot L_2).$$

Union is solved the same way. Only the step $L^+$ remains. Let $e \in S$ be the idempotent that types the expression $L$, and let $K$ be the level 3/2 expression obtained by applying the induction assumption to $L$. We need to find a language $M$ for $L^+$. We set:

$$M = K \cup KB^*K \qquad \text{where } B = \{b \in A : \alpha(b) \preceq e\}$$

Clearly the expression for $M$ is on level 3/2. It remains to show that $M$ satisfies:

$$L^+ \subseteq M \subseteq cl(L^+).$$

By definition of $B$, we have $L \subseteq B^*$, and hence the left inclusion holds. Only the right inclusion $M \subseteq cl(L^+)$ remains. Let then $w$ be a word in $M$. The more difficult case is when $w = w_1 w_2 w_3$, with $w_1, w_3 \in K$ and $w_2 \in B^*$. By

assumption on $K \subseteq cl(L)$, there are words $v_1, v_3 \in L$ with $v_1 \Rightarrow_\alpha w_1$ and $v_3 \Rightarrow_\alpha w_3$. The desired conclusion—actually, a stronger result: $M \subseteq cl_\alpha(LL)$—follows by

$$v_1 v_3 \Rightarrow_\alpha v_1 w_2 v_3 \Rightarrow_\alpha w_1 w_2 w_3$$

The first rewriting uses $\alpha(w_2) \preceq e$, which follows by definition of $w_2$ and idempotency of $e$. The second rewriting follows by the assumption on $v_1, v_3$. □

### 2.3 Characterization of $\Sigma_2$ for Infinite Words

In this section we will be working with infinite words ($\omega$-words). Our approach will be the same, in particular we will need to use a syntactic monoid $\alpha : A^* \to S$ for a language of infinite words. What is a syntactic monoid for a language $L \subseteq A^\omega$ of infinite words? It is also obtained by using a Myhill-Nerode congruence. Two finite $v, v'$ words are called *$L$-equivalent* if both:

- For every $u \in A^*$ and $w \in A^\omega$, either both or none of $uvw, uv'w$ are in $L$.
- For every $u, w \in A^*$, either both or none of $u(vw)^\omega, u(v'w)^\omega$ are in $L$.

It turns out that $L$-equivalence is a congruence on $A^*$, and the mapping that assigns a word its $L$-equivalence class is a semigroup morphism, called the *syntactic morphism* of an infinite language. See [7] for more details.

We will follow the same approach as in the previous section, by introducing a rewriting relation. This relation will work on infinite words. It is generated by two types of rule. The first type is the same one as in the previous section, i.e. the rule (7), which is used to rewrite finite infixes of the infinite word. The second rule works on infinite suffixes of the word:

$$w^\omega \to_\alpha wuv^\omega \qquad \text{if } \alpha(u), \alpha(v) \preceq \alpha(w) \text{ and } \alpha(w) \text{ is idempotent.} \tag{8}$$

We denote by $\Rightarrow_\alpha^\omega$ the rewriting system generated by both rules (7) and (8). We write $cl_\alpha^\omega(L)$ for the closure of $L \subseteq A^\omega$ under this rewriting system.

**Theorem 5.** *For a regular language $L$ of infinite words, the following are equivalent:*

1. *$L$ is definable by a sentence of $\Sigma_2$;*
2. *$L$ is on level 3/2 of the concatenation hierarchy, i.e. equivalent to a finite union of expressions $A_0^* a_1 A_1^* \cdots A_{n-1}^* a_n A_n^\omega$;*
3. *$L = cl_\alpha^\omega(L)$ holds for $\alpha$ the syntactic morphism of $L$.*

**Proof**

As for Theorem 4, we only We only do the proof for the implication from 3 to 2.

Let $X$ be the set of pairs $(s, e) \in S^2$ such that $e$ is an idempotent, and some word of the form $uv^\omega$ belongs to $L$, with $\alpha$ mapping $u, v$ to $s, e$ respectively. We first claim that the following equality holds:

$$L \quad = \quad \bigcup_{(s,e)\in X} \alpha^{-1}(s)\alpha^{-1}(e)A_e^\omega \ , \tag{9}$$

where $A_e$ is the set of letters that may appear in a word mapped to $e$. For the left to right inclusion, fix some word $w \in L$. Using the (infinite) Ramsey theorem, this word can be decomposed as $w = w_0w_1w_2\cdots$, with all the words $w_1, w_2, \ldots$ being mapped by $\alpha$ to the same idempotent $e$. If $s$ is the value $\alpha(w_0)$, then we clearly have $(s, e) \in X$. Furthermore, clearly all the letters in $w_2, w_3, \ldots$ belong to $A_e$, which shows that the word $w$ belongs to the right side of (9).

Consider now the right to left inclusion (9). Let $uvw$ be a word belonging to the right hand side, i.e. with $\alpha(u) = s$, $\alpha(v) = e$ and $w \in A_e^\omega$ for some $(s, e) \in X$. As above, there is a decomposition $w = w_0w_1\cdots$ with all the words $w_1, w_2, \ldots$ being mapped to the same element by $\alpha$. Since all the words $w_1, w_2, \cdots$ are equivalent, it suffices to show that $uvw_0w_1^\omega$ belongs to $L$. By assumption $w_i \in A_e^*$, we have $\alpha(w_i) \preceq e$ for $i = 0, 1$. Therefore, we have $L \ni uv^\omega \Rightarrow_\alpha^\omega uvw_0w_1^\omega$, and the right hand side of the rewriting must belong to $L$.

The rest of the reasoning is as in the case for finite words. For each $(s, e) \in X$, we treat $\alpha^{-1}(s)\alpha^{-1}(e)$ as a language $L_{s,e}$ of finite words. Using Lemma 3, we show that the closure $cl_\alpha(L_{s,e})$ of each such language is defined by a level 3/2 expression. As in the previous section, we get an expression for a language that is between $L$ and $cl_\alpha(L)$, this expression must then describe $L$ itself. □

## 3 Complexity

In this section, we show that the conditions in Theorems 4 and 5 can be effectively checked. To the author's best knowledge, the result below is the first criterion that can be checked in polynomial time with respect to a finite automaton, and not just a semigroup.

**Theorem 6.** *Given a deterministic finite automaton over finite words, one can decide in polynomial time if the language it recognizes belongs level 3/2 of the concatenation hierarchy.*

**Proof**
We assume that all states in the automaton are reachable. However, the automaton need not be minimal. The automaton state assumed after reading the word $w$ when beginning in state $p$ is denoted below by $pw$. Consider the following property, which can be viewed as a forbidden pattern:

(*) Let $p, q$ be states such that for some words $v \preceq w$, $pw = p$, $qw = q$ and $pv = q$ hold. Every word accepted from $p$ is also accepted from $q$.

We first claim that (*) is equivalent to condition 3 in Theorem 4. Then, we wil show that (*) can be checked in polynomial time.

We begin with the left to right implication in the claim. Take a language $L$ recognized by an automaton where (*) holds, and let $\alpha$ be the syntactic morphism of $L$. We need to show that the language is closed under applying the rule $\rightarrow_\alpha$. In other words, we need to show that for any words $w_1, w_2, v \in A^*$ with

$$\alpha(v) \preceq \alpha(w_1) = \alpha(w_2) = \alpha(w_1w_2)$$

and for any two words $u_1, u_2 \in A^*$, we have:

$$u_1 w_1 w_2 u_2 \in L \quad \Rightarrow \quad u_1 w_1 v w_2 u_2 \in L.$$

By assumption, there is some word $w \in A^*$ with $v \preceq w$ and $\alpha(w) = \alpha(w_1)$. Recall that for any function $\delta : Q \to Q$ there is some power $k \in \mathbb{N}$ such that $\delta^k = \delta^{2k}$. Since $\alpha(w_1) = \alpha(w_1)\alpha(w_1)$, we may pick the word $w$ so that its transformation on states is idempotent, i.e. $qw = qww$ holds for all $w$. By assumption on $w$, and therefore also $ww$, being equivalent to $w_1, w_2$ under the morphism $\alpha$, it is sufficient to show

$$u_1(ww)(ww)u_2 \in L \quad \Rightarrow \quad u_1(ww)v(ww)u_2 \in L.$$

Let $p$ be the state assumed by the automaton after reading $u_1 w$, and let $q$ be the state $pvw$. By assumption on $w$ we have $p = pw$ and $q = qw$. Since we have $vw \preceq ww$ and $q = pvw$, we can use the assumption (*) to obtain that any word accepted from $p$ is also accepted from $q$. But the result follows, since $p$ (resp. $q$) is the state assumed by the automaton after reading the word on the left (resp. right) hand side of the implication, wihtout the $u_2$ suffix.

We now show the right to left implication of the claim. Let then $p, q$ and $v, w$ be as in the assumption of (*). Let $v_0$ a word after reading which the automaton assumes state $p$, which exists by assumption on all states being accessible. For some power $k$, we have $\alpha(w^k) = \alpha(w^k)\alpha(w^k)$. This allows us to use the assumption assumption (7) to obtain

$$v_0 w^k w^k u \in L \quad \Rightarrow \quad v_0 w^k v w^k u \in L \qquad \text{for all } u \in A^*.$$

Since $p$ (resp. $q$) is the state assumed by the automaton after reading the word on the left (resp. right) hand side of the implication, wihtout the $u$ suffix, we obtain the desired conclusion of (*): any word accepted from state $p$ is also accepted from state $q$.

We now show that condition (*) can be tested in polynomial time. The basic idea is that a standard dynamic algorithm for verifying forbidden patterns can be adapted to use the non-connected subword relation $\preceq$. The polynomial time algorithm runs as follows. In a first step, it calculates the tuples of states

$$X = \{(p, p', q, q', r, r') : p' = pw, q' = qw, r' = rv \text{ holds for some } v \preceq w\}.$$

This calculation can be done in polynomial time by a least fix-point algorithm: we begin with the tuples that correspond to word pairs $v \preceq w$ of length at most 1, and then apply the rule

$$(p, p', q, q', r, r'), (p', p'', q', q'', r', r'') \in X \Rightarrow (p, p'', q, q'', r, r'') \in X$$

until $X$ saturates. Once $X$ has been calculated, we identify the pairs $(p, q)$ such that $(p, p, q, q, p, q)$ belongs to $X$. For each such pair, we verify that all words accepted from $p$ are also accepted from $q$. □

The same idea can be used for the infinite case:

**Theorem 7.** *Given a deterministic parity automaton over infinite words, one can decide in polynomial time if the language it recognizes belongs level 3/2 of the concatenation hierarchy.*

**Proof**
The proof follows the same lines as for finite words, with the difference that we need to check the second rewriting rule (8). This corresponds to the following pattern:

(**) Let $p$ be a state and $w$ a word with $pw = p$, furthermore assume that the top priority labeling the cycle $pw$ is even. For every words $v, u \preceq w$, the word $v^\omega$ is accepted from state $pwu$.

□

A problem with the above theorem is that the input automaton is deterministic parity, and not nondeterministic Büchi, as often used in verification. We do not know if the algorithm can be adapted to nondeterministic automata, or if the exponential blowup due to determinization is indeed necessary.

The above theorem, when combined with Theorem 2, shows that the common fragment of ACTL and LTL is decidable:

**Theorem 8.** *It is decidable if a regular tree language belongs to the common fragment of ACTL and LTL.*

**Proof**
Let $K$ be the tree language, given e.g. by a parity tree automaton. We first find if there is a word language $L \subseteq A^\omega$ such that $K$ is the same as "on all paths $L$". This language, if it exists, can be easily calculated. If the language does not exist, $K$ is clearly not in the common fragment. Otherwise, we we use Theorem 7 to test if if the complement $A^\omega \setminus L$ is on level 3/2 of the concatenation hierarchy. By Theorem 2, this is equivalent to definability in ACTL. □

## 4 Concluding Remarks on CTL

Probably the most interesting remaining problem is this: what are the tree languages that can be defined in CTL? Is there an effective characterization? The most commonly cited tree property that cannot be defined in CTL is "on some path, there are infinitely many $a$'s". However, the weakness of CTL appears not only in infinitary behavior, and a lot of combinatorially interesting phenomena appear already in finite trees. For instance, the property "the prefix of some path is $(ab)^*c$" cannot be defined in CTL; both over finite and infinite trees (this provides a finitary language that can be expressed in LTL, but not CTL). For finite trees, there are algebraic tools to examine regular languages, such as syntactic objects and morphisms. Even with these tools, it is a nontrivial task to analyze CTL, for instance to show nondefinability of the $(ab)^*c$ language referred to above. But for infinite trees, the situation is exponentially worse, mainly because

there is no reasonable notion of a canonical representation of a tree language, or even a deterministic automaton model! Therefore, it seems a good idea to first understand the expressive power of CTL over finite trees, without tackling both finitary and infinitary aspects at the same time.

## References


1. Arfi, M.: Polynomial operations on rational languages. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 198–206. Springer, Heidelberg (1987)
2. Arfi, M.: Opérations polynomiales et hiérarchies de concaténation. Theor. Comput. Sci. 91(1), 71–84 (1991)
3. Clarke, E.M., Draghicescu, I.A.: Expressibility results for linear-time and branching-time logics. In: REX Workshop, pp. 428–437 (1988)
4. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
5. Kamp, J.A.: Tense Logic and the Theory of Linear Order. PhD thesis, Univ. of California, Los Angeles (1968)
6. Maidl, M.: The common fragment of CTL and LTL. In: Foundations of Computer Science, pp. 643–652 (2000)
7. Perrin, D., Pin, J.-É.: Infinite Words. Elsevier, Amsterdam (2004)
8. Pin, J.-É., Weil, P.: Polynomial closure and unambiguous product. Theory Comput. Systems 30, 1–30 (1997)
9. Potthoff, A.: First-order logic on finite trees. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) TAPSOFT 1995. LNCS, vol. 915, pp. 125–139. Springer, Heidelberg (1995)
10. Simon, I.: Factorization forests of finite height. Theoretical Computer Science 72, 65–94 (1990)
11. Thomas, W.: Classifying regular events in symbolic logic. Journal of Computer and System Sciences 25, 360–375 (1982)

# The Complexity of CTL* + Linear Past

Laura Bozzelli

[1] Università di Napoli Federico II, Via Cintia, 80126 - Napoli, Italy

**Abstract.** We investigate the complexity of satisfiability and finite-state model-checking problems for the branching-time logic $\mathrm{CTL}^*_{lp}$, an extension of CTL* with past-time operators, where past is linear, finite, and cumulative. It is well-known that $\mathrm{CTL}^*_{lp}$ has the same expressiveness as standard CTL*, but the translation of $\mathrm{CTL}^*_{lp}$ into CTL* is of non-elementary complexity, and no elementary upper bounds are known for its satisfiability and finite-state model checking problems. In this paper, we provide an elegant and uniform framework to solve these problems, which non-trivially extends the standard automata-theoretic approach to CTL* model-checking. In particular, we show that the satisfiability problem for $\mathrm{CTL}^*_{lp}$ is 2EXPTIME-complete, which is the same complexity as that of CTL*, but for the existential fragment of $\mathrm{CTL}^*_{lp}$, the problem is EXPSPACE-complete, hence exponentially harder than that of the existential fragment of CTL*. For the model-checking, the problem is already EXPSPACE-complete for the existential and universal fragments of $\mathrm{CTL}^*_{lp}$. For full $\mathrm{CTL}^*_{lp}$, the proposed algorithm runs in time polynomial in the size of the Kripke structure and doubly exponential in the size of the formula. Thus, the exact complexity of model-checking full $\mathrm{CTL}^*_{lp}$ remains open: it lies somewhere between EXPSPACE and 2EXPTIME.

## 1 Introduction

Temporal logics provide a fundamental framework for the description of dynamic behavior of reactive systems [Pnu77]. Usually, in standard temporal logics such as CTL* [EH86], CTL [CE81] and LTL [Pnu77], the modalities only refer to the future of the current time. On the other hand, it is well-known that temporal logics combining past and future modalities make some specifications easier to write and more natural, and for standard linear-time temporal logics, these extensions do not increase the complexity of basic decision problems [Var88]. For the branching-time setting, there are essentially two possible views regarding the nature of the past. In the first view, past is branching and each moment in time may have several possible futures and several possible pasts. In the second view, *past is linear* and each moment in time may have several possible futures and a unique past. Usually, the past is assumed to be finite (since program computations have a definite starting time) and cumulative (i.e., the history of the current situation increases with time and is never forgotten). However, the linear past (rather that branching-past) approach is more suited to the specification of dynamic behavior because it considers states in a computation tree, while

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 186–200, 2008.

the branching-past approach consider machine states (where past is not very meaningful to specify behavioral constraints) [LS95].

For the future (regular) branching-time temporal logic CTL*, the most simple linear-past extension is the logic PCTL* [HT87], obtained by adding the past counterparts of the standard linear-time modalities 'next' and 'until'. However, since the semantics of the path quantifiers in PCTL* is the same as for CTL* (i.e., path quantification ranges over paths that starts in the current node of the computation tree), the usage of past-time modalities is very limited. In other terms, past cannot go beyond the present. It is not surprising then, that PCTL* has the same expressivity and complexity as CTL*. A more interesting and meaningful linear past extension of CTL* is the logic $\mathrm{CTL}^*_{lp}$ [KP95]. $\mathrm{CTL}^*_{lp}$ has the same syntax as PCTL*. However, path quantification is 'memoryful', i.e., it ranges over paths that start at the root and visits the current node. $\mathrm{CTL}^*_{lp}$ is as expressive as CTL*, but the translation of $\mathrm{CTL}^*_{lp}$ into CTL* is of non-elementary complexity, and no elementary upper bounds are known for its satisfiability and finite-state model checking problems [KP95, LS95, LS00, KV06]. More recently, Kupferman and Vardi [KV06] introduce a memoryful variant of CTL*, called mCTL*, which unifies CTL* and the Pistore-Vardi logic [PV03]. This new logic has the same syntax as CTL*. The unique difference is the adding of a special proposition `present` which is needed to emulate the ability of CTL* to talk about the 'present'. By letting path quantification to range over paths that start at the root, an mCTL* formula can refer to events that happen in the past. However, since mCTL* do not contain explicit past-time operators, the ability to refer to the past is limited. The logic mCTL* is as expressive as CTL*, but while satisfiability for mCTL* is 2Exptime-complete, not harder than that of CTL*, its model checking problem is Expspace-complete, exponentially harder than that of CTL* (and this last result holds also for the fragment $\mathrm{mCTL}^*_{-}$ obtained by disallowing the special atom `present`). Moreover, $\mathrm{mCTL}^*_{-}$ can be linearly translated into $\mathrm{CTL}^*_{lp}$ and mCTL* can be linearly translated into the extension of $\mathrm{CTL}^*_{lp}$ with the special atom `present`.

***Our contribution.*** In this paper, we study the complexity of the satisfiability and (finite-state) model checking problems for $\mathrm{CTL}^*_{lp}$ and its existential and universal fragments $\mathrm{ECTL}^*_{lp}$ and $\mathrm{ACTL}^*_{lp}$. The existential (resp., universal) fragment consists of formulas where the only allowed path quantifier is the existential (resp., universal) one, assuming that formulas are written in positive normal form. We also consider the extension $\mathrm{CTL}^*_{lp}+$ of $\mathrm{CTL}^*_{lp}$ obtained by adding the special atom `present` and its existential and universal fragments $\mathrm{ECTL}^*_{lp}+$ and $\mathrm{ACTL}^*_{lp}+$. Our results are summarized in Figure 1 in which we also recall the well-known results about the complexity of the considered problems for CTL* and its existential and universal fragments (see, e.g., [KV00]). For the satisfiability problem, the complexity for $\mathrm{CTL}^*_{lp}$ and $\mathrm{CTL}^*_{lp}+$ is the same as that of CTL*, i.e. 2Exptime-complete. However, for the universal and existential fragments of the considered logics, the situation is quite different. While the complexity of $\mathrm{ACTL}^*_{lp}$ is the same as that of ACTL* (i.e., Pspace-complete), for the fragments $\mathrm{ECTL}^*_{lp}$, $\mathrm{ECTL}^*_{lp}+$, and $\mathrm{ACTL}^*_{lp}+$, the problem is

significantly harder being EXPSPACE-complete. For the model checking problem, the complexity is already EXPSPACE-complete for the existential and universal fragments of $\mathrm{CTL}^*_{lp}$. For full $\mathrm{CTL}^*_{lp}$ and $\mathrm{CTL}^*_{lp}+$, our algorithm runs in time polynomial in the size of the Kripke structure and doubly exponential in the size of the formula. Thus, the exact complexity of model-checking full $\mathrm{CTL}^*_{lp}$ and $\mathrm{CTL}^*_{lp}+$ remains open: it lies somewhere between EXPSPACE and 2EXPTIME.

The upper bounds of the considered problems are established by a uniform automata-theoretic framework which non-trivially generalizes the standard one for $\mathrm{CTL}^*$ [KVW00], and is based on the translation of $\mathrm{CTL}^*_{lp}+$ formulas, with a single exponential blow-up, into a two-way extension of the *symmetric* version of *hesitant alternating* (*finite-state*) *tree automata* (HAA, for short) [KVW00]. Two-way symmetric alternating tree automata (two-way SAA, for short), and in particular, two-way (symmetric) HAA, operate on *arbitrary* (also infinite-branching) $\Sigma$-labelled trees for a given alphabet $\Sigma$. However, SAA cannot distinguish between the different children of a node, and send copies to the children of the current input node in either a universal or an existential manner. Moreover, *two-way* SAA can send copies to the parent (if any) of the current node.

The key aspects of our approach which enable us to solve *partially* the open problems regarding the complexity of $\mathrm{CTL}^*_{lp}$ are the following:

- the complementation result for tree languages accepted by alternating tree automata based on the construction of the dual automaton [MS87], holds also for pointed tree languages (i.e., languages consisting of pairs $(T, x)$ where $T$ is a labelled tree and $x$ is a $T$-node) accepted by two-way SAA. This is a consequence of determinacy of (finitely-coloured) parity games, that holds also when a vertex in the underlying graph has *infinite* successors [Zie98].
- We show that parity two-way SAA can be linearly translated in the full modal $\mu$-calculus (with both backward and forward modalities), which satisfies the bounded-degree tree-model property [Var98]. As a consequence nonemptiness of parity two-way SAA can be linearly reduced to nonemptiness of standard parity two-way alternating tree automata operating on complete $n$-ary trees [Var98], where $n$ is the size of the given two-way SAA.
- The ability of combining both forward and backward moves in two-way HAA is restricted in such a way in every run each (infinite) path has a suffix which is fully *downward*. This allows us to solve the model checking problem for this class of automata by a direct construction. In particular, for the existential fragments of the considered logics, the model checking for the corresponding two-way HAA can be reduced to nonemptiness of 1-letter (one–way) HAA (over infinite words) [KVW00]. For full $\mathrm{CTL}^*_{lp}$ instead, we obtain an extended version of 1-letter HAA in which the 'universal requirement' for universal components of the automaton is relaxed. This explains our difficulty in obtaining membership in EXPSPACE for model checking of full $\mathrm{CTL}^*_{lp}$. However, the extended 1-letter HAA obtained in the construction have a special structure, but actually we do not know if this is sufficient to solve nonemptiness with the same complexity as for 1-letter HAA.

The full version of this paper can be asked to the author by e-mail.

| | | | Satisfiability | Model checking |
|---|---|---|---|---|
| CTL* | | | 2EXPTIME-complete | PSPACE-complete |
| ACTL* | ECTL* | | PSPACE-complete | PSPACE-complete |
| $CTL^*_{lp}+$ | $CTL^*_{lp}$ | | 2EXPTIME-complete | $\in$2EXPTIME |
| $ACTL^*_{lp}+$ | $ECTL^*_{lp}+$ | $ECTL^*_{lp}$ | EXPSPACE-complete | EXPSPACE-complete |
| $ACTL^*_{lp}$ | | | PSPACE-complete | EXPSPACE-complete |

**Fig. 1.** Summary of known and new results

## 2 Linear-Past Branching-Time Temporal Logic

In this section we recall syntax and semantics of the linear-past branching-time temporal logic $CTL^*_{lp}$ [KP95] and its extension, denoted $CTL^*_{lp}+$, obtained by adding the special atomic proposition `present` [KV06], which intuitively allows to refer to the 'present'. We also define the problems addressed in this paper.

Let $\mathbb{N}$ be the set of natural numbers. A *tree* $T$ is a prefix closed subset of $\mathbb{N}^*$. The elements of $T$ are called *nodes* and the empty word $\varepsilon$ is the *root* of $T$. For $x \in T$, the set of *children* of $x$ (in $T$) is $\mathsf{children}(x,T) = \{x \cdot i \in T \mid i \in \mathbb{N}\}$, and the *branching degree* of $x$ is the cardinality (possibly infinite) of $\mathsf{children}(x,T)$. The tree $T$ is *infinite* if each its node has at least a child. A *path* of $T$ is an infinite sequence $\pi = x_0 x_1 \ldots$ of $T$-nodes such that $x_{i+1} \in \mathsf{children}(x_i,T)$ for each $i \geq 0$. Let $\pi(i)$ be the $i^{th}$ node of $\pi$. For $x \in T$, an $x$-path is a path starting from $x$. For an alphabet $\Sigma$, a $\Sigma$-labelled tree is a pair $\langle T, V\rangle$ where $T$ is a tree and $V : T \to \Sigma$. For $x \in T$, the pair $(\langle T, V\rangle, x)$ is called *pointed* $\Sigma$-labelled tree.

The logic $CTL^*_{lp}+$ combines both branching-time and linear-time operators. A path quantifier, $\mathsf{E}$ ("for some path") or $\mathsf{A}$ ("for all paths"), can be followed by an arbitrary linear-time formula over the usual future linear temporal operators $\mathsf{X}^+$ ("forward next"), $\mathsf{U}^+$ ("forward until"), and $\mathsf{G}^+$ ("forward always"), and their past counterparts $\mathsf{X}^-$, $\mathsf{U}^-$, and $\mathsf{G}^-$. As in standard CTL*, for a given finite set of atomic propositions $AP$, there are two types of formulas in $CTL^*_{lp}+$: *state formulas* $\varphi$, whose satisfaction is related to a specific node of a $2^{AP}$-labelled tree, and *path formulas* $\xi$, whose satisfaction is related to a specific path. Their syntax (in *positive normal form*) is defined as follows:

$$\varphi := \top \mid prop \mid \neg prop \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{E}\,\xi \mid \mathsf{A}\,\xi$$
$$\xi := \varphi \mid \mathtt{present} \mid \neg\mathtt{present} \mid \xi \wedge \xi \mid \xi \vee \xi \mid \mathsf{X}^{dir}\xi \mid \neg\mathsf{X}^{-}\top \mid \xi\,\mathsf{U}^{dir}\xi \mid \mathsf{G}^{dir}\xi$$

where $\top$ denotes `true`, $prop \in AP$, $\mathtt{present} \notin AP$ and $dir \in \{+,-\}$. We also use the classical shortcut $\mathsf{F}^{dir}\xi$ for $dir \in \{+,-\}$ ("backward and forward eventually") which stands for $\top\,\mathsf{U}^{dir}\xi$. The set of state formulas $\varphi$ forms the language $CTL^*_{lp}+$.[1] $CTL^*_{lp}$ is the fragment of $CTL^*_{lp}+$ obtained by disallowing the atom `present`. We also study the existential fragment $ECTL^*_{lp}$ (resp., $ECTL^*_{lp}+$) and

[1] Note that the given syntax is complete since the dual $\widetilde{\mathsf{U}}^{dir}$ of the until operator $\mathsf{U}^{dir}$ can be expressed as follows: $\xi_1\,\widetilde{\mathsf{U}}^{dir}\,\xi_2 \equiv \mathsf{G}^{dir}\xi_2 \vee (\xi_2\,\mathsf{U}^{dir}(\xi_1 \wedge \xi_2))$.

the universal fragment $\mathrm{ACTL}^*_{lp}$ (resp., $\mathrm{ACTL}^*_{lp}+$) of $\mathrm{CTL}^*_{lp}$ (resp., $\mathrm{CTL}^*_{lp}+$) obtained by disallowing the path quantifier $\mathsf{A}$ and $\mathsf{E}$, respectively.

$\mathrm{CTL}^*_{lp}+$ formulas are interpreted over $2^{AP}$-labelled infinite trees. Fix a $2^{AP}$-labelled infinite tree $\langle T, V\rangle$ and let $\pi$ be an $\varepsilon$-path of $T$, $x \in T$, and $k, k_0 \in \mathbb{N}$. For a state formula $\varphi$, we write $x \models \varphi$ to mean that $\varphi$ holds at node $x$. Similarly, for a path formula $\xi$, we write $(\pi, k, k_0) \models \xi$ to indicate that $\xi$ holds at position $k$ along the $\varepsilon$-path $\pi$ of $\langle T, V\rangle$, where $k_0$ is the reference position (intuitively, the 'present'). Formally, we have the following (we omit the rules for atoms in $AP$ and boolean connectives, which are standard):

| | | |
|---|---|---|
| $x \models \mathsf{E}\xi$ | `iff` | there is an $\varepsilon$-path $\pi = x_0 x_1 \ldots$ and $k \geq 0$ such that $x_k = x$ and $(\pi, k, k) \models \xi$ |
| $x \models \mathsf{A}\xi$ | `iff` | for each $\varepsilon$-path $\pi = x_0 x_1 \ldots$ such that $x_k = x$ for some $k \geq 0$, we have $(\pi, k, k) \models \xi$ |
| $(\pi, k, k_0) \models \varphi$ | `iff` | $\pi(k) \models \varphi$ |
| $(\pi, k, k_0) \models$ `present` | `iff` | $k = k_0$ |
| $(\pi, k, k_0) \models \mathsf{X}^+\xi$ | `iff` | $(\pi, k+1, k_0) \models \xi$ |
| $(\pi, k, k_0) \models \mathsf{X}^-\xi$ | `iff` | $k > 0$ and $(\pi, k-1, k_0) \models \xi$ |
| $(\pi, k, k_0) \models \xi_1 \,\mathsf{U}^+ \xi_2$ | `iff` | $\exists\, n \geq k.\, (\pi, n, k_0) \models \xi_2$ and $\forall\, k \leq i < n.\, (\pi, i, k_0) \models \xi_1$ |
| $(\pi, k, k_0) \models \xi_1 \,\mathsf{U}^- \xi_2$ | `iff` | $\exists\, n \leq k.\, (\pi, n, k_0) \models \xi_2$ and $\forall\, n < i \leq k.\, (\pi, i, k_0) \models \xi_1$ |
| $(\pi, k, k_0) \models \mathsf{G}^+\xi$ | `iff` | $\forall\, n \geq k.\, (\pi, n, k_0) \models \xi$ |
| $(\pi, k, k_0) \models \mathsf{G}^-\xi$ | `iff` | $\forall\, n \leq k.\, (\pi, n, k_0) \models \xi$ |

For a $\mathrm{CTL}^*_{lp}+$ formula $\varphi$, we denote by $\mathcal{L}_p(\varphi)$ the set of pointed $2^{AP}$-labelled infinite trees $(\langle T, V\rangle, x)$ such that $x \models \varphi$. Note that while in standard $\mathrm{CTL}^*$, path quantification ranges over paths that start in the current node, in $\mathrm{CTL}^*_{lp}+$ path quantification ranges over paths that start at the root and visit the current node. For example, $\mathsf{AG}^+\mathsf{EF}^-(\xi \wedge \neg\mathsf{X}^-\top)$, when viewed as a formula of $\mathrm{CTL}^*$ extended with backward modalities, is unsatisfiable. When viewed as a $\mathrm{CTL}^*_{lp}+$ formula, it holds iff for each node $x$ of the given tree, the partial path from the root to $x$ can be extended to a path (initially) satisfying $\xi$.

In the following we also consider the linear temporal logic PLTL+ (LTL + Past + `present`) corresponding to $\mathrm{CTL}^*_{lp}+$ formulas which do not contain occurrences of $\mathsf{A}$ and $\mathsf{E}$. PLTL+ is interpreted on *pointed* (*infinite*) *words* over $2^{AP}$, i.e. pairs $(w, k)$ such that $w \in (2^{AP})^\omega$ and $k \in \mathbb{N}$. The satisfaction relation $(w, k, k_0) \models \xi$, meaning that $\xi$ holds at position $k$ of $w$ w.r.t. the reference position $k_0$, is defined similarly to the relation $(\pi, k, k_0) \models \xi'$ for path formulas $\xi'$ of $\mathrm{CTL}^*_{lp}+$. Let $\mathcal{L}_p(\xi)$ be the set of pointed words $(w, k)$ such that $(w, k, k) \models \xi$.

A *Kripke structure* over $AP$ is a tuple $\mathcal{K} = \langle S, s_0, \Delta, L\rangle$, where $S$ is a *finite* set of states, $s_0 \in S$ is an initial state, $\Delta \subseteq S \times S$ is a transition relation that must be total, and $L : S \to 2^{AP}$ maps each state $s$ to the set of atomic propositions true in $s$. The Kripke structure $\mathcal{K}$ induces a $2^{AP}$-labelled tree, denoted by $CT_{\mathcal{K}}$, which corresponds to the unwinding of $\mathcal{K}$ from $s_0$ (defined in the usual way).

We address the following problems for $\mathrm{CTL}^*_{lp}+$ (and its mentioned fragments):

- the *satisfiability* problem is to decide, given a $\mathrm{CTL}^*_{lp}+$ formula $\varphi$ over $AP$, whether $(\langle T, V\rangle, \varepsilon) \in \mathcal{L}_p(\varphi)$ for some $2^{AP}$-labelled infinite tree $\langle T, V\rangle$;

– the (*finite-state*) *model checking* problem is to decide, given a Kripke structure $\mathcal{K}$ and a CTL$^*_{lp}$+ formula $\varphi$ over $AP$, whether $(CT_{\mathcal{K}}, \varepsilon) \in \mathcal{L}_p(\varphi)$.

## 3 Alternating Finite–State Automata for Linear Past

In order to solve satisfiability and model-checking for CTL$^*_{lp}$+ and its fragments, we propose an extension of the automata-theoretic approach to branching-time model checking [KVW00]. In particular, we consider *two-way symmetric alternating* (*finite-state*) *tree automata* (two-way SAA), and more specifically we focus on a subclass of such automata. One–way SAA were first introduced in [Wil99] and operate on *arbitrary* $\Sigma$-labeled infinite trees (whose nodes can have infinite branching degrees) for a given alphabet $\Sigma$. SAA cannot distinguish between the different children of a node, and send copies to the children of the current input node in either a universal or an existential manner. Moreover, *two-way* SAA can send copies to the parent (if any) of the current node. In order to formally define such a class of automata, we need additional notation.

For a set $X$, $\mathcal{B}_+(X)$ denotes the set of *positive* boolean formulas over $X$, built from elements in $X$ using $\vee$ and $\wedge$ (we also allow the formulas `true` and `false`). A subset $Y$ of $X$ *satisfies* $\theta \in \mathcal{B}_+(X)$ iff the truth assignment that assigns `true` to the elements in $Y$ and `false` to the elements of $X \setminus Y$ satisfies $\theta$; $Y$ *exactly satisfies* $\theta$ if $Y$ satisfies $\theta$ and every proper subset of $Y$ does not satisfy $\theta$.

A two-way SAA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, Acc \rangle$, where $\Sigma$ is the input alphabet, $Q$ is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to \mathcal{B}_+\big((\{\Box, \Diamond\} \times Q) \cup (\{\uparrow\} \times Q \times \{\texttt{true}, \texttt{false}\})\big)$ is the transition function, and $Acc$ is an acceptance condition. Intuitively, a target of a move of $\mathcal{A}$ is encoded by an element in $(\{\Box, \Diamond\} \times Q) \cup (\{\uparrow\} \times Q \times \{\texttt{true}, \texttt{false}\})$. An atom $(\Diamond, q)$ means that a copy of $\mathcal{A}$ in state $q$ moves to some child of the current node, while an atom $(\Box, q)$ means that for each child $x$ of the current node, a copy of $\mathcal{A}$ in state $q$ is sent to node $x$. Finally, an atom $(\uparrow, q, b)$ can be chosen iff either the current node is not the root or $b = \texttt{true}$. In the first case, a copy of $\mathcal{A}$ in state $q$ is sent to the parent of the current node. A *one-way* SAA is a two-way SAA whose transition function satisfies $\delta(q, a) \in \{\Box, \Diamond\} \times Q$ for each $(q, \sigma) \in Q \times \Sigma$.

For a pointed $\Sigma$-labelled infinite tree $(\langle T, V \rangle, x_0)$, a *run* of $\mathcal{A}$ over $(\langle T, V \rangle, x_0)$ is a $Q \times T$-labelled tree $r = \langle T_r, V_r \rangle$, where each node of $T_r$ labelled by $(q, x)$ describes a copy of $\mathcal{A}$ that is in state $q$ and reads the node $x$ of $T$. Moreover, we require that $r(\varepsilon) = (q_0, x_0)$ (initially, $\mathcal{A}$ is in state $q_0$ reading node $x_0$), and for each $y \in T_r$ with $r(y) = (q, x)$, there is a (possibly empty) set $H \subseteq (\{\Box, \Diamond\} \times Q) \cup (\{\uparrow\} \times Q \times \{\texttt{true}, \texttt{false}\})$ exactly satisfying $\delta(q, V(x))$ such that $H$ does not contain atoms $(\uparrow, q', \texttt{false})$ if $x = \varepsilon$, and $\mathsf{children}(y, T_r)$ satisfies the following for each $at \in H$:

- if $at = (\Diamond, q')$, then $\exists x' \in \mathsf{children}(x, T)$, $\exists y' \in \mathsf{children}(y, T_r)$. $r(y') = (q', x')$;
- if $at = (\Box, q')$, then $\forall x' \in \mathsf{children}(x, T)$, $\exists y' \in \mathsf{children}(y, T_r)$. $r(y') = (q', x')$;
- if $at = (\uparrow, q', b)$ and $x = x' \cdot i$, then $\exists y' \in \mathsf{children}(y, T_r)$. $r(y') = (q', x')$.

For a path $\pi = y_0 y_1 \ldots$ of the run $r = \langle T_r, V_r \rangle$, let $inf(\pi)$ be the set of states in $Q$ that appear in $V_r(y_0)V_r(y_1)\ldots$ infinitely often. We say that $\pi$ is accepting

iff $inf(\pi)$ *satisfies* the acceptance condition $Acc$ of $\mathcal{A}$. The run $r = \langle T_r, V_r \rangle$ is *accepting* iff each its path is accepting. Here, we consider *parity acceptance conditions*, specified by mappings[2] $\Omega : Q \to \mathbb{N}$ assigning to each state $q \in Q$ an integer (called *priority*). A path $\pi$ trough a run satisfies $\Omega$ if the smallest priority of the states in $inf(\pi)$ is *even*. The *index* of a parity two-way SAA with parity acceptance condition $\Omega$ is the cardinality of the set $\{\Omega(q) \mid q \in Q\}$.

For a two-way SAA over $\Sigma$, the *pointed language* $\mathcal{L}_p(\mathcal{A})$ of $\mathcal{A}$ is the set of pointed $\Sigma$-labeled infinite trees $PT$ such that $\mathcal{A}$ has an accepting run over $PT$. The *language* $\mathcal{L}(\mathcal{A})$ is the set of $\Sigma$-labelled trees $\langle T, V \rangle$ s.t. $(\langle T, V \rangle, \varepsilon) \in \mathcal{L}_p(\mathcal{A})$.

Given a parity two-way SAA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \Omega \rangle$, the *dual automaton of* $\mathcal{A}$ is the parity two-way SAA $\widetilde{\mathcal{A}} = \langle \Sigma, Q, q_0, \widetilde{\delta}, \widetilde{\Omega} \rangle$, where for each $q \in Q$, $\widetilde{\Omega}(q) = \Omega(q) + 1$, and for each $(q, \sigma)$, $\widetilde{\delta}(q, \sigma)$ is obtained from $\delta(q, \sigma)$ by switching $\Box$ and $\Diamond$, switching $\vee$ and $\wedge$, and switching `true` and `false`. If, for example, $\delta(q, \sigma) = (\Box, p) \vee (\uparrow, q, \texttt{true})$, then $\widetilde{\delta}(q, \sigma) = (\Diamond, p) \wedge (\uparrow, q, \texttt{false})$.

We can give a game-theoretic interpretation of acceptance in two-way SAA $\mathcal{A}$ by (finitely coloured) parity games. Since the determinacy result for such a class of games holds also when the number of successors of a vertex in the underlying graph is infinite [Zie98], by a readaptation of the proof given in [MS87], it follows that the *dual automaton* of $\mathcal{A}$ accepts the complement of $\mathcal{L}_p(\mathcal{A})$, i.e., the set of pointed $\Sigma$-labeled infinite trees $PT \notin \mathcal{L}_p(\mathcal{A})$.

**Proposition 1.** *The dual automaton of a parity two-way* SAA $\mathcal{A}$ *accepts the complement of* $\mathcal{L}_p(\mathcal{A})$.

As we will see in order to capture $\text{CTL}^*_{lp}+$ formulas, it suffices to consider a subclass of parity two-way SAA of index 3 corresponding to a two-way extension of *hesitant alternating tree automata* (HAA) introduced in [KVW00] as an optimal automata-theoretic framework for $\text{CTL}^*$. Formally, a *two-way* HAA is a two-way SAA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, Acc \rangle$ satisfying the following conditions. As in weak alternating automata, there is a partition of $Q$ into disjoint sets $Q_1, \ldots, Q_m$ (called *components* of $\mathcal{A}$) and a partial order $\leq$ on these sets such that transitions from a state in $Q_i$ lead to states in either the same $Q_i$ or components $Q_j$ such that $Q_j < Q_i$ (*partial order requirement*). Moreover, each component $Q_i$ is classified either as *transient*, *existential*, or *universal*, and the following holds:

1. for each transient set $Q_i$ and $q \in Q_i$, $\delta(q, \sigma)$ contains no states of $Q_i$;
2. for each existential component $Q_i$ and $q \in Q_i$, if $\delta(q, a)$ is rewritten in disjunctive normal form, then there is at most one (forward) atom $(c, q')$ with $q' \in Q_i$ in each disjunct. Moreover, $c = \Diamond$ (*existential requirement*);
3. for each universal component $Q_i$ and $q \in Q_i$, if $\delta(q, a)$ is rewritten in conjunctive normal form, then there is at most one (forward) atom $(c, q')$ with $q' \in Q_i$ in each conjunct. Moreover, $c = \Diamond$ (*universal requirement*);
4. $Acc$ consists of a pair $\langle G, B \rangle$ of sets of states interpreted as the parity condition $\Omega_{\langle G,B \rangle}$ of index 3 assigning 0 to the states in $Q_\exists \cap G$, assigning 1 to the states in $(Q_\exists \setminus G) \cup (Q_\forall \cap B)$, and assigning 2 to the remaining states, where $Q_\exists$ (resp., $Q_\forall$) is the set of existential (resp., universal) states.

[2] We use the symbol $\Omega$ instead of $Acc$ to specify such acceptance conditions.

5. in addition for a *two-way* HAA $\mathcal{A}$, we require that $Q$ is also partitioned into a set $Q_+$ of *positive* states and a set $Q_-$ of *negative* states such that: (i) $q_0 \in Q_+$, (ii) for each atom $(\uparrow, q, b)$ (backward choice) occurring in $\delta$, $q \in Q_-$, and for each atom $(c, q)$ (forward choice) occurring in $\delta$, $q \in Q_+$, and (iii) for each component $Q_i$ and negative state $q_- \in Q_i$, $\delta(q_-, \sigma)$ does not contain atoms of the form $(\Diamond, q)$ or $(\Box, q)$ with $q \in Q_i$.

The partial order requirement and Condition 1 ensure that every path $\pi$ of a run of $\mathcal{A}$ gets trapped within some existential or universal component $Q_i$. Then, by Condition 4, the path satisfies the acceptance condition $Acc = \langle G, B\rangle$ iff either $Q_i$ is an existential set and $inf(\pi) \cap G \neq \emptyset$ (Büchi condition), or $Q_i$ is a universal set and $inf(\pi) \cap B = \emptyset$ (co-Büchi condition). Condition 5 ensures that every path $\pi$ get trapped in $Q_+$, and in particular from a certain point on, $\pi$ becomes fully *downward*, i.e. there is a suffix of $\pi$ such that each node along this suffix is obtained from the previous by applying a forward choice (corresponding to an atom of the form $(c, q)$ with $c \in \{\Box, \Diamond\}$). The *depth* of $\mathcal{A}$ is the number of components of $\mathcal{A}$. The two-way HAA $\mathcal{A}$ is *existential* if each its component is not universal, and is *strictly existential* if its transition function does not contain atoms of the form $(\Box, q)$. Note that the dual automaton $\widetilde{\mathcal{A}} = \langle \Sigma, Q, q_0, \widetilde{\delta}, \widetilde{\Omega_{\langle G,B\rangle}}\rangle$ of $\mathcal{A}$ is still a two-way HAA. Indeed, the components of $\widetilde{\mathcal{A}}$ are the same as $\mathcal{A}$ (with the same partial order) with the difference that a component that is existential in $\mathcal{A}$ is universal in $\widetilde{\mathcal{A}}$, and vice versa. Moreover, Condition 5 continue to hold (the sets of positive states and negative states of $\widetilde{A}$ are the same as $\mathcal{A}$). Finally, it is easy to show that the parity condition $\widetilde{\Omega_{\langle G,B\rangle}}$, when interpreted on runs of $\widetilde{A}$, is equivalent to the parity condition $\Omega_{\langle B,G\rangle}$ associated with $\langle B, G\rangle$. Thus, by Proposition 1 we obtain the following result.

**Proposition 2.** *The dual automaton of a two-way* HAA $\mathcal{A}$ *is a two-way* HAA *accepting the complement of* $\mathcal{L}_p(\mathcal{A})$.

We address the following problems for the class of two-way HAA:

- the *nonemptiness* problem is to decide, for a two-way HAA, whether $\mathcal{L}(\mathcal{A}) \neq \emptyset$;
- the (*finite-state*) *model checking* problem is to decide, given a Kripke structure $\mathcal{K}$ over $AP$ and a two-way HAA $\mathcal{A}$ over $2^{AP}$, whether $CT_{\mathcal{K}} \in \mathcal{L}(\mathcal{A})$.

In the following, we also consider (one-way) HAA on *infinite words* over a 1-letter alphabet (1-letter HAA). Note for such a class of automata, choices represented by atoms $(\Diamond, q)$ and $(\Box, q)$ are equivalent, and thus the transition function can be given as a mapping $\delta : Q \to \mathcal{B}_+(Q)$, where $Q$ is the set of states.

### 3.1 Decision Procedures for Two-Way HAA

**Model checking.** We reduce the model checking problem for two-way HAA to the nonemptiness problem of *extended* 1-letter HAA corresponding to 1-letter HAA in which the universal requirement for universal components (see Condition 3 in the definition of HAA) is relaxed.

**Theorem 1.** *For a Kripke structure $\mathcal{K}$ over AP of size $n$ and a* two-way HAA *$\mathcal{A}$ over $2^{AP}$ with depth $d$ and size $m$, one can build an extended* 1*-letter* HAA *$\mathcal{A}_{\mathcal{K}}$ with depth $d$ and size $O(n \cdot m \cdot 2^{O(m)})$ s.t. $\mathcal{L}(\mathcal{A}_{\mathcal{K}}) \neq \emptyset$ iff $CT_{\mathcal{K}} \in \mathcal{L}(\mathcal{A})$. Moreover, if $\mathcal{A}$ is an existential two-way* HAA*, then $\mathcal{A}_{\mathcal{K}}$ is an existential* 1*-letter* HAA*.*

*Proof.* Let $\mathcal{K} = \langle S, s_0, \Delta, L\rangle$ and $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \langle G, B\rangle\rangle$ with $\Sigma = 2^{AP}$. Essentially, the 1-letter extended HAA $\mathcal{A}_{\mathcal{K}}$ guesses a run of $\mathcal{A}$ over $(CT_{\mathcal{K}}, \varepsilon)$ and checks that it is accepting. At a given node $x$ of a run of $\mathcal{A}_{\mathcal{K}}$, $\mathcal{A}_{\mathcal{K}}$ keeps track by its finite control of the following information: (i) the positive state $q_+$ associated with the current 'positive' copy of $\mathcal{A}$ in the guessed run, (ii) the state $s$ of $\mathcal{K}$ associated with the node $x_{curr}$ of $CT_{\mathcal{K}}$ which is read by the current copy of $\mathcal{A}$, and (iii) the guessed set $P_-$ of negative states of $\mathcal{A}$ associated with the copies of $\mathcal{A}$ which read $x_{curr}$ and belong to the subrun starting from the current copy of $\mathcal{A}$. Note that since $\mathcal{A}$ is a two-way HAA, $P_-$ cannot contain states in components $Q_i$ of $\mathcal{A}$ that are upper in the partial order than the $q_+$-component. The paths of a run of $\mathcal{A}_{\mathcal{K}}$ correspond to the *downward* paths of the simulated run of $\mathcal{A}$ over $(CT_{\mathcal{K}}, \varepsilon)$. Since $\mathcal{A}$ is a two-way HAA, each infinite path of a run of $\mathcal{A}$ has a suffix which is downward. Thus, a run of $\mathcal{A}_{\mathcal{K}}$ keeps track of all meaningful information associated with the corresponding simulated run of $\mathcal{A}$ over $(CT_{\mathcal{K}}, \varepsilon)$.

The extended 1-letter HAA $\mathcal{A}_{\mathcal{K}} = \langle \{a\}, Q_{\mathcal{K}}, q^0_{\mathcal{K}}, \delta_{\mathcal{K}}, \langle G_{\mathcal{K}}, B_{\mathcal{K}}\rangle\rangle$ is formally defined as follows. Let $Q_1, \ldots, Q_d$ be a fixed total ordering of the components of $\mathcal{A}$ extending the partial order $\leq$ of $\mathcal{A}$, and let $Q_+$ (resp., $Q_-$) be the set of positive states (resp., negative states) of $\mathcal{A}$. For a state $q \in Q_i$, let $index(q) := i$, and for $q \in Q$, let $\Pi(q) = \{P_- \subseteq Q_- \mid \forall q_- \in P_-, index(q_-) \leq index(q)\}$.

A state of $\mathcal{A}_{\mathcal{K}}$ is either of the form $(q_+, s, P_{up}, root) \in Q_+ \times S \times 2^{Q_-} \times \{0,1\}$ or of the form $(q_+, s, P_{up}, P_{curr}, root) \in Q_+ \times S \times 2^{Q_-} \times 2^{Q_-} \times \{0,1\}$ such that $P_{curr} \in \Pi(q_+)$, where: (i) $q_+$ represents the state associated with the current 'positive' copy of $\mathcal{A}$ which reads a node $x$ of the computation tree of $\mathcal{K}$ labelled by state $s$, (ii) $P_{curr}$ represents the guessed set of negative states of $\mathcal{A}$ associated with the copies of $\mathcal{A}$ which read $x$ and belong to the 'subrun' of $\mathcal{A}$ starting from the current copy, (iii) $P_{up}$ represents the set of negative states of $\mathcal{A}$ associated with the copies of $\mathcal{A}$ which read the parent node $y$ of $x$ and belong to the 'subrun' of $\mathcal{A}$ associated with a positive copy (reading $y$) which has generated (in one step or many steps) the current copy, (iv) *root* is a flag which is 1 iff the current node $x$ of $CT_{\mathcal{K}}$ is the root. The initial state is $q^0_{\mathcal{K}} = (q_0, s_0, \emptyset, 1)$. The components of $\mathcal{A}_{\mathcal{K}}$ are $Q'_1, \ldots, Q'_d$ with $Q'_i \leq Q'_j$ iff $i \leq j$, where $Q'_i$ is the set of states $(q_+, s, P_{up}, root)$ or $(q_+, s, P_{up}, P_{curr}, root)$ such that $q_+ \in Q_i$. Moreover, $Q'_i$ is existential if $Q_i$ is either existential or transient, and is universal otherwise.

The transition function $\delta_{\mathcal{K}}$ is defined as follows:

1. $\delta_{\mathcal{K}}(q_+, s, P_{up}, root) = \bigvee_{P_{curr} \in \Pi(q_+)} (q_+, s, P_{up}, P_{curr}, root)$;
2. $\delta_{\mathcal{K}}(q_+, s, P_{up}, P_{curr}, root) = \theta(q_+, s, P_{up}, P_{curr}) \wedge \bigwedge_{q_- \in P_{curr}} \theta(q_-, s, P_{up}, P_{curr})$, where $\theta(q, s, P_{up}, P_{curr})$ is obtained from $\delta(q, L(s))$ as follows:
   - each atom $(\Box, p_+)$ (resp., $(\Diamond, p_+)$) occurring in $\delta(q, L(s))$ is replaced with $\bigwedge_{s' \in succ_{\mathcal{K}}(s)} (p_+, s', P_{curr}, 0)$ (resp., $\bigvee_{s' \in succ_{\mathcal{K}}(s)} (p_+, s', P_{curr}, 0)$);
   - each atom $(\uparrow, p_-, b)$ in $\delta(q, L(s))$ is replaced with `true` if either $root = 0$ and $p_- \in P_{up}$ or $root = 1$ and $b =$ `true`, and with `false` otherwise.

where $succ_{\mathcal{K}}(s)$ denotes the set of successors of $s$ in $\mathcal{K}$.

The acceptance condition $\langle G_{\mathcal{K}}, B_{\mathcal{K}} \rangle$ is defined as follows:

- $G_{\mathcal{K}} = \{(q_+, s, P_{up}, root), (q_+, s, P_{up}, P_{curr}, root) \mid q_+ \in G\}$;
- $B_{\mathcal{K}} = \{(q_+, s, P_{up}, root), (q_+, s, P_{up}, P_{curr}, root) \mid q_+ \in B\}$.

Note that $\delta_{\mathcal{K}}$ satisfies the partial order requirement. Moreover, a path of a run of $\mathcal{A}_{\mathcal{K}}$ cannot be trapped within an existential component $Q'_i$ corresponding to a transient component of $\mathcal{A}$. Also, Condition 5 in def. of two-way HAA ensures that an existential component of $\mathcal{A}_{\mathcal{K}}$ satisfies the existential requirement corresponding to condition 2 in def. of HAA. However, if $\mathcal{A}$ contains universal components, then the universal components of $\mathcal{A}_{\mathcal{K}}$ do not satisfy the universal requirement due to the nondeterministic choice of the set $P_{curr}$ in the transitions from states of the form $(q_+, s, P_{up}, root)$. Thus, if $\mathcal{A}$ is existential, then $\mathcal{A}_{\mathcal{K}}$ is an existential 1-letter HAA. Otherwise, $\mathcal{A}_{\mathcal{K}}$ is an *extended* 1-letter HAA. □

In [KVW00] it is shown that nonemptiness of 1-letter HAA (hence, also 1-letter existential HAA) of depth $d$ and size $n$ can be solved in space $O(d \log^2 n)$. Since extended 1-letter HAA are also 1-letter parity alternating automata over infinite words of index 3, and for such class of automata, nonemptiness can be solved in cubic time [KV98], by Theorem 1, we obtain the following upper bounds for the model checking of two-way HAA and two-way existential HAA.

**Theorem 2.** *Given a Kripke structure $\mathcal{K}$ over $AP$ of size $n$ and a* two-way HAA *$\mathcal{A}$ over $2^{AP}$ with depth $d$ and size $m$, the model checking problem for $\mathcal{K}$ and $\mathcal{A}$ can be solved in time $O(n^3 \cdot m^3 \cdot 2^{O(m)})$. Moreover, if $\mathcal{A}$ is* existential, *then the same problem can be solved in space $O(d \cdot \log^2(n \cdot m \cdot 2^{O(m)}))$.*

**Nonemptiness problem.** For the nonemptiness problem of two-way HAA and, more in general, parity two-way SAA, we obtain the following result.

**Theorem 3.** *The nonemptiness problem of parity two-way* SAA (*hence, also two-way* HAA ) *is in* Exptime.

*Proof.* We can show that for a parity two-way SAA $\mathcal{A}$ over $\Sigma$ of size $n$ and index $h$, it is possible to build a formula $\varphi_{\mathcal{A}}$ of the *full modal $\mu$-calculus* (with both forward and backward modalities) over $\Sigma$ [Var98] such that $\varphi_{\mathcal{A}}$ has size bounded by $2n$ and for each $\Sigma$-labelled tree $LT$, $LT \in \mathcal{L}(\mathcal{A})$ iff $LT$ is a tree-model of $\varphi_{\mathcal{A}}$. Since a formula $\varphi$ of the full $\mu$-calculus is satisfiable iff there is a tree-model of $\varphi$ whose branching degrees are bounded by the size of $\varphi$ [Var98], it follows that for the given parity two-way SAA $\mathcal{A}$ over $\Sigma$ of size $n$ and index $h$, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \cap \Upsilon_{2n}(\Sigma) \neq \emptyset$, where $\Upsilon_{2n}(\Sigma)$ denotes the set of $\Sigma$-labelled trees whose branching degrees are bounded by $2n$. Now, let $\perp$ be a symbol non in $\Sigma$. We can encode a labelled tree $LT$ in $\Upsilon_{2n}(\Sigma)$ as a $\Sigma \cup \{\perp\}$-labelled complete $2n$-ary tree $\langle \{1, \ldots, 2n\}^*, V \rangle$ as follows: first, for each node $x$ of $LT$ with $i$ children (note that $i \leq 2n$), we add $2n - i$ new children and label these new nodes with $\perp$; finally, for each node $x$ labelled by $\perp$, we add recursively $2n$ children labelled by $\perp$. Then, starting from $\mathcal{A}$, it is easy to construct a standard parity two-way alternating

tree automaton [Var98] (parity two-way ATA) $\mathcal{A}'$ operating on $\Sigma \cup \{\bot\}$-labelled complete $2n$-ary trees having the same index as $\mathcal{A}$ and size linear in the size of $\mathcal{A}$, and such that $\mathcal{A}'$ accepts all and only the trees encoding $\Sigma$-labelled trees in $\mathcal{L}(\mathcal{A}) \cap \Upsilon_{2n}(\Sigma)$. Hence, $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Since nonemptiness of parity two-way ATA of size $n$, index $h$ over $k$-ary trees can be solved in time $2^{O(n^2 \cdot k \cdot h)}$ [Var98], Theorem 3 follows. □

However, for nonemptiness of strictly existential two-way HAA $\mathcal{A}$, we can do better. Indeed, the transition function of $\mathcal{A}$ does not contain atoms of the form $(\Box, q)$ corresponding to universal choices. Thus, the only forward choices are existential (nondeterminism). Moreover, the automaton cannot distinguish between the different children of the current input node. Thus, if $\mathcal{A}$ is one-way, then it actually corresponds to a nondeterministic tree automaton, hence nonemptiness can be trivially reduced to nonemptiness of 1-letter HAA. If instead $\mathcal{A}$ is two-way, then we need to keep track only of the downward paths of a run of $\mathcal{A}$. These observations enable us to solve nonemptiness for $\mathcal{A}$ by a direct reduction to nonemptiness of an (existential) 1-letter HAA $\mathcal{A}_W$ having the same depth as $\mathcal{A}$ and exponential size. Thus, we obtain the following result.

**Theorem 4.** *Nonemptiness of* strictly existential *two-way* HAA *is in* PSPACE.

## 4 Decision Procedures for $\mathrm{CTL}^*_{lp}+$ and Its Fragments

In this section we describe an automata-theoretic approach to solve satisfiability and model-checking for $\mathrm{CTL}^*_{lp}+$ based on the translation (with a single exponential blow-up) of $\mathrm{CTL}^*_{lp}+$ formulas $\varphi$ into equivalent two-way HAA $\mathcal{A}_\varphi$ accepting the set of pointed labelled trees satisfying $\varphi$. Before illustrating this, we need a preliminary result concerning the translation of the linear temporal logic PLTL+ into a simple variant of two-way Büchi word automata [Var88].

A *simple two-way Büchi* (*nondeterministic*) *word automaton* (Büchi SNWA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \rho, F_-, F_+ \rangle$, where $\Sigma$ is the input alphabet, $Q$ is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\rho : Q \times \Sigma \times \{+, -\} \to 2^Q$ is a transition function, and $F_-$ and $F_+$ are sets of accepting states. A run of $\mathcal{A}$ over a pointed word $(w, i)$, where $w = w(0)w(1)\ldots$, is a pair $r = (r_-, r_+)$ such that $r_+ = q_i^+, q_{i+1}^+ \ldots$ is an infinite sequence of states, $r_- = q_i^-, q_{i-1}^- \ldots q_0^- q_{-1}^-$ is a finite sequence of states, and: (i) $q_i^+ = q_i^- \in Q_0$; (ii) for each $h \geq i$, $q_{h+1}^+ \in \rho(q_h^+, w(h), +)$; and (iii) for each $0 \leq h \leq i$, $q_{h-1}^- \in \rho(q_h^-, w(h), -)$.

Thus, starting from the initial position $i$ in the input pointed word $(w, i)$, the automaton splits in two copies: the first one moves forwardly along the suffix of $w$ starting from position $i$ and the second one moves backwardly along the prefix $w(0) \ldots w(i)$. The run $r = (r_-, r_+)$ is *accepting* if $q_{-1}^- \in F_-$ and $r_+$ visits infinitely often some state in $F_+$. A pointed word $(w, i)$ is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over $(w, i)$. By a readaptation of the standard translation of LTL into Büchi word automata [VW94], we obtain the following.

**Proposition 3.** *Given a* PLTL+ *formula* $\xi$ *over* $AP$*, one can construct a Büchi* SNWA *over* $2^{AP}$ *of size* $2^{O(|\xi|)}$ *accepting the set of pointed words satisfying* $\xi$.

**Theorem 5.** *For a* CTL$^*_{lp}$+ *formula* $\psi$ *over* $AP$*, one can build a two-way* HAA $\mathcal{A}_\psi$ *over* $2^{AP}$ *of size* $2^{O(|\psi|)}$ *and depth* $O(|\psi|)$ *such that* $\mathcal{L}_p(\mathcal{A}_\psi) = \mathcal{L}_p(\psi)$*. Also, if* $\psi$ *is a* ECTL$^*_{lp}$+ *formula, then* $\mathcal{A}_\psi$ *is a* strictly existential *two-way* HAA.

*Proof.* We need some definitions. A CTL$^*_{lp}$+ formula $\varphi$ is *trivial* if either $\varphi = p$ or $\varphi = \neg p$, where $p \in AP$. For two CTL$^*_{lp}$+ formulas $\varphi_{sub}$ and $\varphi$, $\varphi_{sub}$ is *maximal* in $\varphi$ if $\varphi_{sub}$ is a strict state subformula of $\varphi$ and there is an occurrence of $\varphi_{sub}$ in $\varphi$ s.t. there is no occurrence of a strict state subformula of $\varphi$ which strictly contains the considered occurrence of $\varphi_{sub}$. We denote by $max(\varphi)$ the set of all formulas maximal in $\varphi$. For example, $max(\mathsf{A}((\mathsf{X}^+\neg p)\,\mathsf{U}^+(\mathsf{EX}^-\neg p))) = \{\neg p, \mathsf{EX}^-\neg p\}$.

As in the case of the one-way HAA for CTL$^*$ [KVW00], we construct the two-way HAA $\mathcal{A}_\psi$ by induction on the structure of $\psi$. With each state subformula $\varphi$ of $\psi$, we associate a two-way HAA $\mathcal{A}_\varphi$ over $\Sigma = 2^{AP}$ of size $2^{O(|\varphi|)}$ and depth $O(|\varphi|)$ such that $\mathcal{L}_p(\mathcal{A}_\varphi) = \mathcal{L}_p(\varphi)$. For the base of the induction, either $\varphi = p$ or $\varphi = \neg p$, where $p \in AP$. In both cases, $\mathcal{A}_\varphi$ has one state $q_0$, which is positive and transient, acceptance condition $\langle \emptyset, \emptyset \rangle$, and transition function $\delta$ defined as follows. In the first case ($\varphi = p$), $\delta(q_0, \sigma) = \texttt{true}$ if $p \in \sigma$, and $\delta(q_0, \sigma) = \texttt{false}$ if $p \notin \sigma$. In the second case ($\varphi = \neg p$), $\delta(q_0, \sigma) = \texttt{true}$ if $p \notin \sigma$, and $\delta(q_0, \sigma) = \texttt{false}$ if $p \in \sigma$. Now, assume that $\varphi$ is a non-trivial state subformula of $\psi$ (induction step). Let $max(\varphi) = \{\varphi_1, \ldots, \varphi_n\}$. By the induction hypothesis for each $1 \leq i \leq n$, we can construct a two-way HAA $\mathcal{A}_{\varphi_i} = \langle \Sigma, Q^i, q_0^i, \delta^i, \langle G^i, B^i \rangle \rangle$ of size $2^{O(|\varphi_i|)}$ and depth $O(|\varphi_i|)$ such that $\mathcal{L}_p(\mathcal{A}_{\varphi_i}) = \mathcal{L}_p(\varphi_i)$. We assume that the state sets of the two-way HAA $\mathcal{A}_{\varphi_1}, \ldots, \mathcal{A}_{\varphi_n}$ are disjoint (otherwise, we rename the states). We construct a two-way HAA $\mathcal{A}_\varphi$ composed from $\mathcal{A}_{\varphi_1}, \ldots, \mathcal{A}_{\varphi_n}$ as follows. Since $\varphi$ is in positive normal form, there are only the following cases:

- $\varphi = \varphi_1 \wedge \varphi_2$ ($n = 2$). We define $\mathcal{A}_\varphi = \langle \Sigma, Q^1 \cup Q^2 \cup \{q_0\}, q_0, \delta, \langle G^1 \cup G^2, B^1 \cup B^2 \rangle \rangle$, where $q_0$ is a new (positive) state and $\delta$ is defined as follows. For states in $Q^1$ and $Q^2$, $\delta$ agrees with $\delta^1$ and $\delta^2$, respectively (recall that $Q^1$ and $Q^2$ are disjoint). For the state $q_0$ and for each $\sigma \in \Sigma = 2^{AP}$, $\delta(q_0, \sigma) = \delta^1(q_0^1, \sigma) \wedge \delta^2(q_0^2, \sigma)$. Thus, from the initial node of the pointed input tree and in the initial state $q_0$, $\mathcal{A}_\varphi$ sends all the copies sent (initially) by both $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$. The singleton $\{q_0\}$ constitutes a transient component, with the ordering $\{q_0\} > Q'$ for all components $Q'$ of $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$. Evidently, $\mathcal{L}_p(\mathcal{A}_\varphi) = \mathcal{L}_p(\mathcal{A}_{\varphi_1}) \cap \mathcal{L}_p(\mathcal{A}_{\varphi_2})$. Moreover, by the induction hypothesis, we have that $\mathcal{L}_p(\mathcal{A}_\varphi) = \mathcal{L}_p(\varphi)$, and $\mathcal{A}_\varphi$ has size $2^{O(|\varphi|)}$ and depth $O(|\varphi|)$.
- $\varphi = \varphi_1 \vee \varphi_2$. The construction of $\mathcal{A}_\varphi$ is similar to the previous case with the difference that now $\delta(q_0, \sigma) = \delta^1(q_0^1, \sigma) \vee \delta^2(q_0^2, \sigma)$.
- $\varphi = \mathsf{E}\xi$. Let $max(\varphi) = \{\varphi_1, \ldots, \varphi_n\}$ and let $\widehat{AP} = \{p_1, \ldots, p_n\}$ be a set of fresh propositions. Moreover,let $\widehat{\xi}$ be the PLTL+ formula over $\widehat{AP}$ obtained from the path formula $\xi$ by replacing each occurrence of $\varphi_i$ in $\varphi$ which is maximal in $\varphi$ with proposition $p_i$. Since $\widehat{\xi}$ and $\xi$ are in positive normal form and $\widehat{\xi}$ does not contain subformulas of the form $\neg p_i$ for each $p_i \in \widehat{AP}$, it easily follows that for each pointed $2^{AP}$-labelled tree $(\langle T, V \rangle, x)$,

**Claim 1:** $(\langle T, V\rangle, x) \models \mathsf{E}\xi$ *if and only if* there is a $\varepsilon$-path of $T$ $\pi = x_0 x_1 \dots x_k \dots$ with $x_k = x$ and an infinite word $w$ over $2^{\widehat{AP}}$ such that $(w, k, k) \models \widehat{\xi}$ and for each $i \geq 0$ and $p_h \in w(i)$, $(\langle T, V\rangle, x_i) \models \varphi_h$.

Claim 1 suggests the following construction. First, we build a two-way HAA $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ over $\widehat{\Sigma} = 2^{\widehat{AP}}$ accepting $\mathcal{L}_p(\mathsf{E}\widehat{\xi})$ as follows. Let $\mathcal{A}'_{\widehat{\xi}} = \langle \widehat{\Sigma}, Q, Q_0, \rho, F_-, F_+\rangle$ be the Büchi SNWA accepting the set of infinite pointed words over $\widehat{\Sigma}$ satisfying $\widehat{\xi}$ (whose existence is guaranteed by Proposition 3). Then, $\mathcal{A}_{\mathsf{E}\widehat{\xi}} = \langle \widehat{\Sigma}, \widehat{Q}, \widehat{q}_0, \widehat{\delta}, \langle \widehat{F}, \emptyset\rangle\rangle$ extends $\mathcal{A}'_{\widehat{\xi}}$ to trees by simulating it along a single path. Formally, $\widehat{Q} = \{\widehat{q}_0\} \cup (Q \times \{+, -\})$, $\widehat{F} = F_+ \times \{+\}$, where $\{\widehat{q}_0\} \cup (Q \times \{+\})$ is the set of positive states and $Q \times \{-\}$ is the set of negative states. The transition function $\widehat{\delta}$ is defined as follows, where for each $p \in Q$, $b_p$ denotes `true` if $p \in F_-$, and $b_p$ denotes `false` otherwise:

- $\widehat{\delta}((q, +), \widehat{\sigma}) = \bigvee_{p \in \rho(q, \widehat{\sigma}, +)} (\Diamond, (p, +))$;
- $\widehat{\delta}((q, -), \widehat{\sigma}) = \bigvee_{p \in \rho(q, \widehat{\sigma}, -)} (\uparrow, (p, -), b_p)$;
- $\widehat{\delta}(\widehat{q}_0, \widehat{\sigma}) = \bigvee_{q_0 \in Q_0} \bigvee_{q \in \rho(q_0, \widehat{\sigma}, +)} \bigvee_{p \in \rho(q_0, \widehat{\sigma}, -)} [(\Diamond, (q, +)) \wedge (\uparrow, (p, -), b_p)]$.

Note that $\widehat{Q}$ constitutes a single existential component (in particular, $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ is an existential two-way HAA). Intuitively, starting from the initial node $x$ of the input tree, $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ guesses an $\varepsilon$-path $\pi = x_0 x_1 \dots x_k \dots$ such that $x_k = x$ and simulates an infinite run $(r_-, r_+)$ of $\mathcal{A}'_{\widehat{\xi}}$ over the pointed word $(V(x_0)V(x_1)\dots, k)$ by simulating $r_-$ by backward moves along the prefix $x_0 x_1 \dots x_k$ of $\pi$ and by simulating $r_+$ by existential moves along the suffix $x_k x_{k+1} \dots$ of $\pi$. Thus, $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ accepts all the pointed $\widehat{\Sigma}$-labelled trees satisfying $\mathsf{E}\widehat{\xi}$. Now, we define $\mathcal{A}_\varphi$ as follows. Intuitively, $\mathcal{A}_\varphi$ simulates $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ and starts additional copies of the HAA $\mathcal{A}_{\varphi_i}$. According to Claim 1 these copies guarantee that whenever $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ assumes that proposition $p_i$ labels the current node along the guessed path, then formula $\varphi_i$ holds at this node.

Formally, $\mathcal{A}_\varphi = \langle \Sigma, \widehat{Q} \cup \bigcup_{i=1}^n Q^i, \widehat{q}_0, \delta, \langle \widehat{F} \cup \bigcup_{i=1}^n G^i, \bigcup_{i=1}^n B^i\rangle\rangle$, where for states in $\bigcup_{i=1}^n Q^i$, the transition function $\delta$ agrees with the corresponding $\delta^i$. For $q \in \widehat{Q}$ and $\sigma \in \Sigma$, $\delta(q, \sigma) = \bigvee_{\widehat{\sigma} \in \widehat{\Sigma}} \big(\widehat{\delta}(q, \widehat{\sigma}) \wedge \bigwedge_{p_i \in \widehat{\sigma}} \delta^i(q_0^i, \sigma)\big)$.

Each conjunction in $\delta(q, \sigma)$ corresponds to a label $\widehat{\sigma} \in \widehat{\Sigma}$. Some copies of $\mathcal{A}_\varphi$ (those originated from $\widehat{\delta}(q, \widehat{\sigma})$) proceed as $\mathcal{A}_{\mathsf{E}\widehat{\xi}}$ when it reads $\widehat{\sigma}$. The other copies guarantee that for each $p_i \in \widehat{\sigma}$, $\varphi_i$ holds at the current node. The set $\widehat{Q}$ constitutes an existential component, with the ordering $\widehat{Q} > Q'$ for each component $Q'$ of $\mathcal{A}_{\varphi_i}$ $(i = 1, \dots, n)$. Correctness of the construction follows from Claim 1 and the induction hypothesis. Since the SNWA $\mathcal{A}'_{\widehat{\xi}}$ has size $2^{O(|\widehat{\xi}|)}$, $\mathcal{A}_{\varphi_i}$ has size $2^{O(|\varphi_i|)}$ and depth $O(|\varphi_i|)$, and the size of $\widehat{\Sigma}$ is $2^{O(|max(\varphi)|)}$, it holds that $\mathcal{A}_\varphi$ has size $2^{O(|\varphi|)}$ and depth $O(|\varphi|)$.

– $\varphi = \mathsf{A}\xi$. We have that $\mathsf{A}\xi \equiv \neg\mathsf{E}\neg\xi$. Let $\mathsf{E}\xi'$ be the positive normal form of $\mathsf{E}\neg\xi$ (note that $|\mathsf{E}\xi'| = O(|\mathsf{E}\neg\xi|)$). By ind. hyp. we can construct a two-way HAA $\mathcal{A}_{\mathsf{E}\xi'}$ over $\Sigma$ of size $2^{O(|\mathsf{E}\xi'|)} = 2^{O(|\varphi|)}$ and depth $O(|\mathsf{E}\xi'|) = O(|\varphi|)$ such that $\mathcal{L}_p(\mathcal{A}_{\mathsf{E}\xi'})$ is the complement of $\mathcal{L}_p(\varphi)$. By Proposition 2, the dual of $\mathcal{A}_{\mathsf{E}\xi'}$ is a two-way HAA accepting $\mathcal{L}_p(\varphi)$ of size $2^{O(|\varphi|)}$ and depth $O(|\varphi|)$.

Note that if $\psi$ is a ECTL$^*_{lp}$+ formula, then there is no state subformula of $\psi$ of the form $\mathsf{A}\xi$, and the construction given for the cases $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \wedge \varphi_2$, and $\varphi = \mathsf{E}\xi$ ensures that $\mathcal{A}_\psi$ is a strictly existential two-way HAA. □

Now, we can prove the main results of this paper.

**Theorem 6 (Model-checking).** *Model-checking of* CTL$^*_{lp}$+ *can be solved in time polynomial in the size of the structure and doubly exponential in the size of the formula. Moreover, for the existential and universal fragments of* CTL$^*_{lp}$+ *and* CTL$^*_{lp}$*, the problem is* Expspace*-complete and can be solved in space logarithmic in the size of the structure and exponential in the size of the formula.*

*Proof.* The first part follows from Theorems 2 and 5. For the second part, since ACTL$^*_{lp}$+ (resp., ACTL$^*_{lp}$) is the dual of ECTL$^*_{lp}$+ (resp., ECTL$^*_{lp}$) and co-Expspace = Expspace, it suffices to prove the result for ECTL$^*_{lp}$+ and ECTL$^*_{lp}$. The upper bounds follows from Theorems 2 and 5. The lower bound for ECTL$^*_{lp}$, (hence, the lower bound for ECTL$^*_{lp}$+ follows) can be proved by a reduction from the word problem for Expspace-bounded Turing machines. □

**Theorem 7 (Satisfiability).** *Satisfiability of* CTL$^*_{lp}$+ *and* CTL$^*_{lp}$ *is* 2Exptime*-complete. Moreover, for the fragment* ACTL$^*_{lp}$*, the problem is* Pspace*-complete, and for the fragments* ACTL$^*_{lp}$+*,* ECTL$^*_{lp}$*,* ECTL$^*_{lp}$+*, it is* Expspace*-complete.*

*Proof.* 2Exptime-completeness for satisfiability of CTL$^*_{lp}$+ and CTL$^*_{lp}$ directly follows from Theorems 3 and 5 and 2Exptime-hardness of satisfiability of standard CTL* [VS85] (note that CTL* can be trivially linearly translated into CTL$^*_{lp}$). For the logic ACTL$^*_{lp}$, it suffices to observe that a ACTL$^*_{lp}$ formula $\varphi$ is satisfiable iff the PLTL formula $[\varphi]$ is (initially) satisfiable, where $[\varphi]$ is obtained from $\varphi$ by omitting all its (universal) path quantifiers. Thus, satisfiability of ACTL$^*_{lp}$ is linearly reducible to satisfiability of PLTL. Since the converse also holds, and satisfiability of PLTL is PSPACE-complete [Var88], the result for ACTL$^*_{lp}$ follows. For the fragments ACTL$^*_{lp}$+, ECTL$^*_{lp}$, ECTL$^*_{lp}$+, the lower bounds are proved by a reduction from the word problem for Expspace-bounded Turing machines. For the upper bounds, membership in Expspace for ECTL$^*_{lp}$ and ECTL$^*_{lp}$+ follows from Theorems 4 and 5. Finally, it remains to prove membership in Expspace for the universal fragment ACTL$^*_{lp}$+ First, we extend the linear temporal logic PLTL+ by a new unary modality $\mathsf{R}$ (which reads as 'reset'), whose semantics is defined as follows: for a given word $w$, $(w, i, k) \models \mathsf{R}\xi$ iff $(w, i, i) \models \xi$. Intuitively, the modality $\mathsf{R}$ emulates the ability of the path quantifiers of CTL$^*_{lp}$+ to 'reset' the present. We denote by RLTL the extension of PLTL+ with $\mathsf{R}$. Fix an ACTL$^*_{lp}$+ formula $\varphi$ and let $[\varphi]$ be the RLTL formula obtained from $\varphi$ by replacing each occurrence of the path quantifier $\mathsf{A}$ with $\mathsf{R}$. Evidently, $\varphi$ is satisfiable iff $[\varphi]$ is satisfiable. Thus, it suffices to show that satisfiability of RLTL is in Expspace. This is proved by a translation, with a single exponential blow-up, of RLTL into Büchi two-way alternating *word* automata (with $\varepsilon$-moves). By [Var98], these automata can be converted,

with a single exponential blow-up, into parity nondeterministic *word* automata whose nonemptiness problem is in NLOGSPACE. Hence, the result follows. □

## References


[CE81] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

[EH86] Emerson, E.A., Halpern, J.Y.: Sometimes and not never revisited: On branching versus linear time. Journal of the ACM 33(1), 151–178 (1986)

[HT87] Hafer, T., Thomas, W.: Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 269–279. Springer, Heidelberg (1987)

[KP95] Kupferman, O., Pnueli, A.: Once and For All. In: Proc. 10th LICS, pp. 25–35. IEEE Comp. Soc. Press, Los Alamitos (1995)

[KV98] Kupferman, O., Vardi, M.Y.: Weak alternating automata and tree automata emptiness. In: Proc. 30th STOC, pp. 224–233. ACM, New York (1998)

[KV00] Kupferman, O., Vardi, M.Y.: An automata-theoretic approach to modular model checking. ACM Trans. Program. Lang. Syst. 22(1), 87–128 (2000)

[KV06] Kupferman, O., Vardi, M.Y.: Memoryful branching-time logic. In: Proc. 21th LICS, pp. 265–274. IEEE Comp. Soc. Press, Los Alamitos (2006)

[KVW00] Kupferman, O., Vardi, M.Y., Wolper, P.: An Automata-Theoretic Approach to Branching-Time Model Checking. J. ACM 47(2), 312–360 (2000)

[LS95] Laroussinie, F., Schnoebelen, P.: A hierarchy of temporal logics with past. Theoretical Computer Science 148(2), 303–324 (1995)

[LS00] Laroussinie, F., Schnoebelen, P.: Specification in CTL+past for verification in CTL. Information and Computation 156(1–2), 236–263 (2000)

[MS87] Muller, D.E., Schupp, P.E.: Alternating Automata on Infinite Trees. Theoretical Computer Science 54, 267–276 (1987)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–57 (1977)

[PV03] Pistore, M., Vardi, M.Y.: The planning spectrum - one, two, three, infinity. In: Proc. 18th LICS, pp. 234–243. IEEE Comp. Soc. Press, Los Alamitos (2003)

[Var88] Vardi, M.Y.: A temporal fixpoint calculus. In: Proc. 15th Annual POPL, pp. 250–259. ACM, New York (1988)

[Var98] Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)

[VS85] Vardi, M.Y., Stockmeyer, L.: Improved upper and lower bounds for modal logics of programs. In: Proc. 17th STOC, pp. 240–251 (1985)

[VW94] Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)

[Wil99] Wilke, T.: $CTL^+$ is exponentially more succinct than CTL. In: Pandu Rangan, C., Raman, V., Ramanujam, R. (eds.) FST TCS 1999. LNCS, vol. 1738, pp. 110–121. Springer, Heidelberg (1999)

[Zie98] Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. 200(1–2), 135–183 (1998)

# Footprints in Local Reasoning

Mohammad Raza and Philippa Gardner

Department of Computing,
Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
{mraza,pg}@doc.ic.ac.uk

**Abstract.** Local reasoning about programs exploits the natural local behaviour common in programs by focussing on the footprint - that part of the resource accessed by the program. We address the problem of formally characterising and analysing the footprint notion for abstract local functions introduced by Calcagno, O'Hearn and Yang. With our definition, we prove that the footprints are the only essential elements required for a complete specification of a local function. We also show that, for well-founded models (which is usually the case in practice), a smallest specification always exists that only includes the footprints, thus formalising the notion of small axioms in local reasoning. We also present results for the non-well-founded case, and introduce the natural class of one-step local functions for which the footprints are the smallest safe states.

**Keywords:** Footprints, Hoare Logic, Local Reasoning, Separation Logic.

## 1 Introduction

Local reasoning about programs focusses on the collection of resources directly acted upon by the program. It has recently been introduced and used to substantial effect in *local* Hoare reasoning about memory update. Researchers previously used Hoare reasoning based on First-order Logic to specify how programs interacted with the *whole* memory. O'Hearn, Reynolds and Yang instead introduced local Hoare reasoning based on Separation Logic [13,10]. The idea is to reason only about the local parts of the memory—the *footprints*—that are directly accessed by a program. Intuitively, the footprints form the pre-conditions of the *small* axioms, which provide the smallest complete specification of the program. All the true Hoare triples are derivable from the small axioms and the general Hoare rules. In particular, the *frame rule* extends the reasoning to properties about the rest of the heap which has not been changed by the command.

O'Hearn, Reynolds and Yang originally introduced Separation Logic to solve the problem of how to reason about the mutation of data structures in memory. They have applied their reasoning to several memory models, including heaps based on pointer arithmetic [13], heaps with permissions [4], and the combination of heaps with variable stacks which views variables as resource [5,15]. In each case, the basic soundness and completeness results for local Hoare reasoning are essentially the same. For this reason, Calcagno, O'Hearn and Yang [9] recently introduced abstract local functions over abstract resource models (separation algebras), generalising their specific examples of local imperative commands for manipulating memory models. They develop Abstract

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 201–215, 2008.

Separation Logic to provide local Hoare reasoning about such local functions, and give general soundness and completeness results.

We believe that the general concept of a local function is a fundamental step towards establishing the theoretical foundations of local reasoning, and Abstract Separation Logic is an important generalisation of the local Hoare reasoning systems now widely studied in the literature. However, Calcagno, O'Hearn and Yang do not characterise the footprints and small axioms in this general theory, which is a significant omission. O'Hearn, Reynolds and Yang, in one of their first papers on the subject [13], state the local reasoning viewpoint as:

> 'to understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged.'

A complete understanding of the foundations of local Hoare reasoning therefore requires a formal characterisation of the footprint notion. O'Hearn tried to formalise footprints in his work on Separation Logic (personal communication with O'Hearn). His intuition was that the footprints should be the smallest states on which the program is safe, and the *small axioms* arising from these footprints should give rise to a complete specification using the general rules for local Hoare reasoning. However, Yang discovered this notion of footprint does not work, since it does not always yield complete specifications. In this paper, we resolve this problem, providing a definition of footprint which does give rise to complete specifications.

Consider the local program[1]

$$AD ::= \quad x := new(); dispose(x)$$

This *allocate-deallocate* program allocates a new cell, stores its address value in the stack variable $x$, and then deallocates the cell. It is local because all its atomic constituents are local. This tiny example captures the essence of a common type of program; there are many programs which, for example, create a list, work on the list, and then destroy the list.

The smallest heap on which the *AD* program is safe is the empty heap $emp$. The specification using this pre-condition is:

$$\{emp\} \quad AD \quad \{emp\} \tag{1}$$

We can extend our reasoning to larger heaps by applying the frame rule: for example, extending to a one-cell heap with arbitrary address $l$ and value $v$ gives

$$\{l \mapsto v\} \quad AD \quad \{l \mapsto v\} \tag{2}$$

However, axiom (1) does not give the complete specification of the *AD* program. In fact, it captures very little of the spirit of allocation followed by de-allocation. For example, the following triple is also true:

$$\{l \mapsto v\} \quad AD \quad \{l \to v \wedge x \neq l\} \tag{3}$$

[1] Yang's example was the 'allocate-deallocate-test' program *ADT* ::= 'x := new();dispose(x); if (x=1) then z:=0 else z:=1;x=0'. Our *AD* program provides a more standard example of program behaviour.

This triple (3) is true because, if $l$ is already allocated, then the new address cannot be $l$ and hence $x$ cannot be $l$. It cannot be derived from (1). However, the combination of axiom (1) and axiom (3) for arbitrary one-cell heaps does provide the smallest complete specification. This example illustrates that O'Hearn's intuitive view of the footprints as the minimal safe states just does not work for common imperative programs.

In this paper, we introduce the definition of the footprint of a local function. For our *AD* example, our definition identifies $emp$ and the arbitrary one-cell heaps $l \mapsto v$ as footprints, as expected. We prove the general result that, for any local function, the footprints are the only elements which are *essential* to specify completely the behaviour of this function. For well-founded resource, which is almost always the case in practice, we show that the footprints are also always *sufficient*: that is, a complete specification always exists that only uses the footprints. We also explore results for non-well-founded resource: for models without negativity (no inverse elements except the identity), such as heaps with permissions, either the footprints are sufficient (such as for the *write* command in the permissions model) or there is no smallest complete specification (such as for the *read* command in the permissions model); for models with negativity, we show that there can exist smallest complete specifications based on elements which are not essential and hence not footprints. Finally, we identify a natural class of local functions, which we call *one-step* local functions, which satisfy O'Hearn's original intuition regarding footprints. For well-founded resource, the one-step local functions are precisely the local functions whose footprints are the smallest safe states.

## 2 Background

This is a background section that describes the separation algebras, local functions and Hoare reasoning introduced in [9]. Further details and motivation can be found in [9].

**Definition 1 (Separation Algebra).** *A* **separation algebra** *is a cancellative, partial commutative monoid* $(\Sigma, \bullet, u)$. *$\Sigma$ is a set, and $\bullet$ is a partial binary operator with unit $u$ which satisfies the familiar axioms of associativity, commutativity and unit, using a partial equality on $\Sigma$ where either both sides are defined and equal, or both are undefined. It also satisfies the cancellative property stating that, for each $\sigma \in \Sigma$, the partial function $\sigma \bullet (\cdot) : \Sigma \mapsto \Sigma$ is injective.* **Separateness** *($\#$) and* **substate** *($\preceq$) relations are given by $\sigma_0 \# \sigma_1$ iff $\sigma_0 \bullet \sigma_1$ is defined and $\sigma_0 \preceq \sigma_2$ iff $\exists \sigma_1 . \sigma_2 = \sigma_0 \bullet \sigma_1$.*

Examples of separation algebras include multisets under union (with unit $\emptyset$), the natural numbers with addition (with unit 0), heaps as finite partial functions from locations to values ([9] and example 1), heaps with permissions [9,4], and the combination of heaps and variable stacks enabling us to model programs with variables as local functions ([9], [15] and example 1). These example all have an intuition of resource, with $\sigma_1 \bullet \sigma_2$ intuitively giving more resource than just $\sigma_1$ and $\sigma_2$ for $\sigma_1, \sigma_2 \neq u$. However, there are also examples that do not conform to this resource intuition, such as $\{a, u\}$ with $a \bullet a = u$. We shall overload notation, using $\Sigma$ to denote $(\Sigma, \bullet, u)$.

**Lemma 1 (Subtraction).** *For $\sigma_1, \sigma_2 \in \Sigma$, if $\sigma_1 \preceq \sigma_2$ then there exists a unique element $\sigma_2 - \sigma_1 \in \Sigma$ such that $(\sigma_2 - \sigma_1) \bullet \sigma_1 = \sigma_2$.*

Following [9], we model commands on separation algebras as functions of the form $f : \Sigma \to P(\Sigma)^\top$, where $\top$ is an extra top element added to the powerset. The range $P(\Sigma)^\top$ is used to model non-determinism and faulting: elements can map either to a set of elements (to allow for non-determinism) or $\top$ (which represents faulting and returning an error). Mapping to the empty set represents divergence (non-termination).

**Definition 2.** *Define the set $P(\Sigma)^\top = P(\Sigma) \cup \{\top\}$ with the standard subset relation extended with a new greatest element $\top$: that is, $p \sqsubseteq \top$ for all $p \in P(\Sigma)^\top$. It is a total commutative monoid with $\{u\}$ as the unit and a binary operator $*$ given by:*

$$\begin{aligned} p * q &= \{\sigma_0 \bullet \sigma_1 \mid \sigma_0 \# \sigma_1 \wedge \sigma_0 \in p \wedge \sigma_1 \in q\} \quad \textit{if } p, q \in P(\Sigma) \\ &= \top \quad \textit{otherwise} \end{aligned}$$

*For functions $f : \Sigma \to P(\Sigma)^\top$, $f \sqsubseteq g$ iff $f(\sigma) \sqsubseteq g(\sigma)$ for all $\sigma \in \Sigma$.*

Intuitively, we think of a command acting on resource to be *local* if whenever the command executes safely on any resource element, then any more resource that may be added will not be affected by the command. This intuition was first formalised in [19] (for the RAM model) with the following constraints:

- *Safety monotonicity*: if the command is safe on some resource element, then it does not fault when more resource is added.
- *Frame property*: if the command is safe on some resource element, then any additional resource will remain unchanged after execution if the execution terminates.

In the abstract setting of [9] which we use in this paper, these two restrictions were amalgamated into the following succinct definition of a local function.

**Definition 3 (Local Function).** *A* **local function on** $\Sigma$ *is a total function $f : \Sigma \to P(\Sigma)^\top$ which satisfies the* **locality condition***:*

$$\sigma \# \sigma' \;\; \textit{implies} \;\; f(\sigma' \bullet \sigma) \sqsubseteq \{\sigma'\} * f(\sigma)$$

$f$ faults on $\sigma$ when it is not sufficient resource for safe execution ($f(\sigma) = \top$). Adding more resource may make the execution safe ($f(\sigma' \bullet \sigma) \sqsubseteq f(\sigma) = \top$). Safety monotonicity follows since if $f$ is safe on $\sigma$ ($f(\sigma) \sqsubset \top$) then $f(\sigma' \bullet \sigma)$ is also safe ($f(\sigma' \bullet \sigma) \sqsubseteq \{\sigma'\} * f(\sigma) \sqsubset \top$). The frame property follows by the fact that $\sigma'$ is preserved by $f$.

*Example 1 (Local functions on separation algebras).*

1. **Heap dispose command.** We model heaps by the separation algebra $(H, \bullet, u_H)$, where $H = L \rightharpoonup_{fin} Val$ are finite partial functions from a set of locations to a set of values, the partial operator $\bullet$ is the union of partial functions with disjoint domains, and the unit $u_H$ is the empty function. For $h \in H$, let $dom(h)$ be the domain of $h$. We write $l \mapsto v$ for the partial function with domain $\{l\}$ that maps $l$ to $v$. For $h_1, h_2 \in H$, if $h_2 \preceq h_1$ then $h_1 - h_2 = h_1 \mid_{dom(h_1) - dom(h_2)}$, and is undefined otherwise. For $h \in H$, the $dispose[l]$ command that deletes the cell at location $l$ in $h$ is given by

$$dispose[l](h) = \begin{cases} \{h - (l \mapsto v)\} & h \succeq (l \mapsto v) \text{ for some } v \\ \top & \text{otherwise} \end{cases}$$

The function is local: if $h \not\succeq (l \mapsto v)$ then $dispose[l](h) = \top$, and $dispose[l](h' \bullet h) \sqsubseteq \top$. Otherwise, $dispose[l](h' \bullet h) = \{(h' \bullet h) - (l \mapsto v)\} \sqsubseteq \{h'\} * \{h - (l \mapsto v)\} = \{h'\} * dispose[l](h)$.

2. ***AD* command.** The *AD* command $x := new(); dispose(x)$ described in the introduction can be modelled as a local function on a separation algebra that includes the heap and the variable stack. We use the algebra $H \times S$ with $H$ as before and $S = Var \rightharpoonup_{fin} Val$. The $\bullet$ operator in this case combines states with disjoint heap and stack domains, and is undefined otherwise. The unit is $(u_H, u_S)$, where $u_S$ is the empty stack. Although this approach is limited to disjoint reference to stack variables, this constraint can be lifted by enriching the separation algebra with *permissions* [4]. However, this added complexity can be avoided for the discussion in this paper. For a state $h \in H \times S$, let $loc(h)$ denote the set of heap locations in $h$.

$$AD(h) = \begin{cases} \{h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h')\} & h = h' \bullet (u_H, x \mapsto v) \text{ for some } v \\ \top & \text{otherwise} \end{cases}$$

The function is local: if $h \neq h' \bullet (u_H, x \mapsto v)$ for some $h'$ and $v$, then $AD(h) = \top$, and for any $h''$, $AD(h'' \bullet h) \sqsubseteq \top$. Otherwise, $h = h' \bullet (u_H, x \mapsto v)$ for some $h'$ and $v$. Then for any $h''$, $AD(h'' \bullet h) = \{h'' \bullet h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h'' \bullet h')\} \sqsubseteq \{h''\} * \{h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h')\} = \{h''\} * AD(h)$.

3. **Operations on Integers.** We consider the separation algebra of integers under addition with identity 0. It can be seen that any 'adding' function $f(x) = \{x + c\}$ that adds a constant $c$ is local, while a function that multiplies by a constant $c$, $f(x) = \{cx\}$, is non-local.

We now present the local Hoare reasoning framework for local functions on separation algebras. We treat predicates simply as subsets of the separation algebra.

**Definition 4.** *A* **predicate** $p$ *over* $\Sigma$ *is an element of the powerset* $P(\Sigma)$.

Note that the top element $\top$ is not a predicate and that the $*$ operator, although defined on $P(\Sigma)^\top \times P(\Sigma)^\top \to P(\Sigma)^\top$, acts as a binary connective on predicates. We have the distributive law for union: $(\bigsqcup X) * p = \bigsqcup\{x * p \mid x \in X\}$ where $X \subseteq P(\Sigma)$. The same is not true for intersection in general, but does hold for *precise* predicates.

**Definition 5 (Precise Predicates).** *A predicate* $p \in P(\Sigma)$ *is* **precise** *iff, for every* $\sigma \in \Sigma$*, there exists at most one* $\sigma_p \in p$ *such that* $\sigma_p \preceq \sigma$.

$\{l \mapsto v \mid v \in Val\}$ for some $l$ is precise, while $\{l \mapsto v \mid l \in Loc\}$ for some $v$ is not. Also, any singleton predicate $\{\sigma\}$ is precise.

**Lemma 2 (Precision Characterization).** *A predicate* $p$ *is precise iff, for all* $X \subseteq P(\Sigma)$, $(\bigsqcap X) * p = \bigsqcap\{x * p \mid x \in X\}$.

In the introduction we discussed the intuition of how the footprints are expected to correspond to the *smallest* safe states. We will return to this point in section 6, employing the notion of a *saturated* predicate, which is one that is always falsified on bigger states. Saturated can also be called 'anti-intuitionistic', because an intuitionistic predicate is one that is never falsified on bigger states. Every precise predicate is also saturated.

**Definition 6 (Saturated predicate).** *A predicate $p \in P(\Sigma)$ is* **saturated** *if for every $\sigma \in p$, there is no $\sigma'$ in $p$ such that $\sigma \prec \sigma'$.*

Our Hoare reasoning system is a slight adaptation of Abstract Separation Logic [9], the difference being that we emphasise the notion of a specification as a tuple of pre- and post- conditions, rather than the usual Hoare triples that include the function. A triple is then equivalent to saying that a function $f$ *satisfies* a tuple $(p, q)$, written $f \models (p, q)$. This approach is very similar to the notion of the *specification statement* (a Hoare triple with a 'hole') introduced in [11], which is used in refinement calculi, and was also used to prove completeness of a local reasoning system in [19].

**Definition 7 (Specification).** *Let $\Sigma$ be a separation algebra. A* **statement** *on $\Sigma$ is a tuple $(p, q)$, where $p, q \in P(\Sigma)$ are predicates representing pre- and post- conditions. A* **specification** *$\phi$ on $\Sigma$ is a set of statements. We let $\Phi_\Sigma = P(P(\Sigma) \times P(\Sigma))$ be the set of all specifications on $\Sigma$. We shall exclude the subscript when it is clear from the context. The* **domain** *of a specification is defined as $D(\phi) = \bigsqcup\{p \mid (p, q) \in \phi\}$.* **Domain equivalence** *is defined as $\phi \cong_D \psi$ iff $D(\phi) = D(\psi)$.*

Thus the domain is the union of the preconditions of all the statements in the specification. It is one possible measure of *size*: how much of $\Sigma$ the specification is referring to. We also adapt the notions of precise and saturated predicates to specifications.

**Definition 8.** *A specification is saturated iff its domain is saturated. It is precise iff its domain is precise.*

**Definition 9 (Satisfaction).** *A local function $f$ satisfies a statement $(p, q)$, written $f \models (p, q)$, iff, for all $\sigma \in p$, $f(\sigma) \sqsubseteq q$. $f$ satisfies a specification $\phi \in \Phi$, written $f \models \phi$, iff $f \models (p, q)$ for all $(p, q) \in \phi$.*

**Definition 10 (Semantic consequence).** *Let $p, q, r, s \in P(\Sigma)$ and $\phi, \psi \in \Phi$. Each judgement $(p, q) \models (r, s), \phi \models (p, q)$, $(p, q) \models \phi$, and $\phi \models \psi$ holds iff all local functions that satisfy the left hand side also satisfy the right hand side.*

**Proposition 1 (Order Characterization).** *$f \sqsubseteq g$ iff, for all $p, q \in P(\Sigma)$, $g \models (p, q)$ implies $f \models (p, q)$.*

For every specification $\phi$, there is a 'best' local function satisfying $\phi$ (lemma 3). This is of the same nature as the best local action of [9], except that we generalise to specifications rather than statements (single pre- and post-condition pairs).

**Definition 11 (Best local action).** *For a specification $\phi \in \Phi$, the* **best local action** *of $\phi$, written $bla[\phi]$, is the function of type $\Sigma \to P(\Sigma)^\top$ defined by*

$$bla[\phi](\sigma) = \bigsqcap_{\sigma' \preceq \sigma, \sigma' \in p, (p,q) \in \phi} \{\sigma - \sigma'\} * q$$

**Lemma 3.** *Let $\phi \in \Phi$. The following hold:*

- *$bla[\phi]$ is local*
- *$bla[\phi] \models \phi$*
- *if $f$ is local and $f \models \phi$ then $f \sqsubseteq bla[\phi]$.*

$$\frac{(p,q)}{(p*r,q*r)}\ Frame \qquad \frac{p' \sqsubseteq p \quad (p,q) \quad q \sqsubseteq q'}{(p',q')}\ Consequence$$

$$\frac{(p_i,q_i),\ \text{all } i \in I}{(\bigsqcup_{i\in I} p_i, \bigsqcup_{i\in I} q_i)}\ Union \qquad \frac{(p_i,q_i),\ \text{all } i \in I, I \neq \emptyset}{(\bigsqcap_{i\in I} p_i, \bigsqcap_{i\in I} q_i)}\ Intersection$$

**Fig. 1.** Inference Rules for local Hoare reasoning

**Lemma 4.** *For $\phi \in \Phi$ and $p, q \in P(\Sigma)$, $bla[\phi] \models (p,q) \Leftrightarrow \phi \models (p,q)$.*

The inference rules of the proof system are given in figure 1. The system is sound and complete with respect to the satisfaction relation; the proof uses lemmas 3 and 4.

**Definition 12 (Proof-theoretic consequence).** *For statements $p, q, r, s$ and specifications $\phi, \psi$, each of the judgements $(p,q) \vdash (r,s)$, $\phi \vdash (p,q)$, $(p,q) \vdash \phi$, and $\phi \vdash \psi$ holds iff the right-hand side is derivable from the left-hand side by the rules in figure 1.*

**Theorem 1 (Soundness and Completeness).** $\phi \models (p,q) \Leftrightarrow \phi \vdash (p,q)$.

## 3 Properties of Specifications

We discuss certain properties of specifications as a prerequisite for our main discussion on footprints in Section 4. We introduce the notion of a *complete* specification for a local function: a specification from which all properties that hold for the function can be derived in the proof system. However, a function may have many complete specifications, so we introduce a canonical form for specifications. We show that of all the complete specifications of a local function, there exists a unique canonical complete specification for every domain. As discussed in the introduction, an important notion of local reasoning is the *small specification* which completely describes the behaviour of a local function by mentioning only the footprint. Thus, as a prerequisite to investigating their existence, we formalise small specifications as complete specifications with the smallest possible domain. Similarly, we define *big* specifications as complete specifications with the biggest domain.

**Definition 13 (Complete Specification).** *A specification $\phi \in \Phi$ is a* **complete specification** *for $f$, written $complete(\phi, f)$, iff, for all $p, q \in P(\Sigma)$, $f \models (p,q) \Leftrightarrow \phi \models (p,q)$. Let $\Phi_{comp(f)}$ be the set of all complete specifications of $f$.*

$\phi$ is complete for $f$ whenever the tuples that hold for $f$ are *exactly* the tuples that follow from $\phi$. This also means that any two complete specfications $\phi$ and $\psi$ for a local function are semantically equivalent, that is, $\phi \dashv\models \psi$. The following proposition illustrates how the notions of best local action and complete specification are closely related.

**Proposition 2.** *For all $\phi \in \Phi$ and local functions $f$, $complete(\phi, f) \Leftrightarrow f = bla[\phi]$.*

Any specification is therefore only complete for a unique local function, which is its best local action. However, a local function may have lots of complete specifications. We therefore introduce a canonical form for specifications.

**Definition 14 (Canonicalisation).** *The* **canonicalisation** *of a specification $\phi$ is defined as $\phi_{can} = \{(\{\sigma\}, bla[\phi](\sigma)) \mid \sigma \in D(\phi)\}$. A specification is in* **canonical** *form if it is equal to its canonicalisation. Let $\Phi_{can(f)}$ denote the set of all canonical complete specifications of $f$.*

**Proposition 3.** *For any specification $\phi$, we have $\phi \dashv\models \phi_{can}$.*

Thus, the canonicalisation of a specification is logically equivalent to the specification. The following corollary shows that all complete specifications that have the same domain have a unique canonical form, and specifications of different domains have different canonical forms.

**Corollary 1.** *$\Phi_{can(f)}$ is isomorphic to the quotient set $\Phi_{comp(f)}/\ \cong_D$, under the equality-preserving isomorphism that maps $[\phi]_{\cong_D}$ to $\phi_{can}$.*

**Definition 15 (Small and Big specifications).** *$\phi$ is a* **small specification** *for $f$ iff $\phi \in \Phi_{comp(f)}$ and there is no $\psi \in \Phi_{comp(f)}$ such that $D(\psi) \sqsubset D(\phi)$. A* **big specification** *is defined similarly.*

*Small* and *big* specifications are thus the specifications with the smallest and biggest domains respectively. The question is if/when small and big specifications exist. The following result shows that a canonical big specification exists for every local function.

**Proposition 4 (Big Specification).** *For any local function $f$, the canonical big specification for $f$ is given by $\phi_{big(f)} = \{(\{\sigma\}, f(\sigma)) \mid f(\sigma) \sqsubset \top\}$.*

Small specifications are used in local reasoning to completely specify the behaviour of an update command by only mentioning the behaviour of the command on the part of the resource that is affected by the command [13,4,7]. The question of the existence of small specifications is therefore strongly related to the concept of footprints. Finding a small specification is about finding the complete specification with the smallest possible domain, and therefore enquiring about which elements of $\Sigma$ are essential and sufficient for a complete specification. This requires a formal characterisation of the footprint notion, which we shall now present.

## 4 Footprints

In the introduction we discussed how the *AD* program demonstrates that the footprints of a local function do not correspond simply to the smallest safe states, as these states alone do not always yield complete specifications. In this section we introduce the definition of footprint that does yield complete specifications. In order to understand what the footprint of a local function should be, we begin by analysing the definition of locality. Recall that the locality definition 3 says that the action on a certain state $\sigma_1$ imposes a *limit* on the action on a bigger state $\sigma_2 \bullet \sigma_1$. This limit is $\{\sigma_2\} * f(\sigma_1)$, that is, we have $f(\sigma_2 \bullet \sigma_1) \sqsubseteq \{\sigma_2\} * f(\sigma_1)$. A reformulation of this definition is that for any state $\sigma$, the action on that state has to be within the limit imposed by *every* state $\sigma'$ that is smaller than it, and we therefore have

$$f(\sigma) \sqsubseteq \bigsqcap_{\sigma' \prec \sigma} \{\sigma - \sigma'\} * f(\sigma')$$

We define this overall constraint imposed on $\sigma$ by all the smaller states as the *local limit* of $f$ on $\sigma$, and show that the locality of a function is equivalent to it satisfying the local limit constraint.

**Definition 16 (Local limit).** *For a local function $f$ on $\Sigma$, and $\sigma \in \Sigma$, the* **local limit** *of $f$ on $\sigma$ is defined as*

$$L_f(\sigma) = \bigsqcap_{\sigma' \prec \sigma} \{\sigma - \sigma'\} * f(\sigma')$$

**Proposition 5.** *$f$ is local* $\Leftrightarrow$ $f(\sigma) \sqsubseteq L_f(\sigma)$ *for all $\sigma \in \Sigma$*

With this intuition that the locality of $f$ is determined by the local limit restricting the action of $f$, we define the footprints as those elements which further reduce this limit: $\sigma$ is a footprint of $f$ if and only if $f(\sigma) \sqsubset L_f(\sigma)$. If the result of the function is equal to the local limit on a certain element, then this can be determined just by the fact that $f$ is a local function. If the result is strictly smaller, then this behaviour is additional to $f$ being local. The intuition is that this characteristic property of the function would have to be explicitly stated in a complete specification of the function, which would make footprints the essential elements required to describe this function's behaviour. We formally prove this central result after stating the definition and illustrating it with examples.

**Definition 17 (Footprint).** *For a local function $f$ and $\sigma \in \Sigma$, $\sigma$ is a footprint of $f$, written $F_f(\sigma)$, iff $f(\sigma) \sqsubset L_f(\sigma)$. We denote the set of footprints of $f$ by $F(f)$.*

Note that an element $\sigma$ is therefore not a footprint iff the action of $f$ on $\sigma$ is at the local limit, that is $f(\sigma) = L_f(\sigma)$. With this definition of footprint, the smallest elements on which $f$ is safe are always footprints. This is because $f$ faults on everything smaller, and thus the local limit is $\top$. However, these elements are not always the only footprints. An example is the *AD* command discussed in the introduction. The empty heap is a footprint as it is the smallest safe heap, but the heap cell $l \mapsto v$ is also a footprint.

*Example 2 (Dispose command).* The footprints of the $dispose[l]$ command are the cells at location $l$. We check this by considering the following cases

1. The empty heap, $u_H$, is not a footprint since $L_{dispose[l]}(u_H) = \top = dispose[l](u_H)$
2. Every cell $l \mapsto v$ for some $v$ is a footprint

$$L_{dispose[l]}(l \mapsto v) = \{l \mapsto v\} * dispose[l](u_H) = \{l \mapsto v\} * \top = \top$$
$$dispose[l](l \mapsto v) = \{u_H\} \sqsubset L_{dispose[l]}(l \mapsto v)$$

3. Every state $\sigma$ such that $\sigma \succ (l \mapsto v)$ for some $v$ is not a footprint

$$L_{dispose[l]}(\sigma) \sqsubseteq \{\sigma - (l \mapsto v)\} * dispose[l](l \mapsto v) = \{\sigma - (l \mapsto v)\} = dispose[l](\sigma)$$

By proposition 5, we have $L_{dispose[l]}(\sigma) = dispose[l](\sigma)$. The intuition is that $\sigma$ does not characterise any 'new' behaviour of the function: its action on $\sigma$ is just a consequence of its action on the cells at location $l$ and the locality property of the function.

4. Every state $\sigma$ such that $\sigma \not> (l \mapsto v)$ for some $v$ is not a footprint

$$L_{dispose[l]}(\sigma) \sqsubseteq \{\sigma\} * dispose[l](u_H) = \{\sigma\} * \top = \top = dispose[l](\sigma)$$

Again by proposition 5, $L_{dispose[l]}(\sigma) = dispose[l](\sigma)$.

*Example 3 (*AD *command).* The *AD* (Allocate-Deallocate) command was defined in example 1. We have the following cases for $\sigma$.

1. $\sigma \not> (u_H, x \mapsto v_1)$ for some $v_1$ is not a footprint, since $L_{AD}(\sigma) = \top = AD(\sigma)$.
2. $\sigma = (u_H, x \mapsto v_1)$ for some $v_1$ is a footprint since $L_{AD}(\sigma) = \top$ (by case (1)) and $AD(\sigma) = \{(u_H, x \mapsto w) \mid w \in L\} \sqsubset L_{AD}(\sigma)$.
3. $\sigma = (l \mapsto v_1, x \mapsto v_2)$ for some $l, v_1, v_2$ is a footprint.

$$\begin{aligned} L_{AD}(\sigma) &= \{(l \mapsto v_1, u_S)\} * AD((u_H, x \mapsto v_2)) \\ &\quad \text{(AD faults on all other elements strictly smaller than } \sigma) \\ &= \{(l \mapsto v_1, u_S)\} * \{(u_H, x \mapsto w) \mid w \in Loc\} \\ &= \{(l \mapsto v_1, x \mapsto w) \mid w \in Loc\} \end{aligned}$$

$$AD(\sigma) \;= \{(l \mapsto v_1, x \mapsto w) \mid w \in Loc, w \neq l\} \sqsubset L_{AD}(\sigma)$$

4. $\sigma = (h, x \mapsto v_1)$ for some $v_1$, and where $|loc(h)| > 1$, is not a footprint.

$$L_{AD}(\sigma) \sqsubseteq \prod_{(l \mapsto v) \prec h} \{(h - (l \mapsto v), u_S)\} * AD((l \mapsto v, x \mapsto v_1))$$

$$= \{(h, x \mapsto w) \mid w \notin loc(h)\} = AD(\sigma)$$

By proposition 5 we get $L_{AD}(\sigma) = AD(\sigma)$.

5. $\sigma = (h, s \bullet (x \mapsto v_1))$ for some $v_1$ and $s \neq u_S$, is not a footprint.

$$L_{AD}(\sigma) \sqsubseteq \{(u_H, s)\} * AD((h, x \mapsto v_1)) = AD(\sigma)$$

By proposition 5 we get $L_{AD}(\sigma) = AD(\sigma)$.

Our footprint definition therefore works properly for these specific examples. Now we give the formal general result which captures the underlying intuition of local reasoning that the footprints of a local function are the only essential elements for a complete specification of the function.

**Theorem 2 (Essentiality).** *The footprints of a local function are the essential domain elements for any complete specification of that function, that is,*

$$F_f(\sigma) \quad \Leftrightarrow \quad \forall \phi \in \Phi_{comp(f)}.\, \sigma \in D(\phi).$$

## 5 Sufficiency and Small Specifications

We know that the footprints are the only elements that are *essential* for a complete specification of a local function in the sense that every complete specification must include

them. Now we ask when a set of elements is *sufficient* for a complete specification of a local function, in the sense that there exists a complete specification of the function that only includes these elements. In particular, we wish to know if the footprints alone are sufficient.

**Definition 18 (Local limit imposed by a set).** *For a subset $A$ of $\Sigma$, the* **local limit** *imposed by $A$ on the action of $f$ on $\sigma$ is defined by*

$$L_{A,f}(\sigma) = \bigsqcap_{\sigma' \preceq \sigma, \sigma' \in A} \{\sigma - \sigma'\} * f(\sigma')$$

Sometimes, the local limit imposed by $A$ is enough to determine the complete behaviour of $f$. In this case, we call $A$ a *basis* for $f$.

**Definition 19 (Basis).** *$A \sqsubseteq \Sigma$ is a* **basis** *for $f$, written $basis(A, f)$, iff $L_{A,f} = f$.*

This means that, when given the action of $f$ on elements in A alone, we can determine the action of $f$ on any element in $\Sigma$ by just using the locality property of $f$. Every local function has at least one basis, namely the trivial basis $\Sigma$ itself. We next show the correspondence between the bases and complete specifications of a local function.

**Lemma 5.** *Let $\phi_{A,f} = \{(\{\sigma\}, f(\sigma)\} \mid \sigma \in A, f(\sigma) \sqsubset \top\}$. Then we have $basis(A, f) \Leftrightarrow complete(\phi_{A,f}, f)$.*

For every canonical complete specification $\phi \in \Phi_{can(f)}$, we have $\phi = \phi_{D(\phi),f}$. By the previous lemma it follows that $D(\phi)$ forms a basis for $f$. The lemma therefore shows that every basis determines a complete canonical specification, and vice versa. This correspondence also carries over to all complete specifications for $f$ by the fact that every domain-equivalent class of complete specifications for $f$ is represented by the canonical complete specification with that domain (corollary 1). It is a simple consequence of the Essentiality theorem (2) that the footprints are present in every basis for a local function.

**Lemma 6.** *The footprints of $f$ are included in every basis of $f$.*

The question of sufficiency is about how small the basis can get. Given a local function, we wish to know if it has a smallest basis. We know that every basis contains the footprints, but we would also like to know whether the footprints alone form a basis. This would mean that the function would have a complete specification that just mentions the footprints, so it would be a *smallest* complete specification. We find that for well-founded resource models, this is indeed the case.

**Theorem 3 (Sufficiency I).** *If a separation algebra $\Sigma$ is well-founded under the $\preceq$ relation, then the footprints of any local function form a basis for it, that is, $f = L_{F(f),f}$.*

In section 3, the notions of big and small specifications were introduced, and the existence of a big specification was shown. We are now in a position to show the existence of the small specification for well-founded resource. If $\Sigma$ is well-founded, then every local function has a small specification whose domain is the footprints of the function.

**Corollary 2 (Small specification).** *For well-founded separation algebras, every local function has a small specification given by $\phi_{F(f),f}$.*

Thus for well-founded resource, the footprints are always essential and sufficient, and specifications need not consider any other elements. In practice, small specifications may not always be in canonical form even though they always have the same domain as the canonical form. For example, the heap dispose command can have the specification $\{(\{l \mapsto v \mid v \in Val\}, \{u_H\})\}$ rather than $\{(\{l \mapsto v\}, \{u_H\}) \mid v \in Val\}$. Well-founded resource is usually the case in practice. One notable exception is the fractional permissions model [4] in which the resource includes permissions that can be indefinitely divided. In analysing the non-well-founded case, we found it important to distinguish between models that do or do not have *negativity*.

**Definition 20 (Negativity).** *A separation algebra $\Sigma$ has* **negativity** *iff there exists a non-unit element $\sigma \in \Sigma$ that has an inverse, that is, $\sigma \neq u$ and $\sigma \bullet \sigma' = u$ for some $\sigma' \in \Sigma$. We say that $\Sigma$ is* **non-negative** *if no such element exists.*

If a model has negativity then it is non-well-founded, because for elements $\sigma$ and $\sigma'$ such that $\sigma \bullet \sigma' = u$, the set $\{\sigma, u\}$ forms an infinite descending chain (there is no least element). All well-founded models are therefore non-negative. In the general non-negative case, we find that either the footprints form a basis, or there is no smallest basis.

**Theorem 4 (Sufficiceny II).** *If $\Sigma$ is non-negative then, for any local $f$, either the footprints form a smallest basis or there is no smallest basis for $f$.*

**Corollary 3 (Small Specification).** *If $\Sigma$ is non-negative, then every local function either has a small specification given by $\phi_{F(f),f}$ or there is no smallest complete specification for that function.*

*Example 4 (Permissions).* The fractional permissions model [4] is non-well-founded and non-negative. It can be represented by the separation algebra $HPerm = L \rightharpoonup_{fin} Val \times P$ where $L$ and $Val$ are as in example 1, and $P$ is the interval (0, 1] of rational numbers. Elements of $P$ represent 'permissions' to access a heap cell. A permission of 1 for a cell means both read and write access, while any permission less than 1 is read-only access. $\bullet$ joins disjoint heaps and adds the permissions together for any cells that are present in both heaps only if the resulting permission for each heap cell does not exceed 1, and the operation is undefined otherwise. In this case, the write function that updates the value at a location requires a permission of at least 1 and faults on any smaller permission. It therefore has a small specification with precondition being the cell with permission 1. The read function, however, can execute safely on any positive permission, no matter how small. Thus this function can be completely specified with a specification that has a precondition given by the cell with permission $z$, for all $0 < z \leq 1$. However, this is not a *smallest* specification, as a smaller one can be given by further restricting $0 < z \leq 0.5$. We can therefore always find a smaller specification by reducing the value of $z$ but keeping it positive.

For resource with negativity, we find that it is possible to have small specifications that include non-essential elements (which by theorem 2 are not footprints). These elements are non-essential in the sense that complete specifications exist that do not include them, but there is no complete specification that includes only essential elements.

*Example 5 (Integers).* An example of a model with negativity is the separation algebra of integers $(\mathbb{Z}, +, 0)$. In this case there can be local functions which can have small specifications that contain non-footprints. Let $f : \mathbb{Z} \to P(\mathbb{Z})^\top$ be defined as $f(n) = \{n + c\}$ for some constant $c$, as in example 1. $f$ is local, but it has no footprints. This is because for any $n$, $f(n) = 1 + f(n-1)$, and so $n$ is not a footprint of $f$. However, $f$ does have small specifications, for example, $\{(\{0\}, \{c\})\}$, $\{(\{5\}, \{5 + c\})\}$, or indeed $\{(\{n\}, \{n + c\})\}$ for any $n \in \mathbb{Z}$. So although every element is non-essential, some element is required to give a complete specification.

## 6 One-Step Local Functions

In the introduction, we described a common, but mistaken, intuition that the footprint should correspond to the minimal resource on which the function is safe, and that the behaviour of the function on larger states should be derivable from these minimal states. The intuition fails due to the subtle nature of the locality condition as shown by the *AD* example. However, based on our investigation of footprints, we are now in a position to determine a natural class of local functions for which this basic intuition indeed holds. We call these the *one-step* local functions.

**Definition 21 (One-step local function).** *A local function is* **one-step** *if it has a saturated basis. It is* **precise** *if it has a precise basis.*

Note that every precise local function is one-step, because every precise predicate is saturated. For any local function, the footprints are the 'stepping points' in the sense that if we start from the bottom element of $\Sigma$ and go up any ascending chain, the footprints are the points which *add* to the locality constraint by restricting the action on elements above them. For one-step local functions, for any such ascending chain, there is at most a *single* such point along any chain (which is actually the point at which the function becomes safe as shown by proposition 6), and beyond that point, the action of the function is just given by the local limit. So there is at most a single footprint on any ascending chain. Hence the name 'one-step' in analogy with the *unit (or heaviside)* step functions that act as 'on/off switches'. Along any ascending chain in the partial order of $\Sigma$, there is a single point at which the function 'turns on', that is, becomes safe.

Precise local functions are one-step functions with the added property that those ascending chains never intersect, that is, any two footprints would never have a common ancestor. So for example, the dispose function is one-step as its behaviour changes at a single point, which is the cell being deleted. It is also precise because there is no heap that contains two cells with location $l$ pointing to different values. For the *AD* function, there are two stepping points ($u_H$ and the single heap cells) along an ascending chain, so it is not a one-step function.

*Example 6 (One-step local functions)*

1. Dispose. The heap command $dispose[l]$ is a precise local function. This is because the heap algebra is well-founded, so the footprints form a basis and, as shown in section 4, the set of footprints is $\{l \mapsto v \mid v \in Val\}$ which is a precise predicate.

2. *AD*. We showed in section 4 that the set of footprints is $\{(u_H, x \mapsto u) \mid u \in Val\} \cup \{(l \mapsto u, x \mapsto v) \mid u, v \in Val\}$. This is a non-saturated predicate, so there is no saturated basis for the *AD* function. It is therefore not one-step.
3. Multiple Dispose. This is the command $dispose[l_1, l_2]$ that faults if neither $l_1$ or $l_2$ are present, disposes $l_1$ if it is present and $l_2$ is not, disposes $l_2$ if it is present and $l_1$ is not, and diverges if both are present. Note that it has to diverge if both are present, otherwise it would not be local. This is an example of a one-step local function that is not precise: the set of footprints is $\{l_1 \mapsto v \mid v \in Val\} \cup \{l_2 \mapsto v \mid v \in Val\}$ and this is a saturated but imprecise predicate.

**Proposition 6.** *If $f$ is a one-step local function, then its footprints are the smallest states on which the function is safe: $F(f) = min(\{\sigma \mid f(\sigma) \sqsubset \top\})$.*

One-step and precise local functions also have advantageous properties with respect to their specifications. We can determine whether a function is one-step or precise by just looking at its specifications (checking specifications to be saturated or precise is easier than checking functions to be one-step or precise).

**Proposition 7.** *A local function $f$ is one-step iff it has at least one saturated complete specification. It is precise iff it has at least one precise complete specification.*

We can also sometimes determine the footprints of local functions directly from the specifications, without knowing what the function itself is. If the resource is non-negative and $f$ has a saturated complete specification, then the domain of this specification is the set of footprints of $f$.

**Proposition 8.** *If $\Sigma$ is non-negative and $\phi$ is a saturated complete specification for $f$, then the domain of $\phi$ is the set of footprints of $f$: that is, $F(f) = D(\phi)$.*

## 7 Conclusion and Future Work

We have defined the footprints of a local function and demonstrated that they are the only essential elements necessary to obtain complete specifications for local Hoare reasoning. For well-founded resource, the footprints are also sufficient, meaning that they do indeed yield the smallest complete specification. We have therefore solved the footprint problem highlighted in the introduction. We have also given results for the non-well-founded models and have introduced the natural class of one-step local functions for which the footprints are the smallest safe states.

Although we now know what footprints are, there is still much to investigate about their properties. Two important questions for future work are how footprints behave under the sequential composition of functions (we know from examples that it is not simply the union of the sets of footprints) and how the footprints impact on concurrent reasoning where identifying the minimal resource is of paramount importance.

Another question of interest is whether we can regain the simple intuition of footprints as minimal safe states by refining the semantics. We have already identified the natural class of one-step local functions for which the footprints do indeed have this property. However, these one-step functions are not compositional in general, as our *AD* program on the standard heap model illustrates. We are currently exploring an adapted

heap model, where the state keeps track of the allocated cells, aiming for the result that sequential composition does preserve our one-step property for this model. More generally, we will seek conditions under which one-step functions do indeed compose; such conditions should be satisfied by our alternative heap model.

**Acknowledgements.** We thank Calcagno, O'Hearn and Yang for detailed discussions on footprints. Raza acknowledges support of an ORS award. Gardner acknowledges support of a Microsoft/Royal Academy of Engineering Senior Research Fellowship.

## References

1. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape Analysis for Composite Data Structures. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, Springer, Heidelberg (2007)
2. Berdine, J., Calcagno, C., OHearn, P.W.: Smallfoot: Automatic modular assertion checking with separation logic. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, Springer, Heidelberg (2006)
3. Birkedal, L., Yang, H.: Relational parametricity and separation logic. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, Springer, Heidelberg (2007)
4. Bornat, R., Calcagno, C., O'Hearn, P., Parkinson, M.: Permission accounting in separation logic. In: 32nd POPL (2005)
5. Bornat, R., Calcagno, C., Yang, H.: Variables as resource in separation logic. In: 21st MFPS (2005)
6. Brookes, S.D.: A semantics for concurrent separation logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, Springer, Heidelberg (2004)
7. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic and tree update. In: 32nd POPL (2005)
8. Calcagno, C., Gardner, P., Zarfaty, U.: Local Reasoning about Data Update. In: Gordon Plotkin's festschrift, ENTCS (2007)
9. Calcagno, C., O'Hearn, P., Yang, H.: Local Action and Abstract Separation Logic (Longer version). In: LICS (2007)
10. Isthiaq, S., O'Hearn, P.W.: BI as an assertion language for mutable data structures. In: 28th POPL (2001)
11. Morgan, C.C.: The specification statement. In: ACM Transactions on Programming Languages and Systems (1988)
12. O'Hearn, P.: Resources, concurrency and local reasoning. Theoretical Computer Science, Preliminary version appeared in CONCUR 2004 (2007)
13. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, Springer, Heidelberg (2001)
14. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. In: Bulletin of Symbolic Logic (1999)
15. Parkinson, M., Bornat, R., Calcagno, C.: Variables as resource in Hoare logics. In: 21st LICS (2006)
16. Pym, D., O'Hearn, P., Yang, H.: Possible worlds and resources: The semantics of BI. In: Theoretical Computer Science (2004)
17. Pym, D.J.: The Semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series. Kluwer Academic Publishers, Dordrecht (2002)
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: 17th LICS (2002)
19. Yang, H., O'Hearn, P.: A semantic basis for local reasoning. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, Springer, Heidelberg (2002)

# A Modal Deconstruction of Access Control Logics

Deepak Garg[1] and Martín Abadi[2,3]

[1] Carnegie Mellon University
[2] University of California, Santa Cruz
[3] Microsoft Research, Silicon Valley

**Abstract.** We present a translation from a logic of access control with a "says" operator to the classical modal logic S4. We prove that the translation is sound and complete. We also show that it extends to logics with boolean combinations of principals and with a "speaks for" relation. While a straightforward definition of this relation requires second-order quantifiers, we use our translation for obtaining alternative, quantifier-free presentations. We also derive decidability and complexity results for the logics of access control.

## 1 Introduction

In computer systems, access control checks restrict the operations that users, machines, and other principals can execute on objects such as files [27]. These checks are governed by access control policies—often by the combination of several policies at different layers and from different entities. In practice, the principals, the objects, the formulations of policies, and their implementations can be quite varied. The resulting gaps, inconsistencies, and obscurity endanger security.

In response to these concerns, specialized logics have been proposed as frameworks for describing, analyzing, and enforcing access control policies (e.g., [3,30, 20,19,2,6,10,29]). A number of research projects have applied these logics for designing or explaining various languages and systems (e.g., [26,35,6,8,7,9,13, 4,16,10,14,18,29]). On the other hand, there have been only few, limited efforts to study the logics themselves (e.g., [3,20,19,2]). Accordingly, the decidability, expressiveness, and semantics of these logics are largely unexplored.

Our objective in the present paper is to fill this gap. Specifically, we study a class of access control logics via sound and complete translations to the classical modal logic S4.

- Relying on the theory of S4 (e.g., [24,25]), we obtain Kripke semantics for the logics. In the quantifier-free case, we also establish the decidability of the logics and their PSPACE complexity. The translations also open the possibility of re-using existing decision procedures for S4.
- Translating several logics to S4 enables us to compare their expressiveness. In particular, while a straightforward definition of the "speaks for" relation [26, 28] requires second-order quantifiers, we use our translations for obtaining

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 216–230, 2008.

alternative, quantifier-free presentations. We prove that these quantifier-free presentations yield the same consequences as the second-order definition.
– The translations also suggest a logic with a boolean structure on principals. Although propositional, this new logic is rich and quite expressive. Previous logics with similar constructs allowed conjunctions and disjunctions of principals (but not negations); the present logic goes beyond them in ways that we consider both elegant and useful.

Access control logics (those studied here and most of those in the literature) include formulas of the form $\mathsf{A}$ `says` $s$, where $\mathsf{A}$ is a principal and $s$ is a formula. Intuitively, $\mathsf{A}$ `says` $s$ means that $\mathsf{A}$ asserts (or supports) $s$. For example, the administrator admin of a domain might certify that Alice is an authorized user; this assertion may be represented as admin `says` `auth_user`(Alice). In addition, many logics support the use of the "speaks for" relation: $\mathsf{A} \Rightarrow \mathsf{B}$ means that $\mathsf{A}$ speaks for $\mathsf{B}$, that is, $\mathsf{A}$ `says` $s$ implies $\mathsf{B}$ `says` $s$ for every $s$. For example, when $\mathsf{Key}_{\mathsf{Alice}}$ represents the public key of Alice, one may write $\mathsf{Key}_{\mathsf{Alice}} \Rightarrow \mathsf{Alice}$. When a server $\mathsf{S}$ acts on Alice's behalf impersonating her, one may also write $\mathsf{S} \Rightarrow \mathsf{Alice}$. Despite these similarities, logics differ in their axioms. A 2003 survey discusses some of the options [1]. Recently, several works [20,2,19,29] have basically relied upon the rules of lax logic and the computational lambda calculus [17,11,33] for the operator `says`. This approach has several benefits, for example validating the "handoff axiom" [26,2]; a detailed discussion of its features is beyond the scope of this paper. We follow this approach in the logics that we consider.

The first of these logics, called ICL, extends propositional intuitionistic logic with the operator `says` which behaves as a principal-indexed lax modality (Section 2). ICL can be viewed as an indexed version of CL [11], hence its name, and also as the common propositional fragment of CDD [2, Section 8] and other systems [20, 29]. An extension of ICL, called $\mathrm{ICL}^{\Rightarrow}$, allows formulas of the form $\mathsf{A} \Rightarrow \mathsf{B}$ (Section 3). Another extension, called $\mathrm{ICL}^{\mathcal{B}}$, allows compound principals formed with boolean connectives (Section 4). Our translations and the resulting theorems apply to each of these logics. In addition, we show that $\mathsf{A} \Rightarrow \mathsf{B}$ can be encoded using either compound principals or a second-order universal quantifier (Sections 5 and 6). We conclude with a discussion of directions for further work (Section 7). Proofs are available on-line at `www.cs.cmu.edu/~dg/papers/modal-decons-full.pdf`.

*Related Work.* Our translations are partly based on a translation from intuitionistic logic to S4 that goes back to Gödel [22]. Moreover, ICL can be seen as a rather direct generalization of lax logic. Nevertheless, our translation from ICL (and, as a special case, from lax logic) to S4 appears to be new.

Partly following Curry [15], Fairtlough and Mendler suggested interpreting lax logic in intuitionistic logic by mapping $\bigcirc s$ to $C \vee s$ or to $C \supset s$, where $\bigcirc$ is a lax modality and $C$ is a fixed proposition [17]. These interpretations are sound but not complete. Composing them with a translation from intuitionistic logic to S4, one can map $\bigcirc s$ to $\Box((\Box C) \vee s)$ or to $\Box((\Box C) \supset s)$. A similar translation from lax logic to S4 follows from our definitions, as a special case; however, our translation does not put a $\Box$ on $C$, and it is sound and complete.

Other interpretations of lax logic have targeted multimodal logics or intuitionistic S4 [17,5,11,34]. Our translations seem simpler; in particular, they target classical S4. Semantically, those interpretations lead to Kripke models with at least two accessibility relations, while we need only one.

Fairtlough and Mendler also deduced the decidability of lax logic from a subformula property [17]. Further, Howe developed a PSPACE decision procedure for lax logic [23]. It seems possible to extend Howe's approach to obtain an alternative proof of decidability for ICL. We do not know whether it would also apply to richer logics such as $\text{ICL}^{\Rightarrow}$ and $\text{ICL}^{\mathcal{B}}$, for which we have not established a subformula property.

Going beyond basic lax logic, not much is known about the theory of logics with compound principals or with a "speaks for" relation (such as $\text{ICL}^{\Rightarrow}$ and $\text{ICL}^{\mathcal{B}}$). Some of the early work on the subject started to explore semantics and decidability results [3]. Although sometimes helpful, the semantics were not sound and complete, and the decidability results applied only to fragments needed for certain access-control decisions. More recent systems like RT and SecPAL (where the "can act as" relation resembles $\Rightarrow$) include decision procedures for useful classes of formulas similar to Horn clauses [31,32,10].

## 2 ICL: A Basic Logic of Access Control

We start with a basic access control logic ICL that includes the operator $\mathtt{says}$ but not $\Rightarrow$. Although minimal in its constructs, the logic is reasonably expressive. We describe a translation from ICL to classical S4. From this translation we derive a Kripke semantics and a decidability result.

### 2.1 The Logic

Formulas in ICL may be atomic propositions ($p$, $q$, etc.) or constructed from standard connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\top$ (true), and $\bot$ (false), and the operator $\mathtt{says}$.

$$s \ ::= \ p \mid s_1 \wedge s_2 \mid s_1 \vee s_2 \mid s_1 \supset s_2 \mid \top \mid \bot \mid \mathsf{A}\ \mathtt{says}\ s$$

The letters $\mathsf{A}$, $\mathsf{B}$, etc., denote principals, which are atomic and distinct from atomic propositions. They may be simple bit-string representations of names; in Section 4, we generalize principals to a richer algebra.

ICL inherits all the inference rules of intuitionistic propositional logic, which we elide here. For each principal $\mathsf{A}$, the formula $\mathsf{A}\ \mathtt{says}\ s$ satisfies the following axioms:

$$\begin{array}{ll} \vdash s \supset (\mathsf{A}\ \mathtt{says}\ s) & \text{(unit)} \\ \vdash (\mathsf{A}\ \mathtt{says}\ (s \supset t)) \supset (\mathsf{A}\ \mathtt{says}\ s) \supset (\mathsf{A}\ \mathtt{says}\ t) & \text{(cuc)} \\ \vdash (\mathsf{A}\ \mathtt{says}\ \mathsf{A}\ \mathtt{says}\ s) \supset \mathsf{A}\ \mathtt{says}\ s & \text{(idem)} \end{array}$$

These mean that $\mathsf{A}\ \mathtt{says}\ \cdot$ is a lax modality [17]. We describe them briefly, referring the reader to the literature on lax logic and computational lambda calculus for more details and applications.

- (unit) states that every true formula $s$ is supported by every principal $\mathsf{A}$. (The converse is not true: principals may make false statements.)
- (cuc) allows us to reason with $\mathsf{A}$'s statements. It says that whenever $\mathsf{A}$ states $s \supset t$ and $s$, it also states $t$. Thus $\mathsf{A}$'s statements are closed under logical consequence.
- (idem) collapses applications of $\mathsf{A}\ \mathsf{says}\ \cdot$. In the context of (unit), (idem) implies that $\mathsf{A}\ \mathsf{says}\ \cdot$ is idempotent.

**Example 1.** We illustrate the use of ICL through a simple example. Consider a file-access scenario with an administrating principal admin, a user Bob, one file file1, and the following policy:

1. If admin says that file1 should be deleted, then this must be the case.
2. admin trusts Bob to decide whether file1 should be deleted.
3. Bob wants to delete file1.

Intuitively, from these facts we should be able to conclude that file1 should be deleted. We describe a logical presentation of this example in ICL. Suppose that the proposition deletefile1 means that file1 should be deleted. The three facts above can be written:

1. $(\mathsf{admin\ says\ deletefile1}) \supset \mathsf{deletefile1}$
2. $\mathsf{admin\ says}\ ((\mathsf{Bob\ says\ deletefile1}) \supset \mathsf{deletefile1})$
3. $\mathsf{Bob\ says\ deletefile1}$

Using (unit) and (cuc), (1)–(3) imply deletefile1.

### 2.2 Translation from ICL to S4

Next we describe a central technical result of our work: a sound and complete translation from ICL to S4. Before describing the translation, we briefly sketch S4. More details may be found in standard references (e.g., [24]); S4 has been studied thoroughly over the years.

*S4.* S4 is an extension of classical logic with one modality $\Box$, and the rules:

From $\vdash s$ infer $\vdash \Box s$.
$\vdash \Box(s \supset t) \supset \Box s \supset \Box t$
$\vdash \Box s \supset s$
$\vdash \Box s \supset \Box\Box s$

*Translation.* Our translation $\ulcorner\cdot\urcorner$ from ICL to S4 is summarized in Figure 1. It is defined by induction on the structure of formulas. For atomic formulas and non-modal connectives, the translation is a slight simplification of Gödel's translation from intuitionistic logic to S4 [22]. (In Gödel's words, the basic idea is to "put a box around everything"; we simplify the translation by putting boxes only around atomic formulas and implications.) The core of our work is the translation of $\mathsf{A}\ \mathsf{says}\ s$.

$$\ulcorner \mathsf{A\ says}\ s \urcorner \ = \ \Box(\mathsf{A} \vee \ulcorner s \urcorner)$$

$$
\begin{array}{rcl}
\ulcorner p \urcorner & = & \Box p \\
\ulcorner s \wedge t \urcorner & = & \ulcorner s \urcorner \wedge \ulcorner t \urcorner \\
\ulcorner s \vee t \urcorner & = & \ulcorner s \urcorner \vee \ulcorner t \urcorner \\
\ulcorner s \supset t \urcorner & = & \Box(\ulcorner s \urcorner \supset \ulcorner t \urcorner) \\
\ulcorner \top \urcorner & = & \top \\
\ulcorner \bot \urcorner & = & \bot \\
\ulcorner \mathsf{A\ says}\ s \urcorner & = & \Box(\mathsf{A} \vee \ulcorner s \urcorner)
\end{array}
$$

**Fig. 1.** Translation from ICL to S4

We interpret the principal $\mathsf{A}$ as an atomic formula in S4 and assume that such atomic formulas are distinct from the usual atomic formulas $p$, $q$, etc.. Informally, if we read $\Box$ as "in all possible worlds" and the atomic formula $\mathsf{A}$ as "principal $\mathsf{A}$ is unhappy", then $\Box(\mathsf{A} \vee \ulcorner s \urcorner)$ means that $\ulcorner s \urcorner$ holds in all possible worlds in which $\mathsf{A}$ is happy.

Alternatively, but equivalently, we could set: $\ulcorner \mathsf{A\ says}\ s \urcorner = \Box(\mathsf{A} \supset \ulcorner s \urcorner)$. Since the target of the translation is a classical logic, the difference between $\Box(\mathsf{A} \vee \ulcorner s \urcorner)$ and $\Box(\mathsf{A} \supset \ulcorner s \urcorner)$ is only superficial. We prefer $\Box(\mathsf{A} \vee \ulcorner s \urcorner)$ because it leads to a more memorable interpretation of $\Rightarrow$ in Section 3.

This simple translation is correct in the sense that it is both sound and complete:

**Theorem 1 (Soundness and Completeness).** *For every ICL formula $s$, $\vdash s$ in ICL if and only if $\vdash \ulcorner s \urcorner$ in S4.*

The proof of this theorem relies heavily on the proof theory of ICL and S4.

### 2.3 Decidability and Kripke Models for ICL

Decidability is a desirable property in an access control logic: it allows the possibility of completely automated tools for analyzing policies. In the case of ICL, Theorem 1 implies PSPACE decidability since the same complexity bound is known for S4 [25]. This bound is the best we could expect, since PSPACE-hardness holds for plain intuitionistic propositional logic.

**Corollary 1 (Decidability).** *There is a polynomial space procedure that decides whether a given ICL formula is provable or not.*

Kripke models are attractive for access control logics from several perspectives. First, they provide a semantic grounding of the logics. They are also useful as mathematical objects, for instance for showing that certain formulas are not derivable. We use Theorem 1 and standard models of S4 to derive Kripke models for ICL.

**Definition 1 (Kripke Models).** A Kripke model for ICL is a tuple $\langle W, \leq, \rho, \theta \rangle$ where

- $W$ is a set, whose elements are called *worlds* (denoted using the letter $w$ and its decorated variants).

- $\leq$ is a binary relation on $W$ called the *accessibility relation*. When $w \leq w'$, we say that $w'$ is accessible from $w$. We assume that $\leq$ is reflexive and transitive. We often write $\geq$ for $(\leq)^{-1}$.
- $\rho$ is a mapping from atomic formulas of ICL to $\mathcal{P}(W)$ (the power set of W), called the *assignment*. Intuitively, $\rho(p)$ is the set of worlds in which $p$ holds. We assume that $\rho$ is hereditary with respect to $\leq$, that is, if $w \in \rho(p)$, then for all $w'$ such that $w' \geq w$, $w' \in \rho(p)$.
- $\theta$ is a mapping from principals of ICL to $\mathcal{P}(W)$, called the *view map*. When $w \in \theta(\mathsf{A})$, we say that $w$ is *invisible* to $\mathsf{A}$, else it is *visible* to $\mathsf{A}$.

**Definition 2 (Satisfaction).** Given an ICL formula $s$ and a Kripke model $\mathcal{K} = \langle W, \leq, \rho, \theta \rangle$, we define the satisfaction relation at a particular world ($w \models s$) by induction on $s$.

- $w \models p$ iff $w \in \rho(p)$
- $w \models s \wedge t$ iff $w \models s$ and $w \models t$
- $w \models s \vee t$ iff $w \models s$ or $w \models t$
- $w \models s \supset t$ iff for each $w' \geq w$, $w' \models s$ implies $w' \models t$
- $w \models \top$ for every $w$
- not$(w \models \bot)$ for every $w$
- $w \models \mathsf{A}$ says $s$ iff for every $w' \geq w$, either $w' \in \theta(\mathsf{A})$ or $w' \models s$

Thus, this definition implies that a world satisfies $\mathsf{A}$ says $s$ iff every reachable world that is visible to $\mathsf{A}$ satisfies $s$. For other constructs, the definition of satisfaction mirrors that in standard Kripke models of intuitionistic logic.

We say that $\mathcal{K} = \langle W, \leq, \rho, \theta \rangle \models s$ if $w \models s$ for every $w \in W$. A formula $s$ is *valid* (written $\models s$) if $\mathcal{K} \models s$ for every Kripke model $\mathcal{K}$. The following result shows that provability in ICL coincides with validity.

**Corollary 2.** *For every ICL formula $s$, $\vdash s$ if and only if $\models s$.*

## 3 ICL$^{\Rightarrow}$: A Logic with a Primitive "Speaks For" Relation

In this section we extend the logic ICL to include a primitive "speaks for" relation. We call the new logic ICL$^{\Rightarrow}$. We also extend the results of Section 2 to ICL$^{\Rightarrow}$.

### 3.1 The Logic

ICL$^{\Rightarrow}$ extends ICL with formulas of the form $\mathsf{A} \Rightarrow \mathsf{B}$ and with the following axioms for these formulas:

| | |
|---|---|
| $\vdash \mathsf{A} \Rightarrow \mathsf{A}$ | (refl) |
| $\vdash (\mathsf{A} \Rightarrow \mathsf{B}) \supset (\mathsf{B} \Rightarrow \mathsf{C}) \supset (\mathsf{A} \Rightarrow \mathsf{C})$ | (trans) |
| $\vdash (\mathsf{A} \Rightarrow \mathsf{B}) \supset (\mathsf{A}$ says $s) \supset (\mathsf{B}$ says $s)$ | (speaking-for) |
| $\vdash (\mathsf{B}$ says $(\mathsf{A} \Rightarrow \mathsf{B})) \supset (\mathsf{A} \Rightarrow \mathsf{B})$ | (handoff) |

– (refl) and (trans) state that $\Rightarrow$ is reflexive and transitive.
– (speaking-for) states that if $\mathsf{A} \Rightarrow \mathsf{B}$ and $\mathsf{A}$ says $s$, then $\mathsf{B}$ says $s$.

– (handoff) states that whenever B says that A speaks for B, then A does indeed speak for B. This axiom allows every principal to decide which principals speak on their behalf [26].

**Example 2.** We modify Example 1: instead of having Bob says deletefile1 directly, Bob delegates his authority to Alice (fact 3), who wants to delete file1 (fact 4).

1. (admin says deletefile1) $\supset$ deletefile1
2. admin says ((Bob says deletefile1) $\supset$ deletefile1)
3. Bob says Alice $\Rightarrow$ Bob
4. Alice says deletefile1

Using (handoff) and (speaking-for), we can again derive deletefile1.

### 3.2 Translation from ICL$^{\Rightarrow}$ to S4

We extend to ICL$^{\Rightarrow}$ the translation from ICL to S4 by adding the clause:

$$\ulcorner \mathsf{A} \Rightarrow \mathsf{B} \urcorner \;=\; \Box(\mathsf{A} \supset \mathsf{B})$$

As above, A and B are interpreted as atomic formulas in S4, and these atomic formulas are assumed distinct from the atomic propositions of ICL$^{\Rightarrow}$. We have:

**Theorem 2 (Soundness and Completeness).** *For every ICL$^{\Rightarrow}$ formula $s$, $\vdash s$ in ICL$^{\Rightarrow}$ if and only if $\vdash \ulcorner s \urcorner$ in S4.*

### 3.3 Decidability and Kripke Models for ICL$^{\Rightarrow}$

Much as for ICL, Theorem 2 yields a decidability result:

**Corollary 3 (Decidability).** *There is a polynomial space procedure that decides whether a given ICL$^{\Rightarrow}$ formula is provable or not.*

It also leads to Kripke models for ICL$^{\Rightarrow}$. These are the same as those for ICL (Definition 1), with the satisfaction relation for $\mathsf{A} \Rightarrow \mathsf{B}$ at world $w$ given by the clause:

$w \models \mathsf{A} \Rightarrow \mathsf{B}$ iff for every $w' \geq w$, $w' \in \theta(\mathsf{A})$ implies $w' \in \theta(\mathsf{B})$.

These models are sound and complete in the sense of Corollary 2.

## 4 ICL$^{\mathcal{B}}$: A Logic with Boolean Principals

Principals in ICL and ICL$^{\Rightarrow}$ are atomic and cannot be composed in any logically meaningful way. Early on it was observed that the use of compound principals can help in expressing policies [26,3]. For example, the conjunction of two principals may be employed for representing joint statements, with the property

$$(\mathsf{A} \wedge \mathsf{B})\ \mathsf{says}\ s \equiv (\mathsf{A}\ \mathsf{says}\ s) \wedge (\mathsf{B}\ \mathsf{says}\ s)$$

Disjunctions also arise, though they are more complex. Going further, we describe and study a systematic extension $\mathrm{ICL}^{\mathcal{B}}$ of ICL that allows arbitrary Boolean combinations of principals with the connectives $\wedge$, $\vee$, $\supset$, $\top$, and $\bot$. (However, we do not include operators such as "quoting" and "for".) We extend the results of Section 2 to $\mathrm{ICL}^{\mathcal{B}}$.

### 4.1 The Logic

The formulas of $\mathrm{ICL}^{\mathcal{B}}$ are the same as those of ICL, except that principals may contain Boolean connectives. We use the letters $a$, $b$, ... for denoting atomic principals (distinct from atomic propositions), and $\mathsf{A}$, $\mathsf{B}$, ... for denoting arbitrary principals.

$$\mathsf{A}, \mathsf{B} \ ::= \ a \mid \mathsf{A} \wedge \mathsf{B} \mid \mathsf{A} \vee \mathsf{B} \mid \mathsf{A} \supset \mathsf{B} \mid \top \mid \bot$$

We write $\neg\mathsf{A}$ for $(\mathsf{A} \supset \bot)$. We equip the set of principals with a notion of equality by letting $\mathsf{A} \equiv \mathsf{B}$ if $\mathsf{A}$ and $\mathsf{B}$ are provably equivalent when viewed as formulas in classical logic. With these definitions, the set of principals becomes a Boolean algebra.

$\mathrm{ICL}^{\mathcal{B}}$ inherits all the inference rules of ICL, and also includes the following additional rules:

$$\begin{array}{ll} \vdash (\bot \ \mathtt{says}\ s) \supset s & \text{(trust)} \\ \text{If } \mathsf{A} \equiv \top \text{ then } \vdash \mathsf{A} \ \mathtt{says}\ \bot. & \text{(untrust)} \\ \vdash ((\mathsf{A} \supset \mathsf{B}) \ \mathtt{says}\ s) \supset (\mathsf{A} \ \mathtt{says}\ s) \supset (\mathsf{B} \ \mathtt{says}\ s) & \text{(cuc')} \end{array}$$

- (trust) states that $\bot$ is a truth teller.
- (untrust) states that any principal equivalent to $\top$ says false; it can be seen as a variant of the necessitation rule of modal logics.
- Similarly, (cuc') is the analogue of (cuc) for principals. It states that $\mathsf{A}$ says $s$ and $(\mathsf{A} \supset \mathsf{B})$ says $s$ imply $\mathsf{B}$ says $s$.

We define $\mathrm{ICL}^{\mathcal{B}}$ as an extension of ICL, rather than $\mathrm{ICL}^{\Rightarrow}$, because we do not need built-in formulas of the form $\mathsf{A} \Rightarrow \mathsf{B}$. The "speaks for" relation is definable in $\mathrm{ICL}^{\mathcal{B}}$. As we show in Section 5, $\mathsf{A} \Rightarrow \mathsf{B}$ can be seen as an abbreviation for $(\mathsf{A} \supset \mathsf{B})$ says $\bot$.

We can explain the intuitive meaning of $\mathsf{A}$ says $s$ when principal $\mathsf{A}$ is compound, as follows:

- $(\mathsf{A} \wedge \mathsf{B})$ says $s$ is the same as $(\mathsf{A}$ says $s) \wedge (\mathsf{B}$ says $s)$.
- $(\mathsf{A} \vee \mathsf{B})$ says $s$ means that, by combining the statements of $\mathsf{A}$ and $\mathsf{B}$, we can conclude $s$. In particular, if $\mathsf{A}$ says $(s \supset t)$ and $\mathsf{B}$ says $s$ then $(\mathsf{A} \vee \mathsf{B})$ says $t$. Disjunctions can be used in modeling groups in access control.
- $(\mathsf{A} \supset \mathsf{B})$ says $s$ means that $\mathsf{A}$ speaks for $\mathsf{B}$ on $s$ and on its consequences. We can show that if $(\mathsf{A} \supset \mathsf{B})$ says $s$ and $s \supset s'$, then $(\mathsf{A}$ says $s') \supset (\mathsf{B}$ says $s')$. In the special case where $s$ is $\bot$, we obtain the usual $\Rightarrow$ relation.
- $\top$ says $s$ is provable for every formula $s$ (including $\bot$). In access control terms, $\top$ may be seen as a completely untrustworthy principal.
- $\bot$ says $s$ implies that $s$ is true. Thus, $\bot$ is a completely trustworthy principal.

**Example 3.** The following policy is analogous to that of Example 1:

1. $(\mathsf{admin} \supset \bot)$ says deletefile1
2. admin says $(\mathsf{Bob} \supset \mathsf{admin})$ says deletefile1
3. Bob says deletefile1

The first statement means that admin is trusted on deletefile1 and its consequences. The second statement means that admin further delegates this authority to Bob.

From (3) and (unit) it follows that admin says Bob says deletefile1. From (2), (cuc), and (cuc') we get (admin says Bob says deletefile1) $\supset$ (admin says admin says deletefile1). Hence we have admin says admin says deletefile1. Using (idem), we obtain admin says deletefile1. From (1) and (cuc'), we obtain (admin says deletefile1) $\supset \bot$ says deletefile1, and hence $\bot$ says deletefile1. Finally, using (trust), we conclude deletefile1.

### 4.2 Translation from ICL$^{\mathcal{B}}$ to S4

The translation from ICL to S4 works virtually unchanged for ICL$^{\mathcal{B}}$. In the clause $\ulcorner \mathsf{A} \text{ says } s \urcorner = \Box(\mathsf{A} \vee \ulcorner s \urcorner)$, we interpret A as a formula in S4 in the most obvious way: each Boolean connective in A is mapped to the corresponding connective in S4, and each atomic principal in A is interpreted as an atomic formula in S4 (without any added boxes). For instance, the translation of

$$(\mathsf{Bob} \supset \mathsf{admin}) \text{ says deletefile1}$$

is

$$\Box((\mathsf{Bob} \supset \mathsf{admin}) \vee \Box \mathsf{deletefile1})$$

Again, we have soundness and completeness results:

**Theorem 3 (Soundness and Completeness).** *For every ICL$^{\mathcal{B}}$ formula $s$, $\vdash s$ in ICL$^{\mathcal{B}}$ if and only if $\vdash \ulcorner s \urcorner$ in S4.*

### 4.3 Decidability and Kripke Models for ICL$^{\mathcal{B}}$

Once more we obtain a decidability result:

**Corollary 4 (Decidability).** *There is a polynomial space procedure that decides whether a given ICL$^{\mathcal{B}}$ formula is provable or not.*

Furthermore, Kripke models for ICL$^{\mathcal{B}}$ may be obtained by generalizing those for ICL. The view map $\theta$ is defined only for atomic principals $a$. It is lifted to the function $\hat{\theta}$ that maps all principals to $\mathcal{P}(W)$ as follows:

$$\begin{array}{rcc}
\hat{\theta}(a) & = & \theta(a) \\
\hat{\theta}(\mathsf{A} \wedge \mathsf{B}) & = & \hat{\theta}(\mathsf{A}) \cap \hat{\theta}(\mathsf{B}) \\
\hat{\theta}(\mathsf{A} \vee \mathsf{B}) & = & \hat{\theta}(\mathsf{A}) \cup \hat{\theta}(\mathsf{B}) \\
\hat{\theta}(\mathsf{A} \supset \mathsf{B}) & = & (W - \hat{\theta}(\mathsf{A})) \cup \hat{\theta}(\mathsf{B}) \\
\hat{\theta}(\top) & = & W \\
\hat{\theta}(\bot) & = & \emptyset
\end{array}$$

The definition of satisfaction ($w \models s$) is modified to use $\hat{\theta}$ instead of $\theta$:

$$w \models \mathsf{A\ says}\ s \text{ iff for all } w' \geq w, \text{ either } w' \in \hat{\theta}(\mathsf{A}) \text{ or } w' \models s.$$

Again, these Kripke models are sound and complete in the sense of Corollary 2.

Thus, while the analysis of the translations requires special (and often difficult) arguments for each logic, the way in which decidability and semantics follow from translations is almost identical across logics. In the remainder of the paper, we turn to more unexpected consequences of the translations.

## 5 From ICL$^{\Rightarrow}$ to ICL$^{\mathcal{B}}$: Expressing "Speaks For" Via Boolean Principals

We prove that $\mathsf{A} \Rightarrow \mathsf{B}$ can be encoded as $(\mathsf{A} \supset \mathsf{B})\ \mathsf{says}\ \bot$. More precisely, we analyze the following translation $(\overline{\cdot})$ from ICL$^{\Rightarrow}$ to ICL$^{\mathcal{B}}$. It maps every connective except $\Rightarrow$ to itself.

$$\begin{array}{rcl}
\overline{p} & = & p \\
\overline{s \wedge t} & = & \overline{s} \wedge \overline{t} \\
\overline{s \vee t} & = & \overline{s} \vee \overline{t} \\
\overline{s \supset t} & = & \overline{s} \supset \overline{t} \\
\overline{\top} & = & \top \\
\overline{\bot} & = & \bot \\
\overline{\mathsf{A\ says}\ s} & = & \mathsf{A\ says}\ \overline{s} \\
\overline{\mathsf{A} \Rightarrow \mathsf{B}} & = & (\mathsf{A} \supset \mathsf{B})\ \mathsf{says}\ \bot
\end{array}$$

(Alternatively, we could translate an extension of ICL$^{\mathcal{B}}$ with $\Rightarrow$ to ICL$^{\mathcal{B}}$.) The encoding of $\Rightarrow$ is correct, in the following sense:

**Theorem 4.** *For every ICL$^{\Rightarrow}$ formula $s$, $\vdash s$ in ICL$^{\Rightarrow}$ if and only if $\vdash \overline{s}$ in ICL$^{\mathcal{B}}$.*

This theorem is easy to establish using the translations from ICL$^{\Rightarrow}$ and ICL$^{\mathcal{B}}$ to S4. First we show that for every formula $s$ in ICL$^{\Rightarrow}$, $\ulcorner s \urcorner$ and $\ulcorner \overline{s} \urcorner$ are provably equivalent in S4. This argument is by a structural induction on $s$. The only interesting case is for $s$ of the form $\mathsf{A} \Rightarrow \mathsf{B}$, where we observe that $\ulcorner \mathsf{A} \Rightarrow \mathsf{B} \urcorner = \Box(\mathsf{A} \supset \mathsf{B}) \equiv \Box((\mathsf{A} \supset \mathsf{B}) \vee \bot) = \ulcorner \overline{\mathsf{A} \Rightarrow \mathsf{B}} \urcorner$. It then follows from Theorems 2 and 3 that $\vdash s$ iff $\vdash \ulcorner s \urcorner$ iff $\vdash \ulcorner \overline{s} \urcorner$ iff $\vdash \overline{s}$.

## 6 On Second-Order Quantification

In this section we consider a logic with second-order quantification. In this logic, $\mathsf{A} \Rightarrow \mathsf{B}$ has a well-known, compelling definition, as an abbreviation for

$$\forall X.\ \mathsf{A\ says}\ X \supset \mathsf{B\ says}\ X$$

Our main technical goal is to relate this definition to the quantifier-free axiomatizations of Sections 3–5. We prove that those axiomatizations are sound and complete with respect to the second-order definition. Thus, the full power and complexity of second-order quantification is not required for reasoning about $\Rightarrow$. A decidable fragment of the second-order logic suffices.

(This result was far from obvious to us: a priori, it seemed entirely possible that the axiomatizations were missing some subtle consequence of the second-order definition. Its proof was also surprising, as it includes a non-constructive detour through Kripke models, thus leveraging the work of Sections 3–5.)

### 6.1 The Logic

The second-order logic is the straightforward extension of ICL with universal quantification over propositions, with the rules of System F [21,12].

This logic is not entirely new. It has previously been defined [2, Section 8] and used [18] under the name CDD (with only minor syntactic differences). Here we call it $\mathrm{ICL}^{\forall}$ for the sake of uniformity.

The addition of second-order quantification provides great expressiveness, as illustrated by the definition of $\Rightarrow$ given above. On the other hand, it immediately leads to undecidability as well as to other difficulties. Nevertheless, this logic is an obvious and elegant extension of ICL.

### 6.2 Main Results

Though we do not discuss the theory of $\mathrm{ICL}^{\forall}$ in detail, we have had to develop some of it in the course of our study of $\Rightarrow$. In this section we present only our main result on $\Rightarrow$ and mention other developments to the extent that they are relevant to this result.

There is an obvious embedding of $\mathrm{ICL}^{\Rightarrow}$ into $\mathrm{ICL}^{\forall}$:

$$
\begin{array}{rcl}
[\![p]\!] & = & p \\
[\![s \wedge t]\!] & = & [\![s]\!] \wedge [\![t]\!] \\
[\![s \vee t]\!] & = & [\![s]\!] \vee [\![t]\!] \\
[\![s \supset t]\!] & = & [\![s]\!] \supset [\![t]\!] \\
[\![\top]\!] & = & \top \\
[\![\bot]\!] & = & \bot \\
[\![\mathsf{A\ says}\ s]\!] & = & \mathsf{A\ says}\ [\![s]\!] \\
[\![\mathsf{A} \Rightarrow \mathsf{B}]\!] & = & \forall X.\ \mathsf{A\ says}\ X \supset \mathsf{B\ says}\ X
\end{array}
$$

This embedding is correct, in the following sense:

**Theorem 5.** *For every $\mathrm{ICL}^{\Rightarrow}$ formula $s$, $\vdash s$ in $\mathrm{ICL}^{\Rightarrow}$ if and only if $\vdash [\![s]\!]$ in $\mathrm{ICL}^{\forall}$.*

Soundness (the implication from left to right) is easy to establish. It suffices to show that each axiom of $\mathrm{ICL}^{\Rightarrow}$ can be simulated in $\mathrm{ICL}^{\forall}$ after translation.

Completeness (the implication from right to left) is much harder. Complications arise because a proof of $[\![s]\!]$ may contain formulas which are not in the image of $[\![\cdot]\!]$. Even if we wish to restrict attention to a fragment in which the universal quantifier is restricted to formulas of the form $\forall X.\ \mathsf{A}\ \mathtt{says}\ X \supset \mathsf{B}\ \mathtt{says}\ X$, the proofs of theorems in this fragment may mention formulas that contain universal quantifiers in other positions. Although it is conceivable that a constructive proof-theoretic argument would be viable, this difficulty leads us to a non-constructive argument through acyclic Kripke models.

Our approach seems to be new, so we discuss it in some detail. It is as follows.

- First we define a translation from $\mathrm{ICL}^{\forall}$ to second-order S4 (called $\mathrm{S4}^{\forall}$), that is, classical S4 with a second-order universal quantifier. Let us call this translation $\ulcorner\cdot\urcorner$. This translation essentially mimics the translation of ICL to S4, and in addition maps $\forall X.\ s$ to $\Box\forall X.\ \ulcorner s\urcorner$.

  We show that this translation is sound, in the sense that $\vdash s$ in $\mathrm{ICL}^{\forall}$ implies $\vdash \ulcorner s\urcorner$ in $\mathrm{S4}^{\forall}$. It follows immediately that $\vdash [\![s]\!]$ in $\mathrm{ICL}^{\forall}$ implies $\vdash \ulcorner[\![s]\!]\urcorner$ in $\mathrm{S4}^{\forall}$.

  (We do not need to be concerned about the completeness of this translation for our purposes.)
- Next we may try to show that for every $\mathrm{ICL}^{\Rightarrow}$ formula $s$, if $\vdash \ulcorner[\![s]\!]\urcorner$ in $\mathrm{S4}^{\forall}$ then $\vdash \ulcorner s\urcorner$ in S4. If this were true, Theorem 2 would yield that $\vdash [\![s]\!]$ in $\mathrm{ICL}^{\forall}$ implies $\vdash s$ in $\mathrm{ICL}^{\Rightarrow}$ (because $\vdash [\![s]\!]$ in $\mathrm{ICL}^{\forall}$ implies $\vdash \ulcorner[\![s]\!]\urcorner$ in $\mathrm{S4}^{\forall}$, as noted above).

  Thus, it would suffice to establish that $\vdash \ulcorner[\![s]\!]\urcorner$ in $\mathrm{S4}^{\forall}$ implies $\vdash \ulcorner s\urcorner$ in S4. We try to prove this by induction on $s$. Unfortunately, the proof does not go through. The argument fails for a formula of the form $\mathsf{A} \Rightarrow \mathsf{B}$, since

$$\ulcorner[\![\mathsf{A} \Rightarrow \mathsf{B}]\!]\urcorner = \Box\forall X.\ \Box\left(\Box\left(\mathsf{A} \vee \Box X\right) \supset \Box\left(\mathsf{B} \vee \Box X\right)\right)$$

  and

$$\ulcorner\mathsf{A} \Rightarrow \mathsf{B}\urcorner = \Box\left(\mathsf{A} \supset \mathsf{B}\right)$$

  In $\mathrm{S4}^{\forall}$, the latter implies the former, but the former does not imply the latter.
- Two observations allow the proof to go through:
  1. On all acyclic models, $\ulcorner[\![\mathsf{A} \Rightarrow \mathsf{B}]\!]\urcorner$ implies $\ulcorner\mathsf{A} \Rightarrow \mathsf{B}\urcorner$.
     Therefore, we can establish that all acyclic models satisfy $\ulcorner[\![s]\!]\urcorner$ if and only if all acyclic models satisfy $\ulcorner s\urcorner$.
  2. Quantifier-free S4 is sound and complete with respect to acyclic models. (A model can be "unrolled", and the resulting acyclic model satisfies the same quantifier-free formulas as the original model.)
- Using these observations we can complete our proof as follows.
  - Suppose that $\vdash [\![s]\!]$ in $\mathrm{ICL}^{\forall}$.
  - By the soundness of the translation from $\mathrm{ICL}^{\forall}$ to $\mathrm{S4}^{\forall}$, we obtain $\vdash \ulcorner[\![s]\!]\urcorner$ in $\mathrm{S4}^{\forall}$.
  - Therefore every acyclic model of $\mathrm{S4}^{\forall}$ satisfies $\ulcorner[\![s]\!]\urcorner$.
  - By (1), every acyclic model of $\mathrm{S4}^{\forall}$ satisfies $\ulcorner s\urcorner$.

- Since, for S4 formulas (without quantifiers), the models of $S4^{\forall}$ are the same as the models of S4, every acyclic model of S4 satisfies $\ulcorner s \urcorner$.
- By (2), every model of S4 satisfies $\ulcorner s \urcorner$.
- By the completeness of S4 for its models, it follows that $\vdash \ulcorner s \urcorner$ in S4.
- By Theorem 2, we conclude that $\vdash s$ in $\mathrm{ICL}^{\Rightarrow}$.

## 7 Conclusion

Starting with a basic logic with a `says` operator, this paper describes simple translations of three logics of access control to S4. The translations lead to decidability results and semantics, and also to comparison of the logics. In particular, the translations enable us to study definitions and axiomatization of the "speaks for" relation.

Going further, one may attempt to carry out a similar programme for some of the diverse logics that appear in the literature. At present, there is no metric to compare these logics against each other, nor a method for integrating more than one logic into a single system. Translation to a standard logic such as S4 seems a promising approach for addressing both of these issues. Of course, first-order and second-order constructs may sometimes be necessary, and more substantial deviations from S4 may arise too—for instance, towards S5, or by the addition of special-purpose operators. Understanding those deviations may be instructive.

Going further, too, our results may be of practical value. They may serve as the basis for theorem provers for logics of access control, with the help of existing algorithms and provers for S4. More speculatively, finite models (of the kind that we obtain from our semantics) may also play a role in a new variant of proof-carrying authentication [6]. A model can serve as evidence that a particular formula is not valid, thus enabling the use of such negative information as an input to authorization decisions. These applications of our results are intriguing; they still require considerable design and experimentation.

**Acknowledgments.** This work was mostly done at Microsoft Research Silicon Valley. The first author was also funded by NSF Grant CNS-0716469. We are grateful to Valeria de Paiva and to Frank Pfenning for discussions on related work.

## References

1. Abadi, M.: Logic in access control. In: Proceedings of the 18th Annual Symposium on Logic in Computer Science (LICS 2003), pp. 228–233 (June 2003)
2. Abadi, M.: Access control in a core calculus of dependency. Electronic Notes in Theoretical Computer Science, Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin 172, 5–31 (April 2007)
3. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems 15(4), 706–734 (1993)

4. Abadi, M., Wobber, T.: A logical account of NGSCB. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 1–12. Springer, Heidelberg (2004)
5. Alechina, N., Mendler, M., de Paiva, V., Ritter, E.: Categorical and kripke semantics for constructive S4 modal logic. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, pp. 292–307. Springer, Heidelberg (2001)
6. Appel, A.W., Felten, E.W.: Proof-carrying authentication. In: Proceedings of the 6th ACM Conference on Computer and Communications Security, pp. 52–62 (November 1999)
7. Bauer, L., Garriss, S., McCune, J.M., Reiter, M.K., Rouse, J., Rutenbar, P.: Device-enabled authorization in the Grey system. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 431–445. Springer, Heidelberg (2005)
8. Bauer, L.: Access Control for the Web via Proof-Carrying Authorization. PhD thesis, Princeton University (November 2003)
9. Bauer, L., Garriss, S., Reiter, M.K.: Distributed proving in access-control systems. In: Proceedings of the 2005 Symposium on Security and Privacy, pp. 81–95 (May 2005)
10. Becker, M.Y., Fournet, C., Gordon, A.D.: Design and semantics of a decentralized authorization language. In: 20th IEEE Computer Security Foundations Symposium, pp. 3–15 (2007)
11. Benton, P.N., Bierman, G.M., de Paiva, V.C.V.: Computational types from a logical perspective. Journal of Functional Programming 8(2), 177–193 (1998)
12. Cardelli, L.: Type systems. In: Tucker, A.B. (ed.) The Computer Science and Engineering Handbook, ch. 103, pp. 2208–2236. CRC Press, Boca Raton, FL (1997)
13. Cederquist, J.G., Corin, R., Dekker, M.A.C., Etalle, S., den Hartog, J.I., Lenzini, G.: Audit-based compliance control. Int. J. Inf. Secur. 6(2), 133–151 (2007)
14. Cirillo, A., Jagadeesan, R., Pitcher, C., Riely, J.: Do as I SaY! programmatic access control with explicit identities. In: 20th IEEE Computer Security Foundations Symposium, pp. 16–30 (July 2007)
15. Curry, H.B.: The elimination theorem when modality is present. Journal of Symbolic Logic 17(4), 249–265 (1952)
16. DeTreville, J.: Binder, a logic-based security language. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 105–113 (May 2002)
17. Fairtlough, M., Mendler, M.V.: Propositional lax logic. Information and Computation 137(1), 1–33 (1997)
18. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization in distributed systems. In: 20th IEEE Computer Security Foundations Symposium, pp. 31–45 (2007)
19. Garg, D., Bauer, L., Bowers, K.D., Pfenning, F., Reiter, M.K.: A linear logic of authorization and knowledge. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 297–312. Springer, Heidelberg (2006)
20. Garg, D., Pfenning, F.: Non-interference in constructive authorization logic. In: Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW 19), pp. 283–296 (2006)
21. Girard, J.-Y.: Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur. Thèse de doctorat d'état, Université Paris VII (June 1972)
22. Gödel, K.: Eine Interpretation des intuitionistischen Aussagenkalkuls. Ergebnisse eines mathematischen Kolloquiums 8, 39–40 (1933)
23. Howe, J.M.: Proof search in lax logic. Mathematical Structures in Computer Science 11(4), 573–588 (2001)

24. Hughes, G.E., Cresswell, M.J.: An Introduction to Modal Logic. Methuen Inc., New York (1968)
25. Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM Journal on Computing 6(3), 467–480 (1977)
26. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in distributed systems: Theory and practice. ACM Transactions on Computer Systems 10(4), 265–310 (1992)
27. Lampson, B.W.: Protection. In: Proceedings of the 5th Princeton Conference on Information Sciences and Systems, pp. 437–443 (1971)
28. Lampson, B.W.: Computer security in the real world. IEEE Computer 37(6), 37–46 (2004)
29. Lesniewski-Laas, C., Ford, B., Strauss, J., Kaashoek, M.F., Morris, R.: Alpaca: Extensible authorization for distributed services. In: 14th ACM Conference on Computer and Communications Security, pp. 432–444 (2007)
30. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Transactions on Information and System Security 6(1), 128–171 (2003)
31. Li, N., Mitchell, J.C.: Datalog with constraints: A foundation for trust management languages. In: Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages, pp. 58–73 (2003)
32. Li, N., Mitchell, J.C., Winsborough, W.H.: Beyond proof-of-compliance: security analysis in trust management. J. ACM 52(3), 474–514 (2005)
33. Moggi, E.: Notions of computation and monads. Information and Computation 93(1), 55–92 (1991)
34. Pfenning, F., Davies, R.: A judgmental reconstruction of modal logic. Mathematical Structures in Computer Science 11, 511–540 (2001)
35. Wobber, E., Abadi, M., Burrows, M., Lampson, B.: Authentication in the Taos operating system. ACM Transactions on Computer Systems 12(1), 3–32 (1994)

# Coalgebraic Logic and Synthesis of Mealy Machines

M.M. Bonsangue[1,2], Jan Rutten[2,3,⋆], and Alexandra Silva[2,⋆⋆]

[1] LIACS - Leiden University
[2] Centrum voor Wiskunde en Informatica (CWI)
[3] Vrije Universiteit Amsterdam (VUA)

**Abstract.** We present a novel coalgebraic logic for deterministic Mealy machines that is sound, complete and expressive w.r.t. bisimulation. Every finite Mealy machine corresponds to a finite formula in the language. For the converse, we give a compositional synthesis algorithm which transforms every formula into a finite Mealy machine whose behaviour is exactly the set of causal functions satisfying the formula.

## 1 Introduction

A Mealy machine $(S, f)$ consists of a set $S$ of states and a transition function $f{:}S \to (B \times S)^A$ assigning to each state $s \in S$ and input symbol $a \in A$ a pair $\langle b, s' \rangle$, consisting of an output symbol $b \in B$ and a next state $s' \in S$. Typically one writes

$$f(s)(a) = \langle b, s' \rangle \iff s \xrightarrow{a|b} s'$$

One of the most important applications of Mealy machines is their use in the specification of sequential digital circuits. Taking binary inputs and outputs, there is a well-known correspondence between such binary Mealy machines, on the one hand, and sequential digital circuits built out of logical gates and some kind of memory elements, on the other. In present day text books on logic design [11] — on the construction of sequential digital circuits — Mealy machines are still the most important mathematically exact means for the specification of the intended behaviour of circuits.

There does not seem to exist, however, a generally accepted way of formally specifying Mealy machines themselves. The only formal approach we are aware of is the general model for categories with feedback in [6], which can be instantiated to Mealy machines. However, Mealy machines are typically "defined" in a natural language such as English. This obviously leads to ambiguities, inconsistencies and plain errors [4].

In this paper, we propose a simple but adequate and expressive logical language for the specification of Mealy machines. Here adequate means that the logical equivalence corresponds to a natural behavioural equivalence on Mealy machines, whereas expressive means that every finite Mealy machine can be represented by a (finite) formula. Finally, simple means that the logic contains precisely what is needed to obtain this

⋆ Partially supported by EU project IST-33826 CREDO (`http://credo.cwi.nl`).
⋆⋆ Partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 231–245, 2008.

goal, and nothing more. The latter point is an important distinguishing factor in comparison with some already existing formalisms in the literature, discussed below.

Briefly stated, our approach is coalgebraic. Mealy machines are a basic and well-understood family of coalgebras, of the functor $M(S) = (B \times S)^A$. The crucial coalgebraic insight is that the properties of Mealy machines (coalgebras) are fully dictated by (the shape of) their defining functor $M$. This has led, for instance, to the identification of a *final* Mealy machine, in [14], as the set of all causal stream functions from $A^\omega$ to $B^\omega$.

Following coalgebraic methodology, we apply general insights from coalgebraic modal logic (see e.g. [12,2]) and define a logic whose basic operations derive directly from the functor $M$. The equivalence induced by the logic coincides with that induced by the functor $M$. Further, the logic comes equipped with a proof system for reasoning about universal validity that we prove sound and complete.

All finite Mealy machines can be specified as a formula in the logic. The main technical contribution of the paper is the construction, for every formula in the logic, of a finite Mealy Machine whose behaviour is exactly characterised by the formula.

### 1.1 Related Work

Automata synthesis is a popular and very active research area [13,8,4,15,5]. Most of the work done on synthesis has as main goal to find a proper and sufficiently expressive type of automata to encode a specific type of logic (such as LTL [15] or $\mu$-calculus [8]).

Technically, the synthesis from a $\mu$-calculus formula $\varphi$ consists in translating $\varphi$ into an alternating automaton $\mathcal{A}_\varphi$, reducing $\mathcal{A}_\varphi$ into a non-deterministic automaton which is then checked for non-emptiness [8]. The same process has been recently generalized to $F$-coalgebras in [10]. In this paper, we use a different approach. We construct a deterministic Mealy machine for a formula directly, by considering the formula as a state of the automaton containing enough information about its successors.

Although Mealy machines are in one-to-one correspondence with sequential digital circuits, not much work has been done for their specification and synthesis. In [6], an algebra for systems with feedback is given, but no synthesis is presented. In [15], a compositional algorithm for synthesizing Generalized Mealy machines (GMMs) from LTL formulae is presented. GMMs are a special class of non-deterministic Mealy machines that have the acceptance condition of generalized Büchi automata. In this paper, we will remain in the world of deterministic Mealy machines, the one corresponding to sequential digital circuits. Moreover, our work exploits the structure of the Mealy machine and, therefore, the resulting logic is simpler than LTL (but expressive enough for deterministic Mealy machines).

The logic most similar to ours is the one presented in [4]. There a logic for formal specification of hardware protocols is presented, and an algorithm for the synthesis of a Mealy machine is given. Their logic corresponds to the conjunctive fragment of LTL. Their synthesis process is standard: first a non-deterministic Büchi automaton is synthesized, secondly a powerset construction is used to make the automaton deterministic and, finally, the propositions on the states are used to determine the inputs and outputs for each state of the Mealy machine. Because of our coalgebraic approach, the equivalence induced by our logic is canonical, and the logic comes with a proof system that

is sound and complete. Further, our synthesis process remains within standard Mealy machines and the behaviour of the synthesized automata is exactly characterized by the original formula. Apart from [14,5], where synthesis for a special case of 2-adic arithmetic is treated, we did not find any other work on the direct synthesis of deterministic Mealy machines. From these papers we inherit the basic coalgebraic approach, that we use here to derive our expressive logical specification language for Mealy machines.

In summary, the work presented in this paper distinguishes itself from all existing work as follows. Our specification logic is derived directly from the functor, which results in a very simple and consistent logic that has exactly the operators needed to fully specify Mealy machines. Note that being simple does not mean this logic has less expressive power than others. In the context of applications (such as circuits logic design), this logic has all the relevant operators.

## 2 Mealy Machines

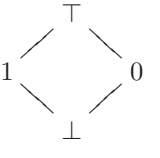
We give the basic definitions on Mealy machines and introduce the notions of simulation and bisimulation.

First we recall the following definition. A (bounded) meet-semilattice is a set $B$ equipped with a binary operation $\wedge_B$ and a constant $\top_B \in B$, such that $\wedge_B$ is commutative, associative and idempotent. The element $\top_B$ is neutral w.r.t. $\wedge_B$. As usual, $\wedge_B$ gives rise to a partial ordering $\leq_B$ on the elements of $B$:

$$b_1 \leq_B b_2 \Leftrightarrow b_1 \wedge_B b_2 = b_1$$

Every set $S$ can be transformed into a meet-semilattice by taking the collection $\mathcal{P}S$ of all subsets of $S$ with intersection as meet. We use semilattices to represent data structures equipped with an information order: $b_1 \leq_B b_2$ means that $b_1$ is more *concrete* than $b_2$.

Our running examples will all use the four element meet-semilattice:

$\mathcal{B} =$ 


Here, the $\top$ element is used for *abstracting* (under-specification) from any concrete data; the $\bot$ element denotes *inconsistency* (over-specification) of information; and the elements $0$ and $1$ are concrete output values.

Now let $A$ be a finite set and let $B$ be a (possibly infinite) meet-semilattice. A Mealy machine $(S, f)$ with inputs in $A$ and outputs in $B$ consists of a set of states $S$ together with a function

$$f{:}S \to (B \times S)^A$$

For a given state $s \in S$ and an input $a \in A$, the function $f$ returns a pair $f(s)(a) = \langle b, s' \rangle$, consisting of an output value $b \in B$ and a state $s' \in S$. Typically we will write

$$f(s)(a) = \langle s[a],\ s_a \rangle$$

and call $s[a]$ the *(initial) output on input* $a$ and $s_a$ the *next state on input* $a$. We shall also use the following convention for the representation of Mealy machines:

$$f(s)(a) = \langle b, s' \rangle \iff s \xrightarrow{a|b} s'$$

In coalgebraic terms, a Mealy machine $(S, f)$ is a coalgebra of the functor $M{:}Set \to Set$ defined, for any set $X$, as $M(X) = (B \times X)^A$.

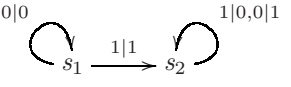
A *homomorphism* from a Mealy machine $(S, f)$ to a Mealy machine $(T, g)$ is a function $h{:}\, S \to T$ preserving initial outputs and next states:

$$h(s)[a] = s[a] \quad \text{and} \quad h(s_a) = h(s)_a$$

(which is equivalent to the condition that $g \circ h = M(h) \circ f$, where the functor $M$ is defined on functions as usual).

Machines where $A$ is the two-element set $\{0, 1\}$ and $B$ is the meet-semilattice $\mathcal{B}$ are called *binary*, and they are *fully specified* if only 0 or 1 are used as output elements (and never $\bot$ or $\top$).

For an example, consider the following binary Mealy machine with $S = \{s_1, s_2\}$ and the transition function defined by the following picture.




This machine computes the two's complement of a given binary number.

Next we define the notion of simulation, which can be used to obtain abstraction, and bisimulation, which plays an important role in the minimization of Mealy machines.

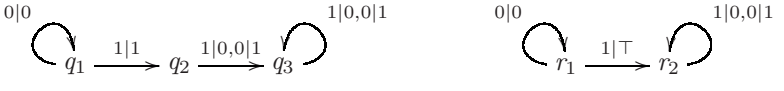
**Definition 1 ((Bi)simulation for Mealy).** *Let $(S, f)$ and $(T, g)$ be two Mealy machines. We call a relation $R \subseteq S \times T$ a* simulation *if for all $(s, t) \in S \times T$ and $a \in A$*

$$s\, R\, t \;\Rightarrow\; (\, s[a] \leq_B t[a] \;\text{ and }\; s_a\, R\, t_a\,)$$

*We call $R$ a* bisimulation *relation if both $R$ and its (relational) inverse $R^{-1}$ are simulations.*

We write $s \lesssim t$ (resp. $s \sim t$) whenever there exists a simulation relation (bisimulation relation) containing $(s, t)$; and we call $\lesssim$ and $\sim$ the similarity and bisimilarity relations. By definition, we have $\lesssim \cap \lesssim^{-1} = \sim$.

As an example, consider the following two binary Mealy machines:




Observe that $q_3$ and $q_2$ are bisimilar, since $R = \{(q_2, q_3), (q_3, q_3)\}$ is a bisimulation. A minimal machine is obtained by identifying all bisimilar states, yielding our two's complement machine above.

Now, note that the rightmost machine can be simulated by any fully specified binary machine substituting either 0 or 1 as output for the abstract $\top$ value in the transition from $r_1$ to $r_2$. For example, considering the above two's complement machine, we have $s_1 \lesssim r_1$ because $S = \{(s_1, r_1), (s_2, r_2)\}$ is a simulation relation.

Next we recall the construction of a *final* Mealy machine with inputs in $A$ and outputs in $B$. Finality plays an important role in minimization as well as in the proof system (in Section 3).

Let $A^\omega = \{\, \sigma \mid \sigma{:}\{0, 1, 2, \ldots\} \to A\}$, the set of all infinite *streams* over $A$. For $a \in A$ and $\sigma \in A^\omega$, we define:

$$a{:}\sigma = (a, \sigma(0), \sigma(1), \sigma(2), \ldots) \quad \sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$$

We call a function $f{:}A^\omega \to B^\omega$ *causal* if for all $\sigma \in A^\omega$ and $n \geq 0$, the $n$th output value $f(\sigma)(n)$ depends only on the first $n$ input values $(\sigma(0), \ldots, \sigma(n-1))$. Let

$$\Gamma = \{\, f{:}A^\omega \to B^\omega \mid f \text{ is causal}\}$$

The set $\Gamma$ can be turned into a Mealy machine $(\Gamma, \gamma)$ by defining $\gamma(f)(a) = \langle f[a], f_a\rangle$ as follows:

$$f[a] = f(a{:}\sigma)(0) \text{ (where } \sigma \text{ is arbitrary)} \qquad f_a(\sigma) = (f(a{:}\sigma))'$$

(Note that by causality the value of $f(a{:}\sigma)(0)$ depends only on $a$.) The following theorem is a minor variation on [14, Prop.2.3 and Corr.2.3].

**Theorem 2 (Finality of $(\Gamma, \gamma)$).** *For every Mealy machine $(S, f)$ there exists a unique homomorphism $h{:}S \to \Gamma$. It satisfies, for all $s, s' \in S$:*

$$s \lesssim s' \iff h(s) \lesssim h(s')$$

*where on $\Gamma$, similarity coincides with the elementwise ordering induced by $B$:*

$$f \lesssim g \iff \forall \sigma \in A^\omega \, \forall n \geq 0 \,.\, f(\sigma)(n) \leq_B g(\sigma)(n)$$

Since the identity function is always a homomorphism, bisimilarity is equality on $\Gamma$. As a consequence, the image $h(S)$ of a Mealy machine $S$ is in fact its minimisation with respect to bisimilarity.

## 3 Mealy Logic

We present a logic for Mealy machines and define its semantics and a satisfaction relation.

**Definition 3 (Mealy formulae).** *Let $A$ be a set of input actions and let $B$ be a meet-semilattice of output actions. Furthermore, let $X$ be a set of (recursion or) fixed point variables. The set $L$ of* Mealy formulae *is given by the following BNF syntax. For $a \in A$, $b \in B$, and $x \in X$:*

$$\phi ::= tt \mid x \mid a(\phi) \mid a{\downarrow}b \mid \phi \wedge \phi \mid \nu x.\psi$$

*where* $\psi \in L_g$*, the set of* guarded *formulae, which is given by:*

$$\psi ::= tt \mid a(\phi) \mid a{\downarrow}b \mid \psi \wedge \psi \mid \nu x.\psi$$

We call $a(\phi)$ a *transition formula* and $a{\downarrow}b$ an *output formula*. Note that our language does not include disjunction or negation. As we will discuss in 3.2, this is a natural restriction and does not decrease the expressiveness of our logic. Moreover, in the same section we will also point out the reasons for only having one type of fixed point operator. Also note that for every unguarded Mealy formula there exists an equivalent guarded formula, as a consequence of [9, Theorem 2.1].

The modal fragment of our logic (*i.e*, the set of closed formulae without the $\nu$ operator) is a special case of the coalgebraic logic obtained by a Stone-type duality [1,2].

In what follows, we shall concentrate on the set $L_g^c$ of formulae that are both guarded and *closed*, that is, without free occurrences of fixed point variables $x$. We turn the set $L_g^c$ into a Mealy machine (coalgebra)

$$\lambda \colon L_g^c \to (B \times L_g^c)^A$$

by defining $\lambda$ as follows. For $a \in A$ and $\phi \in L_g^c$, we write $\lambda(\phi) = \langle \phi[a],\, \phi_a \rangle$ and we define $\phi[a]$ and $\phi_a$ by

$$\begin{array}{lll}
tt[a] & = \top_B & tt_a = tt \\
a(\phi)[a'] & = \top_B \ \text{(for any } a' \in A) & (a(\phi))_{a'} = \begin{cases} \phi & \text{if } a = a' \\ tt & \text{otherwise} \end{cases} \\
(a{\downarrow}b)[a'] & = \begin{cases} b & \text{if } a = a' \\ \top_B & \text{otherwise} \end{cases} & (a{\downarrow}b)_{a'} = tt \ \text{(for any } a' \in A) \\
(\phi_1 \wedge \phi_2)[a] & = \phi_1[a] \wedge_B \phi_2[a] & (\phi_1 \wedge \phi_2)_a = (\phi_1)_a \wedge (\phi_2)_a \\
(\nu x.\psi)[a] & = (\psi[\nu x.\psi/x])[a] & (\nu x.\psi)_a = (\psi[\nu x.\psi/x])_a
\end{array}$$

Here, $\psi[\nu x.\psi/x]$ denotes syntactic substitution, replacing in $\psi$ every free occurrence of $x$ by $\nu x.\psi$.

The above definition uses induction on the following complexity measure, which is based on the number of nested unguarded occurrences of $\nu$-formulae:

$$\begin{array}{ll}
N(tt) & = N(a{\downarrow}b) = N(a(\phi)) = 0 \\
N(\phi_1 \wedge \phi_2) & = max\{N(\phi_1),\, N(\phi_2)\} + 1 \\
N(\nu x.\psi) & = 1 + N(\psi)
\end{array}$$

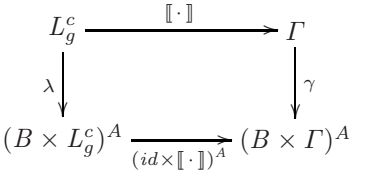
In order to see that the definition of $\phi[a]$ and $\phi_a$ is well-formed, note that in the case of $\nu x.\psi$, we have:

$$N(\psi) = N(\psi[\nu x.\psi/x])$$

This can easily be proved by (standard) induction on the syntactic structure of $\psi$, since $\psi$ is guarded (in $x$).

Note that the (sub)machine generated by a formula $\phi \in L_g^c$ by repeatedly applying $\lambda$ will in general be infinite. In Section 4, an algorithm to produce a finite Mealy machine from a formula $\phi \in L_g^c$ will be presented.

Having a Mealy coalgebra structure on $L_g^c$ has two advantages. First, it provides us, by finality of $\Gamma$, directly with a natural semantics because of the existence of a (unique) homomorphism:




$$[\![\phi]\!][a] = \phi[a] \quad \text{and} \quad [\![\phi]\!]_a = [\![\phi_a]\!]$$

It assigns to every formula $\phi$ a causal stream function $[\![\phi]\!]\colon A^\omega \to B^\omega$.

The second advantage of the Mealy structure on $L_g^c$ is that it lets us use the notion of Mealy simulation to define when a state $s \in S$ of a Mealy machine $(S, f)$ satisfies a formula $\phi \in L_g^c$, by defining:

$$s \models \phi \quad \Leftrightarrow \quad s \lesssim \phi$$

For brevity, we say that a Mealy machine $(S, f)$ satisfies a formula $\phi$ if some state in $S$ satisfies $\phi$.

Proving satisfaction then amounts to the construction of a simulation relation $R \subseteq S \times L_g^c$ between $(S, f)$ and $(L, \lambda)$ such that $sR\phi$.

The above definition is equivalent to the following, more classical definition of satisfaction. For every valuation $\eta\colon Var \to \mathcal{P}(S)$, we define a satisfaction relation $\models_\eta$, by induction, as follows:

$$\begin{array}{lll}
s \models_\eta tt & & \text{for all } s \\
s \models_\eta a(\phi) & \textit{iff} & s_a \models_\eta \phi \\
s \models_\eta a{\downarrow}b & \textit{iff} & s[a] \leq_B b \\
s \models_\eta \phi_1 \wedge \phi_2 & \textit{iff} & s \models_\eta \phi_1 \textit{ and } s \models_\eta \phi_2 \\
s \models_\eta x & \textit{iff} & s \in \eta(x) \\
s \models_\eta \nu v.\psi & \textit{iff} & \exists T \subseteq S. s \in T \textit{ and } \forall t \in T. t \models_{\eta[T/v]} \psi
\end{array}$$

Here, $\eta[T/v]$ denotes the valuation such that, for every $x \in Var$, with $x \neq v$, returns $\eta(x)$ and for $x = v$ returns $T$.

Note that in this definition single occurrences of $x \in X$ are allowed. It can be shown, by a fairly straightforward and not very instructive proof, that the two definitions of satisfaction are equivalent. More precisely, if $\emptyset$ denotes the everywhere empty valuation, we have:

$$s \lesssim \phi \quad \Leftrightarrow \quad s \models_\emptyset \phi$$

for every $\phi \in L_g^c$. We omit the proof and will work in what follows with the definition of satisfaction as simulation.

The following theorem shows that our logic is sufficiently expressive to characterise bisimilarity.

**Theorem 4**

*(1) For all states $s, s'$ of a Mealy machine $(S, f)$,*

$$s \sim s' \quad \textit{iff} \quad \forall \phi \in L_g^c \,.\, s \models \phi \Leftrightarrow s' \models \phi$$

*(2) If $S$ is* finite *then there exists for any $s \in S$ a formula $\phi_s \in L_g^c$ such that*

$$\forall s' \in S \,.\, s \sim s' \quad \textit{iff} \quad s' \models \phi_s$$

*Proof.* (1) Because $s \sim s'$ implies $s \lesssim s'$ and $s' \lesssim s$ we have, for any $\phi \in L_g^c$,

$$s \models \phi \iff s \lesssim \phi \iff s' \lesssim \phi \iff s' \models \phi$$

For the converse, note, for any $s \in S$, $a \in A$, and $\phi \in L_g^c$, that $s \models a{\downarrow}s[a]$ and

$$s_a \models \phi \iff s_a \lesssim \phi \iff s \lesssim a(\phi) \iff s \models a(\phi)$$

As a consequence, the following relation

$$R = \left\{ \langle s, s' \rangle \in S \times S \mid \forall \phi \in L_g^c \,.\, s \models \phi \Leftrightarrow s' \models \phi \right\}$$

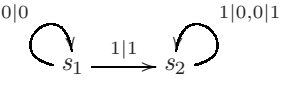
and its inverse $R^{-1}$ are simulation relations on $S$. Thus $R$ is a bisimulation.

(2) It is sufficient to construct for a given $s \in S$ a formula $\phi_s$ with $s \sim \phi_s$. To this end, we associate with every state $s \in S$ a variable $x_s \in X$ and a formula $\phi_s = \nu x_s \,.\, \psi_s$ defined by

$$\psi_s = \bigwedge_{a \in A} a(x_{s_a}) \wedge a{\downarrow}s[a]$$

Syntactically replacing free occurrences of $x_{s'}$ by $\phi_{s'}$ in $\phi_s$ $(s \neq s')$ will ensure that all $\phi_s$ will be in $L_g^c$. By construction, $s \sim \phi_s$. □

Let us illustrate the last construction above. Recall the two's complement Mealy machine presented before:




We define $\phi_1 = \nu x_1 \,.\, \psi_1$ and $\phi_2 = \nu x_2 \,.\, \psi_2$ by

$$\psi_1 = 0(x_1) \wedge 0{\downarrow}0 \wedge 1(x_2) \wedge 1{\downarrow}1 \qquad \psi_2 = 0(x_2) \wedge 0{\downarrow}1 \wedge 1(x_2) \wedge 1{\downarrow}0$$

Substituting $\phi_2$ for $x_2$ in $\psi_1$ then yields

$$\phi_1 = \nu x_1 \,.\, 0(x_1) \wedge 0{\downarrow}0 \wedge 1(\phi_2) \wedge 1{\downarrow}1 \qquad \phi_2 = \nu x_2 \,.\, 0(x_2) \wedge 0{\downarrow}1 \wedge 1(x_2) \wedge 1{\downarrow}0$$

By construction we have $s_1 \sim \phi_1$ and $s_2 \sim \phi_2$.

### 3.1 Proof System

We now introduce a proof system for assertions of the form $\phi_1 \leq \phi_2$, where $\leq$ is the relation of logical entailment between the closed formulae $\phi_1$ and $\phi_2$.

$$(refl)\quad \phi \leq \phi \qquad (top)\quad \phi \leq tt$$
$$(\wedge - e1)\quad \phi_1 \wedge \phi_2 \leq \phi_1 \qquad (\wedge - e2)\quad \phi_1 \wedge \phi_2 \leq \phi_2$$
$$(trans)\quad \frac{\phi_1 \leq \phi_2 \quad \phi_2 \leq \phi_3}{\phi_1 \leq \phi_3} \qquad (\wedge - i)\quad \frac{\phi \leq \phi_1 \quad \phi \leq \phi_2}{\phi \leq \phi_1 \wedge \phi_2}$$

$$(a\downarrow - \top)\ tt \leq a\downarrow \top_B \qquad (a() - \top)\ tt \leq a(tt)$$
$$(a\downarrow - \wedge)\ a\downarrow b_1 \wedge a\downarrow b_2 \leq a\downarrow(b_1 \wedge_B b_2) \qquad (a() - \wedge)\ a(\phi_1) \wedge a(\phi_2) \leq a(\phi_1 \wedge \phi_2)$$
$$(a\downarrow - \leq)\ \frac{b_1 \leq_B b_2}{a\downarrow b_1 \leq a\downarrow b_2} \qquad (a() - \leq)\ \frac{\phi_1 \leq \phi_2}{a(\phi_1) \leq a(\phi_2)}$$

$$(\nu - i)\quad \frac{\phi \leq \psi[\phi/x]}{\phi \leq \nu x.\psi} \qquad (\nu - e)\quad \frac{\psi[\nu x.\psi/x] \leq \phi}{\nu x.\psi \leq \phi}$$

The first group of axioms and rules gives to the set of formulae the structure of a meet-semilattice. Further, there are axioms and rules for the two modal operators, showing the interactions between the transition and output formulae with the meet-semilattice structure. Finally, the last two rules $(\nu - i)$ and $(\nu - e)$ can be explained as stating that the term $\nu x.\psi$ is the greatest postfixed point, when viewing the formula $\psi$ as a (monotone) map on formulae.

We write $\vdash \phi_1 \leq \phi_2$ to indicate that the assertion $\phi_1 \leq \phi_2$ is derivable from the above axioms and rules. Note that the converse of $(a\downarrow - \wedge)$ is derivable from $(a\downarrow - \leq)$ and $(\wedge - i)$. Similarly, also the converses of $(a\downarrow - \top)$, $(a() - \top)$ and $(a() - \wedge)$ are derivable.

**Theorem 5 (Soundness).** *The above proof system is sound, that is, for closed formulae $\phi_1$ and $\phi_2$, $\vdash \phi_1 \leq \phi_2$ implies that for all Mealy machines $(S, f)$ and $s \in S$ if $s \models \phi_1$ then $s \models \phi_2$.*

*Proof.* By induction on the length of proofs. □

Next we turn to the completeness for the *modal* fragment $L_m$ of our Mealy logic $L$, where a modal formula is a formula with neither fixed point operators nor variables. Note that the (Lindenbaum algebra of) $L_m$ is a meet-semilattice.

Let $\Theta$ be the set of all filters of (the Lindenbaum algebra of) $L_m$, where a *filter* of a meet-semilattice is a non-empty upper closed subset $\mathcal{F}$ such that if $a, b \in \mathcal{F}$ then also $a \wedge b \in \mathcal{F}$. The set $\Theta$ can be turned into a Mealy machine $(\Theta, \theta)$ by defining, for $F \in \Theta$ and $a \in A$, $\theta(F)(a) = \langle F[a], F_a \rangle$, where

$$F[a] = \bigwedge \{b | a\downarrow b \in F\} \qquad F_a = \{\phi | a(\phi) \in F\}\,.$$

Note that in order for $F[a]$ to be well defined we assume $B$ to be a *finite* meet-semilattice. In case $B$ is infinite, we would need $B$ to be a complete meet-semilattice.

**Theorem 6.** *For every Mealy machine $(S, f)$ there exists a unique homomorphism $k_S : S \to \Theta$. In particular, the homomorphism $k_\Gamma : \Gamma \to \Theta$ is an isomorphism.*

As a consequence of Theorem 4, the isomorphism $k_\Gamma : \Gamma \to \Theta$ is also an order isomorphism, where the order on $\Theta$ is subset inclusion. The logical significance of the

above result is that a finitary logic with only finite conjunctions suffices to completely describe all Mealy machines up to bisimilarity. In fact the modal fragment of our logic is a special case of coalgebraic logic obtained by a Stone-type duality [1,2].

Theorem 6 together with the next lemma gives a logical interpretation of the final coalgebra: its elements correspond to canonical models (in the logical sense) of the Mealy logic.

**Lemma 7.** *For every modal formula $\phi$ and filter $F \in \Theta$, $F \models \phi$ if and only if $\phi \in F$.*

*Proof.* By induction on the structure of $\phi$, using the fact that $F$ is a filter and the above definition of $\theta{:}\Theta \to (B \times \Theta)^A$. □

We can finally prove the completeness of the modal fragment of our Mealy logic.

**Theorem 8 (Completeness).** *For modal formulae $\phi_1$ and $\phi_2$, if $s \models \phi_1$ implies $s \models \phi_2$ for all Mealy machines $(S, f)$ and $s \in S$, then $\vdash \phi_1 \leq \phi_2$.*

*Proof.* Assume $\nvdash \phi_1 \leq \phi_2$. It is enough to find a state $s$ in a Mealy machine $(S, f)$ such that $s \models \phi_1$ but $s \not\models \phi_2$. Define $F_{\phi_1} = \{\psi \mid \phi_1 \leq \psi\}$. It is not very difficult to verify that $F_{\phi_1}$ is a filter, hence it is an element of $\Theta$. Clearly, $\phi_1 \in F_{\phi_1}$ but, by our assumption $\phi_2 \notin F_{\phi_1}$. We can now conclude by applying Lemma 7. □

### 3.2 Adding Negation

The logic we have considered so far contains no negation. Extending the logic with negated formulae is not problematic as long as we consider Mealy machines with outputs in a Boolean algebra $B$ (like the two-element set). In this case, we can still turn the set of (possibly negated) formulae into a Mealy coalgebra by extending our definition of $\lambda$ at the beginning of section 3 with

$$(\neg\phi)[a] = \neg_B(\phi[a]) \qquad (\neg\phi)_a = \neg(\phi)_a\,.$$

It is easy to see that according to this definition negation distributes up to bisimulation over conjunction (de Morgan law), and over the modal operators (a sign that the machine is indeed deterministic). Further, negation is classical, meaning that $\neg(\neg\phi) \sim \phi$. Clearly, disjunctions and $\mu$-recursive formulae can be defined as derived operators.

From the logical point of view, this means that the Lindenbaum algebra of the resulting logic with negation is the free Boolean algebra over the meet-semilattice of the Mealy logic we considered here. In this case one can apply the isomorphism $\mathit{UFilt}(B(L)) \cong \mathit{Filt}(L)$ to obtain analogous soundness and completeness results as above, where $L$ is a meet-semilattice, $B(L)$ is the free Boolean algebra over $L$ and $\mathit{UFilt}(B(L))$ is the set of ultrafilters of $B(L)$.

## 4 Synthesis

We will now describe the synthesis process that produces a Mealy machine from an arbitrary (closed and guarded) Mealy formula[1]. Each state of the resulting Mealy machine will be a formula constructed in such a way that if $s$ is the state corresponding to

[1] The source code in HASKELL can be downloaded from `www.cwi.nl/~ams/mealy`

a formula $\phi$, then $s \sim \phi$. This implies that the semantics of $s$ is exactly the set of causal functions satisfying $\phi$.

### 4.1 Formulae Normalization

We have seen that the first group of six axioms and rules of our proof system gives to the set of formulae the structure of a meet-semilattice. In order to guarantee the termination of the synthesis process we will need to identify formulae that are provably equivalent using only these axioms and rules. For instance, the formulae

$$a(tt) \wedge a{\downarrow}b \wedge tt \wedge a{\downarrow}b \text{ and } a(tt) \wedge a{\downarrow}b$$

are equivalent.

To *normalize* a formula $\phi$, we need to eliminate any redundancy present in the formula: in a conjunction, $tt$ can be eliminated and, by idempotency, the conjunction of two syntactically equivalent formulae can be simplified.

The function $norm$ encodes this procedure. We define it by induction on the formula structure as follows:

$$\begin{array}{ll} norm(tt) & = tt \\ norm(a(\phi)) & = a(norm(\phi)) \\ norm(a{\downarrow}b) & = a{\downarrow}b \\ norm(\phi_1 \wedge \phi_2) & = conj(rem(\mathit{flatten}(norm(\phi_1) \wedge norm(\phi_2)))) \\ norm(\nu x.\phi) & = \nu x.(norm(\phi))\,. \end{array}$$

Here, $conj$ takes a list of formulae $[\phi_1, \ldots, \phi_n]$ and returns the formula $\phi_1 \wedge \ldots \wedge \phi_n$ ($conj$ applied to the empty list yields $tt$), $rem$ removes duplicates in a list and $\mathit{flatten}$ takes a formula $\phi$ and produces a list of formulae by omitting brackets and replacing $\wedge$-symbols by commas:

$$\begin{array}{ll} \mathit{flatten}(\phi_1 \wedge \phi_2) & = \mathit{flatten}(\phi_1) \cdot \mathit{flatten}(\phi_2) \\ \mathit{flatten}(tt) & = [] \\ \mathit{flatten}(\phi) & = [\phi],\ \phi \in \{a{\downarrow}b, a(\phi_1), \nu x.\phi_1\} \end{array}$$

In this definition, $\cdot$ denotes list concatenation and $[\phi]$ the singleton list containing $\phi$. Note that an occurrence of $tt$ in a conjunction is eliminated because $\mathit{flatten}(tt) = []$.

For example, the normalization of the two formulae above will result in the same formula – $a(tt) \wedge a{\downarrow}b$.

Note that $norm$ still distinguishes the formulae $\phi_1 \wedge \phi_2$ and $\phi_2 \wedge \phi_1$. For simplifying the presentation of the normalization algorithm, we decided not to identify these formulae, since this will not influence termination. However, in the implementation, in order to reduce the number of states, those formulae are identified. In the examples below this situation will never occur.

### 4.2 Synthesis

We first describe what happens in a single step of the synthesis process.

The function $\delta$, which does *one-step synthesis* for a single formula, takes a formula $\phi \in L_g^c$ and produces a partial Mealy machine. Below, $\delta$ will be used in the function $\Delta$, which synthesises the total Mealy machine.

The function $\delta$ is defined, by induction on the complexity measure $N$ defined in Section 3, as follows:

$$\begin{array}{ll}
\delta(tt)(a) & = \langle \top_B, tt \rangle \\
\delta(a'(\phi))(a) & = \begin{cases} \langle \top_B, norm(\phi) \rangle & a = a' \\ \langle \top_B, tt \rangle & otherwise \end{cases} \\
\delta(a' \downarrow b)(a) & = \begin{cases} \langle b, tt \rangle & a = a' \\ \langle \top_B, tt \rangle & otherwise \end{cases} \\
\delta(\phi_1 \wedge \phi_2)(a) & = \delta(\phi_1)(a) \sqcap \delta(\phi_2)(a) \\
\delta(\nu x.\phi)(a) & = \langle b, norm(\phi') \rangle \ \textbf{where} \ \langle b, \phi' \rangle = \delta(\phi[\nu x.\phi / x])(a)
\end{array}$$

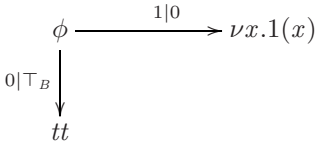
where $\sqcap$ is defined as: $\langle b_1, \phi_1 \rangle \sqcap \langle b_2, \phi_2 \rangle = \langle b_1 \wedge_B b_2, norm(\phi_1 \wedge \phi_2) \rangle$.

Note that this function is very similar to the function $\lambda$ presented in Section 3. In fact, the difference is the normalization that is now being applied to the formulae so that a finite machine will be produced.

As an example, consider the formula $\phi = 1 \downarrow 0 \wedge (\nu x.1(x))$, specifying a binary Mealy machine. We can easily compute that $\delta(\phi)(0) = \langle \top_B, tt \rangle$ and

$$\begin{array}{rl}
\delta(\phi)(1) & = \delta(1 \downarrow 0)(1) \sqcap \delta(\nu x.1(x))(1) \\
& = \langle 0, tt \rangle \sqcap \langle \top_B, \nu x.1(x) \rangle \\
& = \langle 0, \nu x.1(x) \rangle
\end{array}$$

So, $\delta(\phi)$ is a (partial) finite function represented by the following diagram.




To compute the entire Mealy machine that satisfies $\phi$, we need to apply $\delta$ to the new states generated at each step repeatedly until all states in the automata have their transitions/outputs fully defined.

We implement this procedure with the auxiliary function $D$. The arguments of this function are two sets of states: $sts \subseteq L_g^c$, the states that still need to be processed and $vis \subseteq L_g^c$, the states that already have been visited (synthesized). For each $\phi \in sts$, $D$ computes $\delta(\phi)$ and produces an intermediate transition function (possibly partial) by taking the union of all those $\delta(\phi)$. Then, it collects all new states appearing in this step and recursively computes the transition function for those.

$$\begin{array}{l}
D(sts, vis) = \begin{cases} \emptyset & sts = \emptyset \\ trans \cup D(newsts, vis') & otherwise \end{cases} \\
\quad \textbf{where} \ trans = \{ \langle \phi, \delta(\phi) \rangle \mid \phi \in sts \} \\
\qquad sts' = collectStates(trans) \\
\qquad vis' = sts \cup vis \\
\qquad newsts = sts' \setminus vis'
\end{array}$$

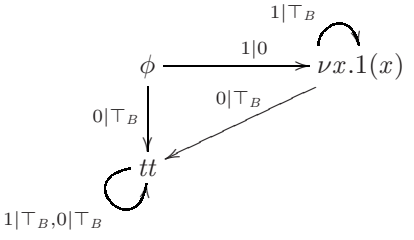
The function $\Delta$ takes a Mealy formula $\phi \in L_g^c$ and returns a Mealy machine that satisfies $\phi$:

$$\Delta(\phi) = (dom(f), f) \quad \textbf{where } f = D(\{norm(\phi)\}, \emptyset)$$

The function $dom$ returns the domain of a finite function.

Due to lack of space, the proof of finiteness and termination of the synthesis algorithm is not included. They are included in the extended version of this paper [3].

Let us look at an example. For the formula $\phi$ presented above $\Delta(\phi) = (S, f)$, where $S = \{tt, \phi, \nu x.1(x)\}$ and $f$ is represented by the following diagram.




Note that the Mealy machine produced by $\Delta$ is not minimal. In this example, the states $tt$ and $\nu x.1(x)$ are bisimilar and could be identified.

The (special) output value $\top_B$ allows us to define *underspecified* machines: if a given formula does not contain information about the output value for a given input $a$, then we do not return as output a concrete value but instead $\top_B$. If $\top_B$ is replaced by any other element $b \in B$ the resulting machine will still satisfy $\phi$.

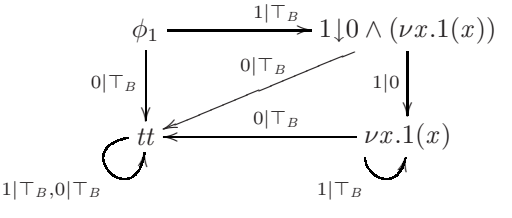
Let us see a few other examples of the synthesis process. To simplify the presentation, we consider again binary machines and, moreover, the formulae presented below will only have information for the input 1. Therefore, for the 0 input $\delta$ will always return $\langle \top_B, tt \rangle$.

Let us start with $\phi_1 = 1(1\downarrow 0) \wedge (\nu x.1(x))$. We have:

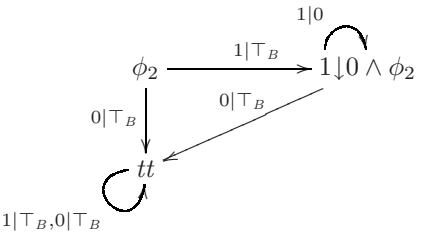
$$\begin{aligned} \delta(\phi_1)(1) &= \delta(1(1\downarrow 0))(1) \sqcap \delta(\nu x.1(x))(1) \\ &= \langle \top_B, 1\downarrow 0 \rangle \sqcap \langle \top_B, \nu x.1(x) \rangle \\ &= \langle \top_B, 1\downarrow 0 \wedge (\nu x.1(x)) \rangle \end{aligned}$$

We now repeat the process for $1\downarrow 0 \wedge (\nu x.1(x))$, which will yield $\delta(1\downarrow 0 \wedge (\nu x.1(x)))(1) = \langle 0, \nu x.1(x) \rangle$. Finally, we calculate $\delta(\nu x.1(x))(1) = \langle \top_B, \nu x.1(x) \rangle$.

The complete Mealy machine is represented in the following diagram:




Now, take $\phi_2 = \nu x.1(1\downarrow 0) \wedge 1(x)$. Because $1(1\downarrow 0)$ has no $x$'s one could be tempted to assume that the automaton for $\phi_2$ would be the same as the one for $\phi_1$. However, that is not the case. The synthesis algorithm will produce the following automaton for $\phi_2$.




As a last example, let $\phi_3 = \nu x.1(x \wedge (\nu y.1(y) \wedge 1{\downarrow}0))$. We have:

$$\begin{aligned}\delta(\phi_3)(1) &= \delta(1(\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0)))(1)\\ &= \langle \top_B, \phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0)\rangle\end{aligned}$$

and

$$\begin{aligned}&\delta(\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0))(1)\\ &= \delta(\phi_3)(1) \sqcap \delta(\nu y.1(y) \wedge 1{\downarrow}0)(1)\\ &= \langle \top_B, \phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0)\rangle \sqcap \langle 0, \nu y.1(y) \wedge 1{\downarrow}0\rangle\\ &= \langle 0, norm(\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0) \wedge (\nu y.1(y) \wedge 1{\downarrow}0))\rangle\\ &= \langle 0, \phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0)\rangle\end{aligned}$$

Note that if $norm$ would not have been applied, the resulting state $\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0) \wedge (\nu y.1(y) \wedge 1{\downarrow}0)$ would be regarded as a new state, even though it is equivalent to $\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0)$. Moreover, applying $\delta$ to this state (for input 1) would yield again an equivalent but (syntactically) different state, namely $\phi_3 \wedge (\nu y.1(y) \wedge 1{\downarrow}0) \wedge (\nu y.1(y) \wedge 1{\downarrow}0) \wedge (\nu y.1(y) \wedge 1{\downarrow}0)$. This illustrates that the function $\lambda$, defined in Section 3, generally produces an infinite machine. However, the identifications made by $norm$ ensure the termination of the synthesis process.

## 5 Conclusions and Future Work

We have given a coalgebraic account of Mealy machines and provided a logical specification language for them. Despite its simplicity, the logic is expressive in the sense that all Mealy machines can be characterized by finite formulae, but also in the sense that logical equivalence corresponds to bisimulation. Further, the logic is sound and the modal fragment complete for all Mealy machines.

The specification language is finitary and includes a fixed point operator.

Other temporal operators can be defined as derived operators. Interestingly, the language is already expressive enough to characterize all Mealy machines even without negation and disjunction. Even stronger, for binary Mealy machines the addition of negation does not increase the expressive power of the logic. This situation is typical also of deterministic finite automata: the addition of negation in regular expressions does not increase the class of languages that they characterize, even though regular languages are closed under complement.

Our main result is an algorithm for the synthesis of a Mealy machine from a formula. Our synthesis algorithm is compositional, in the sense that the semantics of the

Mealy machine synthesized from a formula can be obtained by suitably composing the semantics of the Mealy machines synthesized from sub-formulae.

In this paper we have explored the synthesis of one particular type of automata, the Mealy machines. With a small variation of the logic one can easily obtain a similar result for Moore automata. More generally, different type of automata can be obtained by varying the functor under consideration on the category of sets. It would be interesting to generalize the present result in order to synthesize coalgebras for different functors.

*Acknowledgements.* We would like to thank Clemens Kupke, Helle Hvid Hansen and Yde Venema for valuable suggestions and discussions.

## References

1. Bonsangue, M.M., Kurz, A.: Duality for logics of transition systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 455–469. Springer, Heidelberg (2005)
2. Bonsangue, M.M., Kurz, A.: Presenting functors by operations and equations. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 172–186. Springer, Heidelberg (2006)
3. Bonsangue, M.M., Rutten, J.J. M.M., Silva, A.: Coalgebraic Logic and Synthesis of Mealy Machines. CWI Technical report R0705 (2007)
4. Clarke, E.M., German, S.M., Lu, Y., Veith, H., Wang, D.: Executable protocol specification in esl. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 197–216. Springer, Heidelberg (2000)
5. Hansen, H.H., Costa, D., Rutten, J.J.M.M.: Synthesis of mealy machines using derivatives. ENTCS 164(1), 27–45 (2006)
6. Katis, P., Sabadini, N., Walters, R.F.C.: Feedback, trace and fixed-point semantics. ITA 36(2), 181–194 (2002)
7. Kozen, D.: Results on the propositional $\mu$-calculus. TCS 27, 333–354 (1983)
8. Kupferman, O., Vardi, M.: $\mu$-calculus synthesis. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 497–507. Springer, Heidelberg (2000)
9. Kupferman, O., Vardi, M., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM 47(2), 312–360 (2000)
10. Kupke, C., Venema, Y.: Coalgebraic automata theory: basic results. Technical Report SEN-E0701, CWI, The Netherlands (2007)
11. Marcovitz, A.B.: Introduction to Logic Design. McGraw-Hill, New York (2005)
12. Moss, L.: Coalgebraic logic. Annals of Pure and Applied Logic 96 (1999)
13. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL 1989, pp. 179–190 (1989)
14. Rutten, J.J.M.M.: Algebraic specification and coalgebraic synthesis of mealy automata. ENTCS 160, 305–319 (2006)
15. Tini, S., Maggiolo-Schettini, A.: Compositional synthesis of generalized mealy machines. Fundam. Inform. 60(1–4), 367–382 (2004)

# The Microcosm Principle and Concurrency in Coalgebra

Ichiro Hasuo[1,3,4], Bart Jacobs[1,⋆], and Ana Sokolova[2,⋆⋆]

[1] Radboud University Nijmegen, The Netherlands
[2] University of Salzburg, Austria
[3] RIMS, Kyoto University, Japan
[4] PRESTO Research Promotion Program, Japan Science and Technology Agency

**Abstract.** Coalgebras are categorical presentations of state-based systems. In investigating parallel composition of coalgebras (realizing *concurrency*), we observe that the same algebraic theory is interpreted in two different domains in a nested manner, namely: in the category of coalgebras, and in the final coalgebra as an object in it. This phenomenon is what Baez and Dolan have called the *microcosm principle*, a prototypical example of which is "a monoid in a monoidal category." In this paper we obtain a formalization of the microcosm principle in which such a nested model is expressed categorically as a suitable lax natural transformation. An application of this account is a general compositionality result which supports modular verification of complex systems.

## 1 Introduction

Design of systems with *concurrency* is nowadays one of the mainstream challenges in computer science [19]. Concurrency is everywhere: with the Internet being the biggest example and multi-core processors the smallest; also in a modular, component-based architecture of a complex system its components collaborate in a concurrent manner. However, numerous difficulties have been identified in getting concurrency right. For example, a system's exponentially growing complexity is one of the main obstacles. One way to cope with it is a *modular* verification method in which correctness of the whole system $\mathcal{C}_1 \parallel \cdots \parallel \mathcal{C}_n$ is established using correctness of each component $\mathcal{C}_i$. *Compositionality*—meaning that the behavior of $\mathcal{C} \parallel \mathcal{D}$ is determined by the behavior of $\mathcal{C}$ and that of $\mathcal{D}$—is an essential property for such a modular method to work.

**Coalgebras as systems.** This paper is a starting point of our research program aimed at better understanding of the mathematical nature of concurrency. In its course we shall use *coalgebras* as presentations of systems to be run in parallel. The use of coalgebras as an appropriate abstract model of state-based systems is increasingly established [26,11]; the notion's mathematical simplicity and clarity provide us with a sound foundation

⋆ Also part-time at Technical University Eindhoven, The Netherlands.
⋆⋆ Supported by the Austrian Science Fund (FWF) project no. P18913-N15.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 246–260, 2008.

for our exploration. The following table summarizes how ingredients of the theory of systems are presented as coalgebraic constructs.

| | system | behavior-preserving map | behavior |
|---|---|---|---|
| | | morphism of coalgebras | by coinduction |
| coalgebraically | coalgebra $\begin{array}{c} FX \\ \uparrow \\ X \end{array}$ | $\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ \uparrow & & \uparrow \\ X & \xrightarrow[f]{} & Y \end{array}$ | $\begin{array}{ccc} FX & - - \to & FZ \\ {\scriptstyle c}\uparrow & & {\scriptstyle \text{final}}\uparrow\cong \\ X & \underset{\mathsf{beh}(c)}{- - \to} & Z \end{array}$ (1) |

This view of "coalgebras as systems" has been successfully applied in the category $\mathbf{Sets}$ of sets and functions, in which case the word "behavior" in (1) refers (roughly) to bisimilarity. Our recent work [6,5] has shown that "behavior" can also refer to trace semantics by moving from $\mathbf{Sets}$ to a suitable Kleisli category.

**Compositionality in coalgebras.** We start with the following question: what is "compositionality" in this coalgebraic setting? Conventionally compositionality is expressed as: $\mathcal{C} \sim \mathcal{C}'$ and $\mathcal{D} \sim \mathcal{D}'$ implies $\mathcal{C} \parallel \mathcal{D} \sim \mathcal{C}' \parallel \mathcal{D}'$, where the relation $\sim$ denotes the behavioral equivalence of interest. If this is the case the relation $\sim$ is said to be a *congruence*, with its oft-heard instance being "bisimilarity is a congruence."

When we interpret "behavior" in compositionality as the coalgebraic behavior induced by coinduction (see (1)), the following equation comes natural as a coalgebraic presentation of compositionality.

$$\mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array} \;\middle\|\; \begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) = \mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array}\right) \;\middle\|\; \mathsf{beh}\left(\begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) \tag{2}$$

But a closer look reveals that the two "parallel composition operators" $\parallel$ in the equation have in fact different types: the first one $\mathbf{Coalg}_F \times \mathbf{Coalg}_F \to \mathbf{Coalg}_F$ combines systems (as coalgebras) and the second one $Z \times Z \to Z$ combines behavior (as states of the final coalgebra).[1] Moreover, the two domains are actually nested: the latter one $Z \xrightarrow{\cong} FZ$ is an object of the former one $\mathbf{Coalg}_F$.

**The microcosm principle.** What we have just observed is one instance—probably the first one explicitly claimed in computer science—of the *microcosm principle* as it is called by Baez and Dolan [1]. It refers to a phenomenon that the same algebraic theory (or algebraic "specification," consisting of operations and equations) is interpreted twice in a nested manner, once in a category $\mathbb{C}$ and the other time in its object $X \in \mathbb{C}$. This is not something very unusual, because "a monoid in a monoidal category" constitutes a prototypical example.

[1] At this stage the presentation remains sloppy for the sake of simplicity. Later in technical sections the first composition operator will be denoted by $\otimes$; and the second composition operator will have the type $Z \otimes Z \to Z$ instead of $Z \times Z \to Z$.

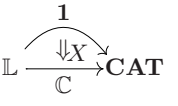
| monoidal category $\mathbb{C}$ | | monoid $X \in \mathbb{C}$ |
|---|---|---|
| $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ | multiplication | $X \otimes X \xrightarrow{\mu} X$ |
| $I \in \mathbb{C}$ | unit | $I \xrightarrow{\eta} X$ |
| $I \otimes X \cong X \cong X \otimes I$ | unit law | $X \to X \otimes X \leftarrow X$ with $X$ |
| $X \otimes (Y \otimes Z) \cong (X \otimes Y) \otimes Z$ | associativity law | $X \otimes X \otimes X \to X \otimes X$, $X \otimes X \to X$ |

(3)

Notice here that the outer operation $\otimes$ appears in the formulation of the inner operation $\mu$. Moreover, to be precise, in the inner "equations" the outer isomorphisms should be present in suitable places. Hence this monoid example demonstrates that, in such nested algebraic structures, the inner structure depends on the outer. What is a mathematically precise formalization of such nested models? Answering this question is a main goal of this paper.

Such a formalization has been done in [11] when algebraic structures are specified in the form of *opetopes*. Here instead we shall formalize the microcosm principle for *Lawvere theories* [18], whose role as categorical representation of algebraic theories has been recognized in theoretical computer science.

As it turns out, our formalization looks like the situation on the right. Here $\mathbb{L}$ is a category (a Lawvere theory) representing an algebraic theory; an outer model $\mathbb{C}$ is a product-preserving functor; and an inner model $X$ is a lax natural transformation. The whole setting is 2-categorical: 2-categories (categories in categories) serve as an appropriate basis for the microcosm principle (algebras in algebras).




**Applications to coalgebras: Parallel composition via sync.** The categorical account we have sketched above shall be applied to our original question about parallel composition of coalgebras. As a main application we prove a *generic compositionality theorem*. For an arbitrary algebraic theory $\mathbb{L}$, compositionality like (2) is formulated as follows: the "behavior" functor $\mathsf{beh} : \mathbf{Coalg}_F \to \mathbb{C}/Z$ via coinduction preserves an $\mathbb{L}$-structure. This general form of compositionality holds if: $\mathbb{C}$ has an $\mathbb{L}$-structure and $F : \mathbb{C} \to \mathbb{C}$ *lax*-preserves the $\mathbb{L}$-structure.

Turning back to the original setting of (2), these general assumptions read roughly as follows: the base category $\mathbb{C}$ has a binary operation $\|$; and the endofunctor $F$ comes with a natural transformation $\mathsf{sync} : FX \parallel FY \to F(X \parallel Y)$. Essentially, this sync is what lifts $\|$ on $\mathbb{C}$ to $\|$ on $\mathbf{Coalg}_F$, hence "parallel composition via sync." It is called a *synchronization* because it specifies the way two systems synchronize with each other. In fact, for a fixed functor $F$ there can be different choices of sync (such as CSP-style vs. CCS-style), which in turn yield different "parallel composition" operators on the category $\mathbf{Coalg}_F$.

**Related work.** Our interest is pretty similar to that of studies of *bialgebraic structures* in computer science (such as [27, 3, 15, 14, 12, 16]), in the sense that we are also concerned about algebraic structures on coalgebras as systems. Our current framework is distinguished in the following aspects.

First, we handle *equations* in an algebraic theory as an integral part of our approach. Equations such as associativity and commutativity appear explicitly as commutative diagrams in a Lawvere theory $\mathbb{L}$. We benefit from this explicitness in e.g. spelling out a condition for the generic associativity result (Theorem 2.4). In contrast, in the bialgebraic studies an algebraic theory is presented either by an endofunctor $X \mapsto \coprod_{\sigma\in\Sigma} X^{|\sigma|}$ or by a monad $T$. In the former case equations are simply not present; in the latter case equations are there but only implicitly.

Secondly and more importantly, by considering higher-dimensional, nested algebraic structures, we can now compose different coalgebras as well as different states of the same coalgebra. In this way the current work can be seen as a higher-dimensional extension of the existing bialgebraic studies (which focus on "inner" algebraic structures).

**Organization of the paper.** We shall not dive into our 2-categorical exploration from the beginning. In Section 2, we instead focus on one specific algebraic theory, namely the one for parallel composition of systems. Our emphasis there is on the fact that the sync natural transformation essentially gives rise to parallel composition $\|$, and the fact that equational properties of $\|$ (such as associativity) can be reduced to the corresponding equational properties of sync.

These concrete observations will provide us with intuition for abstract categorical constructs in Section 3, where we formalize the microcosm principle for an arbitrary Lawvere theory $\mathbb{L}$. Results on coalgebras such as compositionality are proved here in their full generality and abstraction.

In this paper we shall focus on *strict* algebraic structures on categories in order to avoid complicated coherence issues. This means for example that we only consider *strict* monoidal categories for which the isomorphisms in (3) are in fact equalities. However, we have also obtained some preliminary observations on relaxed ("pseudo" or "strong") algebraic structures: see Section 3.3.

## 2 Parallel Composition of Coalgebras

### 2.1 Parallel Composition Via sync Natural Transformation

Let us start with the equation (2), a coalgebraic representation of compositionality. The operator $\|$ on the left is of type $\mathbf{Coalg}_F \times \mathbf{Coalg}_F \to \mathbf{Coalg}_F$. It is natural to require functoriality of this operation, making it a *bifunctor*. A bifunctor—especially an associative one which we investigate in Section 2.3—plays an important role in various applications of category theory. Usually such an (associative) bifunctor is called a *tensor* and denoted by $\otimes$, a convention that we also follow. Therefore the "compositionality" statement now looks as follows.

$$\mathsf{beh}\begin{pmatrix} FX \\ c\!\uparrow \\ X \end{pmatrix} \otimes \begin{pmatrix} FY \\ d\!\uparrow \\ Y \end{pmatrix} = \mathsf{beh}\begin{pmatrix} FX \\ c\!\uparrow \\ X \end{pmatrix} \Big\| \mathsf{beh}\begin{pmatrix} FY \\ d\!\uparrow \\ Y \end{pmatrix} \tag{4}$$

The first question is: when do we have such a tensor $\otimes$ on $\mathbf{Coalg}_F$? In many applications of coalgebras, it is obtained by lifting a tensor $\otimes$ on the base category $\mathbb{C}$ to $\mathbf{Coalg}_F$.[2] Such a lifting is possible in presence of a natural transformation

$$FX \otimes FY \xrightarrow{\mathsf{sync}_{X,Y}} F(X \otimes Y), \quad \text{used in} \quad \begin{array}{c} FX \\ c\uparrow \\ X \end{array} \otimes \begin{array}{c} FY \\ d\uparrow \\ Y \end{array} := \begin{array}{c} F(X \otimes Y) \\ \uparrow \mathsf{sync}_{X,Y} \\ FX \otimes FY \\ \uparrow c \otimes d \\ X \otimes Y \end{array} . \tag{5}$$

We shall call this $\mathsf{sync}$ a *synchronization* because its computational meaning is indeed a specification of the way two systems synchronize. This will be illustrated in the coming examples.

Once we have an outer parallel composition $\otimes$, an inner operator $\|$ which composes behavior (i.e. states of the final coalgebra) is also obtained immediately by coinduction as on the right. Compositionality (4) is also straightforward by finality: both sides of the equation are the unique coalgebra morphism from $c \otimes d$ to the final $\zeta$. The following theorem summarizes the observations so far.

$$\begin{array}{ccc} F(Z \otimes Z) & \dashrightarrow & FZ \\ {\scriptstyle \zeta \otimes \zeta}\uparrow & & \uparrow{\scriptstyle \zeta} \\ Z \otimes Z & \dashrightarrow & Z \end{array} \qquad \text{(final)}$$

**Theorem 2.1 (Coalgebraic compositionality).** *Assume that a category $\mathbb{C}$ has a tensor $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ and an endofunctor $F : \mathbb{C} \to \mathbb{C}$ has a natural transformation $\mathsf{sync}_{X,Y} : FX \otimes FY \to F(X \otimes Y)$. If moreover there exists a final $F$-coalgebra, then:*

1. *The tensor $\otimes$ on $\mathbb{C}$ lifts to an "outer" composition operator $\otimes : \mathbf{Coalg}_F \times \mathbf{Coalg}_F \to \mathbf{Coalg}_F$.*
2. *We obtain an "inner" composition operator $\|: Z \otimes Z \to Z$ by coinduction.*
3. *Between the two composition operators the compositionality property (4) holds.*

□

We can put the compositionality property (4) in more abstract terms as "the functor $\mathsf{beh} : \mathbf{Coalg}_F \to \mathbb{C}/Z$ preserves a tensor," meaning that the diagram below left commutes. Here a tensor $\underline{\otimes}$ on the slice category $\mathbb{C}/Z$ is given as on the right, using the inner composition $\|$.

$$\begin{array}{ccc} \mathbf{Coalg}_F \times \mathbf{Coalg}_F & \xrightarrow{\mathsf{beh} \times \mathsf{beh}} & \mathbb{C}/Z \times \mathbb{C}/Z \\ \otimes\downarrow & & \downarrow\underline{\otimes} \\ \mathbf{Coalg}_F & \xrightarrow[\mathsf{beh}]{} & \mathbb{C}/Z \end{array} \qquad \begin{pmatrix} X & Y \\ \downarrow f & \downarrow g \\ Z & Z \end{pmatrix} \overset{\underline{\otimes}}{\longmapsto} \begin{array}{c} X \otimes Y \\ \downarrow f \otimes g \\ Z \otimes Z \\ \downarrow \| \\ Z \end{array} \tag{6}$$

The point of Theorem 2.1 is as follows. Those parallel composition operators which are induced by $\mathsf{sync}$ are well-behaved ones: good properties like compositionality come for free. We shall present some examples in Section 2.2.

**Remark 2.2.** The view of parallel composition of systems as a tensor structure on $\mathbf{Coalg}_F$ has been previously presented in [13]. The interest there is on categorical

[2] Note that we use boldface $\otimes$ for a tensor on $\mathbf{Coalg}_F$ to distinguish it from $\otimes$ on $\mathbb{C}$.

structures on $\mathbf{Coalg}_F$ rather than on properties of parallel composition such as compositionality. In [13] and other literature an endofunctor $F$ with sync (equipped with some additional compatibility) is called a *monoidal endofunctor*.[3]

### 2.2 Examples

**In Sets: Bisimilarity is a congruence.** We shall focus on LTSs and bisimilarity as their process semantics. For this purpose it is appropriate to take $\mathbf{Sets}$ as our base category $\mathbb{C}$ and $\mathcal{P}_\omega(\Sigma \times \_)$ as the functor $F$. We use Cartesian products as a tensor on $\mathbf{Sets}$. This means that a composition of two coalgebras has the product of the two state spaces as its state space, which matches our intuition. The functor $\mathcal{P}_\omega$ in $F$ is the finite powerset functor; the finiteness assumption is needed for existence of a final $F$-coalgebra. It is standard (see e.g. [26]) that a final $F$-coalgebra captures bisimilarity via coinduction.

In considering parallel composition of LTSs, the following two examples are well-known ones.[4]

- *CSP-style* [7]: $a.P \parallel a.Q \xrightarrow{a} P \parallel Q$. For the whole system to make an $a$-action, each component has to make an $a$-action.
- *CCS-style* [21]: $a.P \parallel \overline{a}.Q \xrightarrow{\tau} P \parallel Q$, assuming $\Sigma = \{a, b, \dots\} \cup \{\overline{a}, \overline{b}, \dots\} \cup \{\tau\}$. When one component outputs on a channel $a$ and another inputs from $a$, then the whole system makes an internal $\tau$ move.

In fact, each of these different ways of synchronization can be represented by a suitable sync natural transformation.

$$\begin{array}{ccc} \mathcal{P}_\omega(\Sigma \times X) \times \mathcal{P}_\omega(\Sigma \times Y) & \xrightarrow[\mathsf{sync}^{\mathrm{CSP}}_{X,Y}]{} & \mathcal{P}_\omega\big(\Sigma \times (X \times Y)\big) \\ (u, v) & \longmapsto & \big\{\, (a,(x,y)) \mid (a,x) \in u \wedge (a,y) \in v \,\big\} \\ (u, v) & \xrightarrow[\mathsf{sync}^{\mathrm{CCS}}_{X,Y}]{} & \big\{\, (\tau,(x,y)) \mid (a,x) \in u \wedge (\overline{a},y) \in v \,\big\} \end{array}$$

By Theorem 2.1, each of these gives (different) $\otimes$ on $\mathbf{Coalg}_F$, and $\parallel$ on $Z$; moreover the behavior functor beh satisfies compositionality. In other words: bisimilarity is a congruence with respect to both CSP-style and CCS-style parallel composition.

**Remark 2.3.** As mentioned in the introduction, in some ways this paper can be seen as an extension of the bialgebraic studies started in [27]. However there is also a drawback, namely the limited expressive power of sync : $FX \otimes FY \to F(X \otimes Y)$.

Our sync specifies the way an algebraic structure interacts with a coalgebraic one. In this sense it is a counterpart of a distributive law $\Sigma F \Rightarrow F\Sigma$ in [27] representing operational rules, where $\Sigma$ is a functor induced by an algebraic signature. However there are many common operational rules which do not allow representation of the

[3] Later in Section 3 we will observe that a functor $F$ with sync is a special case of a *lax* $\mathbb{L}$-functor, by choosing a suitable algebraic theory $\mathbb{L}$. Such a functor $F$ with sync is usually called a monoidal functor (as opposed to a *lax* monoidal functor), probably because it preserves (inner) monoid objects; see Proposition 3.8.1.

[4] Here we focus on synchronous interaction. Both CSP and CCS have an additional kind of interaction, namely an "interleaving" one; see Remark 2.3.

form $\Sigma F \Rightarrow F\Sigma$; therefore in [27] the type of such a distributive law is eventually extended to $\Sigma(F \times \mathrm{id}) \Rightarrow F\Sigma^*$. The class of rules representable in this form coincides with the class of so-called *GSOS-rules*.

At present it is not clear how we can make a similar extension for our sync; consequently there are some operational rules which we cannot model by sync. One important example is an *interleaving* kind of interaction—such as $a.P \parallel Q \xrightarrow{a} P \parallel Q$ which leaves the second component unchanged. This is taken care of in [27] by the identity functor (id) appearing on the left-hand side of $\Sigma(F \times \mathrm{id}) \Rightarrow F\Sigma^*$. For our sync to be able to model such interleaving, we can replace $F$ by the cofree comonad on it, as is done in [13, Example 3.11]. This extension should be straightforward but detailed treatment is left as future work.

**In $\mathcal{K}\ell(T)$: Trace equivalence is a congruence.** In our recent work [6] we extend earlier observations in [25, 10] and show that trace semantics—including trace *set* semantics for non-deterministic systems and trace *distribution* semantics for probabilistic systems—is also captured by coinduction when it is employed in a Kleisli category $\mathcal{K}\ell(T)$. Applying the present composition framework, we can conclude that trace semantics is compositional with respect to well-behaved parallel composition. The details are omitted here due to lack of space.

### 2.3 Equational Properties of Parallel Composition

Now we shall investigate equational properties—associativity, commutativity, and so on—of parallel composition $\otimes$, which we have ignored deliberately for simplicity of argument. We present our result in terms of associativity; it is straightforward to transfer the result to other properties like commutativity. The main point of the following theorem is as follows: if $\otimes$ is associative and sync is "associative," then the lifting $\bar{\otimes}$ is associative. The proof is straightforward.

**Theorem 2.4.** *Let $\mathbb{C}$ be a category with a strictly associative tensor $\otimes$,*[5] *and $F : \mathbb{C} \to \mathbb{C}$ be a functor with* $\mathsf{sync} : FX \otimes FY \to F(X \otimes Y)$. *If the diagram*

$$\begin{array}{ccccc} FX \otimes (FY \otimes FZ) & \xrightarrow{FX \otimes \mathsf{sync}} & FX \otimes F(Y \otimes Z) & \xrightarrow{\mathsf{sync}} & F(X \otimes (Y \otimes Z)) \\ \downarrow \mathrm{id} & & & & \downarrow \mathrm{id} \\ (FX \otimes FY) \otimes FZ & \xrightarrow[\mathsf{sync} \otimes FZ]{} & F(X \otimes Y) \otimes FZ & \xrightarrow[\mathsf{sync}]{} & F((X \otimes Y) \otimes Z) \end{array} \tag{7}$$

*commutes, then the lifted tensor $\bar{\otimes}$ on $\mathbf{Coalg}_F$ is strictly associative.* □

The two identity arrows in (7) are available due to strict associativity of $\otimes$. In the next section we shall reveal the generic principle behind the commutativity condition of (7), namely a coherence condition on a lax natural transformation.

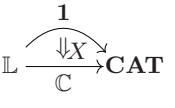
As an example, $\mathsf{sync}^{\mathrm{CSP}}$ and $\mathsf{sync}^{\mathrm{CCS}}$ in Section 2.2 are easily seen to be "associative" in the sense of the diagram (7). Therefore the resulting tensors $\bar{\otimes}$ are strictly associative.

[5] As mentioned already, in this paper we stick to *strict* algebraic structures.

## 3 Formalizing the Microcosm Principle

In this section we shall formalize the microcosm principle for an arbitrary algebraic theory presented as a Lawvere theory $\mathbb{L}$. This and the subsequent results generalize the results in the previous section. In particular, we will obtain a general compositionality result which works for an arbitrary algebraic theory.

As we sketched in the introduction, an outer model will be a product-preserving functor $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$; an inner model inside will be a lax natural transformation $X : \mathbf{1} \Rightarrow \mathbb{C}$. Here $\mathbf{1} : \mathbb{L} \to \mathbf{CAT}$ is the constant functor which maps everything to the category $\mathbf{1}$ with one object and one arrow (which is a special case of an outer model). Mediating 2-cells for the *lax* natural transformation $X$ play a crucial role as inner interpretation of algebraic operations. In this section we heavily rely on 2-categorical notions, about which detailed accounts can be found in [4].




### 3.1 Lawvere Theories

*Lawvere theories* are categorical presentations of algebraic theories. The notion is introduced in [18] (not under this name, though) aiming at a categorical formulation of "theories" and "semantics." An accessible introduction to the notion can be found in [17]. Lawvere theories are known to be equivalent to *finitary monads*. These two ways of presenting algebraic theories have been widely used in theoretical computer science, e.g. for modeling computation with effect [22, 8]. Recent developments (such as [24]) utilize the increased expressive power of *enriched* Lawvere theories.

In the sequel, by an *FP-category* we refer to a category with (a choice of) finite products. An *FP-functor* is a functor between FP-categories which preserves finite products "on-the-nose," that is, up-to-equality instead of up-to-isomorphism.

**Definition 3.1 (Lawvere theory).** By $\mathbf{Nat}$ we denote the category of natural numbers (as sets) and functions between them. Therefore every arrow in $\mathbf{Nat}$ is a (cotuple of) coprojection; an arrow in $\mathbf{Nat}^{\mathrm{op}}$ is a (tuple of) projection.[6]

A *Lawvere theory* is a small FP-category $\mathbb{L}$ equipped with an FP-functor $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$ which is bijective on objects. We shall denote an object of $\mathbb{L}$ by a natural number $k$, identifying $k \in \mathbf{Nat}^{\mathrm{op}}$ and $Hk \in \mathbb{L}$.

The category $\mathbf{Nat}^{\mathrm{op}}$—which is a free FP-category on the trivial category $\mathbf{1}$—is there in order to specify the choice of finite products in $\mathbb{L}$. For illustration, we make some remarks on $\mathbb{L}$'s objects and arrows.

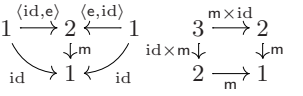
- An object $k \in \mathbb{L}$ is a $k$-fold product $1 \times \cdots \times 1$ of 1.
- An arrow in $\mathbb{L}$ is intuitively understood as an algebraic operation. That is, $k \to 1$ as a $k$-ary operation; and $k \to n$ as an $n$-tuple $\langle \mathsf{f}_1, \ldots, \mathsf{f}_n \rangle$ of $k$-ary operations. To be precise, arrows in $\mathbb{L}$ also include projections (such as $\pi_1 : 2 \to 1$) and *terms* made up of operations and projections (such as $\mathsf{m} \circ \langle \pi_1, \pi_2 \rangle : 3 \to 1$).

[6] An arrow $f : n \to k$ in $\mathbf{Nat}$ can be written as a cotuple $[\kappa_{f(1)}, \ldots, \kappa_{f(n)}]$ where $\kappa_i : 1 \to k$ is the coprojection into the $i$-th summand of $1 + \cdots + 1$ ($k$ times).

Conventionally in universal algebra, an algebraic theory is presented by an *algebraic specification* $(\Sigma, E)$—a pair of a set $\Sigma$ of operations and a set $E$ of equations. A Lawvere theory $\mathbb{L}$ arises from such $(\Sigma, E)$ as its so-called *classifying category* (see e.g. [18, 9]). An arrow $k \to n$ in the resulting Lawvere theory $\mathbb{L}$ is an $n$-tuple $([t_1(\vec{x})], \dots, [t_n(\vec{x})])$ of $\Sigma$-terms with $k$ variables $\vec{x}$, where $[\_]$ denotes taking an equivalence class modulo equations in $E$. An equivalent way to describe this construction is via *sketches*: $(\Sigma, E)$ is identified with an FP-sketch, which in turn induces $\mathbb{L}$ as a free FP-category. See [2] for details.

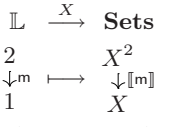
Our leading example is the Lawvere theory $\mathbf{Mon}$ for monoids.[7] It arises as a classifying category from the well-known algebraic specification of monoids. This specification has a nullary operation $\mathsf{e}$ and a binary one $\mathsf{m}$; subject to the equations $\mathsf{m}(x, \mathsf{e}) = x$, $\mathsf{m}(\mathsf{e}, x) = x$, and $\mathsf{m}(x, \mathsf{m}(y, z)) = \mathsf{m}(\mathsf{m}(x, y), z)$.

Equivalently, $\mathbf{Mon}$ is the freely generated FP-category by arrows $0 \xrightarrow{\mathsf{e}} 1$ and $2 \xrightarrow{\mathsf{m}} 1$ subject to the commutativity on the right. These data (arrows and commutative diagrams) form an FP-sketch (see [2]).




### 3.2 Outer Models: $\mathbb{L}$-Categories

We start by formalizing an outer model. It is a category with an $\mathbb{L}$-structure, hence called an $\mathbb{L}$-*category*. It is standard that a (set-theoretic) model of $\mathbb{L}$—a *set* with an $\mathbb{L}$-structure—is identified with an FP-functor $\mathbb{L} \xrightarrow{X} \mathbf{Sets}$. Concretely, let $X = X1$ be the image of $1 \in \mathbb{L}$. Then $k \in \mathbb{L}$ must be sent to $X^k$ due to preservation of finite products. Now the functor's action on arrows is what interprets $\mathbb{L}$'s operations in $X$, as illustrated above right. Equations (expressed as commutative diagrams in $\mathbb{L}$) are satisfied because a functor preserves commutativity.




Turning back to $\mathbb{L}$-*categories*, what we have to do here is to just replace $\mathbf{Sets}$ by the category $\mathbf{CAT}$ of (possibly large and locally small) categories.

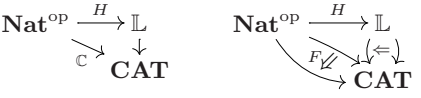
**Definition 3.2 ($\mathbb{L}$-categories, $\mathbb{L}$-functors).** A (strict) $\mathbb{L}$-*category* is an FP-functor $\mathbb{L} \xrightarrow{\mathbb{C}} \mathbf{CAT}$. In the sequel we denote the image $\mathbb{C}1$ of $1 \in \mathbb{L}$ by $\mathbb{C}$; and the image $\mathbb{C}(\mathsf{f})$ of an arrow $\mathsf{f}$ by $[\![\mathsf{f}]\!]$.

An $\mathbb{L}$-*functor* $F : \mathbb{C} \to \mathbb{D}$—a functor preserving an $\mathbb{L}$-structure—is a natural transformation $\mathbb{L} \overset{\mathbb{C}}{\underset{\mathbb{D}}{\Downarrow F}} \mathbf{CAT}$.

Another way to look at the previous definition is to view an $\mathbb{L}$-structure as "factorization through $\mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$." We can identify a category $\mathbb{C} \in \mathbf{CAT}$ with a functor $\mathbf{1} \to \mathbf{CAT}$, which is in turn identified with an FP-functor $\mathbf{Nat}^{\mathrm{op}} \to \mathbf{CAT}$, because $\mathbf{Nat}^{\mathrm{op}}$ is the free FP-category on $\mathbf{1}$. We say that $\mathbb{C}$ has an $\mathbb{L}$-structure, if this FP-functor factors through $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$ (as below left). Note that the factorization is not necessarily

[7] The Lawvere theory $\mathbf{Mon}$ for the theory of monoids should not be confused with the category of (set-theoretic) monoids and monoid homomorphisms (which is often denoted by $\mathbf{Mon}$ as well).

unique, because there can be different ways of interpreting the algebraic theory $\mathbb{L}$ in $\mathbb{C}$. Similarly, a functor $\mathbb{C} \xrightarrow{F} \mathbb{D}$ is identified with a natural transformation $\mathbf{1} \Downarrow F \mathbf{CAT}$; and then with $\mathbf{Nat}^{\mathrm{op}} \Downarrow F \mathbf{CAT}$ due to the 2-universality of $\mathbf{Nat}^{\mathrm{op}}$ as a free object. We say that this $F$ preserves an $\mathbb{L}$-structure, if the last natural transformation factors through $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$ (as below right).

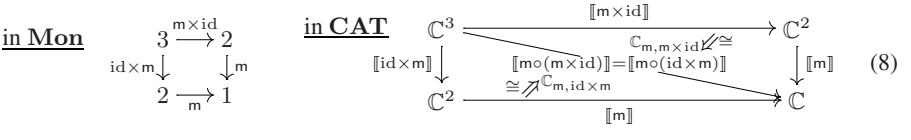



**Example 3.3.** The usual notion of strictly monoidal categories coincides with $\mathbb{L}$-categories for $\mathbb{L} = \mathbf{Mon}$. A tensor $\otimes$ and a unit $I$ on a category arise as interpretation of the operations $2 \xrightarrow{\mathsf{m}} 1$ and $0 \xrightarrow{\mathsf{e}} 1$; commuting diagrams in $\mathbf{Mon}$ such as $\mathsf{m} \circ \langle \mathrm{id}, \mathsf{e} \rangle = \mathrm{id}$ yield equational properties of $\otimes$ and $I$.

### 3.3 Remarks on "Pseudo" Algebraic Structures

As we mentioned in the introduction, in this paper we focus on *strict* algebraic structures. This means that monoidal categories (in which associativity holds only up-to-isomorphism, for example) fall out of our consideration. Extending our current framework to such "pseudo" algebraic structures is one important direction of our future work. Such an extension is not entirely obvious; we shall sketch some preliminary observations in this direction.

The starting point is to relax the definition of $\mathbb{L}$-categories from (strict) functors $\mathbb{L} \to \mathbf{CAT}$ to *pseudo* functors, meaning that composition and identities are preserved only up-to-isomorphism. Then it is not hard to see that a pseudo functor $\mathbf{Mon} \xrightarrow{\mathbb{C}} \mathbf{CAT}$ (which preserves finite products in a suitable sense) gives rise to a monoidal category. Indeed, let us denote a mediating iso-2-cell for composition by $\mathbb{C}_{\mathsf{g},\mathsf{f}} : [\![\mathsf{g}]\!] \circ [\![\mathsf{f}]\!] \xRightarrow{\cong} [\![\mathsf{g} \circ \mathsf{f}]\!]$. The associativity diagram (below left) gives rise to the two iso-2-cells on the right.




(8)

The composition $\mathbb{C}^{-1}_{\mathsf{m},\mathrm{id}\times\mathsf{m}} \bullet \mathbb{C}_{\mathsf{m},\mathsf{m}\times\mathrm{id}}$ is what gives us a natural isomorphism $\alpha : X \otimes (Y \otimes Z) \xrightarrow{\cong} (X \otimes Y) \otimes Z$. Moreover, the coherence condition on such isomorphisms in a monoidal category (like the famous pentagon diagram; see [20]) follows from the coherence condition on mediating 2-cells of a pseudo functor (see [4]).

So far so good. However, at this moment it is not clear what is a canonical construction the other way round, i.e. from a monoidal category to a pseudo functor.[8] In the present paper we side-step these 2-categorical subtleties by restricting ourselves to strict, non-pseudo functors.

[8] For example, given a monoidal category $\mathbb{C}$, we need to define a functor $[\![\mathsf{m} \circ (\mathsf{m} \times \mathrm{id})]\!] = [\![\mathsf{m} \circ (\mathrm{id} \times \mathsf{m})]\!]$ in (8). It's not clear whether it should carry $(X, Y, Z)$ to $X \otimes (Y \otimes Z)$, or to $(X \otimes Y) \otimes Z$.

### 3.4 Inner Models: $\mathbb{L}$-Objects

We proceed to formalize an inner model. It is an object in an $\mathbb{L}$-category which itself carries an (inner) $\mathbb{L}$-structure, hence is called an $\mathbb{L}$-*object*. A monoid object in a monoidal category is a prototypical example. We first present an abstract definition; some illustration follows afterwards.

**Definition 3.4 ($\mathbb{L}$-objects).** An $\mathbb{L}$-*object* $X$ in an $\mathbb{L}$-category $\mathbb{C}$ is a lax natural transformation $X : \mathbf{1} \Rightarrow \mathbb{C}$ (below left) which is "product-preserving": this means that the composition $X \circ H$ (below right) is strictly, non-lax natural. Here $\mathbf{1} : \mathbb{L} \to \mathbf{CAT}$ denotes the constant functor to the trivial one-object category $\mathbf{1}$.

$$\mathbb{L} \xrightarrow[\mathbb{C}]{\mathbf{1} \;\Downarrow X} \mathbf{CAT} \qquad \mathbf{Nat}^{\mathrm{op}} \xrightarrow{H} \mathbb{L} \xrightarrow[\mathbb{C}]{\mathbf{1} \;\Downarrow X} \mathbf{CAT}$$

Such a nested algebraic structure—formalized as an $\mathbb{L}$-object in an $\mathbb{L}$-category—shall be called a *microcosm model* for $\mathbb{L}$.

Let us now illustrate the definition. First, $X$'s component at $1 \in \mathbb{L}$ is a functor $\mathbf{1} \xrightarrow{X_1} \mathbb{C}$ which is identified with an object $X \in \mathbb{C}$. This is the "carrier" object of this inner algebra. Moreover, any other component $\mathbf{1} \xrightarrow{X_k} \mathbb{C}^k$ must be the $k$-tuple $(X, \dots, X) \in \mathbb{C}^k$ of $X$'s. This is because of (strict) naturality of $X \circ H$ (see above right): for any $i \in [1, k]$ the composite $\pi_i \circ X_k$ is required to be $X_1$.

| in $\mathbf{Nat}^{\mathrm{op}}$ | in $\mathbf{CAT}$ |
|---|---|
| $k$ | $\mathbf{1} \xrightarrow{X_k = (X,\dots,X)} \mathbb{C}^k$ |
| $\downarrow \pi_i$ | $\Vert \quad \swarrow \quad \downarrow \llbracket H\pi_i \rrbracket = \pi_i$ |
| $1$ | $\mathbf{1} \xrightarrow[X_1 = X]{} \mathbb{C}$ |

The (inner) algebraic structure on $X$ arises in the form of mediating 2-cells of the *lax* natural transformation. For each arrow $k \xrightarrow{\mathsf{f}} n$ in $\mathbb{L}$, lax naturality of $X$ requires existence of a mediating 2-cell $X_{\mathsf{f}} : \llbracket \mathsf{f} \rrbracket \circ X_k \Rightarrow X_n$. The diagram (above right) shows the situation when we set $\mathsf{f} = \mathsf{m}$, a binary operation. The natural transformation $X_{\mathsf{m}}$ can be identified with an arrow $X \otimes X \xrightarrow{\mu} X$ in $\mathbb{C}$, which gives an inner binary operation on $X$.

| in $\mathbb{L}$ | in $\mathbf{CAT}$ |
|---|---|
| $2$ | $\mathbf{1} \xrightarrow{X_2 = (X,X)} \mathbb{C}^2$ |
| $\downarrow \mathsf{m}$ | $\Vert \quad \swarrow X_{\mathsf{m}} \quad \downarrow \llbracket \mathsf{m} \rrbracket = \otimes$ |
| $1$ | $\mathbf{1} \xrightarrow[X]{} \mathbb{C}$ |

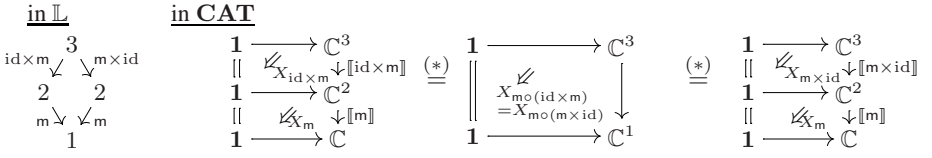
How do such inner operations on $X$ satisfy equations as specified in $\mathbb{L}$? The key is the coherence condition[9] on mediating 2-cells: it requires $X_{\mathrm{id}} = \mathrm{id}$ concerning identities; and $X_{\mathsf{g} \circ \mathsf{f}} = X_{\mathsf{g}} \bullet (\llbracket \mathsf{g} \rrbracket \circ X_{\mathsf{f}})$ concerning composition (as on the right). The following example illustrates how such coherence induces equational properties.

$$X_{\mathsf{g}\circ\mathsf{f}} = \begin{array}{ccc} \mathbf{1} & \longrightarrow & \mathbb{C}^l \\ \Vert & \swarrow X_{\mathsf{f}} & \downarrow \llbracket \mathsf{f} \rrbracket \\ \mathbf{1} & \longrightarrow & \mathbb{C}^k \\ \Vert & \swarrow X_{\mathsf{g}} & \downarrow \llbracket \mathsf{g} \rrbracket \\ \mathbf{1} & \longrightarrow & \mathbb{C}^n \end{array}$$

**Example 3.5.** A monoid object in a strictly monoidal category is an example of an $\mathbb{L}$-object in an $\mathbb{L}$-category. Here we take $\mathbb{L} = \mathbf{Mon}$, the theory of monoids.

[9] This is part of the notion of lax natural transformations; see [4].

For illustration, let us here derive associativity of multiplication $X \otimes X \xrightarrow{\mu} X$. In the current setting the multiplication $\mu$ is identified with a mediating 2-cell $X_{\mathsf{m}}$ as above. The coherence condition yields the two equalities $(*)$ below.




Now it is not hard to see that: the composed 2-cell on the left corresponds to $X^3 \xrightarrow{X\times\mu} X^2 \xrightarrow{\mu} X$; and the one on the right corresponds to $X^3 \xrightarrow{\mu\times X} X^2 \xrightarrow{\mu} X$. The equalities $(*)$ above prove that these two arrows $X^3 \rightrightarrows X$ are identical.

### 3.5 Microcosm Structures in Coalgebras

In this section we return to our original question and apply the framework we just introduced to coalgebraic settings. First we present some basic results, which are used later in our main result of general compositionality. The constructs in Section 2 (such as sync) will appear again, now in their generalized form. Some details and proofs are omitted here due to lack of space. They will appear in the forthcoming extended version of this paper, although the diligent reader will readily work them out.

Let $\mathbb{C}$ be an $\mathbb{L}$-category, and $F : \mathbb{C} \to \mathbb{C}$ be a functor. We can imagine that, for the category $\mathbf{Coalg}_F$ to carry an $\mathbb{L}$-structure, $F$ needs to be somehow compatible with $\mathbb{L}$; it turns out that the following condition is sufficient. It is weaker than $F$'s being an $\mathbb{L}$-functor (see Definition 3.2).

**Definition 3.6 (Lax $\mathbb{L}$-functor).** A functor $F : \mathbb{C} \to \mathbb{D}$ between $\mathbb{L}$-categories is said to be a *lax* $\mathbb{L}$*-functor* if it is identified with[10] some lax natural transformation $\mathbb{L} \xrightarrow[\mathbb{D}]{\mathbb{C}} \mathbf{CAT}$ which is product-preserving (i.e. $F \circ H$ is strictly natural; see Definition 3.4).

Lax $\mathbb{L}$-endofunctors are natural generalization of functors with sync as in Section 2. To illustrate this, look at the lax naturality diagram on the right for a binary operation $\mathsf{m}$. Here we denote the outer interpretation $[\![\mathsf{m}]\!]$ by $\otimes$. The 2-component is $F_2 = (F, F)$ because the lax natural transformation $F$ is product-preserving. The mediating 2-cell $F_{\mathsf{m}}$ can be identified with a natural transformation $FX \otimes FY \to F(X \otimes Y)$; this is what we previously called sync. Moreover, $F_{\mathsf{m}}$ (as generalized sync) is automatically compatible with equational properties (as in Theorem 2.4); this is because of the coherence condition on mediating 2-cells like "$F_{\mathsf{g}\circ\mathsf{f}}$ is a suitable composition of $F_{\mathsf{g}}$ after $F_{\mathsf{f}}$."


The following results follow from a more general result concerning the notion of *inserters*, namely: when $G$ is an oplax $\mathbb{L}$-functor and $F$ is a lax $\mathbb{L}$-functor, then the inserter $Ins(G, F)$ is an $\mathbb{L}$-category.

[10] Meaning: $F : \mathbb{C} \to \mathbb{D}$ is the 1-component of such a lax natural transformation $\mathbb{C} \Rightarrow \mathbb{D}$.

**Proposition 3.7.** *1. Let $\mathbb{C}$ be an $\mathbb{L}$-category and $F : \mathbb{C} \to \mathbb{C}$ be a lax $\mathbb{L}$-functor. Then $\mathbf{Coalg}_F$ is an $\mathbb{L}$-category; moreover the forgetful functor $\mathbf{Coalg}_F \xrightarrow{U} \mathbb{C}$ is a (strict, non-lax) $\mathbb{L}$-functor.*

*2. Given a microcosm model $X \in \mathbb{C}$ for $\mathbb{L}$, the slice category $\mathbb{C}/X$ is an $\mathbb{L}$-category; moreover the functor $\mathbb{C}/X \xrightarrow{\mathrm{dom}} \mathbb{C}$ is an $\mathbb{L}$-functor.* □

Note that $\mathbf{Coalg}_F$ being an $\mathbb{L}$-category means not only that operations are interpreted in $\mathbf{Coalg}_F$ but also that all the equational properties specified in $\mathbb{L}$ are satisfied in $\mathbf{Coalg}_F$. Therefore this result generalizes Theorem 2.4.

Concretely, an operation $\mathsf{f} : k \to 1$ in $\mathbb{L}$ is interpreted in $\mathbf{Coalg}_F$ and $\mathbb{C}/X$ as follows, respectively.

$$\begin{pmatrix} FX_1 & & FX_k \\ \uparrow c_1 & \cdots & \uparrow c_k \\ X_1 & & X_k \end{pmatrix} \mapsto \begin{array}{c} F[\![\mathsf{f}]\!](\overrightarrow{X}) \\ \uparrow(F_{\mathsf{f}})_{\overrightarrow{X}} \\ [\![\mathsf{f}]\!](\overrightarrow{FX}) \\ \uparrow[\![\mathsf{f}]\!](\overrightarrow{c}) \\ [\![\mathsf{f}]\!](\overrightarrow{X}) \end{array} \qquad \begin{pmatrix} Y_1 & & Y_k \\ \downarrow y_1 & \cdots & \downarrow y_k \\ X & & X \end{pmatrix} \mapsto \begin{array}{c} [\![\mathsf{f}]\!](\overrightarrow{Y}) \\ \downarrow[\![\mathsf{f}]\!](\overrightarrow{y}) \\ [\![\mathsf{f}]\!](\overrightarrow{X}) \\ \downarrow X_{\mathsf{f}} \\ X \end{array}$$

Compare these with (5) and (6); these make an essential use of $F_{\mathsf{f}}$ and $X_{\mathsf{f}}$ which generalize sync and $\|$ in Section 2, respectively.

**Proposition 3.8.** *1. A lax $\mathbb{L}$-functor preserves $\mathbb{L}$-objects. Hence so does an $\mathbb{L}$-functor.*

*2. A final object of an $\mathbb{L}$-category $\mathbb{C}$, if it exists, is an $\mathbb{L}$-object. The inner $\mathbb{L}$-structure is induced by finality.* □

We can now present our main result. It generalizes Theorem 2.1, hence is a generalized version of the "coalgebraic compositionality" equation (4).

**Theorem 3.9 (General compositionality).** *Let $\mathbb{C}$ be an $\mathbb{L}$-category and $F : \mathbb{C} \to \mathbb{C}$ be a lax $\mathbb{L}$-functor. Assume further that $\zeta : Z \xrightarrow{\cong} FZ$ is the final coalgebra. Then the functor* $\mathsf{beh} : \mathbf{Coalg}_F \to \mathbb{C}/Z$ *is a (non-lax) $\mathbb{L}$-functor. It makes the following diagram of $\mathbb{L}$-functors commute.*

$$\begin{array}{ccc} \mathbf{Coalg}_F & \xrightarrow{\mathsf{beh}} & \mathbb{C}/Z \\ & {\scriptstyle U}\searrow \quad \swarrow{\scriptstyle \mathrm{dom}} & \\ & \mathbb{C} & \end{array}$$

□

The proof is straightforward by finality. Here $\mathbf{Coalg}_F$ is an $\mathbb{L}$-category (Proposition 3.7.1). So is $\mathbb{C}/Z$ because: $\zeta \in \mathbf{Coalg}_F$ is an $\mathbb{L}$-object (Proposition 3.8.2); $Z = U\zeta$ is an $\mathbb{L}$-object (Propositions 3.8.1 and 3.7.1); hence $\mathbb{C}/Z$ is an $\mathbb{L}$-category (Proposition 3.7.2).

We have also observed some facts which look interesting but are not directly needed for our main result (Theorem 3.9). They include: the category $\mathbb{L}\text{-}\mathbf{obj}_{\mathbb{C}}$ of $\mathbb{L}$-objects in $\mathbb{C}$ and morphisms between them forms the lax limit of a diagram $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$; the simplicial category $\Delta$ is the "universal" microcosm model for $\mathbf{Mon}$ (cf. [20, Proposition VII.5.1]). The details will appear in the forthcoming extended version.

## 4 Conclusions and Future Work

In this paper we have observed that the microcosm principle (as called by Baez and Dolan) brings new mathematical insights into computer science. Specifically, we have looked into parallel composition of coalgebras, which would serve as a mathematical basis for the study of concurrency. As a purely mathematical expedition, we have presented a 2-categorical formalization of the microcosm principle, where an algebraic theory is presented by a Lawvere theory. Turning back to our original motivation, the formalization was applied to coalgebras and yielded some general results which ensure compositionality and equational properties such as associativity.

There are many questions yet to be answered. Some of them have been already mentioned, namely: extending the expressive power of sync (Remark 2.3), and a proper treatment of "pseudo" algebraic structures (Section 3.3).

On the application side, one direction of future work is to establish a relationship between sync and *(syntactic) formats* for process algebras. Our sync represents a certain class of operational rules; formats are a more syntactic way to do the same. Formats which guarantee certain good properties (such as commutativity, see [23]) have been actively studied. Such a format should be obtained by translating e.g. a "commutative" sync into a format.

On the mathematical side, one direction is to identify more instances of the microcosm principle. Mathematics abounds with the (often implicit) idea of nested algebraic structures. To name a few: a topological space in a topos which is itself a "generalized topological space"; a category of domains which itself carries a "structure as a domain." We wish to turn such an informal statement into a mathematically rigorous one, by generalizing the current formalization of the microcosm principle. As a possible first step towards this direction, we are working on formalizing the microcosm principle for finitary monads which are known to be roughly the same thing as Lawvere theories.

Another direction is a search for $n$-folded nested algebraic structures. In the current paper we have concentrated on two levels of interpretation; an example with more levels might be found e.g. in an internal category in an internal category.

**Acknowledgments.** Thanks are due to Kazuyuki Asada, John Baez, Masahito Hasegawa, Bill Lawvere, Duško Pavlović, John Power and the participants of CALCO-jnr workshop 2007 including Alexander Kurz for helpful discussions and comments.

## References

1. Baez, J.C., Dolan, J.: Higher dimensional algebra III: $n$-categories and the algebra of opetopes. Adv. Math. 135, 145–206 (1998)
2. Barr, M., Wells, C.: Toposes, Triples and Theories. Springer, Berlin (1985)
3. Bartels, F.: On generalised coinduction and probabilistic specification formats. Distributive laws in coalgebraic modelling. PhD thesis, Free Univ. Amsterdam (2004)
4. Borceux, F.: Handbook of Categorical Algebra. Encyclopedia of Mathematics, vol. 50, 51, 52. Cambridge Univ. Press, Cambridge (1994)
5. Hasuo, I.: Generic forward and backward simulations. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 406–420. Springer, Heidelberg (2006)

6. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Comp. Sci. 3(4–11) (2007)
7. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
8. Hyland, M., Power, A.J.: Discrete Lawvere theories and computational effects. Theor. Comp. Sci. 366(1–2), 144–162 (2006)
9. Jacobs, B.: Categorical Logic and Type Theory. North Holland, Amsterdam (1999)
10. Jacobs, B.: Trace semantics for coalgebras. In: Adámek, J., Milius, S. (eds.) Coalgebraic Methods in Computer Science. Elect. Notes in Theor. Comp. Sci., vol. 106, Elsevier, Amsterdam (2004)
11. Jacobs, B.: Introduction to coalgebra. Towards mathematics of states and observations (2005) Draft of a book, www.cs.ru.nl/B.Jacobs/PAPERS/index.html
12. Jacobs, B.: A bialgebraic review of deterministic automata, regular expressions and languages. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 375–404. Springer, Heidelberg (2006)
13. Johnstone, P.T., Power, A.J., Tsujishita, T., Watanabe, H., Worrell, J.: An axiomatics for categories of transition systems as coalgebras. In: Logic in Computer Science, IEEE, Computer Science Press, Los Alamitos (1998)
14. Kick, M., Power, A.J., Simpson, A.: Coalgebraic semantics for timed processes. Inf. & Comp. 204(4), 588–609 (2006)
15. Klin, B.: From bialgebraic semantics to congruence formats. In: Workshop on Structural Operational Semantics (SOS 2004). Elect. Notes in Theor. Comp. Sci. 128, 3–37 (2005)
16. Klin, B.: Bialgebraic operational semantics and modal logic. In: Logic in Computer Science, pp. 336–345. IEEE Computer Society, Los Alamitos (2007)
17. Kock, A., Reyes, G.E.: Doctrines in categorical logic. In: Barwise, J. (ed.) Handbook of Mathematical Logic, pp. 283–313. North-Holland, Amsterdam (1977)
18. Lawvere, F.W.: Functorial Semantics of Algebraic Theories and Some Algebraic Problems in the Context of Functorial Semantics of Algebraic Theories. PhD thesis, Columbia University, 1963. Reprints in Theory and Applications of Categories 5, 1–121 (2004)
19. Lee, E.A.: Making concurrency mainstream, Invited talk at CONCUR 2006 (2006)
20. Mac Lane, S.: Categories for the Working Mathematician, 2nd edn. Springer, Berlin (1998)
21. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
22. Moggi, E.: Notions of computation and monads. Inf. & Comp. 93(1), 55–92 (1991)
23. Mousavi, M.R., Reniers, M.A., Groote, J.F.: A syntactic commutativity format for SOS. Inform. Process. Lett. 93(5), 217–223 (2005)
24. Nishizawa, K., Power, A.J.: Lawvere theories enriched over a general base. Journ. of Pure & Appl. Algebra (to appear, 2006)
25. Power, J., Turi, D.: A coalgebraic foundation for linear time semantics. In: Category Theory and Computer Science. Elect. Notes in Theor. Comp. Sci., vol. 29, Elsevier, Amsterdam (1999)
26. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comp. Sci. 249, 3–80 (2000)
27. Turi, D., Plotkin, G.: Towards a mathematical operational semantics. In: Logic in Computer Science, pp. 280–291. IEEE, Computer Science Press, Los Alamitos (1997)

# Systems of Equations Satisfied in All Commutative Finite Semigroups

Paweł Parys*

Warsaw University
parys@mimuw.edu.pl

**Abstract.** The following problem is considered: check if a system of equations has a solution in every commutative finite semigroup. It is shown that the problem is decidable, and NP-complete. The problem is related with the pumping lemma for regular languages.

## 1 Introduction

One of the most famous and deep algorithms existing in formal language theory is *Makanin's algorithm* [8]. The algorithm takes as an input an system of equations and decides whether the system has a solution in a free semigroup. It has been improved several times. The currently best version of Makanin's algorithm works in EXPSPACE [5] and occupies (including the proof of correctness) over forty pages. Recently new algorithms to decide solvability of general word equations, using different ideas, have been found [11,10]. The algorithm in [10] works in PSPACE.

A related problem is to check if a system of equations is satisfied in all semigroups simultaneously. However, it can be easily shown that a system of equations having a solution in the free semigroup, also has a solution in every other semigroup. In this sense, for equations the free semigroup is the most difficult of semigroups. This argument, however, fails for finite semigroups. There are systems, which have solutions in all finite semigroups, but not in the free semigroup. For instance in a finite semigroup, when we add an element to itself several times, we always start looping, which can cause the existence of a solution. It's easy to show that in every finite semigroup there exists an idempotent element (i.e. a solution of an equation $x \cdot x = x$). It is enough to take any element and to add it to itself appropriate number of times. This fails in the free semigroup (no empty words). Solving of other equations will be a generalization of this observation.

This paper is devoted to solving equations over finite semigroups. Formally the following problem may be considered: Given a system of equations (with variables and coefficients), decide if the system has a solution in every finite semigroup and for every evaluation of the coefficients in this semigroups. Intuitively speaking, an opponent chooses a semigroup and values of coefficients and we have to show values of variables such that the equations will be satisfied.

* Author supported by Polish government grant no. N206 008 32/0810.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 261–272, 2008.

For example consider the following system of equations:

$$\begin{cases} x \cdot x = x \\ a \cdot y \cdot b = x \end{cases}$$

In every finite semigroup and for every values of coefficients $a$ and $b$, we want to have such values of variables $x$ and $y$ that the system is satisfied. In other words we need such an idempotent $x$, which ,,begins" with $a$ and ,,ends" with $b$. One can easily prove that such an idempotent exists, regardless of the choice of the semigroup and the coefficients $a$ and $b$.

Checking if a system has a solution in every finite semigroup is interesting itself, but it has also some motivation. There is a correspondence between finite semigroups and regular languages, so questions about semigroups are also questions about regular languages. Solving equations in finite semigroups can be seen as a generalization of pumping. What does it mean that a system of equations has a solution in every finite semigroup? For any finite semigroup (finite automaton) we choose, there would exist such values of the variables such that the left and right sides evaluate to the same element. So using these values of variables we will deceive any semigroup: it cannot distinguish between the left and right side. This idea was used in recent work over tree-walking automata, where standard pumping lemmas proved to be inadequate, and nonexpressiveness results were shown by using equations in semigroups. For example in [1], for every semigroup and for every $a$ and $b$ they need to have $u$ and $v$ such that $u = u \cdot a \cdot u = u \cdot b \cdot v$ and $v = v \cdot a \cdot u = v \cdot b \cdot v$.

In this paper I concentrate on commutative semigroups. I show that solving equations in all finite commutative semigroups is not only decidable, but NP-complete.

**Theorem 1.1.** *The following problem is NP-complete: Given a system of word equations, decide if the system has a solution in every finite commutative semigroup.*

There is the following easy extension of that theorem (proved in Section 4.4).

**Corollary 1.2.** *The following problem is decidable: Given a closed positive $\Pi_2$ formula (i.e. of the form $\forall \ldots \forall \exists \ldots \exists$(positive sentence) where elementary sentences are equations) decide if it is true in every finite commutative semigroup.*

The non-commutative case is left open. Some preliminary results are described in Section 6.

*Related work.* Here we ask if for every choice of coefficients there exist values of variables. A natural extension of this question is deciding whether an arbitrary first-order formula is satisfied in any finite semigroup. The problem is undecidable [4], even for very special case of formulas. For commutative finite semigroups the problem was not considered yet, to the best of the author's knowledge. Checking if a first-order formula is satisfied in a free semigroup is undecidable [12].

However for a commutative free semigroup the problem is decidable, since it can be encoded in well known Presburger arithmetic.

Another connected problem is solving a system of equations in a given finite semigroup [7].

## 2 Notations and Definitions

Vectors of letters like $c_1, \ldots, c_m$ or $X_1, \ldots, X_n$ etc. will be denoted by $\bar{c}, \bar{X}, \ldots$ When I need to consider simultaneously several vectors of the same type, I use superscripts: $\bar{X}^{(1)}, \ldots, \bar{X}^{(h)}, \ldots$

As we have only commutative semigroups I use plus sign for describing the semigroup operation. For $a$ in a semigroup and $k \geq 1$ I write $k \cdot a$ for $a$ added to itself $k$ times.

I fix the following finite nonempty alphabets:

$\Sigma_0 = \{X_1, \ldots, X_\gamma\}$ — the alphabet of *variables*,
$\Sigma_1 = \{C_1, \ldots, C_\omega\}$ — the alphabet of *coefficients*.

An *interpretation of coefficients* in a semigroup $S$ is a vector $\bar{c} = (c_1, \ldots, c_\omega)$ of elements of $S$ (an element $c_i$ corresponds to a coefficient $C_i$). It is easier to treat an interpretation of coefficients as a part of a semigroup. A pair $(S, \bar{c})$ of a semigroup and an interpretation of coefficients in it will be called a *semigroup with coefficients*. Since now as a semigroup I will understand a semigroup with coefficients and sometimes I will simply write $S$ for $(S, \bar{c})$.

A *system of equations* $\bar{\phi}$ is a system of equalities of the form

$$\begin{cases} \phi_{11} = \phi_{12} \\ \quad \ldots \\ \phi_{m1} = \phi_{m2} \end{cases}$$

where $\phi_{11}, \ldots, \phi_{m1}, \phi_{12}, \ldots, \phi_{m2}$ are nonempty words over $\Sigma_0 \cup \Sigma_1$. Sometimes I will write plus signs between symbols in the equations (just for convenience). For a given semigroup with coefficients $(S, \bar{c})$ and vector $\bar{x}$ of elements of $S$, by $\phi_{js}(\bar{x})$ I denote evaluation of $\phi_{js}$ in semigroup $S$ with $c_i$ (and $x_i$) substituted for $C_i$ (and $X_i$). For a given semigroup with coefficients $(S, \bar{c})$, a *solution* of the system $\bar{\phi}$ is a vector $\bar{x} \in S$ such that $\phi_{i1}(\bar{x}) = \phi_{i2}(\bar{x})$ for all $1 \leq i \leq m$.

A *homomorphism* of semigroups with coefficients from $(S, \bar{c})$ to $(S', \bar{c}')$ is a homomorphism of semigroups $f\colon S \to S'$ such that $f(c_i) = c'_i$ for every $i$. See that if a system $\bar{\phi}$ has a solution $\bar{x}$ in $(S, \bar{c})$ and we have any homomorphism $f\colon (S, \bar{c}) \to (S', \bar{c}')$, then image of $\bar{x}$ is obviously a solution in $(S', \bar{c}')$.

We will shortly say that a system *has a solution in the class FinComm* if for every commutative finite semigroup with coefficients there exists a solution of the system. Note the quantifier alternation: for all semigroups and all values of the coefficients, one must be able to find a values of the variables that yield a solution. In the article I present an algorithm for solving the following problem:

Input: a system of equations.
Output: Has the system a solution in every commutative finite semigroup with coefficients?

What would happen if we've skipped the word "finite"? When a system has a solution in every commutative semigroup it also has in a free commutative semigroup. There is a homomorphism from the free commutative semigroup to any other, so from a solution in free commutative semigroup we get a solution in any other commutative semigroup. Therefore the problem would reduce to finding solutions in the free commutative semigroup, which is easy.

## 3 Special Form of Equations: Variables on Both Sides

**Definition 3.1.** *I will say that an equation is* balanced *if on its every side there is at least one variable. I will say that a system of equations is* balanced *if every equation is balanced.*

The problem is somehow easier if we consider only balanced systems. At the beginning I will solve the problem in this special case. The results from this section will be used later for solving the general case.

### 3.1 One Coefficient

At the very beginning we will consider an even simpler form of systems, where at most one coefficient is used in the system. A general balanced system will be later reduced to several such systems. The following two theorems tell us that it is sufficient to solve such systems over $\mathbb{Z}$.

**Theorem 3.2.** *Let $C$ be a coefficient and let $\bar{\phi}$ be a balanced system where no coefficients other than $C$ appear in the system. Then the following statements are equivalent:*

1. *the system $\bar{\phi}$ has a solution in FinComm;*
2. *for every $n \geq 2$ the system $\bar{\phi}$ has a solution in the group $\mathbb{Z}_n$ (integers modulo $n$) with an interpretation $C \mapsto 1$.*

Interpretation of coefficients other than $C$ doesn't matter, as they do not appear in the system, but for completeness we should fix it somehow. Condition 1 above could be replaced by the assertion for all finite semigroups, not only commutative (we do not use commutativity in the proof below).

In a proof of the theorem I will use the following fact:

**Fact 3.3.** *For a given element $c$ of a finite semigroup $S$ there exists $N$ such that $2N \cdot c = N \cdot c$.*

*Proof.* Look at all multiples of $c$. As there is only finite number of elements in $S$, there have to be $k \cdot c = (k+l) \cdot c$ for some $k, l \geq 1$. Adding to this equation $l \cdot c$ several times we get

$$k \cdot c = (k+l) \cdot c = (k+2l) \cdot c = \ldots = (k+kl) \cdot c.$$

Then adding $(l-1)k \cdot c$ to it we get $kl \cdot c = 2kl \cdot c$. So taking $N = kl$ we are done. □

*Proof (of Theorem 3.2).* 1 ⇒ 2. Obvious, because 2 is a special case of 1.

1 ⇐ 2. Fix a commutative finite semigroup $S$ with an interpretation $c \in S$ of the coefficient $C$, for which the system should have a solution. Let $N$ be such that $2N \cdot c = N \cdot c$ (from fact 3.3). We have the following two properties for every $k, l \geq 0$:

- it holds $(N + k) \cdot c + (N + l) \cdot c = (N + k + l) \cdot c$;
- if additionally $k \equiv l \pmod{N}$ it holds $(N + k) \cdot c = (N + l) \cdot c$.

Let $\bar{y}$ will be a solution of the system in $\mathbb{Z}_n$ (understood as numbers from 0 to $N-1$), which exists from point 2. We take $x_j = (N+y_j)\cdot c$ for all $1 \leq j \leq \gamma$. Then for every $1 \leq i \leq m$, $s = 1, 2$ we have $\phi_{is}(\bar{x}) = (N+\phi_{is}(\bar{y}))\cdot c$. Since $\bar{y}$ is a solution in $\mathbb{Z}_n$, we have $\phi_{i1}(\bar{y}) \equiv \phi_{i2}(\bar{y}) \pmod{N}$, so $(N + \phi_{i1}(\bar{y})) \cdot c = (N + \phi_{i2}(\bar{y})) \cdot c$, which means that $\bar{x}$ is a solution in $S$. □

**Theorem 3.4.** *Let $C$ be a coefficient and let $\bar{\phi}$ be a balanced system where no coefficients other than $C$ appear in the system. Then the following statements are equivalent:*

1. *for every $n \geq 2$ the system $\bar{\phi}$ has a solution in the group $\mathbb{Z}_n$ with an interpretation $C \mapsto 1$;*
2. *the system $\bar{\phi}$ has a solution in the group $\mathbb{Z}$ with an interpretation $C \mapsto 1$.*

*Proof.* 1 ⇐ 2. This implication is almost obvious. For every $n$ there is a homomorphism from $\mathbb{Z}$ to $\mathbb{Z}_n$ (taking numbers modulo $n$), so if we have a solution in $\mathbb{Z}$, we also have it in $\mathbb{Z}_n$.

1 ⇒ 2. In this theorem in fact we deal with classical systems of number equations in $\mathbb{Z}$ or $\mathbb{Z}_n$. The system can be written in the form of $\bar{\bar{a}} \cdot \bar{X} = \bar{b}$ where $\bar{\bar{a}}$ and $\bar{b}$ are a matrix and a vector of integer numbers and $\bar{X}$ is a vector of variables. For solving this system in $\mathbb{Z}$ we can perform a Gauss elimination on the pair $(\bar{\bar{a}}, \bar{b})$. To keep all the constants in $\mathbb{Z}$ we cannot divide a equation by a number, we can only multiply, but it is enough. The only difference from normal Gauss elimination (with division allowed) is that we are not able to get ones "on the diagonal", we get there arbitrary nonzero numbers. After possibly changing numeration of variables (order of columns in $\bar{\bar{a}}$) we get the following equivalent system

$$\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 & a_{1,k+1} & a_{1,k+2} & \dots & a_{1\gamma} \\ 0 & a_{22} & 0 & \dots & 0 & a_{2,k+1} & a_{2,k+2} & \dots & a_{2\gamma} \\ 0 & 0 & a_{33} & \dots & 0 & a_{3,k+1} & a_{3,k+2} & \dots & a_{3\gamma} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{kk} & a_{k,k+1} & a_{k,k+2} & \dots & a_{k\gamma} \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_k \\ X_{k+1} \\ X_{k+2} \\ \vdots \\ X_\gamma \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \\ b_{k+1} \\ \vdots \\ b_m \end{bmatrix}$$

where $a_{11}, \dots, a_{kk}$ are nonzero.

The same operations we can do, when we are considering the system over $\mathbb{Z}_n$, and we get the same system (with the exception that all numbers are treated modulo $n$). Here the system we get is not necessarily equivalent to the original one (when multiplying by a number which has common divisors with $n$, a equality may become true, when it wasn't). But if the original system had a solution, then this also has (for every $\mathbb{Z}_n$).

Firstly see that $b_{k+1} = b_{k+2} = \ldots = b_m = 0$. Otherwise for $b_i \neq 0$ we have a equation $0 = b_i$ which should be satisfied in all semigroups $\mathbb{Z}_n$, but is not satisfied in almost all of them, e.g. for any $n > b_i$.

Let $n = a_{11} \cdot a_{22} \cdot \ldots \cdot a_{kk}$. Let $x_1, \ldots, x_\gamma$ be a solution in this $\mathbb{Z}_n$. The $i$-th equation ($1 \leq i \leq k$) says that

$$a_{ii}x_i + a_{i,k+1}x_{k+1} + a_{i,k+2}x_{k+2} + \ldots + a_{i\gamma}x_\gamma \equiv b_i \pmod{a_{11} \cdot a_{22} \cdot \ldots \cdot a_{kk}}$$

from which we have

$$a_{ii}x_i + a_{i,k+1}x_{k+1} + a_{i,k+2}x_{k+2} + \ldots + a_{i\gamma}x_\gamma \equiv b_i \pmod{a_{ii}}$$

which means that

$$a_{ii} | (b_i - a_{i,k+1}x_{k+1} - a_{i,k+2}x_{k+2} - \ldots - a_{i\gamma}x_\gamma) \tag{1}$$

As a solution for $\mathbb{Z}$ we will take:

$$x'_i = \begin{cases} (b_i - a_{i,k+1}x_{k+1} - a_{i,k+2}x_{k+2} - \ldots - a_{i\gamma}x_\gamma) \cdot \frac{1}{a_{ii}} & \text{for } 1 \leq i \leq k \\ x_i & \text{for } k+1 \leq i \leq \gamma \end{cases}$$

Condition (1) guarantees that $x'_i$ is integer. It's easy to see that it is really a solution. □

### 3.2 Many Coefficients

**Definition 3.5.** *For a given balanced system $\bar{\phi}$ and a coefficient $C \in \Sigma_1$ we define a projection $\bar{\phi}^{(C)}$ as a system obtained from $\bar{\phi}$ by erasing all coefficients other than $C$.*

Notice that the assumption that $\bar{\phi}$ has a variable on every side (is balanced) guarantees that $\bar{\phi}^{(C)}$ also has a variable on every side. In particular the sides are nonempty, so we get a well defined system.

**Lemma 3.6.** *Let $\bar{\phi}$ be a balanced system. Then the following statements are equivalent:*

1. *the system $\bar{\phi}$ has a solution in FinComm;*
2. *for every coefficient $C \in \Sigma_1$ and $n \geq 2$ the system $\bar{\phi}^{(C)}$ has a solution in the group $\mathbb{Z}_n$ with an interpretation $C \mapsto 1$.*

*Proof.* 1 ⇒ 2. Almost obvious. Fix $C$ and $n$. As a special case of 1 we get that the system $\bar{\phi}$ has a solution in $\mathbb{Z}_n$ with interpretation $C \mapsto 1$ and $D \mapsto 0$ for all $D \in \Sigma_1$, $D \neq C$. But this solution is also a solution of $\bar{\phi}^{(C)}$, because evaluation of sides of $\bar{\phi}$ and $\bar{\phi}^{(C)}$ are the same (the only difference is that in $\bar{\phi}$ we add 0 several times).

1 ⇐ 2. Fix a commutative finite semigroup $S$ with coefficients $\bar{c}$. From 2 and Theorem 3.2 for every $C_i \in \Sigma_1$ we have a solution $\bar{x}^{(C_i)}$ of the projection $\bar{\phi}^{(C_i)}$ in $S$. As our solution of the whole system we take the sum of these solutions: $x_j = x_j^{(C_1)} + x_j^{(C_2)} + \ldots + x_j^{(C_\omega)}$ for every $1 \leq j \leq \gamma$. Then the evaluation of a side of an equation will be the sum of evaluations of sides of equations for one coefficient: $\phi_{ks}(\bar{x}) = \phi_{ks}^{(C_1)}(\bar{x}^{(C_1)}) + \phi_{ks}^{(C_2)}(\bar{x}^{(C_2)}) + \ldots + \phi_{ks}^{(C_\omega)}(\bar{x}^{(C_\omega)})$. This is because every $c_i$ or $x_j^{(C_i)}$ appears the same number of times in $\phi_{ks}(\bar{x})$ as in $\phi_{ks}^{(C_i)}(\bar{x}^{(C_i)})$ and does not appear in any other element. Of course we've used the assumption that the semigroup is commutative. □

As an immediate corollary of Lemma 3.6 and Theorem 3.4 we get the following theorem:

**Theorem 3.7.** *Let $\bar{\phi}$ be a balanced system. Then the following statements are equivalent:*

1. *the system $\bar{\phi}$ has a solution in FinComm;*
2. *for every coefficient $C \in \Sigma_1$ the system $\bar{\phi}^{(C)}$ has a solution in the group $\mathbb{Z}$ with an interpretation $C \mapsto 1$.*

So we've got an easy to check criterion for testing if such system has a solution in the class *FinComm*.

## 4 General Case

### 4.1 Everywhere Something Is One Everywhere

Now I will prove an important technical lemma, which simplifies further argumentation. Here we need a version of the lemma for commutative finite semigroups, but the same lemma is true for general finite semigroups or for any variety.

**Lemma 4.1.** *Let $\{\bar{\phi}^{(1)}, \ldots, \bar{\phi}^{(N)}\}$ be systems of equations such that in every commutative finite semigroup with coefficients some $\bar{\phi}^{(i)}$ has a solution. Then some of the systems $\bar{\phi}^{(i)}$ has a solution in every of these semigroups.*

*Proof.* Assume that for every system $\bar{\phi}^{(i)}$ there exists a semigroup $S_i$ in which there is no solution of $\bar{\phi}^{(i)}$. Look at the product semigroup $S = S_1 \times \ldots \times S_N$ (naturally interpretation of coefficient in the product is a sequence of its interpretations in every $S_i$). Some system $\bar{\phi}^{(i)}$ has to have a solution in $S$. But we have a homomorphism from $S$ to $S_i$ (a projection), so $\bar{\phi}^{(i)}$ has a solution in $S_i$ too. But we've assumed that it hasn't, we've got contradiction, so the theorem is true. □

### 4.2 Removing "Wrong" Variables

Let $\bar{\phi}$ be a system in which in some equations on one side there are only coefficients and on the other side there are also variables. We will replace the system by a number of simpler systems. For notational simplicity, assume that this happened in the first equation: that on the right side of this equation ($\phi_{12}$) there are only coefficients, and that on its left side ($\phi_{11}$) there is at least the variable $X_1$. For a word of coefficients $w \in \Sigma_1^*$ we define a system $\bar{\phi}^{(w)}$. It will be $\bar{\phi}$ in which we replace every occurrence of $X_1$ in every equation by the word $w$. We will be considering these systems for all nonempty subsequences of the right side of the first equation (in fact the order of coefficients in $w$ doesn't matter, as we have only commutative semigroups, so we can also think about all submultisets of the $\phi_{12}$). Obviously there are only finitely many of these subsequences.

**Theorem 4.2.** *For a system $\bar{\phi}$ as above, the system has a solution in FinComm if and only if for some $w$ — subsequence of $\phi_{12}$, the system $\bar{\phi}^{(w)}$ has a solution in FinComm.*

*Proof.* $\Leftarrow$. Let $w$ be this subsequence of $\phi_{12}$, for which $\phi^{(w)}$ has a solution in *FinComm*. Fix a commutative finite semigroup with coefficients $S$, for which we need a solution of $\bar{\phi}$. Let $\bar{x}$ be a solution of $\bar{\phi}^{(w)}$ there. As a solution of $\bar{\phi}$ we can take an evaluation of $w$ as $x_1$ and values from $\bar{x}$ as the other variables. Then the evaluation of sides of $\bar{\phi}$ and $\bar{\phi}^{(w)}$ are the same, so this is a solution of $\bar{\phi}$ (the only difference between $\bar{\phi}$ and $\bar{\phi}^{(w)}$ is that on the places of $X_1$ we have $w$, but they both evaluates to the same value).

$\Rightarrow$. According to Lemma 4.1 it is enough to show that in every commutative finite semigroup for every interpretation of coefficients at least one of these systems has a solution. Then the lemma would say that one of the systems would have a solution in every of these semigroups.

Fix a commutative finite semigroup with coefficients $S$, for which we need a solution of some system $\bar{\phi}^{(w)}$. We will construct a semigroup $S'$ and basing on a solution of $\bar{\phi}$ in it, we will construct a solution of some $\bar{\phi}^{(w)}$ in $S$. Let $N$ be the length of $\phi_{12}$ (the right side of the first equation, which contains only coefficients). The semigroup $S'$ will contain two disjoint parts:

1. all the elements of $S$;
2. all commutative words (multisets) of coefficients from $\Sigma_1$ up to length $N$.

Now we need to define an operation. Inside $S$ we just add in $S$. When adding between the parts, we evaluate the word of coefficients in $S$ and then add in $S$. When adding two words of coefficients, we concatenate it. If the result fits into second part (has length $\leq N$), we just take it. When it is longer, we evaluate it in $S$. The idea is that $S'$ for short words simulates free commutative semigroup. Words no longer than $N$ we remember as they are, for longer words we remember only their value in $S$.

We take an interpretation of coefficients in $S'$ such that a coefficient is interpreted as an one-letter word containing it. $S'$ is a commutative finite semigroup, so $\bar{\phi}$ has a solution $\bar{x}'$ in it. See that when something is in the first part, then

after adding anything it will remain in this part. The right side of the first equation, which contains just $N$ coefficients, in $S'$ will evaluate to itself in the second part. This means that $x'_1$ (and the whole left side) is also in the second part. Moreover, $x'_1$ is a submultiset of the right side. We will take $x'_1$ as the word $w$. Now we need to have a solution of $\bar{\phi}^{(w)}$ in $S$. As $x_i$ (for $2 \leq i \leq \gamma$) we will take $x'_i$ when it is in the first part or evaluation of $x'_i$ in $S$, when it is in the second part. It's easy to see that $\bar{\phi}^{(w)}$ gives now the same equalities, as $\bar{\phi}$ before (when an equation from $\bar{\phi}$ in $S'$ has given an equality on words, then it also holds after evaluation to $S$). So it's a solution in $S$ and we're done. □

### 4.3 The Algorithm

Theorem 4.2 almost gives us an algorithm. In every moment we will have a set of systems, about which we'd like to know if at least one of the systems has a solution in *FinComm*. At the very beginning the set contains only the original system. Then we repeatedly replace a system, in which in some equations variables are only on one side, by a number of systems, as described above. See that after such step, we get systems containing less variables, so the process has to finish. At the end we get a set of systems, in which every equation contains at least one variable on both sides or no variables at any side.

Now see what happens, if in an equation there are only coefficients. If the number of coefficients of every kind is equal on both sides (the sides are equal as multisets), then the equation is always satisfied, because our semigroups are commutative. In this case we can just remove this equation and we get equivalent system. Otherwise there is an coefficient $C$, which on left side appears $k$ times and on the right side $l$ times, where $k \neq l$. Such equation is not satisfied in *FinComm*, it is even very unlikely to be satisfied in any commutative finite semigroup. For example consider an additive group $\mathbb{Z}_{k+l+1}$ with interpretation $C \mapsto 1$ and $D \mapsto 0$ for all $D \in \Sigma_1$, $D \neq C$. Here left side evaluates to $k$ and right side to $l$, which aren't equal, so the equality isn't satisfied. This means that we can immediately say that system containing such equation cannot have a solution in *FinComm*.

Finally we get a set of balanced systems; we want to say if any of them has a solution in *FinComm*. But for such systems it can be determined by the algorithm from section 3.

### 4.4 Positive $\Pi_2$ Formulas

We will now prove Corollary 1.2. Here on the input we have a closed positive $\Pi_2$ formula instead of a system of equations. In fact original Theorem 1.1 (and the above algorithm) solves a situation, when only conjunctions are used. A formula with disjunctions can be normalized to a form saying that (in every commutative semigroup) for every choice of coefficients there exist values of variables such that some of given systems is satisfied. In the light of Lemma 4.1 we can equivalently say that some of these systems is satisfied in *FinComm*. So it is enough to check each of these systems separately using above algorithm—if one of them

is satisfied in *FinComm*, then the formula is true in every commutative finite semigroup.

However during the mentioned normalization, the size of a formula can grow exponentially, so we lose the NP complexity.

## 5 Complexity

In this section I will show that the problem is NP-complete. To talk about the complexity we need to know how we represent the system and how is its size defined. The easiest (but not good enough) way is to remember an equation as it is, as a list of coefficients and variables. When we use this representation, then during the operation of our algorithm the size of the system can grow exponentially, so it wouldn't work in NP. But we can do better: a side of a equation will be described by numbers of occurrences of every symbol (coefficient or variable). So when a symbol repeats $N$ times, the binary representation of $N$ will be remembered, which has a length of $\Theta(\log N)$.

Firstly I will show that the problem is in NP. Of course I will use the algorithm described above in the paper. Essentially the algorithm consists of two parts. In the first part we remove a variable from unbalanced equation several times and substitute for it some subsequence of the other side of the equation (see subsection 4.2). Note that in nondeterministic model we can just guess the correct substitution instead of checking all the possibilities. In the second part we solve a balanced system of equations over $\mathbb{Z}$. This can be done in NP (see [2], the solution will have polynomial size, so we can just guess it).

The only question is what is the length of the system after the first part. We need to show that it will grow at most polynomially. Let $V_i$ and $B_i$ be a total number of occurrences of variables and of coefficients in the system after the $i$-th step of the algorithm (in particular $V_0$ and $B_0$ are these numbers in the initial system). Note that the length of the initial system is at least $\Omega(\log|V_0+B_0|)$. In $i$-th step we substitute for a variable at most $B_{i-1}$ coefficients, and the variable occurs at most $V_{i-1}$ times, so

$$B_i \le B_{i-1} + V_{i-1}B_{i-1} = (V_{i-1}+1)B_{i-1}$$

The number of occurrences of variables even decrease, so $V_i \le V_{i-1}$. After $k$ steps we have: $V_k \le V_0$ and $B_k \le (V_0+1)^k B_0$. Let $N$ be the length of the initial system, $\gamma \le N$ — number of different variables and $\omega \le N$ — number of different coefficients. In each step we remove one variable, so there are at most $k \le \gamma \le N$ steps. In the resulting system we have $2m$ sides of $m$ equations, every of which needs length at most $\gamma \log V_k$ for describing variables and $\omega \log B_k$ for describing coefficients. So the length of the resulting system will be at most:

$$\begin{aligned} 2m(\gamma \log V_k + \omega \log B_k) &\le 2N^2(\log V_0 + \log((V_0+1)^N B_0)) = \\ &= 2N^2(\log V_0 + N\log(V_0+1) + \log B_0) \le \\ &\le O(N^3)O(\log|V_0+B_0|) \le O(N^4) \end{aligned}$$

so it's polynomial.

Now I will show that the problem is NP-hard. To do that I will reduce to it the NP-complete clique problem (defined in [6]). Almost the same proof would also work for a problem of solving systems of equations in the commutative free semigroup. Assume that we are looking for a clique of size $k$ in a graph $G = (V, E)$. Assume that $V = \{1, 2, \ldots, |V|\}$. We will have only one coefficient $C$. We will have a variable $X_i$ for each vertex and helper variables $X'_i$ and $X''_{ij}$ for each vertex and each pair of vertices. The equations are:

$$\begin{cases} X_1 + X_2 + \ldots + X_{|V|} = (|V| + k) \cdot C & \\ X_i + X'_i = 3 \cdot C & \text{for every } i \in V \\ X_i + X_j + X''_{ij} = 4 \cdot C & \text{for every } (i, j) \notin E \end{cases}$$

If there is a clique of size $k$, then we will have a solution in every commutative finite semigroup (and even in the free commutative semigroup). We take $X_i = 2 \cdot C$ if the vertex $i$ is in the clique or $X_i = C$ if it isn't. First equation is satisfied, because there are exactly $k$ vertices in the clique. The equations of the second and third type can be satisfied by taking $X'_i$ and $X''_{ij}$ equal to $C$ or $2 \cdot C$. When there is no edge $(i, j)$, then at most one of vertices $i$ and $j$ is in the clique, so it's possible to satisfy the equations of the third type.

When there is a solution in every commutative finite semigroup, then there is also some solution $\bar{x}$ in the following semigroup $S$: Elements of $S$ are $\{1, 2, \ldots, M\}$ for large enough $M = \max(5, |V| + k + 1)$. The operation $a + b$ is defined as $\min(M, a + b)$. The coefficient $C$ evaluates to 1. From equations of the second type we see that $x_i = 1$ or 2. We take vertex $i$ to a clique iff $x_i = 2$. The equations of the third type guarantees that two vertices cannot be in the clique if there is no edge between them. First equation says that the clique has size $k$.

## 6 Other Questions

It may be interesting to check if a system has a solution in other classes of semigroups, for example all finite semigroups or all idempotent finite semigroups.

For idempotent finite semigroups this question is easy, as we know that there are only finitely many idempotent semigroups with a given number of generators (see [9]). Moreover all these semigroups can be listed, because there is a formula for maximum number of elements in such semigroup. It is enough to check if a system has a solution in all semigroups generated by our coefficients, which we can do one semigroup after another. For any semigroup, when there is a solution in a subsemigroup generated by coefficients, it is also a solution in whole semigroup. It may be interesting to find an algorithm with better complexity.

It remains open how to check that in class of all finite semigroups. It is not difficult to prove that in every finite semigroup $S$, there are elements $u, v$ such that the subsemigroup $\{usv : s \in S\}$ is a two-sided ideal and a group. So there is a hope that (with more effort that for commutative case) the problem could be reduced to solving systems of equations over all finite groups. However unlike in the commutative case, there are systems of equations satisfied in all finite

groups, but not in the free group [3]. So the reduction to finite groups do not give a solution yet.

It would be also interesting to know if the problem of checking if arbitrary first order formula is satisfied in every commutative finite semigroup is decidable.

## References

1. Bojańczyk, M., Colcombet, T.: Tree-walking automata cannot be determinized. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, p. 246–256. Springer, Heidelberg (2004)
2. Borosh, I., Treybig, L.B.: Bounds on positive integral solutions of linear diophantine equations. Proc. Amer. Math. Soc. 55(2), 299–304 (1976)
3. Coulbois, T., Khelif, A.: Equations in free groups are not finitely approximable. Proc. Amer. Math. Soc. 127(4), 963–965 (1999)
4. Gurevich, Y.: The word problem for certain classes of semigroups. Algebra and Logic 5(5), 25–35 (1966) (in Russian)
5. Gutiérrez, C.: Satisfiability of word equations with constants is in exponential space. In: Foundations of Computer Science, pp. 112–120 (1998)
6. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
7. Klima, O., Tesson, P., Therien, D.: Dichotomies in the complexity of solving systems of equations over finite semigroups. Theor. Comp. Sys. 40(3), 263–297 (2007)
8. Makanin, G.S.: The problem of solvability of equations in a free semigroup. USSR Sbornik 32, 129–198 (1977)
9. McLean, D.: Idempotent semigroups. The American Mathematical Monthly 61(2), 110–113 (1954)
10. Plandowski, W.: Satisfiability of word equations with constants is in pspace. In: Foundations of Computer Science, pp. 731–742 (1999)
11. Plandowski, W., Rytter, W.: Application of lempel-ziv encodings to the solution of word equations. In: International Colloquium on Automata, Languages and Programming, pp. 731–742 (1998)
12. Tarski, A., Mostowski, A., Robinson, M.R.: Undecidable theories. North-Holland, Amsterdam (1953)

# Optimal Lower Bounds on Regular Expression Size Using Communication Complexity

Hermann Gruber and Jan Johannsen

Institut für Informatik, LMU München
Oettingenstr. 67, 80538 München, Germany
{gruberh,jjohanns}@tcs.ifi.lmu.de

**Abstract.** The problem of converting deterministic finite automata into (short) regular expressions is considered. It is known that the required expression size is $2^{\Theta(n)}$ in the worst case for infinite languages, and for finite languages it is $n^{\Omega(\log\log n)}$ and $n^{O(\log n)}$, if the alphabet size grows with the number of states $n$ of the given automaton. A new lower bound method based on communication complexity for regular expression size is developed to show that the required size is indeed $n^{\Theta(\log n)}$.

For constant alphabet size the best lower bound known to date is $\Omega(n^2)$, even when allowing infinite languages and nondeterministic finite automata. As the technique developed here works equally well for deterministic finite automata over binary alphabets, the lower bound is improved to $n^{\Omega(\log n)}$.

## 1 Introduction

One of the most basic theorems in formal language theory is that every finite automaton can be converted into an equivalent regular expression, and vice versa [10]. While algorithms accomplishing these tasks have been known for a long time, there has been a renewed interest in these classical problems during the last few years. For instance, new algorithms for converting regular expressions into finite automata outperforming classical algorithms have been found only recently, as well as a matching lower bound of $\Omega(n \cdot (\log n)^2)$ on the minimum number of transitions required by any equivalent nondeterministic finite automaton (NFA). The lower bound is, however, only reachable for growing alphabets, and a better algorithm is known for constant alphabet size, see [20] for the current state of the art.

In contrast, much less is known about the converse direction, namely of converting finite automata into regular expressions. Apart from the fundamental nature of the problem, some applications of converting finite automata into regular expressions lie in control flow normalization, including uses in software engineering such as automatic translation of legacy code [17]. All known algorithms covering the general case of infinite languages are based on the classical ones, which are compared in the survey [18]. The drawback is that all of these (structurally similar) algorithms return expressions of size $2^{O(n)}$ in the worst case, and Ehrenfeucht and Zeiger exhibit a family of languages over a growing alphabet for which

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 273–286, 2008.

this exponential blow-up is inevitable [2]. This leads to the quest for identifying structural restrictions on the underlying transition graph of the given finite automaton that can guarantee a shorter equivalent regular expression [3,16], as well as for heuristics improving the classical algorithms [6,1]. Another possibility is to concentrate on subfamilies of regular languages. For the important special case of unary languages, it has been established that every $n$-state nondeterministic finite automaton can be converted in polynomial time into an equivalent regular expression of polynomial size [13]. And for finite languages, there exist equivalent regular expressions of size at most $n^{O(\log n)}$ obtained by a classical construction, which is carefully analyzed in [3]. In contrast, results from [2] show that size $n^{\Omega(\log\log n)}$ can be necessary for finite languages—at least for a growing alphabet.

Although there remains a considerable gap between the best known upper bounds and the lower bounds given some 30 years ago, to the best of our knowledge, only little progress has been made on this problem. The most preeminent gap between the upper and lower bounds presented in [2] is in the case of finite languages. There the upper and lower bounds of $n^{O(\log n)}$ and $n^{\Omega(\log\log n)}$, respectively, are essentially the best ones known to date. We close this gap, giving that the blow-up for finite languages is $n^{\Theta(\log n)}$ in the worst case, when switching the representation from a finite automaton to a regular expression. To this end, we develop a new lower bound technique for regular expression size based on communication complexity. Ellul et al. [3] prove a lower bound of $\Omega(n^2)$ on the size of regular expressions for a finite language over the binary alphabet by a reduction to Boolean circuit complexity. We improve this approach by harnessing a technique used to obtain better lower bounds for *monotone* Boolean circuits, using the communication complexity of search problems as introduced by Karchmer and Wigderson [8]. Our approach shows that the lower bound can even be realized for an $n$-state *deterministic* finite automaton over a *binary* alphabet.

We also show that a family of finite languages (over a growing alphabet) studied by Ehrenfeucht and Zeiger [2] captures the combinatorial core of the conversion problem for finite languages, as these form in some precise sense the hardest languages for this problem. We then use this to obtain a slight improvement of the best known upper bounds on this conversion problem.

## 2 Preliminaries

### 2.1 Formal Languages

We assume the reader to be familiar with the basic notions in formal language and automata theory as contained in [7]. In particular, let $\Sigma$ be an alphabet and $\Sigma^*$ the set of all words over the alphabet $\Sigma$, including the empty word $\varepsilon$. The length of a word $w$ is denoted by $|w|$, where $|\varepsilon| = 0$, and the total number of occurrences of the alphabet symbol $a$ in $w$ is denoted by $|w|_a$. In this paper we mainly deal with a special class of finite languages called homogeneous languages. A finite language $L \subset \Sigma^*$ is *homogeneous* if all words in the language have the same length. In order to fix the notation, we briefly recall the definition of regular expressions and the languages described by them:

Let $\Sigma$ be an alphabet. The *regular expressions* over $\Sigma$ and the languages that they denote are defined recursively as follows: $\emptyset$ is a regular expression and denotes the empty language; for $a \in \Sigma \cup \{\varepsilon\}$, $a$ is a regular expression and denotes the language $\{a\}$; if $e$ and $f$ are regular expressions denoting languages $E$ and $F$, then $(e + f)$, $(e \cdot f)$ and $(e)^*$ are regular expressions denoting the languages $E \cup F$, $E \cdot F$ and $E^*$, respectively. Finally, the language described by the regular expression $E$ is denoted by $L(E)$.

For convenience, parentheses are sometimes omitted and the product is simply written as juxtaposition. The priority of operators is specified in the usual fashion: product is performed before disjunction, and star before both product and disjunction. We also write sometimes $L_1 + L_2$ to denote the union of the languages $L_1$ and $L_2$. The *alphabetic width* (or *size*) $\mathrm{alph}(E)$ of a regular expression $E$ is defined as the total number of occurrences of symbols in $\Sigma$ in $E$. For a regular language $L$ we define $\mathrm{alph}(L)$ as the minimum alphabetic width among all regular expressions describing $L$. As we will be primarily concerned with small regular expressions, we recall the notion of uncollapsible regular expressions [3]:

Let $E$ be a regular expression. We say that $E$ is *uncollapsible* if all of the following conditions hold: If $E$ contains the symbol $\emptyset$, then $E = \emptyset$; the expression $E$ contains no subexpression of the form $FG$ or $GF$, with $L(F) = \{\varepsilon\}$; if $E$ contains a subexpression of the form $F + G$ or $G + F$ with $L(F) = \{\varepsilon\}$, then $\varepsilon \notin L(G)$; if $E$ contains a subexpression of the form $F^*$, then $L(F) \neq \{\varepsilon\}$.

The reader might have noticed that we have added a fourth condition not present in the original definition. This condition ensures that the star operator cannot occur in uncollapsible regular expressions describing finite languages. It is easily seen that for every collapsible regular expression, there is an uncollapsible one specifying the same language of at most the same size.

### 2.2 Communication Complexity

Let $X, Y, Z$ be finite sets and $R \subseteq X \times Y \times Z$ a ternary relation on them. In the search problem $R$, we have Alice given some input $x \in X$, Bob is given some input $y \in Y$. Initially, no party knows the other's input, and Alice and Bob both want to output some $z$ such that $(x, y, z) \in R$, by communicating as few bits as possible. A *communication protocol* is a binary tree with each internal node $v$ labeled either by a function $a_v : X \to \{0, 1\}$ if Alice transmits at this node, or $b_v : Y \to \{0, 1\}$ if Bob transmits at this node. Each leaf is labeled by an output $z \in Z$. We say that a protocol solves the search problem for relation $R$ if for every input pair $(x, y) \in X \times Y$, walking down the tree according to the functions $a_v$ and $b_v$ leads to a leaf labeled with some $z$ satisfying $(x, y, z) \in R$. The overall number of bits transmitted for a given input pair $(x, y) \in X \times Y$ and a given protocol is then equal to the length of the walk just described; and the maximum length among these walks equals the depth of the tree. The *(deterministic) communication complexity* $D(R)$ is now defined as the minimum depth among all communication protocols solving $R$, and the *protocol partition number* $C^P(R)$ denotes the minimum number of leaves among all protocols solving the search problem for $R$.

Of course, there exist protocols whose depth is even linear in the number of leaves, but a standard argument about balancing binary trees shows that every such deep protocol can be transformed into a shallow protocol, see e.g. [11, ch. 2]:

**Lemma 1.** $\log C^P(R) \geq \frac{1}{3} D(R)$. □

An important fact about these two complexity measures is a close correspondence with the complexity of Boolean circuits and Boolean formulas, respectively. This relation is based on search problems, which we define next in terms of languages.

For a homogeneous language $\emptyset \subset L \subset \Sigma^n$, the *search problem associated with* $L$ is a ternary relation $R_L \subseteq L \times (\Sigma^n \setminus L) \times [n]$ defined by: $(v, w, i) \in R_L$ iff $v_i \neq w_i$. Karchmer and Wigderson established the following connection to circuit complexity [8]: Take $\Sigma = \{0, 1\}$. If we naturally identify each set $L \subseteq \{0, 1\}^n$ with its characteristic $n$-bit Boolean function, then $D(R_L)$ equals the minimum depth of a Boolean circuit (over the standard basis) computing the characteristic function of $L$. Moreover, $C^P(R_L)$ equals the minimum number of variable occurrences among all Boolean formulas representing $L$.

Proving superpolynomial lower bounds on formula size for specific functions turns out to be an extremely difficult open problem. Fortunately, this is no longer true if we consider monotone Boolean formulas [8]. A similar class of search problems can be defined for the latter setup:

Let $(\Sigma, <) = (a_1 < a_2 < \cdots < a_k)$ be an ordered alphabet. This order on $\Sigma$ is extended componentwise to a partial order on $\Sigma^n$. The *upward closure* of a homogeneous language $L \subseteq \Sigma^n$ (w.r.t. this partial order) is defined as the set

$$\uparrow(L) = \{w \in \Sigma^n \mid u \leq w \text{ for some } u \in L\}.$$

A homogeneous language $L \subseteq \Sigma^n$ is called *monotone*, if $L = \uparrow(L)$. For a monotone homogeneous language $\emptyset \subset L \subset \Sigma^n$, the *monotone search problem associated with* $L$, denoted by $R_L^m$, is defined by $(v, w, i) \in R_L^m$ iff both $(v, w, i) \in R_L$ and overmore $v_i > w_i$. For $\Sigma = \{0, 1\}$ with $0 < 1$, the measure $C^P(R_L^m)$ equals the minimum formula size among all *monotone* Boolean formulas representing $L$, and an similar correspondence holds for $D(R_L^m)$ and *monotone* circuit depth [8].

For more background on communication complexity, the reader might want to consult the book [11].

## 3 A New Lower Bound Technique for Regular Expression Size

The goal of this section is to relate the alphabetic width of a homogeneous language to the protocol partition number of the monotone search problem associated with that language.

**Definition 2.** *A regular expression $E$ describing a homogeneous language is called a* homogeneous expression*, if none of the symbols $\emptyset$, $\varepsilon$ and $^*$ occur in $E$, or $L(E)$ is empty and $E = \emptyset$.*

**Lemma 3.** *For $n \geq 1$, let $L \subseteq \Sigma^n$ be a homogeneous language. If $E$ is an uncollapsible regular expression describing $L$, then $E$ is a homogeneous expression.*

*Proof.* For the case $L(E) = \emptyset$, the statement immediately follows from the definitions. Assume $E$ is uncollapsible and $\emptyset \subset L(E) \subseteq \Sigma^n$. We can rule out that any subexpression $F$ with $L(F) = \emptyset$ occurs in $E$: Every regular expression denoting the empty language contains the symbol $\emptyset$ at least once. Next, finiteness of the described languages is invariant under the operations $+$ and $\cdot$, but not by the Kleene star: For any regular expression $F$, the set denoted by $F^*$ is infinite unless $L(F) = \emptyset$ or $\{\varepsilon\}$. We have already ruled out the existence of $\emptyset$ symbols in $E$. Since $E$ is uncollapsible, it does not contain any subexpression of the form $F^*$ with $L(F) = \{\varepsilon\}$ either. Thus, the language $L(E)$ being finite, $E$ cannot have any subexpression of the form $F^*$.

Finally, we rule out the possibility that $\varepsilon$ occurs in $E$: As all words in $L(E)$ are of length $n$, we make the following observation: If $E$ contains a subexpression of the form $F + G$, then there exists $m \leq n$ such that both $L(F)$ and $L(G)$ contain only strings of length $m$. If $\mathrm{alph}(E) \leq 1$, then clearly $E$ has no $\varepsilon$-subexpression. Assume $\mathrm{alph}(E) > 1$ and $\varepsilon$ occurs in $E$. Since $E$ is uncollapsible, $E$ contains a subexpression of the form $F + \varepsilon$ with $\varepsilon \notin F$ and $F \neq \emptyset$. But then $F + \varepsilon$ describes a set of strings having different lengths, a property which is inherited to $E$, since $E$ has no subexpressions describing the empty language. □

The next proposition shows that for homogeneous languages, the upward closure operator $\uparrow$ commutes with union and concatenation.

**Proposition 4.** *For homogeneous languages $L_1$ and $L_2$,*

$$\uparrow(L_1) + \uparrow(L_2) = \uparrow(L_1 + L_2) \text{ and } \uparrow(L_1) \cdot \uparrow(L_2) = \uparrow(L_1 \cdot L_2)\,. \qquad \square$$

We establish next that homogeneous monotone languages can be described by regular expressions in some normal form, and that the conversion into this normal form increases the expression size at most by a factor of $|\Sigma|$.

A homogeneous expression is called a *sum* if it uses $+$ as the only operator, i.e. it is of the form $(b_1 + \ldots + b_m)$ for $b_i \in \Sigma$. Let $E$ be a homogeneous expression and $F$ a subexpression of $E$. The subexpression $F$ is called a *maximal sum* in $E$ if $F$ is a sum, but each subexpression $G$ having $F$ as a proper subexpression is not a sum. Note that the maximal sums in an expression each describe a subset of $\Sigma$. For a homogeneous expression $E$, the number of maximal sums in $E$ is denoted by $\mathrm{s}(E)$. Since any non-redundant sum is of size at most $|\Sigma|$ and contains at least one alphabetical symbol, we get $\mathrm{s}(L) \leq \mathrm{alph}(L) \leq |\Sigma| \cdot \mathrm{s}(L)$ for every homogeneous language $L$. A homogeneous expression $E$ is called *monotone* if each maximal sum $F$ in $E$ describes a monotone language, that is $L(F) = \uparrow(L(F))$.

**Lemma 5.** *For each homogeneous expression $E$ over an ordered alphabet $\Sigma$, there exists a monotone expression $F$ with $\mathrm{s}(F) = \mathrm{s}(E)$ and $L(F) = \uparrow(L(E))$. In particular, if $E$ describes a monotone language, then $L(F) = L(E)$.*

*Proof.* The claim is shown by induction on $\mathrm{s}(E)$. In the base case $\mathrm{s}(E) = 1$, $E$ is itself a sum. Let $b$ be the minimal letter occurring in $E$, and let $b_1, \ldots, b_m$ be

those letters in $\Sigma$ with $b_i \geq b$. We set $F := (b_1 + \ldots + b_m)$, and we clearly have $L(F) = \uparrow(L(E))$ as well as $\mathrm{s}(F) = \mathrm{s}(E) = 1$, hence the claim holds.

Now let $\mathrm{s}(E) > 1$, and thus $E = E_1 \oplus E_2$ where the symbol $\oplus$ stands for one of the operators $+$ or $\cdot$, and in the latter case, $E_1$ and $E_2$ are not sums. Thus we have $\mathrm{s}(E) = \mathrm{s}(E_1) + \mathrm{s}(E_2)$ and hence $\mathrm{s}(E_i) < \mathrm{s}(E)$ for $i = 1, 2$. By the induction hypothesis, we get expressions $F_i$ with $L(F_i) = \uparrow(L(E_i))$ and $\mathrm{s}(F_i) = \mathrm{s}(E_i)$. We set $F := F_1 \oplus F_2$, and we obtain $\mathrm{s}(F) = \mathrm{s}(F_1) + \mathrm{s}(F_2) = \mathrm{s}(E_1) + \mathrm{s}(E_2) = \mathrm{s}(E)$. By Proposition 4, we obtain $L(F) = \uparrow(L(E_1)) \oplus \uparrow(L(E_2)) = \uparrow(L(E_1) \oplus L(E_2)) = \uparrow(L(E))$, so the claim holds. □

Now we are ready to derive a technique for bounding alphabetic width of homogeneous languages in terms of communication complexity:

**Lemma 6.** *For every homogeneous language $L$ with $\emptyset \subset L \subset \Sigma^n$ and $n \geq 1$,*

$$\mathrm{alph}(L) \geq \mathrm{s}(L) \geq C^P(R_L)\,.$$

*Moreover, if $L$ is monotone, then*

$$\mathrm{alph}(L) \geq \mathrm{s}(L) \geq C^P(R_L^m)\,.$$

*Proof.* Let $E$ be a regular expression with $L(E) = L$. By Lemma 3, we can assume that $E$ is homogeneous. If $E$ is a homogeneous regular expression with $L(E)$ homogeneous, then for every subexpression $F$ of $E$ the language $L(F)$ is homogeneous as well, and we denote by $\lambda(F)$ the length of the words in $L(F)$.

We will now, given a homogeneous regular expression $E$ for $L$, construct a protocol for $R_L$ with $\mathrm{s}(E)$ many leaves.

Recall that Alice is given an input $x \in L$, Bob a $y \notin L$, and they have to find an index $i$ with $x_i \neq y_i$. At each state of the protocol, Alice and Bob keep a subexpression $F$ of $E$ together with an interval $[i, j]$ of length $j - i + 1 = \lambda(F)$, satisfying the invariant that $x_i \ldots x_j \in L(F)$ and $y_i \ldots y_j \notin L(F)$. At the beginning $F = E$ and $[i, j] = [1, n]$, hence the invariant holds.

At a state of the protocol with a subexpression $F = F_0 + F_1$ with $\mathrm{s}(F) > 1$ and interval $[i, j]$, it must hold that either $x_i \ldots x_j \in L(F_0)$ or $x_i \ldots x_j \in L(F_1)$, but $y_i \ldots y_j \notin L(F_0)$ and $y_i \ldots y_j \notin L(F_1)$. Thus Alice can transmit $\delta \in \{0, 1\}$ such that $x_i \ldots x_j \in L(F_\delta)$, and the protocol continues with $F$ updated to $F_\delta$ and $[i, j]$ unchanged.

At a state with subexpression $F = F_0 \cdot F_1$ and interval $[i, j]$, let $\ell := i + \lambda(F_0) - 1$. Then it must hold that $x_i \ldots x_\ell \in L(F_0)$ and $x_{\ell+1} \ldots x_j \in L(F_1)$, but either $y_i \ldots y_\ell \notin L(F_0)$ (case 0) or $y_{\ell+1} \ldots y_j \notin L(F_1)$ (case 1). Then Bob can transmit $\delta = 0, 1$ such that case $\delta$ holds, and the protocol continues with $F$ updated to $F_\delta$ and $[i, j]$ set to $[i, \ell]$ in case 0 and $[\ell + 1, j]$ in case 1.

At a state with a subexpression $F$ that is a maximal sum in $E$, it must be the case that $i = j$, and that $x_i \in L(F)$ and $y_i \notin L(F)$, hence in particular $x_i \neq y_i$ and the protocol can terminate with output $i$.

Obviously, the protocol solves $R_L$, and the tree of the protocol constructed is isomorphic to the parse tree of $E$ with its maximal sums at the leaves, thus the number of leaves is $\mathrm{s}(E)$.

If $L$ happens to be monotone, then by Lemma 5 we can assume that $E$ is a monotone expression. Then also all subexpressions of $E$ that appear in the above proof are monotone, and in the terminating case it must moreover be the case that $x_i > y_i$, therefore the protocol solves $R_L^m$. □

## 4 Lower Bounds for the Conversion Problem

For given integers $\ell, n$, we define a family of graphs $\mathcal{F}_{\ell,n}$ with parameters $\ell, n$ as the set of directed acyclic graphs whose vertex set $V$ is organized in $\ell+2$ layers, with $n$ vertices in each each layer. Hence we assume $V = \{\, \langle i,j\rangle \mid 1 \le i \le n, 0 \le j \le \ell+1\}$. For all graphs in $\mathcal{F}_{\ell,n}$, we require in addition that each edge connects a vertex in some layer $i$ to a vertex in the adjacent layer $i+1$.

The following definition serves to represent the set $\mathcal{F}_{\ell,n}$ as a finite set of strings over the alphabet $\{0,1\}$: Fix a graph $G \in \mathcal{F}_{\ell,n}$ for the moment. Let $e(i,j,k) = 1$ if $G$ has an edge from vertex $i$ in layer $j$ to vertex $k$ in layer $j+1$, and let $e(i,j,k) = 0$ otherwise. Next, for vertex $i$ in layer $j$, the word $f(i,j) = e(i,j,1)e(i,j,2)\cdots e(i,j,n)$ encodes the set of outgoing edges for this vertex. Then for layer $j$, the word $g(j) = f(1,j)f(2,j)\cdots f(n,j)$ encodes the set of edges connecting vertices in layer $j$ to vertices in layer $j+1$, for $0 \le j \le \ell$. Finally, the graph $G$ is encoded by the word $w(G) = g(0)g(1)\cdots g(\ell)$. It is easy to see that each word in the set $\{0,1\}^{n^2(\ell+1)}$ can be uniquely decoded as a graph in the set $\mathcal{F}_{\ell,n}$.

A graph $G \in \mathcal{F}_{\ell,n}$ belongs to the subfamily $\mathsf{fork}_{\ell,n}$, if there exists a simple path starting in $\langle 1,0\rangle$ ending eventually in a fork, i.e., a vertex of outdegree at least two. The goal of this section will be to show that the language

$$L = L_{\ell,n} = \{\, w \in \{0,1\}^{n^2(\ell+1)} \mid w = w(G) \text{ with } G \in \mathsf{fork}_{\ell,n}\,\}$$

can be accepted by a DFA of size polynomial in both parameters, while this cannot be the case for any regular expression describing this language.

**Proposition 7.** *For every pair $(\ell,n)$ with $\ell \ge 2$ and $n \ge 5$, the language $L_{\ell,n}$ can be accepted by a DFA having at most $\ell \cdot n^4$ states.*

*Proof.* We describe a DFA $A$ accepting $L$, which has special states $q_i^j$ for $1 \le i \le n$ and $0 \le j \le \ell+1$. These states will have the following property: If $G$ has a simple non-forking path starting in vertex $\langle 1,0\rangle$ and ending in vertex $\langle i,j\rangle$, then the DFA is in state $q_{i,j}$ after reading the first $j \cdot n^2$ letters of the word $w(G)$. A DFA having this property is obtained by setting $q_1^0$ to be the start state, and by applying the construction shown in Figure 1 one by one to all states $q_i^j$, for $1 \le i \le n$ and $0 \le j \le \ell$. Each transition in Figure 1 labeled with some regular expression has to be unrolled to a simple path.

To complete the construction, we have to ensure that from every state $q$ for which $\delta(q,1)$ is not yet defined, the transition $\delta(q,1)$ leads to a state that leads every suffix of admissible length to an accepting state. This will be achieved by adding the transition structure of another deterministic finite automaton

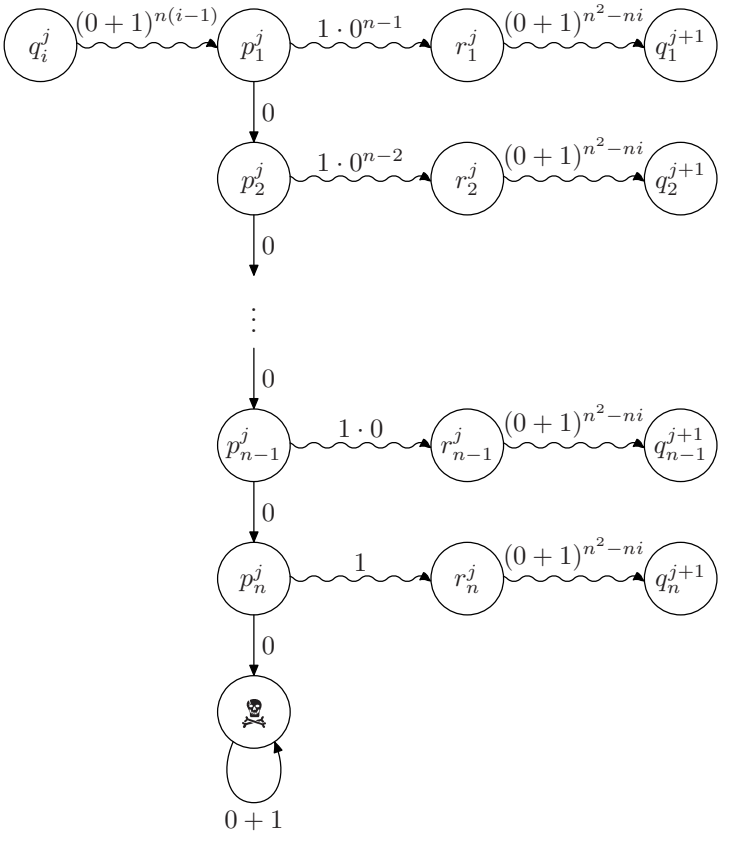



**Fig. 1.** Connecting $q_i^j$ to the corresponding states in the next layer

that accepts $\{0,1\}^{n^2(\ell+1)}$, and routing the lacking transitions into states of this automaton appropriately, in a way that each time the suffixes of admissible length are accepted.

Finally, we count the number of states in $A$: Unrolling the construction depicted in Figure 1 introduces

$$\sum_{j=0}^{\ell}\sum_{i=1}^{n}\left(n\cdot i+\sum_{k=1}^{n}(n+1-k+n^2-n\cdot i)\right)$$

states, excluding the dead state. All dead states can be merged, and the minimal DFA accepting the language $\{0,1\}^{n^2(\ell+1)}$ has $n^2(\ell+1)+2$ states, one of which is already a dead state. By adding up and simplifying, we see that the number of states equals

$$(\ell+1)(\frac{1}{2}n^4+\frac{1}{2}n^3+2n^2)+2.$$

We have $\ell + 1 < \sqrt{2}\ell$ and $2/(\ell+1) < 1$ provided $\ell \geq 2$, and for $n \geq 5$ holds $\frac{1}{2}n^4 + \frac{1}{2}n^3 + 2n^2 + 1 < \frac{n^4}{\sqrt{2}}$, and thus we can conclude that the number of states is bounded above by $\ell \cdot n^4$, provided $\ell \geq 2$ and $n \geq 5$. □

Next, we give a lower bound on the alphabetic width of this language. To this end, we show that the communication complexity of the monotone search problem for $L_{\ell,n}$ is bounded below by the communication complexity of the relation $\mathsf{FORK}_{\ell,n}$, which is defined as follows (cf. [11, ch. 5.3]):

Let $W := \{1, \ldots, n\}^\ell$. The relation $\mathsf{FORK}_{\ell,n}$ is a subset of $W \times W \times \{0, 1, \ldots, \ell\}$ For two strings $x = x_1 x_2 \cdots x_\ell$ and $y = y_1 y_2 \cdots y_\ell$, and $i \in \{0, 1, \ldots, \ell\}$ we have $(x, y, i) \in \mathsf{FORK}_{\ell,n}$ iff $x_i = y_i$ and $x_{i+1} \neq y_{i+1}$, with the convention that $x_0 = y_0 = 1$, $x_{\ell+1} = n - 1$ and $y_{\ell+1} = n$. The following lower bound on this relation is found in the monograph [11] and is due to[1] Grigni and Sipser [4]:

**Lemma 8.** $D(\mathsf{FORK}_{\ell,n}) \geq \lfloor (\log n)/4 \rfloor \cdot \lfloor \log \ell \rfloor$ □

It remains to give a reduction from $\mathsf{FORK}_{\ell,n}$ to the monotone search problem.

**Lemma 9.** *Let $L = L_{\ell,n}$. Then $D(R_L^m) \geq \lfloor \frac{1}{4} \log n \rfloor \cdot \lfloor \log \ell \rfloor$.*

*Proof.* We show that for $L = L_{\ell,n}$, any protocol that solves $R_L^m$ can be used to solve $\mathsf{FORK}_{\ell,n}$ without any additional communication, which implies the stated lower bound. The reduction is similar to one used by Grigni and Sipser [4].

From her input $x \in W$, Alice computes a graph $G_x \in \mathcal{F}_{\ell,n}$ having for every $0 \leq i \leq \ell$ an edge from $\langle x_i, i \rangle$ to $\langle x_{i+1}, i+1 \rangle$, and an additional edge from $\langle x_\ell, \ell \rangle$ to $\langle n, \ell + 1 \rangle$. By construction, $G_x \in \mathsf{fork}_{\ell,n}$ and thus $w(G_x) \in L_{\ell,n}$.

Similarly, from his input $y \in W$, Bob computes a graph $G_y$ having for every $0 \leq i \leq \ell$ an edge from $\langle y_i, i \rangle$ to $\langle y_{i+1}, i+1 \rangle$. Additionally, $G_y$ has all the edges from $\langle i, j \rangle$ to $\langle i', j+1 \rangle$ where $i \neq y_j$ and $i'$ is arbitrary. Therefore $G_y \notin \mathsf{fork}_{\ell,n}$ and thus $w(G_y) \notin L_{\ell,n}$.

Now running the protocol for $R_L^m$ on $w(G_x)$ and $w(G_y)$ yields a position $k$ where $w(G_x)_k = 1$ and $w(G_y)_k = 0$, i.e., an edge that is present in $G_x$, but not in $G_y$. By construction, this edge goes from $\langle x_i, i \rangle$ to $\langle x_{i+1}, i+1 \rangle$ for some $i$, and it must be that $y_i = x_i$ and $y_{i+1} \neq x_{i+1}$, as otherwise the edge would be present in $G_y$. Thus $i$ is a solution such that $(x, y, i) \in \mathsf{FORK}_{\ell,n}$. □

**Theorem 10.** *There exist infinitely many languages $L_m$ such that $L_m$ is acceptable by a DFA with at most $m$ states, but*

$$\operatorname{alph}(L_m) \geq m^{\frac{1}{75} \log m}.$$

*Proof.* For an integer $k$, we choose $n = 2^{4k}$, $\ell = n^4$ and $m = n^5$. Then our witness language is $L = L_m = L_{n^4,n}$. This language is acceptable by a DFA with at most $m = n^5$ states. In contrast, we have $D(R_L^m) \geq (\log n)^2$ by Lemma 9, which together with Lemma 1 implies that $C^P(R_L^m) \geq 2^{\frac{1}{3}(\log n)^2} = m^{\frac{1}{75} \log m}$, and the latter is a lower bound for the alphabetic width of the language $L_m$. □

[1] In fact, Grigni and Sipser investigated a relation that is slightly different from the one used in [11] and in this work.

## 5 Strength and Limitations

In this section, we illustrate the power and limitations of the techniques we introduced. We show that our lower bound technique sometimes gives tight lower bounds, although the gap between the lower bound and the actual minimum regular expression size can be exponential, that is, we cannot hope that $C^P(R_L)$ has a performance guarantee for regular expressions similar to the case of Boolean formulas.

### 5.1 A Poor Lower Bound

For $n$ even, consider the languages of palindromes of length $n$, $L_n = \{\, ww^R \mid w \in \{0,1\}^{n/2} \,\}$. To give an upper bound on $C^P(R_{L_n})$, recall from Section 2.2 that this number equals the minimum number of variable occurrences among all Boolean formulas describing the characteristic function of $L_n$. The following formula of size $2n$ describes the characteristic function:

$$\bigwedge_{i=1}^{n/2} (x_i \wedge x_{n/2+i}) \vee (\neg x_i \wedge \neg x_{n/2+i}).$$

For a lower bound on $\mathrm{alph}(L)$, we use the well known fact that $\mathrm{alph}(L)$ is bounded below by the minimum number of states required by a nondeterministic finite automaton accepting $L$. However, it is well known that every nondeterministic finite automaton accepting $L_n$ has size exponential in $n$ [15].

### 5.2 Optimal Expressions for Parity

Let $\mathsf{par}_n$ denote the parity language $\{w \in \{0,1\}^n \,;\, |w|_1 \text{ odd}\}$. In [3], it is shown that $\mathrm{alph}(\mathsf{par}_n) = \Omega(n^2)$ using Khrapchenko's bound [9] on the Boolean formula size of the parity function. From a recent improvement of this bound by Lee [12], we obtain the following better lower bound:

**Theorem 11.** *If $E$ is a regular expression with $L(E) = \mathsf{par}_n$, and $n = 2^d + k$ with $k < 2^d$, then* $\mathrm{alph}(E) \geq 2^d(2^d + 3k)$.

We will now construct regular expressions for $\mathsf{par}_n$ that exactly match this lower bound. The construction is essentially the same as Lee's [12] upper bound for the size of Boolean formulas for parity, but our analysis is simpler, using only induction and elementary arithmetic.

We have that $\mathsf{par}_n = L(\mathsf{odd}_n)$, where the expressions $\mathsf{even}_n$ and $\mathsf{odd}_n$ are defined inductively by

$$\begin{aligned}
\mathsf{even}_1 &:= 0 \qquad \mathsf{odd}_1 := 1 \\
\mathsf{even}_{2m} &:= (\mathsf{even}_m \cdot \mathsf{even}_m) + (\mathsf{odd}_m \cdot \mathsf{odd}_m) \\
\mathsf{odd}_{2m} &:= (\mathsf{even}_m \cdot \mathsf{odd}_m) + (\mathsf{odd}_m \cdot \mathsf{even}_m) \\
\mathsf{even}_{2m+1} &:= (\mathsf{even}_{m+1} \cdot \mathsf{even}_m) + (\mathsf{odd}_{m+1} \cdot \mathsf{odd}_m) \\
\mathsf{odd}_{2m+1} &:= (\mathsf{even}_{m+1} \cdot \mathsf{odd}_m) + (\mathsf{odd}_{m+1} \cdot \mathsf{even}_m)
\end{aligned}$$

First we observe that $\mathrm{alph}(\mathsf{even}_n) = \mathrm{alph}(\mathsf{odd}_n)$ for every $n$, and we denote it by $r(n) := \mathrm{alph}(\mathsf{even}_n)$. Then the function $r(n)$ satisfies the following recursive equations:

$$r(1) = 1 \qquad r(2m) = 4r(m) \qquad r(2m+1) = 2r(m+1) + 2r(m)$$

We now show that if $n = 2^d + k$ with $k < 2^d$, then $r(n) = 2^d(2^d + 3k)$, by induction on $n$. Thus our expressions match Lee's lower bound.

The case $n = 1$ is obvious. For the induction step, we distinguish three cases. The first case is $n = 2m$ where $m = 2^d + k$, hence $n = 2^{d+1} + 2k$. In this case we have

$$r(n) = 4r(m) = 4 \cdot 2^d(2^d + 3k) = 2^{d+1}(2^{d+1} + 6k).$$

The second case is $n = 2m+1$ where $m = 2^d + k$ and $m+1 = 2^d + (k+1)$ with $k+1 < 2^d$, hence $n = 2^{d+1} + 2k + 1$ with $2k+1 < 2^{d+1}$. In this case we obtain

$$\begin{aligned} r(n) &= 2r(m+1) + 2r(m) &&= 2^{d+1}(2^d + 3k + 3) + 2^{d+1}(2^d + 3k) \\ &= 2^{d+1}(2^{d+1} + 6k + 3)\,. \end{aligned}$$

The final case is $n = 2m+1$, where $m = 2^d + k$ and $m+1 = 2^{d+1}$, thus $k = 2^d - 1$ and $n = 2^{d+1} + (2^{d+1} - 1)$. In this case we calculate

$$\begin{aligned} r(n) &= 2r(m+1) + 2r(m) &&= 2^{2d+3} + 2^{d+1}(2^d + 3(2^d - 1)) \\ &= 2^{d+1}(2^{d+2} + 2^d + 3(2^d - 1)) &&= 2^{d+1}(2^{d+1} + 2^{d+1} + 2^d + 3(2^d - 1)) \\ &= 2^{d+1}(2^{d+1} + 3 \cdot 2^d + 3(2^d - 1) &&= 2^{d+1}(2^{d+1} + 3(2^{d+1} - 1)) \end{aligned}$$

which shows the claim.

## 6 Upper Bounds for Converting NFAs into Regular Expressions

In this section, we identify a family of finite languages $H_n$ which are the hardest finite languages for the NFA to RE conversion problem. The term *hardest* is made precise in the statement of the theorem below. The languages $H_n$ were also studied in [2], where it was shown that $\mathrm{alph}(H_n) = n^{\Omega(\log \log n)}$.

**Theorem 12.** *For $n \geq 1$, let $G_n = (V_n, \Delta)$ be the complete directed acyclic graph on $n$ vertices, that is $V_n = \{1, 2, \ldots, n\}$ and edge set $\Delta = \{\langle i, j\rangle \mid 1 \leq i < j \leq n\}$. Define the language $H_n \subset \Delta^{\leq n-1}$ as the set of all paths in $G$ leading from vertex 1 to vertex $n$. Then the following holds:*

1. *$H_n$ can be accepted by an $n$-state nondeterministic finite automaton.*
2. *Let $\Sigma$ be an alphabet. For every finite language $L$ over $\Sigma$ acceptable by an $n$-state nondeterministic finite automaton holds* $\mathrm{alph}(L) \leq |\Sigma| \cdot \mathrm{alph}(H_n)$.

*Proof.* The first statement is easy to see. For the second statement, assume the theorem holds for all values up to $n-1$, and let $A$ be an $n$-state nondeterministic finite automaton accepting $L \subseteq \Sigma^{\leq n-1}$. Without loss of generality, we assume $A$ has state set $\{q_1, q_2, \ldots, q_n\}$ and the states are in topological order with respect to the transition structure of the directed acyclic graph underlying $A$, that is, the automaton cannot move from state $q_j$ to state $q_i$ if $i \leq j$. Furthermore, we can safely assume that the automaton has start state $q_1$ and single accepting state $q_n$. This can be achieved by the following construction: If $q_1$ is not the start state, and the state set is topologically ordered, then $q_1$ is not reachable from the start state. $q_1$ can be removed, and we can apply the theorem for the obtained $n-1$-state automaton. For similar reasons, we can assume that $q_n$ is a final state. If $A$ has another final state $p$, we add transitions such that for every transition entering $p$ there is now a transition from the same source entering $q_n$. Then we remove $p$ from the set of final states. Clearly, the accepted language is not altered by this construction, and the number of states remains $n$.

Let $H$ be the minimal partial $n$-state deterministic finite automaton accepting $H_n$, i.e. the automaton has no dead state. We again assume that the state set of $H$ is topologically ordered, as for $A$.

Let $F$ and $G$ be the regular expressions obtained by applying the standard state elimination algorithm [7,14] to the automata $H$ and $A$, respectively. Since the algorithm is correct, we have $L(F) = H_n$, and $L(G) = L(A)$. For a pair of states $(q_i, q_j)$ with $i < j$ in $A$, define the regular expression $F_{ij}$ as the minimal expression describing the union of all transition labels under which the automaton can change its state from $q_i$ to $q_j$. Then by the properties of the transformation algorithm holds $G = \mathrm{sub}(F)$, where sub is the substitution replacing every occurrence of the atomic expression $\langle i, j \rangle$ with an occurrence of the expression $(F_{ij})$.

Now let $F'$ be an expression of minimal alphabetic width describing $H_n$, that is $L(F') = L(F)$. Then this equality is derivable using a sound and complete proof system for regular expression equations, e.g. see [19]. Then we can derive the equality $L(\mathrm{sub}(F')) = L(\mathrm{sub}(F))$ by a single application of the substitution rule [19], and recall $\mathrm{sub}(F) = G$. To estimate the size of $L(G)$, we simply observe that $\mathrm{alph}(F_{ij}) \leq |\Sigma|$, for all $i, j$. □

Thus an algorithm which does the job for the $n$-state automaton accepting $H_n$ will not perform much worse given any other finite automaton of equal size. In doing this, we obtain a slightly improved upper bound for the conversion problem for all finite languages — the currently best known method [3, Cor. 22] gives a bound of $(n+1) \cdot kn(n-1)^{\log n + 1}$:

**Corollary 13.** *Let $A$ be an $n$-state nondeterministic finite automaton accepting a finite language $L = L(A)$ over a $k$-symbol alphabet. Then*

$$\mathrm{alph}(L) < k \cdot n(n-1)^{\log(n-1)+1}$$

*Proof.* By the preceding theorem, it suffices to give an upper bound on $\mathrm{alph}(H_n)$. The language $H_n$ coincides with set of all walks (of length at most $n-1$) in $G_n$ that start in vertex 1 and end in vertex $n$. The analysis given in [3, Thm. 20]

implies that there exists a regular expression of size at most $n(n-1)^{\log(n-1)+1}$ describing this set, since for each pair $(i, j)$, there is a regular expression of size at most 1 describing the set of walks of length at most 1 in $G_n$ starting in $i$ and end in $j$. Thus, $\text{alph}(H_n) \leq n(n-1)^{\log(n-1)+1}$. □

## 7 Conclusions and Further Work

We developed a new lower bound technique for regular expression size to show that converting deterministic finite automata accepting finite languages into regular expressions leads to an inevitable blow-up in size of $n^{\Theta(\log n)}$, solving an open problem stated in [2]. This bound still holds when restricting to alphabet size two. Note that finite automata accepting unary finite languages can be easily converted into regular expressions of linear size.

Compared to the finite automaton model, we feel that we have still a limited understanding of the power of regular expressions in terms of descriptional complexity. For instance, although we have determined an asymptotically tight bound of $n^{\Theta(\log n)}$, there is still a considerable gap between the two constants implied by the $\Theta$-Notation. This is in stark contrast with the deterministic finite automaton model, where exact bounds are known for many questions regarding descriptional complexity, see [21] for an overview.

Another interesting line of research concerns lower bounds for the conversion problem on infinite languages, over alphabets of constant size. This problem has been solved in [5], where a corresponding lower bound of $2^{\Omega(n)}$ was established, given an $n$-state DFA over a binary alphabet accepting an infinite language. There a different proof technique based on digraph connectivity was used, which only gives trivial lower bounds for finite languages. That paper also contains lower bounds on alphabetic width of some basic regular language operations, such as intersection, shuffle, and complement. The effect of language operations on alphabetic width in the case of finite languages is also of interest: For finite automata accepting finite languages, the descriptional complexity often differs from the general case, see e.g. [21]. The lower bound techniques developed in this paper might be useful in that context.

**Acknowledgment.** We would like to thank Jeffrey Shallit for kindly providing us a copy of the corrected final version [3], which by now has already appeared.

## References

1. Delgado, M., Morais, J.: Approximation to the smallest regular expression for a given regular language. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 312–314. Springer, Heidelberg (2005)
2. Ehrenfeucht, A., Zeiger, H.P.: Complexity measures for regular expressions. Journal of Computer and System Sciences 12(2), 134–146 (1976)
3. Ellul, K., Krawetz, B., Shallit, J., Wang, M.: Regular Expressions: New Results and Open Problems. Journal of Automata, Languages and Combinatorics 10(4), 407–437 (2005)

4. Grigni, M., Sipser, M.: Monotone separation of logarithmic space from logarithmic depth. Journal of Computer and System Sciences 50, 433–437 (1995)
5. Gruber, H., Holzer, M.: Finite automata, digraph connectivity and regular expression size. Technical report, Technische Universität München (December 2007)
6. Han, Y., Wood, D.: Obtaining shorter regular expressions from finite-state automata. Theoretical Computer Science 370(1–3), 110–120 (2007)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
8. Karchmer, M., Wigderson, A.: Monotone circuits for connectivity require superlogarithmic depth. SIAM Journal on Discrete Mathematics 3, 255–265 (1990)
9. Khrapchenko, V.M.: Methods for determining lower bounds for the complexity of $\pi$-schemes (English translation). Math. Notes Acad. Sciences USSR 10, 474–479 (1972)
10. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, Annals of Mathematics Studies, pp. 3–42. Princeton University Press, Princeton (1956)
11. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, New York (1997)
12. Lee, T.: A new rank technique for formula size lower bounds. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, Springer, Heidelberg (2007)
13. Martinez, A.: Efficient computation of regular expressions from unary NFAs. In: Dassow, J., Hoeberechts, M., Jürgensen, H., Wotschke, D. (eds.) Workshop on Descriptional Complexity of Formal Systems 2002, London, Canada, pp. 216–230 (2002)
14. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRA Transactions on Electronic Computers 9(1), 39–47 (1960)
15. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: IEEE Symposium on Switching and Automata Theory 1971, pp. 188–191 (1971)
16. Morais, J.J., Moreira, N., Reis, R.: Acyclic automata with easy-to-find short regular expressions. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 349–350. Springer, Heidelberg (2006)
17. Morris, P.H., Gray, R.A., Filman, R.E.: Goto removal based on regular expressions. Journal of Software Maintenance 9(1), 47–66 (1997)
18. Sakarovitch, J.: The language, the expression, and the (small) automaton. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 15–30. Springer, Heidelberg (2006)
19. Salomaa, A.: Two complete axiom systems for the algebra of regular events. Journal of the ACM 13(1), 158–169 (1966)
20. Schnitger, G.: Regular expressions and NFAs without $\varepsilon$-transitions. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 432–443. Springer, Heidelberg (2006)
21. Yu, S.: State complexity of finite and infinite regular languages. Bulletin of the EATCS 76, 142–152 (2002)

# On Decision Problems for Probabilistic Büchi Automata

Christel Baier[1], Nathalie Bertrand[1,2], and Marcus Größer[1]

[1] Technische Universität Dresden, Germany
[2] IRISA/INRIA Rennes, France

**Abstract.** Probabilistic Büchi automata (PBA) are finite-state acceptors for infinite words where all choices are resolved by fixed distributions and where the accepted language is defined by the requirement that the measure of the accepting runs is positive. The main contribution of this paper is a complementation operator for PBA and a discussion on several algorithmic problems for PBA. All interesting problems, such as checking emptiness or equivalence for PBA or checking whether a finite transition system satisfies a PBA-specification, turn out to be undecidable. An important consequence of these results are several undecidability results for stochastic games with incomplete information, modelled by partially-observable Markov decision processes and $\omega$-regular winning objectives. Furthermore, we discuss an alternative semantics for PBA where it is required that almost all runs for an accepted word are accepting, which turns out to be less powerful, but has a decidable emptiness problem.

Probabilistic $\omega$-automata have been introduced in [BG05] as probabilistic acceptors for languages over infinite words. The central idea of these models is to resolve all choices by fixed probabilistic distributions and to define the accepted language as the set of infinite words where the probability measure of the accepting runs (according to, *e.g.*, a Büchi, Rabin or Streett acceptance condition) is positive. In the paper [BG05], we mainly concentrated on expressiveness and efficiency and showed that the class of languages that are recognizable by a probabilistic Büchi automaton (PBA) strictly subsumes the class of $\omega$-regular languages (*i.e.*, PBA are more expressive than their nondeterministic counterparts) and agrees with the class of languages that can be accepted by a probabilistic Rabin or Streett automaton (PRA/PSA). Furthermore, there are $\omega$-regular languages that have PBA of polynomial size, while any NBA has at least exponentially many states. Another aspect that makes probabilistic $\omega$-automata interesting is the observation that the verification problem "given a finite Markov chain $\mathcal{M}$ and an $\omega$-regular language $L$ for the undesired behaviors, check whether the undesired behaviors have zero measure in $\mathcal{M}$" can be answered with a PBA-representation of $L$ by means of a simple product-construction and graph-based methods, while the methods known for a representation of $L$ by a standard Büchi automaton (alternating or nondeterministic) are more complex since they rely on some kind of powerset construction [Var85, CY95, BRV04].

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 287–301, 2008.

The purpose of this paper is to study algorithmic problems for PBA in more detail and to provide answers to several questions that were left open in [BG05]. Our main results are:

(1) a complementation operator for PBA
(2) the fact that the language accepted by a PBA might depend on the precise transition probabilities
(3) the undecidability of the emptiness problem for PBA, and various related problems
(4) the decidability of the emptiness problem for PBA with a (non-standard) almost-sure semantics

To provide a complementation operator for PBA we use a technique that resembles the complementation of nondeterministic Büchi automata by means of Safra's algorithm [Saf88] and relies on (i) a transformation of a given PBA $\mathcal{P}$ into an equivalent PRA $\mathcal{P}_R$ that accepts each infinite word with probability 0 or 1, (ii) the complementation of the Rabin acceptance condition in $\mathcal{P}_R$ to obtain a PSA $\mathcal{P}_S$ for the complement language and (iii) a transformation of $\mathcal{P}_S$ into an equivalent PBA by means of techniques presented in [BG05].

At a first glance, the undecidability of the emptiness problem for PBA might not be astonishing given the undecidability of the emptiness problem for Rabin's probabilistic finite automata (PFA) [Rab63,Paz71]. However, PFA are equipped with a positive threshold for the acceptance probability, while the acccepted language of a PBA just requires positive acceptance probability (which is not of interest for PFA as for finite words the criterion "positive acceptance probability" agrees with the existence of an accepting run). In fact, (2) and (3) are surprising since for the verification of finite probabilistic systems (Markov chains or Markov decision processes) against $\omega$-regular properties, the precise transition probabilities are irrelevant and a simple graph analysis suffices, as long as one is interested in qualitative questions [HSP83,Var85,CY95].

The undecidability of the emptiness problem has several important consequences. First, together with the effectiveness of complementation, it implies that all interesting algorithmic problems for PBA (such as checking emptiness, universality or equivalence) as well as all relevant verification problems for finite-state nondeterministic systems (with or without probabilism) where the desired or undesired behaviors are specified by a PBA are undecidable. Second, several undecidability results can be established for *stochastic games with incomplete information*. More precisely, we show the undecidability of the questions whether there exists an observation-based strategy such that a Büchi condition holds with positive probability or whether there is an observation-based strategy such that a coBüchi condition holds almost surely. This even holds for stochastic games with a single nondeterministic player, modelled by partially observable Markov decision processes (POMDP) which can be seen as a generalization of PBA. Although several undecidability results have been established for POMDPs and *quantitative* properties [MHC03,GD07], we are not aware of any other undecidability result for POMDPs and *qualitative* properties. Our results might be of

interest to several research communities as POMDPs are widely used in various applications like elevator controlling, autonomous robot planning, network toubleshooting or health care policymaking (see [Cas98] for a survey).

We finally discuss an alternative semantics for PBA which defines the accepted language of a PBA by the requirement that almost all runs are accepting. PBA with the *almost-sure semantics* turn out to be less powerful than PBA with the standard semantics, but checking emptiness is decidable for them. In contrast to the above undecidability results for POMDPs, we provide a decision algorithm for the almost-sure (repeated) reachability problem for POMDPs, that is, given a POMDP $\mathcal{M}$ and a state set $F$, check whether there is an observation-based strategy for $\mathcal{M}$ that ensures to visit $F$ (infinitely often) with probability 1. This extends former results on the decidability of special cases of the qualitative model checking problem for POMDPs [dA99] where the confinement problem (which asks whether an invariant can hold with positive probability) has been addressed.

*Organization of the paper* Section 1 briefly recalls the basic definitions of probabilistic Büchi automata and (partially observable) Markov decision processes. The complementation of PBA is described in Section 2. Several undecidability results for PBA (as well as POMDPs) are given in Section 3. An alternative semantics for PBA is introduced and studied in Section 4. Section 5 concludes the paper.

## 1 Preliminaries

Throughout the paper, we assume familiarity with formal languages and nondeterministic automata over finite and infinite words, see *e.g.* [Tho90, PP04, GTW02]. We just recall the main concepts of probabilistic $\omega$-automata with Büchi or other acceptance conditions and (partially observable) Markov decision processes. For further details we refer respectively to [BG05] and [Put94].

*Probabilistic Büchi automata* (PBA) can be seen as nondeterministic Büchi automata where the nondeterminism is resolved by a probabilistic choice: for any state $q$ and letter $a \in \Sigma$ either $q$ does not have any $a$-successor or there is a probability distribution for the $a$-successors of $q$. Formally, a PBA over the alphabet $\Sigma$ is a tuple $\mathcal{P} = (Q, \delta, \mu, F)$ where $Q$ is a finite state space, $\delta : Q \times \Sigma \times Q \to [0,1]$ the transition probability function such that for all $q \in Q$ and $a \in \Sigma$,

$$\sum_{p \in Q} \delta(q,a,p) \in \{0,1\},$$

$\mu$ the initial distribution, i.e., $\mu$ is a function $Q \to [0,1]$ with $\sum_{q \in Q} \mu(q) = 1$, and $F \subseteq Q$ the set of accepting states. The states $q \in Q$ where $\mu(q) > 0$ are called initial. A *run* for an infinite word $w = a_1 a_2 \ldots \in \Sigma^\omega$ is an infinite sequence $\pi = p_0, p_1, p_2, \ldots$ of states in $Q$ such that $p_0$ is initial and $p_{i+1} \in \delta(p_i, a_{i+1}) = \{q : \delta(p_i, a_{i+1}, q) > 0\}$ for all $i \geq 0$. $\mathsf{Inf}(\pi)$ denotes the set of states that are visited infinitely often in $\pi$. Run $\pi$ is called *accepting* if $\mathsf{Inf}(\pi) \cap F \neq \emptyset$. Given an

infinite input word $w \in \Sigma^\omega$, the behavior of $\mathcal{P}$ is given by the infinite Markov chain that is obtained by unfolding $\mathcal{P}$ into a tree using $w$ as a "scheduling policy". We can therefore apply standard concepts for Markov chains ($\sigma$-algebra on the infinite paths and probability measure [KSK66,Kul95,Ste94]) to define the *acceptance probability* of $w$ in $\mathcal{P}$, denoted $\Pr_{\mathcal{P}}(w)$ or briefly $\Pr(w)$, by the probability measure of the set of accepting runs for $w$ in $\mathcal{P}$. The *accepted language* of $\mathcal{P}$ is then defined as

$$\mathcal{L}(\mathcal{P}) \;=\; \big\{ w \in \Sigma^\omega \mid \Pr_{\mathcal{P}}(w) > 0 \big\}.$$

The language of a PBA $\mathcal{P}$ might be different from the language of the NBA that is obtained from $\mathcal{P}$ by ignoring the probabilities. However, DBA and NBA that are *deterministic in limit* [Var85,CY95] can be viewed as special instances of PBA (arbitrary probabilities in $]0,1]$ can be attached to the edges). Since each NBA can be transformed into an equivalent one that is deterministic in limit [Var85,CY95], each $\omega$-regular language can be represented by a PBA. However, there are PBA that accept non-$\omega$-regular languages. For example, the PBA $\mathcal{P}_\lambda$ depicted in Fig. 1 with $0 < \lambda < 1$ accepts the following non-$\omega$-regular language:

$$\mathcal{L}(\mathcal{P}_\lambda) = \big\{ a^{k_1} b a^{k_2} b a^{k_3} b \ldots \mid k_1, k_2, k_3, \ldots \in \mathbb{N}_{\geq 1} \text{ s.t. } \prod_{i=1}^{\infty} (1 - \lambda^{k_i}) > 0 \big\}.$$

Here and in the rest of the paper, we depict accepting states by boxes.

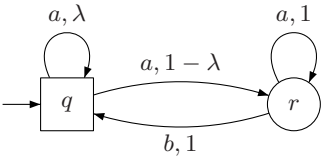



**Fig. 1.** PBA $\mathcal{P}_\lambda$ with $0 < \lambda < 1$

Similarly, *probabilistic Rabin* and *Streett automata* (PRA and PSA, respectively) are defined as tuples $\mathcal{P} = (Q, \delta, \mu, Acc)$ where $Q$, $\delta$ and $\mu$ are as above, and $Acc$ is a finite set of pairs $(H, K)$ with $H, K \subseteq Q$. The accepted language of a PRA or PSA is defined as for PBA, but with an adapted notion of accepting runs. A run $\pi = p_0, p_1, p_2 \ldots$ is accepting in a PRA if there is a pair $(H, K) \in Acc$ such that $\mathsf{Inf}(\pi) \subseteq H$ and $\mathsf{Inf}(\pi) \cap K \neq \emptyset$, whereas it is accepting in a PSA if for all pairs $(H, K) \in Acc$ either $\mathsf{Inf}(\pi) \cap H = \emptyset$ or $\mathsf{Inf}(\pi) \cap K \neq \emptyset$. Note that any PRA or PSA $\mathcal{P}$ can be transformed into an equivalent PBA whose size is polynomially bounded in the size of $\mathcal{P}$ [BG05].

A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (Q, Act, \delta, \mu)$ where $Q$ is a finite set a states, $\delta : Q \times Act \times Q \to [0,1]$ a transition probability function and $\mu$ an initial distribution. The behaviour of an MDP is determined by a device, the *scheduler*, that resolves the nondeterministic choices: a scheduler for $\mathcal{M}$ is any (history-dependent) function that selects an action for the current state i.e., a function $\mathcal{U} : S^* \to Act$ such that $\mathcal{U}(s_0 \ldots s_n) = \alpha$ implies $\delta(s_n, \alpha, t) > 0$ for some state $t$.

A *partially observable* MDP (POMDP) is a pair $(\mathcal{M}, \sim)$ consisting of an MDP and an equivalence relation $\sim \subseteq Q \times Q$ over the states of $\mathcal{M}$ such that for all states $s, t \in Q$, if $s \sim t$ then the sets of actions enabled in $s$ and $t$ are equal. Given a POMDP $(\mathcal{M}, \sim)$, an observation-based scheduler $\mathcal{U}$ is a scheduler for the underlying MDP $\mathcal{M}$ that is consistent with $\sim$, *i.e.* which satisfies $\mathcal{U}(s_0 s_1 \dots s_n) = \mathcal{U}(t_0 t_1 \dots t_m)$ if $n = m$ and $s_i \sim t_i$ for $0 \leq i \leq m$.

Given a total PBA $\mathcal{P}$ (a PBA that has transitions for each pair of a state and input letter) and the trivial equivalence relation $\sim = Q \times Q$, the pair $(\mathcal{P}, \sim)$ forms a POMDP, where an observation-based scheduler represents an input word for the PBA $\mathcal{P}$ (here $Act = \Sigma$).

## 2 Complementation of PBA

The question whether the class of languages recognizable by PBA is closed under complementation was left open in [BG05]. We show here that for each PBA $\mathcal{P}$ there exists a PBA that accepts the complement of $\mathcal{L}(\mathcal{P})$. Before providing a complementation operator for PBA, we consider the PBA $\widetilde{\mathcal{P}_\lambda}$ in Fig. 2 which accepts the following language (see appendix):

$$\widetilde{L_\lambda} = \big\{ a^{k_1} b a^{k_2} b a^{k_3} b \dots \mid k_1, k_2, k_3 \dots \in \mathbb{N}_{\geq 1} \text{ s.t. } \prod_{i=1}^{\infty} (1 - \lambda^{k_i}) = 0 \big\}.$$

$\widetilde{L_\lambda}$ is thus roughly the complement of the language accepted by the PBA $\mathcal{P}_\lambda$ shown in Fig. 1. More precisely, it holds that $\widetilde{L_\lambda} = (a^+ b)^\omega \setminus \mathcal{L}(\mathcal{P}_\lambda)$. Hence, $\widetilde{\mathcal{P}_\lambda}$ combined with a PBA for $(a+b)^* a^\omega$, $b(a+b)^\omega$ and $(a+b)^* bb(a+b)^\omega$ yields a PBA that recognizes the complement of $\mathcal{L}(\mathcal{P}_\lambda)$.

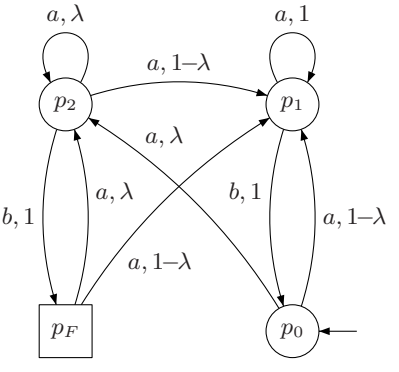



**Fig. 2.** PBA $\widetilde{\mathcal{P}_\lambda}$ with $0 < \lambda < 1$

**Theorem 1.** *For each PBA $\mathcal{P}$ there exists a PBA $\mathcal{P}'$ of size $\mathcal{O}(\exp(|\mathcal{P}|)$ such that $\mathcal{L}(\mathcal{P}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{P})$. Moreover, $\mathcal{P}'$ can be effectively constructed from $\mathcal{P}$.*

*Proof (sketch).* The idea for the complementation of a given PBA $\mathcal{P}$ is to provide the following series of transformations

$$\begin{aligned} \text{PBA } \mathcal{P} &\overset{(1)}{\Longrightarrow} \text{0/1-PRA } \mathcal{P}_R \text{ with } \mathcal{L}(\mathcal{P}_R) = \mathcal{L}(\mathcal{P}) \\ &\overset{(2)}{\Longrightarrow} \text{0/1-PSA } \mathcal{P}_S \text{ with } \mathcal{L}(\mathcal{P}_S) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{P}_R) \\ &\overset{(3)}{\Longrightarrow} \text{PBA } \overline{\mathcal{P}} \text{ with } \mathcal{L}(\overline{\mathcal{P}}) = \mathcal{L}(\mathcal{P}_S) \end{aligned}$$

where 0/1-PRA denotes a PRA with $\Pr_{\mathcal{P}_R}(w) \in \{0,1\}$ for each word $w \in \Sigma^\omega$. Step (2) is obvious as it relies on the duality of Rabin and Streett acceptance. In step (3), we may use the polynomial transformation from PSA to PBA presented in [BG05]. The most interesting part is step (1), which has some similarities with Safra's determinization algorithm for NBA and also relies on some kind of powerset construction. However, we argue that the probabilistic setting is slighty simpler: instead of organizing the potential accepting runs in Safra trees, we may deal with up to $n$ independent *sample runs* (where $n$ is the number of states in $\mathcal{P}$) that are representative for all potential accepting runs. The idea is to represent the current states of the sample runs by tuples $\langle p_1, \ldots, p_k \rangle$ of pairwise distinct states in $\mathcal{P}$. Whenever two sample runs meet at some point, say the next states $p_1'$ and $p_2'$ in the first two sample runs agree, then they are merged, which requires a shift operation for the other sample runs and yields a tuple of the form $\langle p_1', p_3', \ldots, p_k', \ldots, r, \ldots \rangle$ where $p_i \to p_i'$ stands for the move in the $i$-th sample run. Additionally, new sample runs are generated in case the original PBA $\mathcal{P}$ can be in an accepting state $r \notin \{p_1', \ldots, p_k'\}$. The Rabin condition serves to express the condition that at least one of the sample runs enters the set $F$ of accepting states in $\mathcal{P}$ infinitely often and is a proper run in $\mathcal{P}$ (*i.e.*, is affected by the shift operations only finitely many times). The details of this construction are complicated and omitted here. □

## 3 Undecidability Results

A natural question that arises with automata is whether the accepted language of a given automaton is empty. The decidability of this problem for PBAs was open, since in [BG05] the emptiness problem was only treated for *uniform* PBA, a subclass of PBA, which are as expressive as $\omega$-regular languages. It was shown there that checking emptiness is decidable for uniform PBAs and that the problem, given a uniform PBA $\mathcal{P}$, whether $\mathcal{L}(\mathcal{P}) \neq \emptyset$ is NP-hard. The decidability followed from a transformation of a uniform PBA into an equivalent NSA. For the full class of PBA we present the following result.

**Theorem 2.** *The emptiness problem for PBA is undecidable.*

Our proof for Theorem 2 given below relies on a reduction from a variant of the emptiness problem for PFA, using the fact that modifying the transition probabilities can affect the accepted language of a PBA. To see this last point, we consider again the PBA represented in Fig. 1 and show that:

**Proposition 3.** *For* $0 < \lambda < \frac{1}{2} < \eta < 1, \quad \mathcal{L}(\mathcal{P}_\lambda) \neq \mathcal{L}(\mathcal{P}_\eta)$.

Recall that $\mathcal{L}(\mathcal{P}_\lambda) = \{a^{k_1}ba^{k_2}b\cdots \mid \prod_{i\geq 1}(1-\lambda^{k_i}) > 0\}$. Proposition 3 is an immediate consequence of the following lemma (using $n = 2$):

**Lemma 4.** *For each* $n \in \mathbb{N}_{\geq 2}$ *there exists a sequence* $(k_i)_{i\geq 1}$ *such that*

$$\prod_{i\geq 1}(1-\lambda^{k_i}) > 0 \text{ if and only if } \lambda < \tfrac{1}{n}.$$

*Proof.* Given $n \in \mathbb{N}_{\geq 2}$, we define the sequence $(k_i)_{i\geq 1}$ in the following way: the first $n$ elements are set to 1, then the $n^2$ following elements are set to 2, the $n^3$ next elements set to 3, etc. The sequence $(k_i)_{i\geq 1}$ is non-decreasing, and defined by plateaux of increasing values and exponentially increasing length. We show that $\prod_i(1-\lambda^{k_i})$ is positive if and only if $\lambda < \frac{1}{n}$. To see this, we consider the series $\sum_i \log(1-x^{k_i})$ which converges if and only if $\prod_i(1-x^{k_i})$ is positive. Now, $\sum_i \log(1-x^{k_i}) = \sum_i n^i \log(1-x^i)$ by definition of the sequence $(k_i)$, and the latter series behaves as $-\sum_i n^i x^i$ (*i.e.* either both converge, or both diverge) since $\log(1-\varepsilon) \sim_{\varepsilon\mapsto 0} -\varepsilon$. Hence $\sum_i n^i \log(1-x^i) < \infty$ if and only if $x < \frac{1}{n}$, and $\prod_i(1-\lambda^{k_i}) > 0$ if and only if $\lambda < \frac{1}{n}$ which proofs the claim. □

The emptiness problem for PFA is known to be undecidable [Rab63,Paz71]. We use here a variant of this result, due to Madani et al [MHC03]:

**Theorem 5 (Undecidability result for PFA, [MHC03]).** *The following problem is undecidable: Given a constant* $0 < \varepsilon < 1$ *and a PFA that either accepts some string with probability at least* $1-\varepsilon$ *or accepts all strings with probability at most* $\varepsilon$*, decide which is the case.*

To provide an undecidability proof of the emptiness problem for PBA (Theorem 2), we reduce the variant of the emptiness problem for PFA recalled in Theorem 5 to the *intersection problem for PBA* which takes as input two PBA $\mathcal{P}_1$ and $\mathcal{P}_2$ and asks whether $\mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2)$ is empty. As PBA are closed under intersection ( [BG05]), this will complete the proof for Theorem 2.

Let $\mathcal{R}$ be a PFA over some alphabet $\Sigma$ and $0 < \varepsilon < \frac{1}{2}$ as in Theorem 5, *i.e.* such that either there exists some word $w$ accepted by $\mathcal{R}$ with probability strictly greater than $1-\varepsilon$, or all words are accepted with probability less than $\varepsilon$. For $w \in \Sigma^*$, let $\Pr_{\mathcal{R}}(w)$ denote the probability that the word $w$ is accepted by $\mathcal{R}$. From the PFA $\mathcal{R}$ and the constant $\varepsilon$ we derive two PBA $\mathcal{P}_1$ and $\mathcal{P}_2$ such that

$$\mathcal{L}^{>\varepsilon}(\mathcal{R}) = \emptyset \text{ if and only if } \mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2) = \emptyset,$$

where $\mathcal{L}^{>\varepsilon}(\mathcal{R}) = \{w \in \Sigma^* \mid \Pr_{\mathcal{R}}(w) > \varepsilon\}$. The alphabet for both $\mathcal{P}_1$ and $\mathcal{P}_2$ arise from the alphabet $\Sigma$ of $\mathcal{R}$ by adding new symbols $\sharp$ and \$, that is, $\mathcal{P}_1$ and $\mathcal{P}_2$ are PBA over the alphabet $\Sigma' = \Sigma \cup \{\sharp, \$\}$. The rough idea is to use the somehow complementary acceptance behaviour of the automata $\mathcal{P}_\lambda$ and $\widetilde{\mathcal{P}_\lambda}$ (see Fig. 1 and 2). The automata $\mathcal{P}_1$ and $\mathcal{P}_2$ are designed to read words of the form $w_1^1\sharp w_2^1\sharp\cdots w_{k_1}^1\,\$\$w_1^2\sharp w_2^2\sharp\cdots w_{k_2}^2\$\$\cdots$ where $w_i^j \in \Sigma^*$. Roughly speaking, $\mathcal{P}_1$ will mimick the automaton $\mathcal{P}_\lambda$ and $\mathcal{P}_2$ will mimick $\widetilde{\mathcal{P}_\lambda}$, where reading a word

$w_i^j\sharp$ in $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) corresponds to reading a single letter $a$ in $\mathcal{P}_\lambda$ (resp. $\widetilde{\mathcal{P}_\lambda}$). Recall that $\mathcal{P}_\lambda$ and $\widetilde{\mathcal{P}_\lambda}$ accept infinite words of the form $a^{k_1}ba^{k_2}b\ldots$ (depending on the $k_i$). The two \$-symbols serve as a separator for $\mathcal{P}_1$ and $\mathcal{P}_2$, just like the letter $b$ does for $\mathcal{P}_\lambda$ and $\widetilde{\mathcal{P}}_\lambda$. Thus, the number of $\sharp$-symbols between the $(j-1)$st and the $j$th occurence of \$\$ (and therefore the number of words $w_i^j$) corresponds to the value of $k_j$. Automaton $\mathcal{P}_1$ evolves from automaton $\mathcal{P}_\lambda$ by replacing each of its two states $q, r$ by a copy of the PFA $\mathcal{R}$. The transitions for the $\sharp$-symbol will be defined, such that after reading a word $w_i^j\sharp$ in the copy of $\mathcal{R}$ that corresponds to the state $q$ (recall that this corresponds to reading a single letter $a$ in $\mathcal{P}_\lambda$ in state $q$) the automaton $\mathcal{P}_1$ is still in this copy of $\mathcal{R}$ with probability $1-\Pr_{\mathcal{R}}(w_i^j)$ and has moved to the other copy with probability $\Pr_{\mathcal{R}}(w_i^j)$, similar to the behaviour of automaton $\mathcal{P}_\lambda$ upon reading the letter $a$ in state $q$ (it stays in $q$ with probability $\lambda$ and moves to $r$ with probability $1-\lambda$). The structure of $\mathcal{P}_1$ and $\mathcal{P}_2$ is shown in Fig. 3 and 4, respectively.

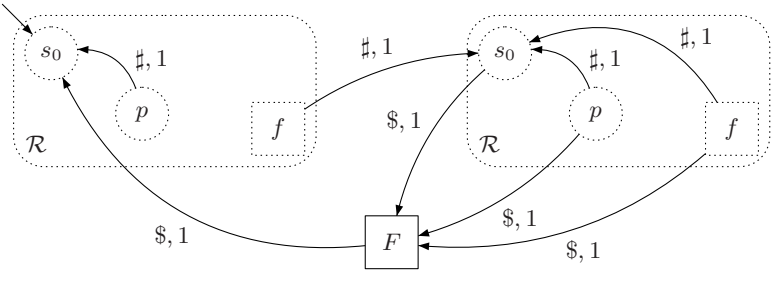



**Fig. 3.** PBA $\mathcal{P}_1$

PBA $\mathcal{P}_1$ is composed of two copies of the PFA $\mathcal{R}$ (respresented in dashed lines) augmented with new edges using the additional symbols $\sharp$ and \$. For simplicity, we represented only one initial and one final state from $\mathcal{R}$, called $s_0$ and $f$ respectively. The initial states of $\mathcal{P}_1$ are the initial states of the first copy of $\mathcal{R}$. From any final state of the first copy of $\mathcal{R}$, the PBA $\mathcal{P}_1$ can reach each initial state of $\mathcal{R}$ in the second copy (but no other state) while reading the symbol $\sharp$. (That means, each initial state of $\mathcal{R}$ of the second copy is reached with the initial probability of $\mathcal{R}$.) Upon reading the symbol $\sharp$ in a non-final state $p$ of the first copy of $\mathcal{R}$, the automaton $\mathcal{P}_1$ proceeds to each initial state of the first copy of $\mathcal{R}$ (again the initial distribution of $\mathcal{R}$ is assumed). Consuming the symbol \$ in some (final or non-final) state of the second copy, $\mathcal{P}_1$ enters with probability 1 the special state $F$, which is the unique accepting state of $\mathcal{P}_1$. Reading the second \$ symbol, $\mathcal{P}_1$ moves on to an initial state.

The language of this PBA is the following:

$$\mathcal{L}(\mathcal{P}_1) = \Big\{ w_1^1\sharp w_2^1\sharp\ldots w_{k_1}^1\,\$\$\, w_1^2\sharp w_2^2\sharp\ldots w_{k_2}^2\,\$\$\ \ldots \mid w_i^j \in \Sigma^* \\ \text{and } \prod_{j\geq 1}\Big(1-\big(\prod_{i=1}^{k_j-1}(1-\Pr_{\mathcal{R}}(w_i^j))\big)\Big) > 0\Big\}.$$

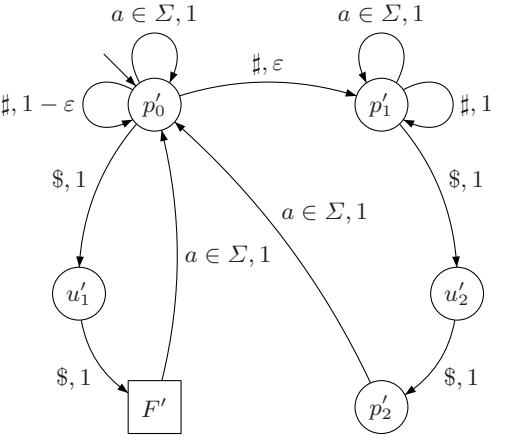



**Fig. 4.** PBA $\mathcal{P}_2$

PBA $\mathcal{P}_2$ (Fig. 4) does not depend on the structure of the given PFA $\mathcal{R}$, but only on $\varepsilon$ and the alphabet $\Sigma$. Its accepted language is :

$$\mathcal{L}(\mathcal{P}_2) = \Big\{ v_1\$\$\, v_2\$\$ \ldots \mid v_i \in (\Sigma \cup \{\sharp\})^* \text{ and } \prod_{i \geq 1} \big(1 - (1-\varepsilon)^{|v_i|_\sharp}\big) = 0 \Big\},$$

where $|v|_\sharp$ is the number of $\sharp$ symbols in the word $v \in (\Sigma \cup \{\sharp\})^*$.

Given $\mathcal{P}_1$ and $\mathcal{P}_2$, let us now detail the correctness of the reduction, *i.e.*, prove that the language $\mathcal{L}^{>\varepsilon}(\mathcal{R}) = \{w \in \Sigma^* \mid \Pr_{\mathcal{R}}(w) > \varepsilon\}$ of $\mathcal{R}$ for the threshold $\varepsilon$ is empty if and only if $\mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2) = \emptyset$.

$\Rightarrow$: Assume that $\mathcal{L}^{>\varepsilon}(\mathcal{R})$ is empty, *i.e.* for all finite words $w \in \Sigma^*$ we have: $\Pr_{\mathcal{R}}(w) \leq \varepsilon$. Let $\tilde{w} \in \mathcal{L}(\mathcal{P}_2)$. The goal is to prove that $\tilde{w} \notin \mathcal{L}(\mathcal{P}_1)$. Since $\tilde{w} \in \mathcal{L}(\mathcal{P}_2)$, $\tilde{w}$ can be written as

$$\tilde{w} = v_1\$\$\, v_2\$\$ \ldots \text{ with } v_i \in (\Sigma \cup \{\sharp\})^* \text{ and } \textstyle\prod_i \big(1 - (1-\varepsilon)^{|v_i|_\sharp}\big) = 0.$$

The subwords $v_i$ can be decomposed according to the occurrences of the symbol $\sharp$. That is,

$$\tilde{w} = w_1^1 \sharp w_2^1 \sharp \ldots w_{k_1}^1 \$\$\, w_1^2 \sharp w_2^2 \sharp \ldots w_{k_2}^2 \$\$ \ldots \text{ with } |v_i|_\sharp = k_i - 1.$$

Hence $\tilde{w} \in \mathcal{L}(\mathcal{P}_2)$ implies $\prod_i \big(1 - (1-\varepsilon)^{k_i - 1}\big) = 0$. However:

$$\begin{aligned} \prod_j \Big(1 - \textstyle\prod_{i=1}^{k_j - 1} \big(1 - \Pr_{\mathcal{R}}(w_i^j)\big)\Big) &\leq \prod_j \Big(1 - \prod_{i=1}^{k_j - 1} (1-\varepsilon)\Big) \quad \text{since } \mathcal{L}^{>\varepsilon}(\mathcal{R}) = \emptyset \\ &= \prod_j \Big(1 - (1-\varepsilon)^{k_j - 1}\Big) \\ &= 0 \qquad \text{since } \tilde{w} \in \mathcal{L}(\mathcal{P}_2). \end{aligned}$$

Hence, $\tilde{w} \notin \mathcal{L}(\mathcal{P}_1)$. Since this holds for any $\tilde{w} \in \mathcal{L}(\mathcal{P}_2)$, we conclude that $\mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2) = \emptyset$.

$\Leftarrow$: Assume now that $\mathcal{L}^{>\varepsilon}(\mathcal{R}) \neq \emptyset$. By assumption on the PFA $\mathcal{R}$, this means that there exists a finite word $w \in \Sigma^*$ such that $\Pr_{\mathcal{R}}(w) > 1-\varepsilon$.
We define

$$\tilde{w}_{k_1,k_2,\ldots} = (w\sharp)^{k_1} w\$\$ \, (w\sharp)^{k_2} w\$\$ \ldots,$$

and prove that there exists a sequence $k_1, k_2, \ldots$, such that $\tilde{w}_{k_1,k_2,\ldots} \in \mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2)$. The acceptance probability of $\tilde{w}_{k_1,k_2,\ldots}$ in $\mathcal{P}_1$ is

$$\begin{aligned}\prod_j \Big(1 - \textstyle\prod_{i=1}^{k_j} \big(1 - \Pr_{\mathcal{R}}(w)\big)\Big) &= \prod_j \Big(1 - \big(1 - \Pr_{\mathcal{R}}(w)\big)^{k_j}\Big) \\ &> \prod_j \Big(1 - \big(1 - (1-\varepsilon)\big)^{k_j}\Big) \\ &= \prod_j \Big(1 - \varepsilon^{k_j}\Big)\end{aligned}$$

On the other hand, the word $\tilde{w}_{k_1,k_2,\ldots}$ can be written as $v_1\$\$v_2\$\$\ldots$ with $v_i \in (\Sigma \cup \{\sharp\})^*$ and $|v_i|_\sharp = k_i$. Hence, $\prod_i \big(1-(1-\varepsilon)^{|v_i|_\sharp}\big) = \prod_i \big(1-(1-\varepsilon)^{k_i}\big)$. We finally apply Lemma 4 (with $n = 2$) which yields the existence of a sequence $(k'_i)_{i\geq 1}$ that will ensure at the same time

$$\prod_{j\geq 1} \Big(1 - \varepsilon^{k'_j}\Big) > 0 \text{ and } \prod_{i\geq 1} \Big(1 - (1-\varepsilon)^{k'_i}\Big) = 0.$$

Hence, $\tilde{w}_{k'_1,k'_2,\ldots} \in \mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2)$ and $\mathcal{L}(\mathcal{P}_1) \cap \mathcal{L}(\mathcal{P}_2) \neq \emptyset$.

This completes the proof of Theorem 2.

Since complementation is effective for PBA, from the undecidability of the emptiness problem, we immediately get that many other interesting algorithmic problems for PBA are undecidable too.

**Corollary 6 (Other undecidability results for PBA).** *Given two PBA $\mathcal{P}_1$ and $\mathcal{P}_2$, the following problems are undecidable.*

| | |
|---|---|
| *universality:* | $\mathcal{L}(\mathcal{P}_1) = \Sigma^\omega$? |
| *equivalence:* | $\mathcal{L}(\mathcal{P}_1) = \mathcal{L}(\mathcal{P}_2)$? |
| *inclusion:* | $\mathcal{L}(\mathcal{P}_1) \subseteq \mathcal{L}(\mathcal{P}_2)$? |

Another immediate consequence of Theorem 2 is that the verification problem for finite transition systems $\mathcal{T}$ and PBA-specifications is undecidable. Here we assume that the states in $\mathcal{T}$ are labelled with sets of atomic propositions of some finite set $\mathsf{AP}$ and consider the traces of the paths in $\mathcal{T}$ that arise by the projection to the labels of the states. Furthermore, we assume that the given PBA has the alphabet $2^{\mathsf{AP}}$:

**Corollary 7 (Verification against PBA-specifications).** *The following problems are undecidable:*

*(a) Given a transition system $\mathcal{T}$ and a PBA $\mathcal{P}$, is there a path in $\mathcal{T}$ whose trace is in $\mathcal{L}(\mathcal{P})$?*
*(b) Given a transition system $\mathcal{T}$ and a PBA $\mathcal{P}$, do the traces of all paths in $\mathcal{T}$ belong to $\mathcal{L}(\mathcal{P})$?*

*Proof.* Consider a transition system $\mathcal{T}$ such that each infinite word over the alphabet of $\mathcal{P}$ is a trace of $\mathcal{T}$. Then the emptiness problem for PBA reduces to (a) and the universality problem for PBA reduces to (b). □

As transition systems are special instances of state-labelled Markov decision processes, the following four cases of the qualitative verification problem for finite state-labelled Markov decision processes $\mathcal{M}$ and PBA-specifications $\mathcal{P}$ are undecidable too. Is there a scheduler $\mathcal{U}$ for $\mathcal{M}$ such that

$$(i)\quad \Pr_{\mathcal{U}}(\mathcal{L}(\mathcal{P})) > 0? \qquad (ii)\quad \Pr_{\mathcal{U}}(\mathcal{L}(\mathcal{P})) = 1?$$
$$(iii)\quad \Pr_{\mathcal{U}}(\mathcal{L}(\mathcal{P})) < 1? \qquad (iv)\quad \Pr_{\mathcal{U}}(\mathcal{L}(\mathcal{P})) = 0?$$

Indeed, problem (a) of Corollary 7 reduces to (i) (resp. (ii)) and problem (b) reduces to (iii) (resp. (iv)) when $\mathcal{T}$ is viewed as an MDP $\mathcal{M}$.

Since PBA are a special case of POMDPs our results immediately imply undecidability results for POMDPs and qualitative properties. In the literature, some undecidability results for POMDPs and quantitative properties (e.g. expected rewards, approximation of the maximal reachability problem) can be found [MHC03,GD07]. However, as far as we know, the undecidability of qualitative $\omega$-regular properties for POMDPs is a new result. As POMDPs are $1\frac{1}{2}$-player games, the following results also apply to the setting of stochastic multi-player games with incomplete information.

**Corollary 8 (Undecidability results for POMDPs).** *The following problems are undecidable:*

*(a) Given $(\mathcal{M}, \sim)$ a finite POMDP and $F$ a set of states in $\mathcal{M}$, is there an observation-based strategy $\mathcal{U}$ for $(\mathcal{M}, \sim)$ such that $\Pr_{\mathcal{U}}(\Box\Diamond F) > 0$?*
*(b) Given $(\mathcal{M}, \sim)$ a finite POMDP and $F$ a set of states in $\mathcal{M}$, is there an observation-based strategy $\mathcal{U}$ for $(\mathcal{M}, \sim)$ such that $\Pr_{\mathcal{U}}(\Diamond\Box F) = 1$?*

## 4 Almost-Sure Semantics and Decidability Results

Despite the undecidability of the emptiness problem for PBA, one way to try to recover decidability results is to consider an altered semantics for PBA. More precisely, we define the almost-sure semantics of a PBA $\mathcal{P}$ as the set of words which generate an almost-sure set of accepting runs:

$$\mathcal{L}^{=1}(\mathcal{P}) \;=\; \left\{ w \in \Sigma^{\omega} \mid \Pr_{\mathcal{P}}(w) = 1 \right\}$$

Let us first observe that for probabilistic Büchi automata, the switch from the standard semantics which requires positive acceptance probability to the almost-sure semantics leads to a loss of expressiveness, and the class of probabilistic Büchi automata under the almost-sure semantics is not closed under complementation. This restricted class of languages is nevertheless not included in the $\omega$-regular languages. Before we summarize the expressiveness results for almost-sure PBA, we fix some notation. By $\mathbb{L}(\mathrm{PBA})$ we denote the class of PBA-definable languages, i.e. $\mathbb{L}(\mathrm{PBA}) = \{\mathcal{L}(\mathcal{P}) \mid \mathcal{P} \text{ is a PBA}\}$. Similarly, $\mathbb{L}(\mathrm{PBA}^{=1})$

denotes the class of languages definable by a PBA with the almost-sure semantics. At last, $\mathbb{L}(\omega\text{-reg})$ denotes the class of $\omega$-regular languages.

**Theorem 9 (Expressiveness of almost-sure PBA)**

*(a)* $\mathbb{L}(PBA^{=1}) \subsetneq \mathbb{L}(PBA)$
*(b)* $\mathbb{L}(\omega\text{-reg}) \not\subseteq \mathbb{L}(PBA^{=1})$
*(c)* $\mathbb{L}(PBA^{=1}) \not\subseteq \mathbb{L}(\omega\text{-reg})$
*(d)* $\mathbb{L}(PBA^{=1})$ *is not closed under complementation.*

*Proof (sketch).* The proofs for items (a) and (b) are omitted here.

(c) The PBA $\widetilde{\mathcal{P}}_\lambda$ of Fig. 2 recognizes a non-$\omega$-regular language and enjoys the property that each word is either accepted with probability 0 or with probability 1, thus $\mathcal{L}^{=1}(\widetilde{\mathcal{P}}_\lambda) = \mathcal{L}(\widetilde{\mathcal{P}}_\lambda)$. This shows that $\widetilde{\mathcal{P}}_\lambda$ with the almost-sure semantics accepts a non-$\omega$-regular language.

(d) It is evident that each DBA $\mathcal{P}$ can be viewed as a PBA and that $\mathcal{L}(\mathcal{P}) = \mathcal{L}^{=1}(\mathcal{P})$. Consider the language $(a^*b)^\omega$. It can be recognized by a DBA and hence by a PBA with the almost-sure semantics. However, its complement $(a+b)^*a^\omega$ cannot be recognized by a PBA with the almost-sure semantics. □

*Remark 10.* It is worth noting that the almost-sure semantics does not lead to a loss of expressiveness if Streett or Rabin acceptance is considered. That is:

$$\mathbb{L}(\text{PSA}) \;=\; \mathbb{L}(\text{PSA}^{=1}) \;=\; \mathbb{L}(\text{PRA}^{=1}) \;=\; \mathbb{L}(\text{PRA}).$$

This follows from the duality of the Streett and Rabin acceptance conditions and the results presented in section 2 as every PRA (resp. PSA) can be transformed into an equivalent PBA [BG05].

PBA with the almost-sure semantics are less expressive but simpler to handle algorithmically. As $\mathcal{L}^{=1}(\widetilde{\mathcal{P}}_\lambda) = \mathcal{L}(\widetilde{\mathcal{P}}_\lambda)$, Lemma 4 implies

**Proposition 11.** *For* $0 < \lambda < \frac{1}{2} < \eta < 1$, $\quad \mathcal{L}^{=1}(\widetilde{\mathcal{P}}_\lambda) \neq \mathcal{L}^{=1}(\widetilde{\mathcal{P}}_\eta)$.

Thus modifying the transition probabilities can affect the accepted language of a PBA with the almost-sure semantics. However the emptiness problem "Given a PBA, does $\mathcal{L}^{=1}(\mathcal{P}) = \emptyset$?" for PBA under the almost-sure semantics is decidable. We will show a more general result, namely the decidability of the almost-sure repeated reachability problem for POMDPs (which asks whether, for a given POMDP $(\mathcal{M}, \sim)$ and a state set $F$, there exists an observation-based scheduler $\mathcal{U}$ such that $\Pr_{\mathcal{U}}(\Box\Diamond F) = 1$).

**Theorem 12.** *The almost-sure repeated reachability problem for POMDPs is decidable.*

*Proof.* The proof splits into two steps. We first show (Lemma 13) that the almost-sure repeated reachability problem for POMDPs reduces to the almost-sure reachability problem for POMDPs (and vice versa) and then we proof the decidability of the latter problem (Theorem 14). □

**Lemma 13.** *The two following problems are reducible to each other:*
*(i) Given a POMDP $(\mathcal{M}, \sim)$ and a set of states $F$, is there an observation-based scheduler $\mathcal{U}$ with $\Pr_{\mathcal{U}}(\Box\Diamond F) = 1$?*
*(ii) Given a POMPD $(\mathcal{M}, \sim)$ and a set of states $F$, is there an observation-based scheduler $\mathcal{U}$ with $\Pr_{\mathcal{U}}(\Diamond F) = 1$?*

*Proof.* Problem $(ii)$ reduces to $(i)$ in a straightforward manner: given an instance for $(ii)$ we transform it into an instance for $(i)$ by making all $F$-states absorbing, i.e. by removing all outgoing edges from states in $F$, and adding self loops for all letters, with probability one (to these same states). We now show that problem $(i)$ is reducible to problem $(ii)$. Let $(\mathcal{M}, \sim), F$ be an instance for $(i)$. We define $\mathcal{M}'$ as follows: $\mathcal{M}'$ consists of a copy of $\mathcal{M}$ and some additional state $f$. All transitions $(r, a, r')$ in $\mathcal{M}$ with $r \notin F$ are left unchanged. The transitions $(r, a, r')$ in $\mathcal{M}$ with $r \in F$ are kept, but their probabilities are divided by 2 in $\mathcal{M}'$. In $\mathcal{M}'$, we add a self-loop with probability 1 to state $f$ for all actions $a \in Act$. Finally, for all $r \in F$ and $a \in Act$, we add a new transition $(r, a, f)$ with probability $\frac{1}{2}$. The transformation is depicted in figure 5.

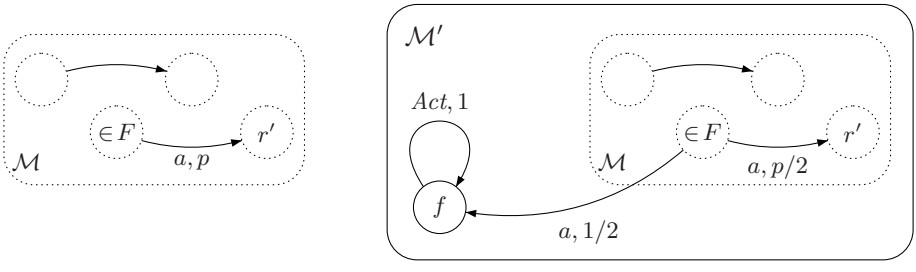



**Fig. 5.** Transformation from $\mathcal{M}$ to $\mathcal{M}'$

The equivalence relation $\sim'$ on $Q \dot\cup \{f\}$ agrees with $\sim$ on $Q$ and $\{f\}$ forms its own equivalence class, i.e. $[s]_{\sim'} = [s]_{\sim}$ for $s \in Q$ and $[f]_{\sim'} = \{f\}$. With $F' = \{f\}$, $(\mathcal{M}', \sim')$, $F'$ is an instance for problem $(ii)$ satisfying the equivalence:

$$\exists \text{ obs.-based scheduler } \mathcal{U} \text{ s.th. } \Pr_{\mathcal{U}}^{\mathcal{M}}(\Box\Diamond F) = 1 \iff$$
$$\exists \text{ obs.-based scheduler } \mathcal{U}' \text{ s.th. } \Pr_{\mathcal{U}'}^{\mathcal{M}'}(\Diamond F') = 1$$

Indeed if $F$ is visited almost surely infinitely often in $\mathcal{M}$ under the scheduler $\mathcal{U}$, $F'$ will be almost surely visited in $\mathcal{M}'$ under the scheduler $\mathcal{U}'$ that mimics $\mathcal{U}$. Conversely, given $\mathcal{U}'$ with $\Pr_{\mathcal{U}'}^{\mathcal{M}'}(\Diamond F') = 1$, we define $\mathcal{U}$ to be the restriction of $\mathcal{U}'$ on the set of path of $\mathcal{M}$. Then $\Pr_{\mathcal{U}}^{\mathcal{M}}(\Box\Diamond F) = 1$, since $\Pr_{\mathcal{U}}^{\mathcal{M}}(\Diamond\Box\neg F) > 0$ implies $\Pr_{\mathcal{U}'}^{\mathcal{M}'}(\Box\neg F') > 0$. □

**Theorem 14.** *The almost-sure reachability problem for POMDPs is decidable.*

*Proof.* We reduce the almost-sure reachability problem for POMDPs to the almost-sure reachability problem for (fully observable) MDPs, which is known

to be solvable by means of graph-algorithms. Let $\mathcal{M} = ((Q, Act, \delta, \mu), \sim)$ be a (w.l.o.g. total) POMDP and $F \subseteq Q$. We define an MDP $\mathcal{M}' = (Q', Act, \delta', \mu')$ as follows. The set of states $Q'$ of $\mathcal{M}$ consists of pairs $(r, R)$ with $r \in R \subseteq [r]_\sim$ and an extra state $q_F$ that has a self-loop with probability one for all $a \in Act$. Given $a \in Act$ and $R \subseteq Q$, let $R' = \delta(R \setminus F, a)$.

If $\delta(r, a) \cap F = \emptyset$ then $\delta'((r, R), a, (r', R' \cap [r']_\sim) = \delta(r, a, r')$ for each $r' \in Q$.

If $\delta(r, a) \cap F \neq \emptyset$ then $\delta'((r, R), a, (r', R' \cap [r']_\sim)) = \frac{1}{2 \cdot |R' \setminus F|}$ for all $r' \in R' \setminus F$ and $\delta'((r, R), a, q_F) = \frac{1}{2}$ (in case $R' \setminus F = \emptyset$, $\delta'((r, R), a, q_F) = 1$).

Moreover $\mu'(q, \{q\}) = \mu(q)$ for all $q \notin F$ and $\mu(q_F) = \Sigma_{r \in F} \mu(r)$. We set $F' = \{q_F\}$. This construction ensures that there exists an observation-based scheduler $\mathcal{U}$ with $\Pr_{\mathcal{U}}^{\mathcal{M}}(\Diamond F) = 1$ if and only if $\mathcal{M}'$ has a scheduler $\mathcal{U}'$ such that $\Pr_{\mathcal{U}'}^{\mathcal{M}'}(\Diamond F') = 1$. □

Our algorithm uses a powerset construction and hence runs in time exponential in the size of the given POMDP. However, given the EXPTIME-hardness results established by Reif [Rei84] for 2-player games with incomplete information and by de Alfaro [dA99] for POMDPs, we do not expect more efficient algorithms.

**Corollary 15.** *The emptiness problem for PBA with the almost-sure semantics is decidable.*

*Proof.* As PBA are a special case of POMDPs, this follows from Theorem 12. □

## 5 Conclusion

This paper answers several open questions on probabilistic Büchi automata. We first provide a complementation operator for PBA, that somehow resembles Safra's complementation operator for NBA, but appears to be simpler as the concept of sample runs (rather than Safra trees) suffices. We then establish the undecidability of the emptiness and universality problem for PBA, which yields the undecidability of the qualitative verification problem for POMDPs against (general) $\omega$-regular properties. Switching to an alternative *almost-sure* semantics for PBA (which turns out to be less expressive) we prove the decidability of the emptiness problem, via showing the decidability of the almost-sure repeated reachability problem and the almost-sure reachability problem for POMDPs.

## References

[BG05] Baier, C., Größer, M.: Recognizing $\omega$-regular languages with probabilistic automata. In: Proc. 20th IEEE Symp. on Logic in Computer Science (LICS 2005), pp. 137–146. IEEE Computer Society Press, Los Alamitos (2005)

[BRV04] Bustan, D., Rubin, S., Vardi, M.: Verifying $\omega$-regular properties of Markov chains. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 189–201. Springer, Heidelberg (2004)

[Cas98] Cassandra, A.R.: A survey of POMD applications. Presented at the AAAI Fall Symposium (1998), http://pomdp.org/pomdp/papers/applications.pdf

[CY95] Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of the ACM 42(4), 857–907 (1995)

[dA99] de Alfaro, L.: The verification of probabilistic systems under memoryless partial-information policies is hard. In: Proc. Workshop on Probabilistic Methods in Verification (ProbMiV 1999), Birmingham University, Research Report CSR-99-9, pp. 19–32 (1999)

[GD07] Giro, S., D'Argenio, P.R.: Quantitative model checking revisited: neither decidable nor approximable. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 179–194. Springer, Heidelberg (2007)

[GTW02] Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)

[HSP83] Hart, S., Sharir, M., Pnueli, A.: Termination of probabilistic concurrent programs. ACM Transactions on Programming Languages and Systems 5(3), 356–380 (1983)

[KSK66] Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov chains. D. Van Nostrand Co (1966)

[Kul95] Kulkarni, V.G.: Modeling and Analysis of Stochastic Systems. Chapman & Hall, Boca Raton (1995)

[MHC03] Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. Artificial Intelligence 147(1–2), 5–34 (2003)

[Paz71] Paz, A.: Introduction to probabilistic automata. Academic Press Inc., London (1971)

[PP04] Perrin, D., Pin, J.-É.: Infinite Words. Pure and Applied Mathematics, vol. 141. Elsevier, Amsterdam (2004)

[Put94] Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, Chichester (1994)

[Rab63] Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)

[Rei84] Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer System Sciences 29(2), 274–301 (1984)

[Saf88] Safra, S.: On the complexity of omega-automata. In: Proc. 29th Symposium on Foundations of Computer Science (FOCS 1988), pp. 319–327. IEEE Computer Society Press, Los Alamitos (1988)

[Ste94] Stewart, W.J.: Introduction to the numerical solution of Markov Chains. Princeton University Press, Princeton (1994)

[Tho90] Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, vol. B, ch. 4, pp. 133–191. Elsevier, Amsterdam (1990)

[Var85] Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. 26th Symposium on Foundations of Computer Science (FOCS 1985), pp. 327–338. IEEE Computer Society Press, Los Alamitos (1985)

# Model-Checking $\omega$-Regular Properties of Interval Markov Chains*

Krishnendu Chatterjee[1], Koushik Sen[1], and Thomas A. Henzinger[1,2]

[1] University of California, Berkeley, USA
[2] EPFL, Switzerland
{c_krish,ksen,tah}@eecs.berkeley.edu

**Abstract.** We study the problem of model checking Interval-valued Discrete-time Markov Chains (IDTMC). IDTMCs are discrete-time finite Markov Chains for which the exact transition probabilities are not known. Instead in IDTMCs, each transition is associated with an interval in which the actual transition probability must lie. We consider two semantic interpretations for the uncertainty in the transition probabilities of an IDTMC. In the first interpretation, we think of an IDTMC as representing a (possibly uncountable) family of (classical) discrete-time Markov Chains, where each member of the family is a Markov Chain whose transition probabilities lie within the interval range given in the IDTMC. We call this semantic interpretation Uncertain Markov Chains (UMC). In the second semantics for an IDTMC, which we call Interval Markov Decision Process (IMDP), we view the uncertainty as being resolved through non-determinism. In other words, each time a state is visited, we adversarially pick a transition distribution that respects the interval constraints, and take a probabilistic step according to the chosen distribution. We introduce a logic $\omega$-*PCTL* that can express liveness, strong fairness, and $\omega$-regular properties (such properties cannot be expressed in PCTL). We show that the $\omega$-PCTL model checking problem for Uncertain Markov Chain semantics is decidable in PSPACE (same as the best known upper bound for PCTL) and for Interval Markov Decision Process semantics is decidable in coNP (improving the previous known PSPACE bound for PCTL). We also show that the qualitative fragment of the logic can be solved in coNP for the UMC interpretation, and can be solved in polynomial time for a sub-class of UMCs. We also prove lower bounds for these model checking problems. We show that the model checking problem of IDTMCs with LTL formulas can be solved for both UMC and IMDP semantics by reduction to the model checking problem of IDTMC with $\omega$-PCTL formulas.

## 1 Introduction

*Discrete Time Markov Chains* (DTMCs) are often used to model and analyze the reliability and performance of computer systems [6,9,15,12]. A DTMC consists

* This research was supported in part by the AFOSR MURI grant F49620-00-1-0327, the NSF grant CNS-0720906, the NSF ITR grant CCR-0225610 and CCR-0234690, the Swiss National Science Foundation (NCCR MICS and Indo-Swiss Research Programme), and the ARTIST2 European Network of Excellence.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 302–317, 2008.

of a finite number of states and a fixed probability of transition from one state to another state. The fixed probability assumption in a DTMC may often not be realistic in practice [11,14,22,13]. For example, in case of an open system that interacts with an environment, transition probabilities may not be known precisely due to incomplete knowledge about the environment. Imprecision in the transition probabilities may arise if the probabilities in the system model are estimated through statistical experiments, which only provide bounds on the transition probabilities.

The model of *Interval-valued Discrete-time Markov Chains (IDTMC)* has been introduced [11,13] to faithfully capture these system uncertainties. IDTMCs are DTMC models where the exact transition probability is not known, and instead the transition probability is assumed to lie within a range. Three valued abstractions of DTMCs also naturally result in IDTMCs [8]. Two semantic interpretations have been suggested for such models. *Uncertain Markov Chains* (UMC) [11,18] is an interpretation of an IDTMC as a family of (possibly uncountably many) DTMCs, where each member of the family is a DTMC whose transition probabilities lie within the interval range given in the IDTMC. In the second interpretation, called *Interval Markov Decision Process* (IMDP) [18], the uncertainty is resolved through non-determinism. In other words, each time a state is visited, a transition distribution that respects the interval constraints is adversarially picked, and a probabilistic step is taken according to the chosen distribution. Thus, IMDPs allow the possibility of modeling a non-deterministic choice made from a set of (possibly) uncountably many choices.

The problem of model checking PCTL specifications for IDTMC was studied in [18]. PSPACE model checking algorithms were given for both UMCs and IMDPs. The model checking problem for UMCs was shown to be both NP-hard and coNP-hard. For IMDPs, a PTIME-hardness was shown; in fact, this is a consequence of the PTIME-hardness of (classical) DTMC model checking [6].

The logic PCTL [9], which extends computation tree logic (CTL) with probabilities, does not allow arbitrarily nested path formulas. Therefore, PCTL cannot express properties that depend on the set of states that appears infinitely often, e.g., liveness properties cannot be expressed in PCTL. *In order to address this limitation of PCTL, we introduce $\omega$-PCTL.* In $\omega$-PCTL, we allow Büchi conditions, that require a set of states to be visited infinitely often, its dual coBüchi conditions, and their boolean combinations. Since we allow Büchi conditions, liveness (or weak-fairness) conditions can be expressed in $\omega$-PCTL. Moreover, since we allow boolean combinations of Büchi and coBüchi conditions, strong fairness conditions can also be expressed in $\omega$-PCTL. The logic $\omega$-PCTL can express all $\omega$-regular conditions, and thus forms a robust specification language to specify properties that commonly arise in verification of probabilistic systems.

In addition to the UMC interpretation, we also consider the sub-class of UMC interpretation that restricts the DTMCs obtained from an IDTMC as follows: if the upper bound of a transition probability is positive, then the actual transition probability is also positive. In many situations the upper bound on a transition probability is positive if the transition is observed (i.e., the actual transition

probability is positive, though no positive lower bound may be known). We call this sub-class as PUMCs (Positive UMCs).

In this paper, we study the problem of model checking $\omega$-PCTL specifications for DTMCs and IDTMCs. We first show that the $\omega$-PCTL model checking problem for DTMCs can be solved in polynomial time. We then show that the $\omega$-PCTL model checking problem for PUMC and UMC interpretations is decidable in PSPACE and for IMDP interpretations is decidable in coNP. These results extend and improve the best known PSPACE bound for PCTL model checking to a much richer logic that can express $\omega$-regular properties. We also show that the qualitative fragment of the logic (called $\omega$-QPCTL) can be solved in polynomial time for the PUMCs and in coNP for the UMCs. The results of PCTL model checking algorithm do not extend straightforwardly to $\omega$-PCTL model checking. We first present the model checking algorithm for PUMC semantics using results on Markov chains, and then reduction to a formula in the existential theory of reals. The result for UMC semantics is then obtained by partitioning the UMCs in equivalence classes of PUMCs. The IMDP model checking algorithm requires a precise characterization of optimal strategies in MDPs with Müller objectives. We also prove lower bounds for these model checking problems: we show that the PCTL model checking problem is both NP-hard and coNP-hard for PUMCs, and the NP and coNP-hardness for PCTL model checking for UMCs follows from [18]. We also present model checking algorithms for IDTMCs with LTL path formulas for PUMC, UMC, and IMDP interpretations, and the result is obtained by reduction to $\omega$-PCTL formulas. Table 1 summarizes the complexity of model checking of the various classes of Markov chains under uncertainty with respect the various fragments of $\omega$-PCTL.

**Table 1.** Complexity of DTMC and IDTMC model checking

| | PCTL | | $\omega$-QPCTL | | $\omega$-PCTL | |
|---|---|---|---|---|---|---|
| Models | Lower Bound | Upper Bound | Lower Bound | Upper Bound | Lower Bound | Upper Bound |
| DTMC | PTIME | PTIME | | PTIME | PTIME | PTIME |
| PUMC | NP and coNP | PSPACE | | PTIME | NP and coNP | PSPACE |
| UMC | NP and coNP | PSPACE | | coNP | NP and coNP | PSPACE |
| IMDP | PTIME | coNP | | coNP | PTIME | coNP |

## 2 Formal Models

In this section, we recall the definitions of IDTMC, UMC, and IMDP from [18] and introduce the definition of PUMC.

**Definition 1.** A discrete-time Markov chain *(DTMC) is a 3-tuple* $\mathcal{M}=(S,\mathbf{P},L)$*, where (1)* $S$ *is a finite set of* states*; (2)* $\mathbf{P}\colon S\times S\to[0,1]$ *is a* transition probability matrix*, such that* $\sum_{s'\in S}\mathbf{P}(s,s')=1$*; and (3)* $L\colon S\to 2^{\mathrm{AP}}$ *is a* labeling *function that maps states to sets of atomic propositions from a set* AP*.*

A non-empty sequence $\pi = s_0 s_1 s_2 \cdots$ is called a *path* of $\mathcal{M}$, if each $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote the $i^{\text{th}}$ state in a path $\pi$ by $\pi[i] = s_i$. We let $Path(s)$ be the set of paths starting at state $s$. A probability measure on paths is induced by the matrix $\mathbf{P}$ as follows.

Let $s_0, s_1, \ldots, s_k \in S$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $0 \leq i < k$. Then $C(s_0 s_1 \ldots s_k)$ denotes a *cylinder set* consisting of all paths $\pi \in Path(s_0)$ such that $\pi[i] = s_i$ (for $0 \leq i \leq k$). Let $\mathcal{B}$ be the smallest $\sigma$-algebra on $Path(s_0)$ which contains all the cylinders $C(s_0 s_1 \ldots s_k)$. The measure $\mu$ on cylinder sets can be defined as follows: $\mu(C(s_0 s_1 \ldots s_k)) = 1$ if $k = 0$; otherwise $\mu(C(s_0 s_1 \ldots s_k)) = \mathbf{P}(s_0, s_1) \cdots \mathbf{P}(s_{k-1}, s_k)$. The *probability measure* on $\mathcal{B}$ is then defined as the unique measure that agrees with $\mu$ (as defined above) on the cylinder sets.

**Definition 2.** An Interval-valued Discrete-time Markov chain *(IDTMC) is a 4-tuple* $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$*, where (1)* $S$ *is a finite set of* states*; (2)* $\check{\mathbf{P}} \colon S \times S \to [0, 1]$ *is a* transition probability matrix*, where each* $\check{\mathbf{P}}(s, s')$ *gives the* lower bound *of the transition probability from the state* $s$ *to the state* $s'$*; (3)* $\hat{\mathbf{P}} \colon S \times S \to [0, 1]$ *is a* transition probability matrix*, where each* $\hat{\mathbf{P}}(s, s')$ *gives the* upper bound *of the transition probability from the state* $s$ *to the state* $s'$*; and (4)* $L \colon S \to 2^{\text{AP}}$ *is a* labeling *function that maps states to sets of atomic propositions from a set* AP*.*

We consider two semantic interpretations of an IDTMC model, namely Uncertain Markov Chains (UMC) and Interval Markov Decision Processes (IMDP).

**Uncertain Markov Chains (UMCs).** An IDTMC $\mathcal{I}$ may represent an infinite set of DTMCs, denoted by $[\mathcal{I}]$, where for each DTMC $(S, \mathbf{P}, L) \in [\mathcal{I}]$ the following is true: $\check{\mathbf{P}}(s, s') \leq \mathbf{P}(s, s') \leq \hat{\mathbf{P}}(s, s')$ for all pairs of states $s$ and $s'$ in $S$. In the Uncertain Markov Chains semantics, or simply, in the UMCs, we assume that the external environment non-deterministically picks a DTMC from the set $[\mathcal{I}]$ at the beginning and then all the transitions take place according to the chosen DTMC. Note that in this semantics, the external environment makes only one non-deterministic choice. Henceforth, we will use the term UMC to denote an IDTMC interpreted according to the Uncertain Markov Chains semantics.

**Positive Uncertain Markov Chains (PUMCs).** We consider Positive Uncertain Markov Chains (PUMCs) semantics, for which we will obtain more efficient model checking algorithms for the qualitative fragment of the logic that we will consider, and the results will also be useful in the analysis of UMCs. Given an IDTMC $\mathcal{I}$, we denote by $[\mathcal{I}]_P \subseteq [\mathcal{I}]$ the infinite set of DTMCs $(S, \mathbf{P}, L)$ such that the following conditions hold: (1) $\check{\mathbf{P}}(s, s') \leq \mathbf{P}(s, s') \leq \hat{\mathbf{P}}(s, s')$ for all pairs of states $s$ and $s'$ in $S$; (2) if $\hat{\mathbf{P}}(s, s') > 0$, then $\mathbf{P}(s, s') > 0$, for all $s, s' \in S$. In the semantics for Positive Uncertain Markov Chains (PUMCs), we assume that the external environment non-deterministically picks a DTMC from $[\mathcal{I}]_P$.

**Interval Markov Decision Processes.** In the Interval Markov Decision Processes semantics, or simply, in the IMDPs, we assume that before every transition the external environment non-deterministically picks a DTMC from the set $[\mathcal{I}]$ and then takes a one-step transition according to the probability distribution of

the chosen DTMC. Note that in this semantics, the external environment makes a non-deterministic choice before every transition. Henceforth, we will use the term IMDP to denote an IDTMC interpreted according to the Interval Markov Decision Processes semantics. We now formally define this semantics.

Let $Steps(s)$ be the set of probability density functions over $S$ defined as follows: $Steps(s) = \{\mu \colon S \to \mathbb{R}^{\geq 0} \mid \sum_{s' \in S} \mu(s') = 1 \text{ and } \check{\mathbf{P}}(s,s') \leq \mu(s') \leq \hat{\mathbf{P}}(s,s') \text{ for all } s' \in S\}$. In an IMDP, at every state $s \in S$, a probability density function $\mu$ is chosen non-deterministically from the set $Steps(s)$. A successor state $s'$ is then chosen according to the probability distribution $\mu$ over $S$.

A *path* $\pi$ in an IMDP $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$ is a non-empty sequence of the form $s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \ldots$, where $s_i \in S$, $\mu_{i+1} \in Steps(s_i)$, and $\mu_{i+1}(s_{i+1}) > 0$ for all $i \geq 0$. A path can be either finite or infinite. We use $\pi_{\text{fin}}$ to denote a finite path. Let $last(\pi_{\text{fin}})$ be the last state in the finite path $\pi_{\text{fin}}$. As in DTMC, we denote the $i^{\text{th}}$ state in a path $\pi$ by $\pi[i] = s_i$. We let $Path(s)$ and $Path_{\text{fin}}(s)$ be the set of all infinite and finite paths, respectively, starting at state $s$. To associate a probability measure with the paths, we resolve the non-deterministic choices by an *adversary*, which is defined as follows:

**Definition 3.** *An* adversary *$A$ of an IMDP $\mathcal{I}$ is a function mapping every finite path $\pi_{\text{fin}}$ of $\mathcal{I}$ onto an element of the set $Steps(last(\pi_{\text{fin}}))$. Let $\mathcal{A}_{\mathcal{I}}$ denote the set of all possible adversaries of the IMDP $\mathcal{I}$. Let $Path^A(s)$ denote the subset of $Path(s)$ which corresponds to $A$.*

The behavior of an IMDP $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$ under a given adversary $A$ is purely probabilistic. The behavior of a IMDP $\mathcal{I}$ from a state $s$ can be described by an infinite-state DTMC $\mathcal{M}^A = (S^A, \mathbf{P}^A, L^A)$ where (a) $S^A = Path_{\text{fin}}(s)$; (b) $\mathbf{P}^A(\pi_{\text{fin}}, \pi'_{\text{fin}}) = A(\pi_{\text{fin}})(s')$ if $\pi'_{\text{fin}}$ is of the form $\pi_{\text{fin}} \xrightarrow{A(\pi_{\text{fin}})} s'$; and 0 otherwise. There is a one-to-one correspondence between the paths of $\mathcal{M}^A$ and $Path^A(s)$ of $\mathcal{I}$. Therefore, we can define a probability measure $Prob^A_s$ over the set of paths $Path^A(s)$ using the probability measure of the DTMC $\mathcal{M}^A$.

## 3 ω-Probabilistic Computation Tree Logic (ω-PCTL)

In this paper, we consider an extension of PCTL that can express $\omega$-regular properties. We call the logic $\omega$-PCTL. The formal syntax and semantics of this logic is as follows.

**ω-PCTL Syntax.** We define the syntax of $\omega$-PCTL and its qualitative fragment as follows:

$$\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}(\psi)$$
$$\psi ::= \phi \,\mathcal{U}\, \phi \mid \mathbf{X}\phi \mid \psi^{\omega}$$
$$\psi^{\omega} ::= \mathsf{Buchi}(\phi) \mid \mathsf{coBuchi}(\phi) \mid \psi^{\omega} \wedge \psi^{\omega} \mid \psi^{\omega} \vee \psi^{\omega}$$

where $a \in \text{AP}$ is an atomic proposition, and $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0,1]$. Here $\phi$ represents a *state* formula, $\psi$ represents a *path* formula, and $\psi^{\omega}$ represents path formulas that depend on the set of states that appear infinitely often in a path

(we call them infinitary path formulas). The qualitative fragment of the logic, denoted as $\omega$-QPCTL, consists of formulas $\phi_Q$ such that in all sub-formulas $\mathcal{P}_{\bowtie p}(\psi)$ of $\phi_Q$ we have $p \in \{0, 1\}$, i.e., the comparison of the probability of satisfying a path formula is only made with 1 and 0 only. The logic PCTL is obtained from $\omega$-PCTL where only path formulas of the form $\phi \,\mathcal{U}\, \phi$ and $\mathbf{X}\phi$ are considered, i.e., formulas obtained as $\psi^\omega$ are not allowed. The canonical Rabin and Streett conditions (strong fairness conditions) can be expressed as conjunction and disjunction of Büchi and coBüchi conditions. Hence $\omega$-PCTL can express Rabin and Streett conditions. Since Rabin and Streett conditions are canonical forms to express $\omega$-regular properties [19], $\omega$-PCTL can express $\omega$-regular properties.

**$\omega$-PCTL Semantics for DTMC.** The notion that a state $s$ (or a path $\pi$) *satisfies* a formula $\phi$ in a DTMC $\mathcal{M}$ is denoted by $s \models_{\mathcal{M}} \phi$ (or $\pi \models_{\mathcal{M}} \phi$), and is defined inductively as follows:

$$
\begin{array}{ll}
s \models_{\mathcal{M}} \mathit{true} & \\
s \models_{\mathcal{M}} a & \text{iff } a \in L(s) \\
s \models_{\mathcal{M}} \neg\phi & \text{iff } s \not\models_{\mathcal{M}} \phi \\
s \models_{\mathcal{M}} \phi_1 \wedge \phi_2 & \text{iff } s \models_{\mathcal{M}} \phi_1 \text{ and } s \models_{\mathcal{M}} \phi_2 \\
s \models_{\mathcal{M}} \mathcal{P}_{\bowtie p}(\psi) & \text{iff } \mathit{Prob}\{\pi \in \mathit{Path}(s) \mid \pi \models_{\mathcal{M}} \psi\} \bowtie p \\
\pi \models_{\mathcal{M}} \mathbf{X}\phi & \text{iff } \pi[1] \models_{\mathcal{M}} \phi \\
\pi \models_{\mathcal{M}} \phi_1 \,\mathcal{U}\, \phi_2 & \text{iff } \exists i \geq 0\ (\pi[i] \models_{\mathcal{M}} \phi_2 \text{ and } \forall j < i.\ \pi[j] \models_{\mathcal{M}} \phi_1) \\
\pi \models_{\mathcal{M}} \mathsf{Buchi}(\phi) & \text{iff } \forall i \geq 0. \exists j \geq i.\ (\pi[j] \models_{\mathcal{M}} \phi) \\
\pi \models_{\mathcal{M}} \mathsf{coBuchi}(\phi) & \text{iff } \exists i \geq 0. \forall j \geq i.\ (\pi[j] \models_{\mathcal{M}} \phi) \\
\pi \models_{\mathcal{M}} \psi_1^\omega \wedge \psi_2^\omega & \text{iff } \pi \models_{\mathcal{M}} \psi_1^\omega \text{ and } \pi \models_{\mathcal{M}} \psi_2^\omega \\
\pi \models_{\mathcal{M}} \psi_1^\omega \vee \psi_2^\omega & \text{iff } \pi \models_{\mathcal{M}} \psi_1^\omega \text{ or } \pi \models_{\mathcal{M}} \psi_2^\omega.
\end{array}
$$

It can shown that for any path formula $\psi$ and any state $s$, the set $\{\pi \in \mathit{Path}(s) \mid \pi \models_{\mathcal{M}} \psi\}$ is measurable [21]. For a path formula $\psi$ we denote by $\mathit{Prob}_s(\psi)$ the probability of satisfying $\psi$ from $s$, i.e., $\mathit{Prob}_s(\psi) = \mathit{Prob}[\{\pi \in \mathit{Path}(s) \mid \pi \models_{\mathcal{M}} \psi\}]$. A formula $\mathcal{P}_{\bowtie p}(\psi)$ is satisfied by a state $s$ if $\mathit{Prob}_s[\psi] \bowtie p$. The path formula $\mathbf{X}\phi$ holds over a path if $\phi$ holds at the second state on the path. The formula $\phi_1 \,\mathcal{U}\, \phi_2$ is true over a path $\pi$ if $\phi_2$ holds in some state along $\pi$, and $\phi$ holds along all prior states along $\pi$. The formula $\mathsf{Buchi}(\phi)$ is true over a path $\pi$ if the path infinitely often visits states that satisfy $\phi$. The formula $\mathsf{coBuchi}(\phi)$ is true over a path $\pi$ if after a finite prefix the path visits only states that satisfy $\phi$. Given a DTMC $\mathcal{M}$ and an $\omega$-PCTL state formula $\phi$, we denote by $[\![\phi]\!]_{\mathcal{M}} = \{s \mid s \models_{\mathcal{M}} \phi\}$ the set of the states that satisfy $\phi$. Given a DTMC $\mathcal{M}$ and an $\omega$-PCTL path formula $\psi$ we denote by $W_{\mathcal{M}}(\psi) = \{s \mid \mathit{Prob}_s(\psi) = 1\}$ the set of states that satisfy $\psi$ with probability 1.

**$\omega$-PCTL Semantics for UMC.** Given an IDTMC $\mathcal{I}$ and an $\omega$-PCTL state formula $\phi$, we denote by $[\![\phi]\!]_{\mathcal{I}} = \bigcap_{\mathcal{M} \in [\mathcal{I}]} [\![\phi]\!]_{\mathcal{M}}$. Note that $s \notin [\![\phi]\!]_{\mathcal{I}}$ does not imply that $s \in [\![\neg\phi]\!]_{\mathcal{I}}$. This is because there may exist $\mathcal{M}, \mathcal{M}' \in [\mathcal{I}]$ such that $s \models_{\mathcal{M}} \phi$ and $s \models_{\mathcal{M}'} \neg\phi$. The semantics of $\omega$-PCTL for PUMCs are obtained similarly: given an IDTMC $\mathcal{I}$ and an $\omega$-PCTL state formula $\phi$, we denote by $[\![\phi]\!]_{\mathcal{I}_P} = \bigcap_{\mathcal{M} \in [\mathcal{I}]_P} [\![\phi]\!]_{\mathcal{M}}$.

**$\omega$-PCTL Semantics for IMDP.** The interpretation of a state formula and a path formula of PCTL for IMDPs is same as for DTMCs except for the state formulas of the form $\mathcal{P}_{\bowtie p}(\psi)$. The notion that a state $s$ (or a path $\pi$) *satisfies* a formula $\phi$ in an IMDP $\mathcal{I}$ is denoted by $s \models_{\mathcal{I}} \phi$ (or $\pi \models_{\mathcal{I}} \phi$), and the semantics is very similar to the one of DTMC other than path formulas with probabilistic operator which is defined below:

$$s \models_{\mathcal{I}} \mathcal{P}_{\bowtie p}(\psi) \text{ iff } Prob_s^A(\{\pi \in Path^A(s) \mid \pi \models_{\mathcal{I}} \psi\}) \bowtie p \text{ for all } A \in \mathcal{A}$$

The model checking of IDTMC with respect to the two semantics can give different results. An example illustrating this fact for the PCTL logic can be found in [18].

## 4 DTMC Model Checking

In this section we outline the basic model checking algorithm for (classical) DTMCs for $\omega$-PCTL. We start with a few notations.

**Graph of a DTMC.** Given a DTMC $\mathcal{M} = (S, \mathbf{P}, L)$ we define a graph $G_{\mathcal{M}} = (S_{\mathcal{M}}, E_{\mathcal{M}}, L_{\mathcal{M}})$ for $\mathcal{M}$ where $S_{\mathcal{M}} = S$, $L_{\mathcal{M}} = L$, and the set of edges $E_{\mathcal{M}} = \{(s, s') \mid \mathbf{P}(s, s') > 0\}$ consists of state pairs $(s, s')$ such that the transition probability from $s$ to $s'$ is positive. Given two DTMCs $\mathcal{M}_1$ and $\mathcal{M}_2$, they are graph equivalent, denoted by $\mathcal{M}_1 \equiv \mathcal{M}_2$, iff $S_{\mathcal{M}_1} = S_{\mathcal{M}_2}$, $E_{\mathcal{M}_1} = E_{\mathcal{M}_2}$, and $L_{\mathcal{M}_1} = L_{\mathcal{M}_2}$, i.e., the set of states, the set of edges, and the labeling function in $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide. Observe that though the set of edges in $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide, the exact transition probabilities in $\mathcal{M}_1$ and $\mathcal{M}_2$ can be different. For a state formula $\phi$ (resp. a set $U \subseteq S$ of states) we denote by $\Diamond\phi$ (resp. $\Diamond U$) eventually $\phi$ (resp. eventually $U$), i.e., the PCTL formula $true\, \mathcal{U}\, \phi$ (resp. $true\, \mathcal{U}\, U$).

**Lemma 1.** *Given a DTMC $\mathcal{M}$ and an infinitary path formula $\psi^\omega$, we have $Prob_s(\psi^\omega) = Prob_s\big(\Diamond(W_{\mathcal{M}}(\psi^\omega))\big)$.*

**Graph equivalence and $\omega$-QPTCL.** The truth of a qualitative PCTL formula $\phi$ (i.e., a QPCTL formula) does not depend on the precise transition probabilities of a DTMC, but depends only on the underlying graph structure of the DTMC. Lemma 2 extends the result to $\omega$-QPCTL formulas. Formally, we have the following lemma.

**Lemma 2.** *For all DTMCs $\mathcal{M}_1$ and $\mathcal{M}_2$, if $\mathcal{M}_1 \equiv \mathcal{M}_2$, then for all $\omega$-QPCTL state formulas $\phi$ we have $[\![\phi]\!]_{\mathcal{M}_1} = [\![\phi]\!]_{\mathcal{M}_2}$.*

**Model checking $\omega$-PCTL for DTMCs.** The model checking algorithm for $\omega$-PCTL for DTMCs is as follows. Given a DTMC $\mathcal{M}$ the set of closed recurrent sets of states in $\mathcal{M}$ can be computed in linear time by computing the maximal strongly connected components of $G_{\mathcal{M}}$ [5]. From the proof of Lemma 1 it follows that once the set of closed recurrent set of states in $\mathcal{M}$ is computed, the

computation of an $\omega$-PCTL formula can be reduced to a PCTL formula. The model checking algorithm for QPCTL formulas on DTMCs is very similar to CTL model checking on graphs, and the CTL like model checking algorithm is applied on the graph of the DTMC. The model checking of PCTL for DTMCs can be solved in polynomial time [6] by solving a set of linear constraints. Thus we have the following result.

**Theorem 1.** *Given a DTMC $\mathcal{M}$ and an $\omega$-PCTL state formula $\phi$, the following assertions hold: (1) the set $[\![\phi]\!]_{\mathcal{M}}$ can be computed in time polynomial in $|\mathcal{M}|$ times $\ell$; (2) if $\phi$ is an $\omega$-QPCTL formula, then the set $[\![\phi]\!]_{\mathcal{M}}$ can be computed in $O(|\mathcal{M}| \cdot \ell)$ time; where $|\mathcal{M}|$ denotes the size of $\mathcal{M}$ and $\ell$ denotes the length of $\phi$.*

**Reduction to existential theory of reals.** We now present a reduction of the model checking problem for DTMCs with $\omega$-PCTL formulas to the existential theory of reals, which is decidable in PSPACE [2]. The reduction will be later useful for model checking algorithms for IDTMCs under the PUMC and UMC semantics. Since the model checking of DTMCs for $\omega$-PCTL formulas can be done in polynomial time and the NP-complete SAT problem can be reduced to the existential theory of reals, it follows that the model checking problem of DTMCs with $\omega$-PCTL formulas can be reduced to the existential theory of reals. Formally, for all DTMCs $\mathcal{M}$, for all $\omega$-PCTL formulas $\phi$, for all states $s$ of $\mathcal{M}$, there is a formula $\Gamma(\mathcal{M}, \phi, s)$ in the existential theory of reals such that (a) $\Gamma(\mathcal{M}, \phi, s)$ is true iff $s \models_{\mathcal{M}} \phi$, (b) $\Gamma(\mathcal{M}, \phi, s)$ is polynomial in size in $\mathcal{M}$ and $\phi$; (c) $\Gamma(\mathcal{M}, \phi, s)$ can be constructed in polynomial time in size of $\mathcal{M}$ and $\phi$.

Here we make an important observation. For two DTMCs $\mathcal{M}_1$ and $\mathcal{M}_2$, if $\mathcal{M}_1 \equiv \mathcal{M}_2$, then $\Gamma(\mathcal{M}_1, \phi, s)$ and $\Gamma(\mathcal{M}_2, \phi, s)$ have the same structure in which the transition probabilities only differ. However, the converse is not true. This important observation makes the model checking algorithms for PUMC and UMC different—the UMC model checking algorithm gets more complex.

## 5 PUMC Model Checking

We first present a polynomial time model checking algorithm for $\omega$-QPCTL for PUMC interpretation of IDTMCs. We then present a PSPACE model checking algorithm for $\omega$-PCTL for PUMC interpretation of IDTMCs, and show that the problem is both NP-hard and coNP-hard. The algorithms exploit the fact that for an IDTMC $\mathcal{I}$ and for all $\mathcal{M}_1, \mathcal{M}_2 \in [\mathcal{I}]_P$, we have $\mathcal{M}_1 \equiv \mathcal{M}_2$.

**Model checking $\omega$-QPCTL.** Given an IDTMC $\mathcal{I}$, all the DTMCs in the PUMC interpretation of $\mathcal{I}$ are graph equivalent. Formally, for all $\mathcal{M}_1, \mathcal{M}_2 \in [\mathcal{I}]_P$ we have $\mathcal{M}_1 \equiv \mathcal{M}_2$. The above observation and Lemma 2 lead directly to the following model checking algorithm: given an IDTMC $\mathcal{I}$, pick a DTMC $\mathcal{M}_1 \in [\mathcal{I}]_P$, then for all $\omega$-QPCTL state formulas $\phi$, we have $[\![\phi]\!]_{\mathcal{M}_1} = [\![\phi]\!]_{\mathcal{I}_P}$. This is because for all $\mathcal{M}_2 \in [\mathcal{I}]_P$ we have $[\![\phi]\!]_{\mathcal{M}_1} = [\![\phi]\!]_{\mathcal{M}_2}$. Thus we obtain a polynomial time model checking algorithm for PUMC semantics for $\omega$-QPCTL, by just picking a DTMC $\mathcal{M}_1$ from $[\mathcal{I}]_P$ and model checking $\mathcal{M}_1$.

**Theorem 2.** *Given an IDTMC $\mathcal{I}$ and an $\omega$-QPCTL state formula $\phi_Q$, the set $[\![\phi_Q]\!]_{\mathcal{I}_P}$ can be computed in $O(|\mathcal{I}| \cdot \ell)$ time, where $|\mathcal{I}|$ denotes the size of $\mathcal{I}$ and $\ell$ denotes the length of the formula $\phi_Q$.*

**Model checking $\omega$-PCTL.** We will now present a PSPACE model checking algorithm for $\omega$-PCTL. The result is obtained by reduction to the existential theory of reals, and using the PSPACE decision procedure for the existential theory of reals [2]. Recall that for a DTMC $\mathcal{M}$, an $\omega$-PCTL formula $\phi$ and a state $s$ of $\mathcal{M}$, there is a formula $\Gamma(\mathcal{M}, \phi, s)$ in the existential theory of reals such that $s \models_{\mathcal{M}} \phi$ if and only if $\Gamma(\mathcal{M}, \phi, s)$ is true; moreover, $\Gamma(\mathcal{M}, \phi, s)$ is polynomial in the size of $\mathcal{M}$ and length of $\phi$, and $\Gamma(\mathcal{M}, \phi, s)$ can be constructed in polynomial time. Given an IDTMC $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$, consider values $0 \leq p_{s,s'} \leq 1$ for all $s, s' \in S$ such that (a) $\check{\mathbf{P}}(s, s') \leq p_{s,s'} \leq \hat{\mathbf{P}}(s, s')$, for all $s, s' \in S$; and (b) $\sum_{s' \in S} p_{s,s'} = 1$, for all $s \in S$. Let us denote $\boldsymbol{p}$ for all the values $p_{s,s'}$. We denote by $\mathcal{I}(\boldsymbol{p}) = (S, \mathbf{P}, L)$ the DTMC obtained by assigning $p_{s,s'}$ for the transition probability $\mathbf{P}(s, s')$. Given an IDTMC $\mathcal{I}$, an $\omega$-PCTL formula $\phi$ and a state $s$ of $\mathcal{I}$, we first observe that $s \in [\![\phi]\!]_{\mathcal{I}_P}$ if and only if for all $\mathcal{M} \in [\mathcal{I}]_P$ we have $s \in [\![\phi]\!]_{\mathcal{M}}$, i.e., in other words, $s \notin [\![\phi]\!]_{\mathcal{I}_P}$ if and only if there is a DTMC $\mathcal{M} \in [\mathcal{I}]_P$ such that $s \models \neg\phi$. Thus for an IDTMC $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$, an $\omega$-PCTL formula and a state $s$ we obtain a formula $\Phi(\mathcal{I}, \phi, s)$ in the existential theory of reals such that $s \notin [\![\phi]\!]_{\mathcal{I}_P}$ if and only if $\Phi(\mathcal{I}, \phi, s)$ is true. The formula $\Phi(\mathcal{I}, \phi, s)$ is as follows:

$$\Phi(\mathcal{I}, \phi, s) = \exists \boldsymbol{p}.\ \bigwedge_{s,s' \in S} \left(\check{\mathbf{P}}(s, s') \leq p_{s,s'} \leq \hat{\mathbf{P}}(s, s')\right) \ \wedge\ \bigwedge_{s \in S} \left(\sum_{s' \in S} p_{s,s'} = 1\right)$$
$$\bigwedge_{s,s' \in S} \left(\hat{\mathbf{P}}(s, s') > 0 \Rightarrow p_{s,s'} > 0\right) \ \wedge\ \bigwedge \Gamma(\mathcal{I}(\boldsymbol{p}), \neg\phi, s)$$

The first two sets of constraints specify the transition probability restriction on $\boldsymbol{p}$ such that $\boldsymbol{p}$ represents a valid probability transition for $\mathcal{M} \in [\mathcal{I}]$. The third set of constraints specify that if $\hat{\mathbf{P}}(s, s') > 0$, then $p_{s,s'} > 0$, and thus ensures that $\boldsymbol{p}$ represents a valid probability transition for $\mathcal{M} \in [\mathcal{I}]_P$. The last constraint specifies that the DTMC $\mathcal{I}(\boldsymbol{p})$ satisfies $\neg\phi$ at $s$. Note that the formula $\Gamma(\mathcal{I}(\boldsymbol{p}), \neg\phi, s)$ has the same form for all $\mathcal{M} \in [\mathcal{I}]_P$, because for all $\mathcal{M}_1, \mathcal{M}_2 \in [\mathcal{I}]_P$, $\mathcal{M}_1 \equiv \mathcal{M}_2$. This is not the case if $\bigwedge_{s,s' \in S} \left(\hat{\mathbf{P}}(s, s') > 0 \Rightarrow p_{s,s'} > 0\right)$ does not hold as in UMC. Therefore, this model checking algorithm is not applicable for UMCs. Since the existential theory of reals can be decided in PSPACE [2], we have the following theorem.

**Theorem 3.** *Given an IDTMC $\mathcal{I}$ and an $\omega$-PCTL state formula $\phi$, the set $[\![\phi]\!]_{\mathcal{I}_P}$ can be computed in space polynomial in size of $\mathcal{I}$ times the length of $\phi$.*

**Hardness of PCTL model checking.** We next demonstrate the intractability of the model checking problem for PUMC by reducing the satisfiability and validity of propositional boolean formulas to the model checking problem. Consider a propositional boolean formula $\varphi$ over the propositions $\{p_1, \ldots, p_m\}$. We consider the UMC $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$ where

- $S = \{s_I, s_1, \ldots, s_m, s_\perp\}$
- $L(s_I) = L(s_\perp) = \{\}$, $L(s_i) = \{p_i\}$ for each $1 \leq i \leq m$
- $\check{\mathbf{P}}(s_I, s_i) = 1/m^3$ and $\hat{\mathbf{P}}(s_I, s_i) = 1/m$ for all $1 \leq i \leq m$
- $\check{\mathbf{P}}(s_I, s_\perp) = 1/m^3$ and $\hat{\mathbf{P}}(s_I, s_\perp) = 1$
- $\check{\mathbf{P}}(s_i, s_i) = \hat{\mathbf{P}}(s_i, s_i) = 1$ for all $1 \leq i \leq m$
- $\check{\mathbf{P}}(s_i, s_j) = \hat{\mathbf{P}}(s_i, s_j) = 0$ for all $1 \leq i \leq m$ and $1 \leq j \leq m$ and $i \neq j$
- $\check{\mathbf{P}}(s_\perp, s_\perp) = \hat{\mathbf{P}}(s_\perp, s_\perp) = 1$

We consider the PCTL formula $\phi'$ obtained from $\phi$ by syntactically replacing every occurrence of $p_i$ in $\phi$ by $\mathcal{P}_{>\frac{1}{2m}}(\mathbf{X}p_i)$ for $1 < i < m$.

**Lemma 3.** *The following assertions hold: (a) $\varphi$ is satisfiable iff $s_I \in [\![\neg\phi]\!]_{\mathcal{I}_P}$; and (b) $\varphi$ is valid iff $s_I \in [\![\phi]\!]_{\mathcal{I}_P}$.*

*Proof.* Suppose $\varphi$ is satisfiable and let $a$ be the satisfying assignment. Consider the DTMC $\mathcal{M}^a$, where $\mathbf{P}(s_I, s_i) = \frac{1}{2m}$ if $a(p_i) = \text{false}$ and $\mathbf{P}(s_I, s_i) = \frac{1}{m+1}$ if $a(p_i) = \text{true}$; $\mathbf{P}(s_I, s_\perp)$ is thus determined by this assignment. It is easy to see that $\mathcal{M}^a \in [\mathcal{I}]$ and $\mathcal{M}^a \models \phi$. Similarly, if $\mathcal{M} \in [\mathcal{I}]$ such that $\mathcal{M} \models \phi$, then we can construct a satisfying assignment for $\varphi$: $a(p_i) = \text{false}$ if $\mathbf{P}(s_I, s_i) \leq \frac{1}{2m}$ and $a(p_i) = \text{true}$ if $\mathbf{P}(s_I, s_i) > \frac{1}{2m}$. These observations also imply that $\varphi$ is valid iff $s_I \in [\![\phi]\!]_{\mathcal{I}_P}$. □

Since the satisfiability of general propositional boolean formulas is NP-hard and the validity of general propositional boolean formulas is coNP-hard [10], the lower bounds follow immediately from Lemma 3.

**Theorem 4.** *Given an IDTMC $\mathcal{I}$, a PCTL formula $\phi$, and a state $s$ of $\mathcal{I}$ the decision problem of whether $s \in [\![\phi]\!]_{\mathcal{I}_P}$ is NP-hard and coNP-hard.*

## 6 UMC Model Checking

In this section we present a PSPACE model checking algorithm for UMC semantics. The PSPACE algorithm is obtained by a reduction to PUMC model checking. The basic reduction is obtained by partitioning the set of DTMCs $[\mathcal{I}]$ of IDTMC $\mathcal{I}$ into several PUMCs.

**Partitioning $[\mathcal{I}]$ of an IDTMC $\mathcal{I}$.** Given an IDTMC $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$, let $\mathcal{B} = \{(s, s') \mid s, s' \in S, \check{\mathbf{P}}(s, s') = 0 \text{ and } \hat{\mathbf{P}}(s, s') > 0\}$ be the set of transitions that have a positive upper bound and the lower bound is 0. We consider the following set of IDTMCs $\mathcal{I}^B$ for $B \subseteq \mathcal{B}$: we have $\mathcal{I}^B = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}^B, L)$ such that $\hat{\mathbf{P}}^B(s, s') = 0$ if $(s, s') \in B$; and $\hat{\mathbf{P}}(s, s')$ otherwise. In other words, in $\mathcal{I}^B$ the upper of the transition probabilities for the set $B$ is set to 0, and otherwise it behaves like $\mathcal{I}$. The key partitioning property is as follows: $[\mathcal{I}] = \bigcup_{B \subseteq \mathcal{B}} [\mathcal{I}^B]_P$, i.e., the union of the DTMCs obtained from the PUMCs semantics of $\mathcal{I}^B$ is the

set of DTMCs obtained from the UMC semantics of $[\mathcal{I}]$. Thus we obtain that for all $\omega$-PCTL formulas $\phi$ we have $[\![\phi]\!]_{\mathcal{I}} = \bigcap_{B \subseteq \mathcal{B}} [\![\phi]\!]_{\mathcal{I}_P^B}$.

**Model checking $\omega$-QPCTL.** The model checking problem for IDTMCs for $\omega$-QPCTL formulas under UMC semantics can be solved in coNP. Given an IDTMC $\mathcal{I}$, an $\omega$-QPCTL formula $\phi$, and a state $s$, to show that $s \notin [\![\phi]\!]_{\mathcal{I}}$, it suffices to guess $B \subseteq \mathcal{B}$ and prove that $s \notin [\![\phi]\!]_{\mathcal{I}_P^B}$. Hence the guess (or the witness) is $B$ and Theorem 2 provides the polynomial time verification procedure. Hence we obtain the following theorem.

**Theorem 5.** *Given an IDTMC $\mathcal{I}$, an $\omega$-QPCTL state formula $\phi_Q$, and a state $s$ of $\mathcal{I}$ whether $s \in [\![\phi_Q]\!]_{\mathcal{I}}$ can be decided in coNP.*

**Model checking $\omega$-PCTL.** Similar to the model checking algorithm for the $\omega$-QPCTL, we can obtain a NPSPACE model checking algorithm, by guessing $B \subseteq \mathcal{B}$ and then using the PSPACE model checking algorithm for $\omega$-PCTL for PUMC semantics.

**Theorem 6.** *Given an IDTMC $\mathcal{I}$ and an $\omega$-PCTL state formula $\phi$, the set $[\![\phi]\!]_{\mathcal{I}}$ can be computed in space polynomial in size of $\mathcal{I}$ times the length of $\phi$.*

**Hardness of PCTL model checking.** The hardness result follows from the result for PUMC. In the hardness proof for PUMC, the IDTMCs $\mathcal{I}$ considered satisfied that $[\mathcal{I}] = [\mathcal{I}]_P$; and hence the UMC and PUMC semantics coincide for $\mathcal{I}$. This gives us the following result.

**Theorem 7.** *Given an IDTMC $\mathcal{I}$, a PCTL formula $\phi$, and a state $s$ of $\mathcal{I}$ the decision problem of whether $s \in [\![\phi]\!]_{\mathcal{I}}$ is NP-hard and coNP-hard.*

## 7 IMDP Model Checking

We consider the problem of model checking IMDPs in this section. We will solve the problem by showing that we can reduce IMDP model checking to model checking (classical) a Markov Decision Process (MDP). Before presenting this reduction we recall some basic properties of the feasible solutions of a linear program and the definition of an MDP.

**Linear programming.** Consider an IMDP $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$. For a given $s \in S$, let $IE(s)$ be the following set of inequalities over the variables $\{p_{ss'} \mid s' \in S\}$: $\sum_{s' \in S} p_{ss'} = 1$, where $\check{\mathbf{P}}(s, s') \leq p_{ss'} \leq \hat{\mathbf{P}}(s, s')$ for all $s' \in S$.

**Definition 4.** *A map $\theta^s \colon S \to [0, 1]$ is called a* basic feasible solution *(BFS) to the above set of inequalities $IE(s)$ iff $\{p_{ss'} = \theta^s(s') \mid s' \in S\}$ is a solution of $IE(s)$ and there exists a set $S' \subseteq S$ such that $|S'| \geq |S| - 1$ and for all $s' \in S'$ either $\theta^s(s') = \check{\mathbf{P}}(s, s')$ or $\theta^s(s') = \hat{\mathbf{P}}(s, s')$.*

Let $\Theta^s$ be the set of all BFS of $IE(s)$. The set of BFS of a linear program has the special property that every other feasible solution can be expressed as a linear combination of basic feasible solutions. This is the content of the next proposition.

**Proposition 1.** *Let $\{p_{ss'} = \bar{p}_{ss'} \mid s' \in S\}$ be some solution of $IE(s)$. Then there are $0 \leq \alpha_{\theta^s} \leq 1$ for all $\theta^s \in \Theta^s$, such that*

$$\bar{p}_{ss'} = \textstyle\sum_{\theta^s \in \Theta^s} \alpha_{\theta^s} \theta^s(s') \text{ for all } s' \in S \qquad \text{and} \qquad \sum_{s \in S} \alpha_{\theta^s} = 1$$

**Lemma 4.** *The number of basic feasible solutions of $IE(s)$ in the worst case can be $O(|S|2^{|S|-1})$.*

**Markov Decision Processes (MDP).** A Markov decision process (MDP) is a Markov chain that has non-deterministic transitions, in addition to the probabilistic ones. In this section we formally introduce this model along with some well-known observations about them.

**Definition 5.** *If $S$ is the set of states of a system, a* next-state probability distribution *is a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$.*

**Definition 6.** *A Markov decision process (MDP) is a 3-tuple $\mathcal{D} = (S, \tau, L)$, where (1) $S$ is a finite set of states; (2) $L\colon S \to 2^{\mathrm{AP}}$ is a labeling function that maps states to sets of atomic propositions from a set* AP*; and (3) $\tau$ is a function which associates to each $s \in S$ a finite set $\tau(s) = \{\mu_1^s, \ldots, \mu_{k_s}^s\}$ of* next-state probability distributions *for transitions from $s$.*

A *path* $\pi$ in an MDP $\mathcal{D} = (S, \tau, L)$ is a non-empty sequence of the form $s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \ldots$, where $s_i \in S$, $\mu_{i+1} \in \tau(s_i)$, and $\mu_{i+1}(s_{i+1}) > 0$ for all $i \geq 0$. A path can be either finite or infinite. We use $\pi_{\mathrm{fin}}$ to denote a finite path. Let $last(\pi_{\mathrm{fin}})$ be the last state in the finite path $\pi_{\mathrm{fin}}$. As in DTMC, we denote the $i^{\mathrm{th}}$ state in a path $\pi$ by $\pi[i] = s_i$. We let $Path(s)$ and $Path_{\mathrm{fin}}(s)$ be the set of all infinite and finite paths, respectively, starting at state $s$. To associate a probability measure with the paths, we resolve the non-deterministic choices by a randomized *adversary*, which is defined as follows:

**Definition 7.** *A randomized history dependent* adversary *$A$ of an MDP $\mathcal{D}$ is a function mapping every finite path $\pi_{\mathrm{fin}}$ of $\mathcal{D}$ and an element of the set $\tau(last(\pi_{\mathrm{fin}}))$ to $[0, 1]$, such that for a given finite path $\pi_{\mathrm{fin}}$ of $\mathcal{D}$, $\sum_{\mu \in \tau(last(\pi_{\mathrm{fin}}))} A(\pi_{\mathrm{fin}})(\mu) = 1$. Let $\mathcal{A}_{\mathcal{D}}$ denote the set of all possible randomized history dependent adversaries of the MDP $\mathcal{D}$. An adversary is memoryless if it is independent of the history and only depends on the current state. Let $Path^A(s)$ denote the subset of $Path(s)$ which corresponds to an adversary $A$.*

The behavior of an MDP under a given randomized adversary is purely probabilistic. If an MDP has evolved to the state $s$ after starting from the state $s_I$ and following the finite path $\pi_{\mathrm{fin}}$, then it chooses the next-state distribution $\mu^s \in \tau(s)$ with probability $A(\pi_{\mathrm{fin}}, \mu^s)$. Then it chooses the next state $s'$ with

probability $\mu^s(s')$. Thus the probability that a direct transition to $s'$ takes place is $\sum_{\mu^s \in \tau(s)} A(\pi_{\text{fin}}, \mu^s)\mu^s(s')$. Thus as for IMDPs, one can define DTMC $\mathcal{D}^A$ that captures the probabilistic behavior of MDP $\mathcal{D}$ under adversary $A$ and also associate a probability measure on execution paths. Given an MDP $\mathcal{D}$, an $\omega$-PCTL formula $\varphi$, and a state $s$ we can define when $s \models_{\mathcal{D}} \varphi$ in a way analogous to the IMDPs.

**The reduction.** We are now ready to describe the model checking algorithm for IMDPs. Consider an IMDP $\mathcal{I} = (S, \check{\mathbf{P}}, \hat{\mathbf{P}}, L)$. Recall from the description of linear programming that we can describe the transition probability distributions from state $s$ that satisfy the range constraints as the feasible solutions of the linear program $IE(s)$. Furthermore, we denote by $\Theta^s$ the set of all BFS of $IE(s)$. Define the following MDP $\mathcal{D} = (S', \tau, L')$ where $S' = S$, $L' = L$, and for all $s \in S$, $\tau(s) = \Theta^s$. Observe that $\mathcal{D}$ is exponentially sized in $\mathcal{I}$, since $\tau(s)$ is exponential (see Lemma 4). The main observation behind the reduction is that the MDP $\mathcal{D}$ "captures" all the possible behaviors of the IMDP $\mathcal{I}$. This is the formal content of the next proposition. Theorem 8 follows from the following proposition.

**Proposition 2.** *For any adversary $A$ for $\mathcal{I}$, we can define a randomized adversary $A'$ such that $Prob_s^{\mathcal{I}^A} = Prob_s^{\mathcal{D}^{A'}}$ for every $s$, where $Prob_s^{X^A}$ is measure on paths from $s$ defined by $X$ under $A$. Similarly for every adversary $A$ for $\mathcal{D}$, there is an adversary $A'$ for $\mathcal{I}$ that defines the same probability measure on paths.*

**Theorem 8.** *Given an IMDP $\mathcal{I}$, for all $\omega$-PCTL formulas $\varphi$ and for all states $s$, we have $s \models_{\mathcal{I}} \varphi$ iff $s \models_{\mathcal{D}} \varphi$.*

Thus, in order to model check IMDP $\mathcal{I}$, we can model check the MDP $\mathcal{D}$. The model checking algorithm for MDPs requires the solution of MDPs with infinitary path formulas, and solution of MDPs with PCTL formulas. Algorithms that run in polynomial time (and space) for MDPs with Büchi and coBüchi conditions are known from [4,7], and it is straightforward to extend the algorithms to infinitary path formulas that are obtained as conjunction and disjunction of Büchi and coBüchi conditions. Algorithms that run in polynomial time (and space) for MDPs with PCTL formulas are available in [1,17]. Thus, if we directly model check $\mathcal{D}$ we get an EXPTIME model checking algorithm for $\mathcal{I}$. However, we can improve this to get a coNP procedure. The reason for this is that it is known that as far as model checking MDPs is concerned, we can restrict our attention to certain special class of *memoryless* adversaries, i.e., adversaries that always pick a fixed probability distribution over a set of non-deterministic choices whenever a state is visited. It follows from the results of [3] that in MDPs with Müller conditions (that subsumes the infinitary path formulas of $\omega$-PCTL) an uniform randomized memoryless optimal strategy exists such that the size of the support of the memoryless optimal strategy is bounded by the size of the state space. Formally, we have the following lemma.

**Lemma 5.** *For an MDP $\mathcal{D} = (S, \tau, L)$ and an infinitary path formula $\psi^\omega$, there exists an randomized memoryless adversary $A$ such that (1)* (Support of size at

most $|S|$). *for all* $s \in S$ *we have* $|Supp(A(s))| \leq |S|$; *(2)* (Uniform). *for all* $s \in S$ *and* $\mu \in Supp(A(s))$ *we have* $A(s)(\mu) = \frac{1}{|Supp(A(s))|}$; *and (3)* (Optimal). *for all* $s \in S$ *we have* $Prob_s^{\mathcal{D}^A}(\psi^\omega) = \sup_{A' \in \mathcal{A}} Prob_s^{\mathcal{D}^{A'}}(\psi^\omega)$.

The existence of deterministic memoryless strategies for formulas in PCTL (where the sub-formulas are already evaluated) for MDPs follows from the results of [1,17]. Thus we obtain the following theorem.

**Proposition 3 ([1,17,3]).** *Let* $\mathcal{D} = (S, \tau, L)$ *be an MDP. Let* $\mathcal{A}_{unf}$ *be the set of uniform randomized memoryless adversaries with support of size at most* $|S|$ *for MDP* $\mathcal{D}$, *i.e., for all* $A \in \mathcal{A}_{unf}$, $A(s)(\mu) = \frac{1}{|Supp(A(s))|}$ *for* $\mu \in Supp(A(s))$ *and* $|Supp(A(s))| \leq |S|$. *Consider an* $\omega$*-PCTL formula* $\varphi = \mathcal{P}_{\bowtie p}(\psi)$ *such that the truth or falsity of every subformula of* $\psi$ *in every state of* $\mathcal{D}$ *is already determined. Then* $\mathcal{D} \models \varphi$ *iff* $\mathcal{D}^A \models \varphi$ *for all* $A \in \mathcal{A}_{unf}$.

For every subformula of the form $\varphi = \mathcal{P}_{\bowtie p}(\psi)$, if the formula $\varphi$ is not true at a state $s$, in the IMDP semantics, then we can guess $A \in \mathcal{A}_{unf}$ and then verify that in $\mathcal{D}^A$ the formula $\varphi$ is not true at $s$. The witness $A$ is the polynomial witness and the polynomial time algorithm for Markov chains presents the polynomial time verification procedure. In case of general formulas, the above procedure needs to be applied in a bottom up fashion.

**Theorem 9.** *Given an IDTMC* $\mathcal{I}$ *and an* $\omega$*-PCTL state formula* $\phi$, *and a state* $s$, *whether the state* $s \models \phi$ *under the IMDP semantics can be decided in coNP.*

**Lower bound.** It follows from the results of [6] that the model checking problem for DTMCs with PCTL formulas is PTIME-hard. Since DTMCs are a special case of IMDPs, the PTIME-time lower bound follows for model checking IMDPs with PCTL and $\omega$-PCTL formulas.

## 8 Model Checking of Linear Time Formulas

Finally, we consider the model checking problem of IDTMCs with LTL formulas. In other words, we consider LTL path formulas $\psi$, and formulas of the form $\mathcal{P}_{\bowtie p}(\psi)$. For the model checking problem we apply the following procedure: we first convert $\psi$ to an equivalent non-deterministic Büchi automata [20], and then determinize it to obtain an equivalent deterministic Rabin automata $Q(\psi)$ [16]. The deterministic Rabin automata $Q(\psi)$ has $2^{2^l}$ states, where $l$ is the length of the formula $\psi$, and has $2^l$ Rabin pairs. Given a IDTMC $\mathcal{I}$ and a formula $\varphi = \mathcal{P}_{\bowtie p}(\psi)$, the model checking problem for the UMC and IMDP semantics are solved as follows. In both case we construct the Rabin automata $Q(\psi)$.

1. For the IMDP semantics, we construct the product IDTMC of $\mathcal{I}$ and $Q(\psi)$, denoted as $\mathcal{I} \times Q(\psi)$, and solve it under the IMDP semantics with respect to a Rabin objective (applying the results of Section 7).

2. For the PUMC semantics, we construct the product IDTMC of $\mathcal{I}$ and $Q(\psi)$, denoted as $\mathcal{I} \times Q(\psi)$. For the formula $\varphi$, we write a formula in the existential theory of reals: the formula is similar to the formula of Section 5 with the additional constraints that for two states in $\mathcal{I} \times Q(\psi)$, if the state component of $\mathcal{I}$ is the same, then the chosen distribution at the states must also be same, i.e., for two states $(s, q_1)$ and $(s, q_2)$ we require that the probability distribution chosen from the interval must be the same. The result for UMC semantics is similar. We thus obtain the following result.

**Theorem 10.** *Given an IDTMC $\mathcal{I}$, an LTL path formula $\psi$, and a state formula $\phi = \mathcal{P}_{\bowtie p}(\psi)$, the following assertions hold: (1) the sets $[\![\phi]\!]_{\mathcal{I}_P}$ and $[\![\phi]\!]_{\mathcal{I}}$ can be computed in PSPACE in the size of $\mathcal{I}$ and 2EXPTIME in the length of the formula $\psi$; and (2) given a state $s$, whether the state $s \models \phi$ under the IMDP semantics can be decided in coNP in the size of $\mathcal{I}$ and 2EXPTIME in the length of the formula $\psi$.*

## 9 Conclusion

We have investigated the model checking problem of $\omega$-PCTL and its qualitative fragment for three semantic interpretations of IDTMCs, namely UMC, PUMC and IMDP. We proved upper bounds and lower bounds on the complexity of the model checking problem for these models. Some of our bounds however are not tight. Finding tight lower and upper bounds for these model checking problems is an interesting open problem. We also present model checking algorithm for LTL formulas.

## References

1. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, Springer, Heidelberg (1995)
2. Canny, J.: Some algebraic and geometric computations in PSPACE. In: STOC 1988, pp. 460–467. ACM, New York (1988)
3. Chatterjee, K., de Alfaro, L., Henzinger, T.: Trading memory for randomness. In: QEST 2004, IEEE, Los Alamitos (2004)
4. Chatterjee, K., Jurdziński, M., Henzinger, T.: Quantitative stochastic parity games. In: SODA 2004, ACM-SIAM (2004)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (1990)
6. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of ACM 42(4), 857–907 (1995)
7. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
8. Fecher, H., Leucker, M., Wolf, V.: Don't know in probabilistic systems. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, Springer, Heidelberg (2006)
9. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)

10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, MA (1979)
11. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS 1991, IEEE, Los Alamitos (1991)
12. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov chains. Springer, Heidelberg (1976)
13. Kozine, I.O., Utkin, L.V.: Interval-valued finite Markov chains. Reliable Computing 8(2), 97–113 (2002)
14. Kuznetsov, V.P.: Interval statistical models. Radio and Communication (1991)
15. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. American Mathematical Society (2004)
16. Safra, S.: Complexity of automata on infinite objects. PhD thesis, Weizmann Institute of Science (1989)
17. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT (1995)
18. Sen, K., Viswanathan, M., Agha, G.: Model-checking Markov chains in the presence of uncertainties. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, Springer, Heidelberg (2006)
19. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages. Beyond Words, ch. 7, vol. 3, Springer, Heidelberg (1997)
20. Vardi, M., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS 1986, IEEE, Los Alamitos (1986)
21. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS 1985, IEEE, Los Alamitos (1985)
22. Walley, P.: Measures of uncertainty in expert systems. Artificial Intelligence 83, 1–58 (1996)

# Prevision Domains and Convex Powercones*

Jean Goubault-Larrecq

LSV, ENS Cachan, CNRS, INRIA Futurs
61, av. du président-Wilson, 94230 Cachan, France
goubault@lsv.ens-cachan.fr

**Abstract.** Two recent semantic families of models for mixed probabilistic and non-deterministic choice over a space $X$ are the convex powercone models, due independently to Mislove, and to Tix, Keimel, and Plotkin, and the continuous prevision model of the author. We show that, up to some minor details, these models are isomorphic whenever $X$ is a continuous, coherent cpo, and whether the particular brand of non-determinism we focus on is demonic, angelic, or chaotic. The construction also exhibits domains of continuous previsions as retracts of well-known continuous cpos, providing simple bases for the various continuous cpos of continuous previsions. This has practical relevance to computing approximations of operations on previsions.

## 1 Introduction

Continuous lower and upper previsions, and forks, were proposed in [5] as adequate models for mixed non-deterministic (demonic, angelic, and chaotic respectively) and probabilistic choice. At the end of this paper, it was claimed that there was a strong relation between this model and that discovered independently by Mislove [13] and by Tix [16,17], consisting of convex non-empty subsets of continuous valuations, which are also compact saturated, resp. closed, resp. (compact) lenses—the so-called *convex powercones*. We make this connection more precise, and to show that this "strong relation" is in fact an isomorphism, provided $X$ is a coherent continuous pointed cpo.

Before we go on, let us mention that Keimel and Plotkin [10] solved a very similar problem, under the guise of finding predicate transformers characterizing convex powercones. Keimel and Plotkin's so-called functional representations of predicate transformers map elements of convex powercones over a cpo $X$ to certain continuous functionals very much like our continuous previsions (essentially, up to the replacement of $\mathbb{R}^+$ by $\overline{\mathbb{R}^+} = \mathbb{R}^+ \cup \{+\infty\}$). However, Keimel and Plotkin's convex powercones are composed of convex subsets of continuous valuations, and the latter may be unbounded. In practice, convex subsets of continuous *probabilities* (such that the measure of the whole space is 1) or *subprobabilities* (at most 1) seem to fit more tightly our needs in modeling probabilistic choice, and Keimel and Plotkin note that "it would be more natural, from the point of view of computer science applications, to restrict to subprobability valuations, rather than allowing all of them." This is what we do here.

Our isomorphism result is not a consequence of the theorems of Keimel and Plotkin, although it is likely that adapting their proofs (and in fact, making them murkier) would

* Partially supported by the INRIA ARC ProNoBis.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 318–333, 2008.

give us the desired results. We take the route of [7, Section 11.7], and prove the isomorphism in two steps: first, isomorphism theorems on fairly general classes of topological spaces $X$, but where convexity has to be replaced by a slightly stronger notion; second, the proof that the strong notions of convexity coincide with ordinary convexity on coherent continuous pointed cpos.

**Outline.** We quickly go over preliminaries in Section 2, then show our first isomorphism theorem in the demonic case, in Section 3. This relies on the notion of strong convexity, and uses notions of barycenters à la Choquet-Bishop-de Leeuw. Showing that strong convexity reduces to convexity in the case of cpos is the subject of Section 4, and is the only section that relies on theorems by Tix, Keimel, and Plotkin. We deal with the angelic case in Section 5, which we reduce to the demonic case by the so-called *convex-concave duality*. The chaotic case is now ripe for treatment in Section 6. We conclude in Section 7.

**Related Work.** Clearly [16,17,10] and [5] are most relevant. Other relevant material will be cited on the fly.

## 2 Preliminaries

See [1,3,12] for background material on domain theory and topology. A *cpo* $X$ is a partially ordered set (poset) in which every directed set has a least upper bound, or sup. We write $\leq$ its ordering. The Scott topology on a poset has as opens all upward-closed subsets $U$ such that whenever $(z_i)_{i\in I}$ is a directed family having a sup $z$ in $U$, then some $z_i$ is in $U$ already. The *way-below* relation $\ll$ on a poset is defined by $x \ll y$ iff whenever $(z_i)_{i\in I}$ is a directed family having a sup $z$ with $y \leq z$, then $x \leq z_i$ for some $i \in I$. A poset $X$ is *continuous* iff every $x \in X$ is the directed sup of all elements $y \ll x$. A *basis* of $X$ is then a subset $B$ of $X$ such that every $x \in X$ is the directed sup of all elements $y \in B$ such that $y \ll x$. The Scott topology then has a basis of open sets of the form $\uparrow\!\!\!\uparrow y = \{x \in X | y \ll x\}$, $y \in B$.

For any topological space (e.g., cpo) $X$, let $\langle X \to \mathbb{R}^+\rangle$ be the poset of all bounded continuous maps from $X$ to $\mathbb{R}^+$. We take $\mathbb{R}^+$ with the Scott topology, whose non-trivial opens are the open intervals $(t, +\infty)$, $t \in \mathbb{R}^+$, and order $\langle X \to \mathbb{R}^+\rangle$ pointwise. A *continuous prevision* $F$ on a topological space (e.g., a cpo) $X$ [5] is a Scott-continuous map from $\langle X \to \mathbb{R}^+\rangle$ to $\mathbb{R}^+$ such that $F(af) = aF(f)$ for every $a \in \mathbb{R}^+$ (*positive homogeneity*). A prevision $F$ is *lower* iff $F(h + h') \geq F(h) + F(h')$ for every $h, h'$, *upper* iff $F(h + h') \leq F(h) + F(h')$ for every $h, h'$, *linear* iff $F(h + h') = F(h) + F(h')$, *normalized* iff $F(a + h) = a + F(h)$ for every function $h$ and constant $a \in \mathbb{R}^+$, *subnormalized* iff $F(a + h) \leq a + F(h)$ for every $h$ and constant $a$. A *fork* is a pair $(F^-, F^+)$ of continuous previsions, where $F^-$ is lower, $F^+$ is upper, and *Walley's condition* $F^-(h + h') \leq F^-(h) + F^+(h') \leq F^+(h + h')$ holds for every $h, h'$. A fork is normalized, resp. sub-normalized, whenever both $F^-$ and $F^+$ are. We shall concentrate on normalized previsions and forks in the sequel.

It was shown in [5] that, among continuous normalized previsions, the lower brand was an adequate model of mixed probabilistic and demonically non-deterministic choice, the upper brand was one of mixed probabilistic and angelically non-deterministic

choice, while normalized forks were an adequate model of mixed probabilistic and chaotically non-deterministic choice. It was essentially well-known since Tix [15] that the space of continuous (subnormalized, resp. normalized) *linear* previsions were isomorphic to Jones' space $\mathbf{V}_{\leq 1}(X)$ (resp., $\mathbf{V}_1(X)$) of subprobability (resp. probability) valuations. A *valuation* is a map $p$ from the set $\mathcal{O}(X)$ of all opens of $X$ to $\overline{\mathbb{R}^+}$ that is strict ($p(\emptyset) = 0$), monotone ($U \subseteq V$ implies $p(U) \leq p(V)$), and modular ($p(U \cup V) + p(U \cap V) = p(U) + p(V)$). *Subprobability* valuations (resp., *probability* valuations) are those such that $p(X) \leq 1$ (resp., $p(X) = 1$). The valuation $p$ is *continuous* iff $p(\bigcup_{i \in I} U_i) = \sup_{i \in I} p(U_i)$ for every directed family $(U_i)_{i \in I}$ of opens of $X$. We shall always equip each cpo, and in fact each poset, $X$ with its Scott topology. However, we shall consider more general topological spaces in the sequel. For every open $U$, let $\chi_U$ map $x$ to 1 if $x \in U$, to 0 otherwise. The isomorphism between the space $\mathbf{P}_1^{\triangle}(X)$ of continuous normalized linear previsions $G$ on $X$ and $\mathbf{V}_1(X)$ maps $G$ to $p = \gamma_{\mathcal{C}}(G)$ defined by $p(U) = G(\chi_U)$ for every open $U$, and conversely, maps $p$ to $G = \alpha_{\mathcal{C}}(p)$ defined by letting $G(h)$ be the Choquet integral of $h$ along $p$ [5]. (The Choquet integral of $h \in \langle X \to \mathbb{R}^+ \rangle$ along $p$, which we shall write $\oint_{x \in X} h(x)dp$, is defined as the ordinary Riemann integral $\int_0^{+\infty} p(h^{-1}(t, +\infty))dt$. This is a continuous linear prevision of $h$ and is also linear and Scott-continuous in $p$, whenever $p$ is a continuous valuation. Note that Choquet integration is also defined when $p$ is merely a so-called *game* [4].)

Let $\nabla \mathbf{P}_1(X)$ be the space of continuous normalized lower previsions on $X$, $\mathbf{P}_1^{\triangle}(X)$ that of all continuous normalized linear previsions (with the Scott topology, ordered pointwise), and $\mathbf{P}_{1\,wk}^{\triangle}(X)$ the same space with the *weak topology*, defined as the smallest that contains the subbasic opens $[f > r] = \{G \in \mathbf{P}_{1\,wk}^{\triangle}(X) | G(f) > r\}$, $f \in \langle X \to \mathbb{R}^+ \rangle$, $r \in \mathbb{R}^+$. The Scott topology is always finer than the weak topology. When $X$ is a continuous cpo with a least element, both topologies coincide, i.e., $\mathbf{P}_{1\,wk}^{\triangle}(X) = \mathbf{P}_1^{\triangle}(X)$. This is easily obtained from the coincidence of the two topologies on spaces of valuations, through the isomorphism between continuous valuations and continuous linear previsions above (see [9], who refers to Tix [15, Satz 4.10], who cites Kirch [11, Satz 8.6]; see also [7, Proposition 3.7.12].)

The relation between spaces of previsions and sets of convex subsets of valuations alluded to in the introduction takes the following form, in the demonic case [5, Proposition 4]. Let the *Smyth powerdomain* $\mathcal{Q}(Y)$ of a topological space be the set of all non-empty compact saturated subsets (see below) of $Y$, ordered by reverse inclusion $\supseteq$. (This is a standard model of demonic non-determinism alone [1].) Then, there is a map $CCoeur_1 : \nabla \mathbf{P}_1(X) \to \mathcal{Q}(\mathbf{P}_{1\,wk}^{\triangle}(X))$ sending each continuous normalized lower prevision $F$ to its *heart* $CCoeur_1(F) = \{G \in \mathbf{P}_1^{\triangle}(X) | F \leq G\}$. Whenever $X$ is stably compact (see below), the heart is non-empty and compact saturated. (Both properties are non trivial.) Moreover, the heart is *convex*: for any two $G, G' \in CCoeur_1(F)$, for any $\alpha \in [0, 1]$, $\alpha G + (1 - \alpha)G'$ is in $CCoeur_1(F)$ (trivial). Conversely, there is a map $\bigsqcap : \mathcal{Q}(\mathbf{P}_{1\,wk}^{\triangle}(X)) \to \nabla \mathbf{P}_1(X)$, sending $\mathcal{Q}$ to $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \min_{G \in \mathcal{Q}} G(h)$. (The min is, indeed, attained.) $CCoeur_1$ and $\bigsqcap$ are Scott-continuous, and form a Galois insertion, i.e., $\bigsqcap \circ CCoeur_1 = \mathrm{id}$ (Rosenmuller's Theorem), and $CCoeur_1 \circ \bigsqcap \supseteq \mathrm{id}$. We shall denote Galois connections, i.e., pairs of monotone maps $\alpha$, $\gamma$ such that $\alpha \circ \gamma \leq \mathrm{id}$ and $\mathrm{id} \leq \gamma \circ \alpha$, by $\alpha \dashv \gamma$; so $CCoeur_1 \dashv \bigsqcap$ (recall that $\leq$ is $\supseteq$ on $\mathcal{Q}(\mathbf{P}_{1\,wk}^{\triangle}(X))$).

A subset $Q$ of $X$ is *compact* iff one can extract a finite subcover from every open cover of $Q$. It is *saturated* iff it is the intersection of all opens containing it, a.k.a. it is upward-closed in the *specialization quasi-ordering* $\leq$, defined by $x \leq y$ iff every open containing $x$ contains $y$. A topological space $X$ is *stably compact* (taking Jung's definitions [9]) iff $X$ is $T_0$ ($\leq$ is an ordering), *well-filtered* (for every filtered family $(Q_i)_{i\in I}$ of compact saturated subsets, for every open $U$, if $\bigcap_{i\in I} Q_i \subseteq U$ then $Q_i \subseteq U$ already for some $i \in I$), *locally compact* (whenever $x \in U$ with $U$ open, there is a compact saturated subset $Q$ such that $x \in int(Q) \subseteq Q \subseteq U$, where $int(Q)$ denotes the interior of $Q$), *coherent* (the intersection of any two compact saturated subsets is again so) and *compact*. Every continuous cpo $X$ is well-filtered and locally compact. If additionally $X$ is *pointed*, i.e., has a least element, then $X$ is compact. If finally $X$ is also coherent, then $X$ is stably compact. Stable compactness has a long history, going back to Nachbin (1948; see [9]).

## 3 Demonic Non-determinism + Probabilistic Choice

The central question of this paper, in the demonic case, is whether the pair $CCoeur_1 \dashv \bigsqcap$ actually defines an isomorphism between $\bigtriangledown \mathbf{P}_1(X)$ and some suitable subset of $\mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$. One natural candidate is $\mathcal{Q}^{cvx}(\mathbf{P}^{\triangle}_{1\,wk}(X))$, the space of all elements of $\mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$ that are convex—since the heart is always convex.

However, the right notion we need is that of *strong convexity*, defined below. (Connoisseurs will note that the same idea is the root of the classic Choquet-Bishop-de Leeuw extension to the Krein-Milman Theorem.) One first notes that convex sets $\mathcal{Q}$ of linear previsions are those that are stable by taking *finite barycenters*, i.e., such that for any finite set $G_0, G_1, \ldots, G_n$ of $n+1$ elements of $\mathcal{Q}$, for any coefficients $a_0, a_1, \ldots, a_n \in \mathbb{R}^+$ with $\sum_{i=0}^n a_i = 1$, $\sum_{i=0}^n a_i G_i$ is again in $\mathcal{Q}$. On any space $Y$, the *Dirac valuation* $\delta_y$ defined so that $\delta_y(V) = 1$ if $y \in V$, 0 otherwise, is a continuous probability valuation, and so are the *simple probability valuations* of the form $\sum_{i=0}^n a_i \delta_{y_i}$, with $a_0, a_1, \ldots, a_n$ as above. The Choquet integral of $h \in \langle Y \to \mathbb{R}^+ \rangle$ along such a simple probability valuation yields $\sum_{i=0}^n a_i h(y_i)$. It follows that we can rewrite the finite barycenter $\sum_{i=0}^n a_i G_i$ as $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} G(h) d\mathcal{P}$, where $\mathcal{P}$ is the simple probability valuation $\sum_{i=0}^n a_i \delta_{G_i}$ on $\mathbf{P}^{\triangle}_{1\,wk}(X)$ (an element of $\mathbf{P}^{\triangle}_1(\mathbf{P}^{\triangle}_{1\,wk}(X))$!). This allows us to define the general notion of *barycenter* $Bary(\mathcal{P})$ of a continuous probability valuation $\mathcal{P} \in \mathbf{P}^{\triangle}_1(\mathbf{P}^{\triangle}_{1\,wk}(X))$ as $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} G(h) d\mathcal{P} \in \mathbf{P}^{\triangle}_1(X)$.

Say that $\mathcal{P}$ is *supported* on $\mathcal{Q} \subseteq \mathbf{P}^{\triangle}_{1\,wk}(X)$ iff $\mathcal{P}(\mathcal{U}) = 1$ for every open set of continuous probability valuations containing $\mathcal{Q}$. This intuitively says that $\mathcal{P}$ bears no mass outside $\mathcal{Q}$. One can check that $\sum_{i=0}^n a_i G_i$ is supported on $\mathcal{Q}$ (where $\mathcal{Q}$ is upward-closed) iff $G_i \in \mathcal{Q}$ for all $i$ such that $a_i \neq 0$.

We then say that $\mathcal{Q}$ is *strongly convex* iff $Bary(\mathcal{P}) \in \mathcal{Q}$ for every $\mathcal{P} \in \mathbf{P}^{\triangle}_1(\mathbf{P}^{\triangle}_{1\,wk}(X))$ that is supported on $\mathcal{Q}$. Whenever $\mathcal{Q}$ is strongly convex, this property must hold whenever $\mathcal{P}$ is a simple probability valuation, showing that every strongly convex upward-closed set is convex. The converse fails, as we now show. Let $X$ be $\mathbb{N} \cup \{+\infty\}$, with the usual ordering and its Scott topology. Let $\mathcal{Q}$ be the set of all linear previsions of the

form $\lambda h \cdot \sum_{i=0}^{n} a_i h(k_i)$, where $a_0, \ldots, a_n$ are as above and $k_i \in \mathbb{N}$. This is the convex hull of the set of linear previsions of the form $\alpha_{\mathcal{C}}(\delta_k)$, $k \in \mathbb{N}$, and is therefore convex. (Recall that $\alpha_{\mathcal{C}}(\delta_k)$ is the image of $\delta_k$ through the isomorphism between continuous valuations and continuous linear previsions, and maps $h$ to $h(k)$.) It is clear that $\delta_{+\infty}$ is not in $\mathcal{Q}$. However, $\delta_{+\infty}$ arises as $Bary(\delta_{\delta_{+\infty}})$, and we check that $\delta_{\delta_{+\infty}}$ is supported on $\mathcal{Q}$: any open containing $\mathcal{Q}$ must contain, say, $\delta_0$, hence also $\delta_{+\infty}$, since $\delta_0 \leq \delta_{+\infty}$.

**Proposition 1.** *Let $X$ be stably compact, and $F \in \bigtriangledown \mathbf{P}_1(X)$. Then $CCoeur_1(F)$ is strongly convex.*

*Proof.* We first observe that, given any compact saturated subset $Q$ of a space $Y$, given any continuous probability valuation $p$ on $Y$ that is supported on $Q$, for any $h \in \langle Y \to \mathbb{R}^+ \rangle$: $(*)$ $\oint_{y \in Y} h(y) dp \geq \min_{y \in Q} h(y)$. Let $a = \min_{y \in Q} h(y)$ (which is attained since $Q$ is compact). For all $t < a$, $f^{-1}(t, +\infty)$ contains $Q$, so $p(f^{-1}(t, +\infty)) = 1$; hence $\oint_{y \in Y} h(y) dp = \int_0^{+\infty} p(f^{-1}(t, +\infty)) dt \geq \int_0^a p(f^{-1}(t, +\infty)) dt = a$.

For $f$ an arbitrary element of $\langle X \to \mathbb{R}^+ \rangle$, apply $(*)$ to the case $Y = \mathbf{P}^{\triangle}_{1\ wk}(X)$, $p = \mathcal{P}$ supported on $\mathcal{Q} = CCoeur_1(F)$, taking $h(G) = G(f)$. (Note that $h$ is continuous, precisely because $Y$ is equipped with the weak topology.) We get $\oint_{G \in \mathbf{P}^{\triangle}_{1\ wk}(X)} G(f) d\mathcal{P} \geq \min_{G \in \mathcal{Q}} G(f) = \bigsqcap \mathcal{Q}(f)$. However, remember that $\bigsqcap \circ CCoeur_1 = \mathrm{id}$, so $\bigsqcap \mathcal{Q} = F$. Also, $\oint_{G \in \mathbf{P}^{\triangle}_{1\ wk}(X)} G(f) d\mathcal{P} = Bary(\mathcal{P})(f)$. So $Bary(\mathcal{P}) \geq F$, i.e., $Bary(\mathcal{P}) \in CCoeur_1(F)$. □

The converse direction, that strongly convex non-empty compact saturated subsets of $\mathbf{P}^{\triangle}_{1\ wk}(X)$ arise from some element of $\bigtriangledown \mathbf{P}_1(X)$, relies on the following key Proposition 2. To appreciate it, look at the case when $\mathcal{Q}$ is the upward closure $\uparrow \{G_1, \ldots, G_n\}$ of $\{G_1, \ldots, G_n\}$ in $\mathbf{P}^{\triangle}_{1\ wk}(X)$: up to some details, the proposition states that if for each $h$, there is an $i$ with $G(h) \geq G_i(h)$ (not necessarily the same $i$ for each $h$), then there are coefficients $a_0, a_1, \ldots, a_n$ with $\sum_{i=0}^{n} a_i = 1$ such that $G(h) \geq \sum_{i=0}^{n} a_i G_i(h)$ for *all* $h$. (This is similar to a key step in some proofs of the minimax theorem.)

The proof relies on Roth's Sandwich Theorem ([14], [17, Theorem 3.1]), which states that on every ordered cone $C$, for every positively homogeneous super-additive function $q : C \to \overline{\mathbb{R}^+}$ and every positively homogeneous sub-additive function $p : C \to \overline{\mathbb{R}^+}$ such that $a \leq b$ implies $q(a) \leq p(b)$ (e.g., when $q \leq p$ and either $q$ or $p$ is monotonic), then there is a monotonic linear function $f : C \to \overline{\mathbb{R}^+}$ such that $q \leq f \leq p$. A *cone* is a set $C$, together with a binary operation $+$ turning it into a commutative monoid and a scalar multiplication $\cdot$ from $\mathbb{R}^+ \times C$ to $C$, such that $1 \cdot a = a$, $0 \cdot a = 0$, $(rs) \cdot a = r \cdot (s \cdot a)$, $r \cdot (a + b) = r \cdot a + r \cdot b$, and $(r + s) \cdot a = r \cdot a + s \cdot a$. An *ordered cone* is equipped in addition with a partial ordering $\leq$ making $+$ and $\cdot$ monotonic. The function $q$ is *positively homogeneous* iff $q(s \cdot a) = sq(a)$ for all $s \in \mathbb{R}^+$, *super-additive* iff $q(a + b) \geq q(a) + q(b)$, *sub-additive* iff $q(a + b) \leq q(a) + q(b)$, and *linear* iff it has all three properties.

**Proposition 2.** *Let $X$ be stably compact, $\mathcal{Q}$ be a non-empty compact saturated subset of $\mathbf{P}^{\triangle}_{1\ wk}(X)$, $G \in \mathbf{P}^{\triangle}_{1\ wk}(X)$ such that $\bigsqcap \mathcal{Q} \leq G$. Then there is a continuous probability valuation $\mathcal{P}$ on $\mathbf{P}^{\triangle}_{1\ wk}(X)$ that is supported on $\mathcal{Q}$, and such that $Bary(\mathcal{P}) \leq G$.*

*Proof.* Consider the ordered cone $C = \langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}^+ \rangle$, with the obvious $+$ and $\cdot$, and the pointwise ordering. For every $\varphi \in \langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}^+ \rangle$, let $q(\varphi) = \min_{G \in \mathcal{Q}} \varphi(G)$. This is clearly positively homogeneous, super-additive, and monotonic. In fact, $q$ is even (Scott-)continuous: recall from [4] that, for any compact saturated subset $Q$ of a space $Y$, the *unanimity game* $\mathfrak{u}_Q$ (mapping each open $U$ containing $Q$ to 1 and all others to 0) is a continuous game, that the Choquet integral $\oint_{y \in Y} f(y) d\mathfrak{u}_Q$ equals $\min_{y \in Q} f(y)$—so that $q(\varphi) = \oint_{G \in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G) d\mathfrak{u}_{\mathcal{Q}}$—and that Choquet integration wrt. a continuous game is Scott-continuous in the integrated function.

Let $p(\varphi) = \inf_{\substack{f \in \langle X \to \mathbb{R}^+ \rangle \\ \forall G' \in \mathcal{Q} \cdot \varphi(G') \leq G'(f)}} G(f)$. (We take this to mean $+\infty$ if the inf is taken over an empty family of values.) It is easy to see that $p(a\varphi) = ap(\varphi)$ for every $a \in \mathbb{R}^+$: the case $a = 0$ works by realizing that $f = 0$ satisfies $\varphi(G') = G'(f)$ for all $G \in \mathcal{Q}$, and then $G(f) = 0$, the case $a \neq 0$ works by substituting $f/a$ for $f$ in the formula for $p$. Checking that $p$ is super-additive is only slightly harder: $p(\varphi) + p(\varphi')$ equals the inf of $G(f) + G(g) = G(f + g)$ when $f$ and $g$ range over functions such that $\varphi(G') \leq G'(f)$ and $\varphi(G') \leq G'(g)$ for all $G' \in \mathbf{P}^{\triangle}_{1\,wk}(X)$. Since every such $G$' is linear, they all satisfy $(\varphi + \varphi')(G') \leq G'(f + g)$, so $p(\varphi) + p(\varphi')$ is greater than of equal to the inf of $G(f + g)$ when $f$ and $g$ satisfy the weaker condition $(\varphi + \varphi')(G') \leq G'(f + g)$. This is then greater than or equal to $p(\varphi + \varphi')$.

We now check that $q \leq p$. This is where we use the assumption $\bigsqcap \mathcal{Q} \leq G$. Fix $\varphi \in \langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}^+ \rangle$. For all $f \in \langle X \to \mathbb{R}^+ \rangle$ such that $\forall G' \in \mathcal{Q} \cdot \varphi(G') \leq G'(f)$, we have $G(f) \geq \bigsqcap \mathcal{Q}(f) = \min_{G' \in \mathcal{Q}} G'(f) \geq \min_{G' \in \mathcal{Q}} \varphi(G') = q(\varphi)$. Now take infs over $f$ on each side, whence $p(\varphi) \geq q(\varphi)$.

Using Roth's Sandwich Theorem, there is a monotonic linear functional $\mathcal{G}_0$ from $\langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}^+ \rangle$ to $\overline{\mathbb{R}^+}$ such that $q \leq \mathcal{G}_0 \leq p$. We claim that $\mathcal{G}_0$ never takes the value $+\infty$. Indeed, for any $\varphi \in \langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}^+ \rangle$, letting $a = \sup_{G' \in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G')$, $\mathcal{G}_0(\varphi) \leq p(\varphi) \leq ap(\chi_{\mathbf{P}^{\triangle}_{1\,wk}(X)}) \geq G(\chi_X) = 1$ because we may take $f = \chi_X$ in the definition of $p$, as $\varphi(G') \leq a = aG'(\chi_X)$, $G'$ being normalized. So $\mathcal{G}_0$ is a linear prevision. Since $q \leq \mathcal{G}_0$ and $q(\chi_{\mathbf{P}^{\triangle}_{1\,wk}(X)}) = 1$, it follows that $\mathcal{G}_0(\chi_{\mathbf{P}^{\triangle}_{1\,wk}(X)}) = 1$; since $\mathcal{G}_0$ is linear, this is enough to show that $\mathcal{G}_0$ is normalized.

But $\mathcal{G}_0$ is not necessarily continuous. We use the machinery, based on the Scott extension formula, developed in [5, Long version, Appendix], and which we recall briefly now. By Claim Q of op.cit., for any stably compact space $Y$ (the result is stated for the slightly more general class of compact, stably core compact spaces), we may define a continuous functional $\mathfrak{r}(F)$ from $\langle Y \to \mathbb{R}^+ \rangle$ to $\overline{\mathbb{R}^+}$ from any functional $F$ from $\langle Y \to \mathbb{R}^+ \rangle$ to $\overline{\mathbb{R}^+}$ by the formula $\mathfrak{r}(F)(f) = \sup_{g \in B, g \ll f} F(g)$, where $B$ is a basis of the continuous poset $\langle Y \to \mathbb{R}^+ \rangle$ (described in Claim K) and $\ll$ is its way-below relation; then $\mathfrak{r}(F)$ is the largest continuous functional below $F$, and is a continuous normalized linear prevision whenever $F$ is a normalized linear prevision.

In our case, observe that $Y = \mathbf{P}^{\triangle}_{1\,wk}(X)$ is stably compact: the isomorphism $\alpha_{\mathcal{C}}$, $\gamma_{\mathcal{C}}$ between $\mathbf{P}^{\triangle}_1(X)$ and $\mathbf{V}_1(X)$ also defines an isomorphism between $\mathbf{P}^{\triangle}_{1\,wk}(X)$ and $\mathbf{V}_{1\,wk}(X)$ (where the latter is defined with the weak topology, whose subbasic opens are $[f > r] = \{p \in \mathbf{V}_1(X) | \oint_{x \in X} f(x) dp > r\}$ are in one to one correspondence to those of $\mathbf{P}^{\triangle}_{1\,wk}(X)$). And $\mathbf{V}_{1\,wk}(X)$ is stably compact as soon as $X$ is, a result

due to Jung [9, Theorem 3.2]. So the machinery applies: $\mathcal{G} = \mathfrak{r}(\mathcal{G}_0)$ is a continuous normalized linear prevision, and $\mathcal{G} \leq \mathcal{G}_0$. Moreover, since $\mathcal{G}$ is the largest continuous functional below $\mathcal{G}_0$, and $q$ is continuous, we have $q \leq \mathcal{G} \leq \mathcal{G}_0 \leq p$.

Using the isomorphism $\alpha_{\mathfrak{C}}$, $\gamma_{\mathfrak{C}}$, let $\mathcal{P} = \gamma_{\mathfrak{C}}(\mathcal{G})$: this is a continuous normalized valuation on $\mathbf{P}^{\triangle}_{1\,wk}(X)$. We claim it is supported on $\mathcal{Q}$. For every open $\mathcal{U}$ containing $\mathcal{Q}$ indeed, $q(\chi_{\mathcal{U}}) \leq \mathcal{G}(\chi_{\mathcal{U}})$. But $q(\chi_{\mathcal{U}}) = \min_{G' \in \mathcal{Q}} \chi_{\mathcal{U}}(G') = 1$, and $\mathcal{G}(\chi_{\mathcal{U}}) = \mathcal{P}(\mathcal{U})$. Since $\mathcal{P}(\mathcal{U}) \leq 1$ anyway, $\mathcal{P}(\mathcal{U}) = 1$.

For every $f \in \langle X \to \mathbb{R}^+ \rangle$, let $\varphi$ be the function mapping $G'$ to $G'(f)$. Note that $\mathcal{G}(\varphi) = \alpha_{\mathfrak{C}}(\mathcal{P})(\varphi) = \oint_{G' \in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G') d\mathcal{P} = \oint_{G' \in \mathbf{P}^{\triangle}_{1\,wk}(X)} G'(f) d\mathcal{P} = Bary(\mathcal{P})(f)$, while $p(\varphi) = \inf_{\substack{g \in \langle X \to \mathbb{R}^+ \rangle \\ \forall G' \in \mathcal{Q} \cdot \varphi(G') \leq G'(g)}} G(g) = \inf_{\substack{g \in \langle X \to \mathbb{R}^+ \rangle \\ \forall G' \in \mathcal{Q} \cdot G'(f) \leq G'(g)}} G(g) \leq G(f)$. So $Bary(\mathcal{P}) \leq G$, as announced. □

Let $Conv(\mathcal{Q})$, the *strong convex closure* of $\mathcal{Q}$, be $\{Bary(\mathcal{P}) \mid \mathcal{P} \text{ supported on } \mathcal{Q}\}$. Using Proposition 2, we may characterize the action of $CCoeur_1 \circ \bigsqcap$ by $CCoeur_1(\bigsqcap \mathcal{Q}) = \uparrow Conv(\mathcal{Q})$ for every $\mathcal{Q} \in \mathcal{Q}(\mathbf{P}^{\triangle}_{1\,wk}(X))$. Recall that $\mathcal{Q} \subseteq CCoeur_1(\bigsqcap \mathcal{Q})$. By Proposition 1, it follows that $Conv(\mathcal{Q}) \subseteq CCoeur_{\leq 1}(\bigsqcap \mathcal{Q})$. Since the heart is upward-closed, $\uparrow Conv(\mathcal{Q}) \subseteq CCoeur_1(\bigsqcap \mathcal{Q})$. Conversely, if $G \in CCoeur_1(\bigsqcap \mathcal{Q})$, i.e., $\bigsqcap \mathcal{Q} \leq G$, then by Proposition 2, there is a continuous probability valuation $\mathcal{P}$ supported on $\mathcal{Q}$, such that $Bary(\mathcal{P}) \leq G$. This means that $Bary(\mathcal{P}) \in Conv(\mathcal{Q})$, so $G \in \uparrow Conv(\mathcal{Q})$.

**Theorem 1 (Isomorphism).** *Let $X$ be stably compact. Then $CCoeur_1$ and $\bigsqcap$ define an isomorphism between $\bigtriangledown \mathbf{P}_1(X)$ and the space $\mathcal{Q}^{Cvx}(\mathbf{P}^{\triangle}_{1\,wk}(X))$ of strongly convex non-empty compact saturated subsets of $\mathbf{P}^{\triangle}_{1\,wk}(X)$, ordered by $\supseteq$.*

*Proof.* It is enough to realize that for every $\mathcal{Q} \in \mathcal{Q}^{Cvx}(\mathbf{P}^{\triangle}_{1\,wk}(X))$, $\uparrow Conv(\mathcal{Q}) = \uparrow \mathcal{Q} = \mathcal{Q}$, while $CCoeur_1(\bigsqcap \mathcal{Q}) = \uparrow Conv(\mathcal{Q})$. The identity $\bigsqcap \circ CCoeur_1 = \mathrm{id}$ is already known from [5]. □

## 4 The Cpo Case

We may refine Theorem 1 and replace strong convexity by the mere notion of convexity, when $X$ is a coherent, continuous and pointed (hence stably compact) cpo. This comes close to the results of Keimel and Plotkin [10], who show that the space of superadditive, positively homogeneous and Scott-continuous functionals from $[X \to \overline{\mathbb{R}^+}]$ to $\overline{\mathbb{R}^+}$, is isomorphic to $\mathcal{Q}^{cvx}(\overline{\mathbf{V}}(X))$, where $[X \to Y]$ denotes the cpo of all continuous maps from $X$ to $Y$ (not just the bounded ones), and $\overline{\mathbf{V}}(X)$ is the set of all valuations (not just the normalized ones, not even those that are bounded, i.e. do not take the value $+\infty$). Note also the use of $\mathcal{Q}^{cvx}$ here instead of $\mathcal{Q}^{Cvx}$. Despite the apparent added generality of the results of [10], they do not seem to entail ours. (Try it!)

The key to our result is to realize that any compact saturated, convex subset of $\mathbf{P}^{\triangle}_{1\,wk}(X)$ is in fact strongly convex. We need the following easily proved variant of [17, Theorem 3.8] first [6, Appendix A]. Call *continuous cone* any ordered cone $C$ which is continuous qua poset, and where $+$ and $\cdot$ are Scott-continuous. (This is as the continuous d-cones of [17], except we don't require $C$ to be a cpo.) It is *additive* iff

$x_1 \ll y_1$ and $x_2 \ll y_2$ imply $x_1 + x_2 \ll y_1 + y_2$. Call a subset $Z$ of an additive continuous cone *regular* whenever $Z$ is continuous as a sub-partial order of $C$, and whenever $x, y \in Z$ are such that $x \ll_Z y$, then $x \ll_C y$, where $\ll_Z$ and $\ll_C$ are the way-below relations of $Z$ and $C$ respectively.

**Proposition 3.** *Let $C$ be an additive continuous cone, and $Z$ a regular subspace of $C$. For every convex compact subset $K$ of $Z$, for every non-empty convex closed subset $F$ of $Z$ disjoint from $K$, there is $a \in \overline{\mathbb{R}^+}$, $a > 1$, and a continuous linear function $f : C \to \overline{\mathbb{R}^+}$ such that $f(z) > a$ for all $z \in K$ and $f(y) \leq 1$ for every $y \in F$.*

Caveat: The topology of $Z$ is that induced by that of $C$ (whose opens are of the form $V \cap Z$, $V$ open in $C$), but there is no reason in general that this should coincide with the Scott topology of the ordering $\leq$ on $Z$. Such pathologies do not arise when $Z$ is a regular subspace of $C$ [6, Claim D, Appendix A]. For every continuous cpo $Y$, the space $C = \mathbf{V}(Y)$ of all continuous bounded valuations is an additive continuous cone, and $Z = \mathbf{V}_{\leq 1}(Y)$ is a regular subspace of $C$ [6, Claim F, Appendix A]. However, $\mathbf{V}_1(Y)$ is *not* regular in $\mathbf{V}(Y)$, when $Y$ has a least element $\bot$, as $\delta_\bot \ll_{\mathbf{V}_1(X)} \delta_\bot$ but $\delta_\bot \not\ll_{\mathbf{V}(X)} \delta_\bot$ (the family $r\delta_\bot$, $r < 1$, has the right-hand side as sup, but no element of this family is greater than or equal the left-hand side).

Recall that a saturated subset is one that is the intersection of all opens containing it. Say that a subset $A$ is *linearly saturated* iff $A$ is the intersection of all convex opens containing it. It is tempting to think that any convex saturated subset should be linearly saturated. This is indeed the case for those subsets that are also compact, as the following variant of [17, Corollary 3.13], due to Jung, shows.

**Proposition 4.** *Let $C$ be a continuous cone, $Z$ be a regular convex subspace of $C$. Every convex compact saturated subset $Q$ of $Z$ is linearly saturated.*

*Proof.* We must show that for every $x \in Z \setminus Q$, there is a convex open subset $V$ containing $Q$ but not $x$. Let $F = \{z \in Z | z \leq x\}$: this is convex, non-empty, and disjoint from $Q$. Build $f$ and $a$ as in Proposition 3. The open $V = f^{-1}(a, +\infty) \cap Z$ of $Z$ fits the bill. In particular, $V$ is convex because $f$ is linear. □

**Theorem 2.** *Let $X$ be a continuous pointed cpo. Every convex compact saturated subset $\mathcal{Q}$ of $\mathbf{P}_1^\triangle(X) = \mathbf{P}_{1\,wk}^\triangle(X)$ is strongly convex.*

*Proof.* Fix $X$, with least element $\bot$. By Edalat's variant of Jones' Theorem [2, Section 3], $Y = \mathbf{V}_1(X)$ is then also a continuous pointed cpo, with least element $\delta_\bot$. Using a trick by Edalat, $X' = X \setminus \{\bot\}$ is a continuous cpo again, and $\mathbf{V}_1(X)$ is isomorphic to $\mathbf{V}_{\leq 1}(X')$. (Send $\nu \in \mathbf{V}_1(X)$ to $\lambda U \in \mathcal{O}(X') \cdot \nu(U)$, and send back $\nu \in \mathbf{V}_{\leq 1}(X')$ to $\nu'$ defined as $\nu'(U) = \nu(U)$ if $U \neq X$, i.e., $\bot \notin U$, $\nu'(U) = 1$ otherwise.) Take $C = \mathbf{V}(X')$, $Z = \mathbf{V}_{\leq 1}(X')$. Let $\mathcal{Q}$ be a convex saturated compact of $Z$, $\mathcal{P}$ an element of $\mathbf{V}_1(Z)$ supported on $\mathcal{Q}$. Since $Z$ is a continuous pointed cpo again, we use again one of Edalat's results [2, Section 3]: every element of $\mathbf{V}_1(Z)$ is the sup of a directed family of simple probability valuations $\mathcal{P}_i$, $i \in I$. Fix an arbitrary convex open $\mathcal{U}$ containing $\mathcal{Q}$. By definition of the Scott topology, some $\mathcal{P}_i$ is in $\mathcal{U}$, from which we deduce easily that the subfamily of those $\mathcal{P}_i$ that are in $\mathcal{U}$ is again directed, with sup $\mathcal{P}$. Let $J$ be the set of indices $i$ such that $\mathcal{P}_i \in \mathcal{U}$. However, since $\mathcal{U}$ is convex and $\mathcal{P}_i$ is simple, the finite

barycenter $Bary(\mathcal{P}_i)$ is in $\mathcal{U}$. It is easy to see that the family $(Bary(\mathcal{P}_i))_{i\in J}$ is directed. Its sup is in $\mathcal{U}$, since $\mathcal{U}$ is upward-closed. Since Choquet integration is continuous in the valuation argument, $Bary$ is continuous, so this sup is just $Bary(\mathcal{P})$. We have shown that $Bary(\mathcal{P}) \in \mathcal{U}$ for every convex open $\mathcal{U}$ containing $\mathcal{Q}$. By Proposition 4, $\mathcal{Q}$ is the intersection of all such convex opens, so $Bary(\mathcal{P}) \in \mathcal{Q}$. This shows that every convex compact saturated subset of $Z = \mathbf{V}_{\leq 1}(X')$ is strongly convex. Now note that $Z$ is isomorphic to $\mathbf{V}_1(X)$, which is isomorphic to $\mathbf{P}_1^{\triangle}(X) = \mathbf{P}_{1\,wk}^{\triangle}(X)$. □

Since convex and strong convexity coincide for compact saturated subsets, the following is immediate.

**Corollary 1 (Isomorphism).** *Let $X$ be a continuous, coherent pointed cpo. $CCoeur_1$ and $\bigsqcap$ define an isomorphism between $\bigtriangledown \mathbf{P}_1(X)$ and $\mathcal{Q}^{cvx}(\mathbf{P}_{1\,wk}^{\triangle}(X)) = \mathcal{Q}^{cvx}(\mathbf{P}_1^{\triangle}(X)) \cong \mathcal{Q}^{cvx}(\mathbf{V}_1(X))$.*

Note that $CCoeur_1$ and $\bigsqcap$ also exhibit $\bigtriangledown \mathbf{P}_1(X)$ as a retract of $\mathcal{Q}(\mathbf{P}_1^{\triangle}(X))$, i.e., they are continuous, and $\bigsqcap \circ CCoeur_1 = \mathrm{id}$. ($\bigsqcap$ is the *retraction*, and $CCoeur_1$ the associated *section*.) By [2], $\mathbf{P}_1^{\triangle}(X) \cong \mathbf{V}_1(X)$ has a basis of *simple normalized linear previsions*, i.e., previsions of the form $\alpha_{\mathcal{C}}(p)$, with $p$ a simple probability valuation. Concretely, these are previsions of the form $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \sum_{i=0}^n a_i h(x_i)$, where $a_0, a_1, \ldots, a_n \in \mathbb{R}^+$, $\sum_{i=0}^n a_i = 1$. It is well-known that, whenever $Y$ is a continuous cpo, $\mathcal{Q}(Y)$ is also a continuous cpo with basis given by the *finitary compacts* $\uparrow \mathcal{E}$, $\mathcal{E}$ a finite subset of $Y$ [1]. It is also known that any retract $Z'$ of a continuous cpo $Z$ is again a continuous cpo, with basis given by the image of any basis of $Z$ by the retraction. So:

**Theorem 3.** *For any continuous, coherent pointed cpo $X$, $\bigtriangledown \mathbf{P}_1(X)$ is a continuous, coherent pointed cpo. A basis is given by previsions of the form $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \min_{i=1}^m \sum_{j=1}^n a_{ij} h(x_j)$, where $a_{ij} \in \mathbb{R}^+$ and $\sum_{j=1}^n a_{ij} = 1$ for each $i$.*

*Proof.* The only things that remain to be proved are that $\bigtriangledown \mathbf{P}_1(X)$ has a least element ($\lambda h \cdot h(\bot)$, i.e., $\alpha_{\mathcal{C}}(\delta_\bot)$), and that $\bigtriangledown \mathbf{P}_1(X)$ is coherent. Note that $\mathcal{Q}(\mathbf{P}_1^{\triangle}(X))$ is a bc-domain, i.e., a continuous cpo where any two elements $Q_1$ and $Q_2$ having an upper bound have a least upper bound (namely $Q_1 \cap Q_2$, which is non-empty because $Q_1$ and $Q_2$ have an upper bound $Q \subseteq Q_1, Q_2$). Every bc-domain is coherent, hence stably compact (see, e.g., [8]), and by a result of Lawson quoted by Jung [9], every retract of a stably compact space is again stably compact. □

The above theorem means that we can always approximate, from below, any continuous normalized lower prevision by one that is computable, using only finitely many $\min$, $+$ and $\cdot$ operations. It is remarkable that we know no proof of this fact that would avoid the relatively daunting constructions above.

## 5 Angelic Non-determinism + Probabilistic Choice

Let the *Hoare powerdomain* $\mathcal{H}(Y)$ be the set of all non-empty closed subsets of $Y$, ordered by inclusion (a standard model of angelic non-determinism alone). This is also a cpo, which is continuous as soon as $Y$ is, and is usually used to model angelic

non-determinism. Let $\bigtriangleup \mathbf{P}_1(X)$ be the space of all continuous normalized *upper* previsions on $X$. Then [5, Proposition 5] there is a map $CPeau_1 : \bigtriangleup \mathbf{P}_1(X) \to \mathcal{H}(\mathbf{P}_{1\,wk}^{\bigtriangleup}(X))$, defined by $CPeau_1(F) = \{G \in \mathbf{P}_1^{\bigtriangleup}(X) | G \leq F\}$, and a map $\bigsqcup : \mathcal{H}(\mathbf{P}_{1\,wk}^{\bigtriangleup}(X)) \to \bigtriangleup \mathbf{P}_1(X)$, sending $\mathcal{F}$ to $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \sup_{G \in \mathcal{F}} G(h)$. When $X$ is stably compact, $\bigsqcup \dashv CPeau_1$ defines what we called a Galois surrection, i.e., $\bigsqcup$ and $CPeau_1$ are monotonic, $\bigsqcup \circ CPeau_1 = \mathrm{id}$, and $CPeau_1 \circ \bigsqcup(\mathcal{F}) \supseteq \mathcal{F}$ for all $\mathcal{F}$. $\bigsqcup$ is continuous, but we do not know whether $CPeau_1$ is continuous in general.

We shall prove that $\bigsqcup$ and $CPeau_1$ define an isomorphism similar to those of the previous sections. The main trick is in using a nice duality between demonic and angelic non-determinism, which we called *convex-concave duality* on games in [4], and which extends to previsions. Very roughly, the idea is to turn any prevision $F$ into the functional $F^\perp = \lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot -F(-h)$. If $F$ is lower, then $F^\perp$ will be upper, and conversely, moreover $F^{\perp\perp} = F$. Unfortunately, $F(-h)$ is in general ill-defined: First, $-h$ does not take its values in $\mathbb{R}^+$ (easy to repair, see below); second, $-h$ is very far from being continuous from $X$ to $\mathbb{R}^+$: the inverse image of the (Scott-)open $(t, +\infty)$ by $-h$ is $h^{-1}(-\infty, -t)$, of which we know nothing.

To correct the first problem, extend any normalized prevision $F$ on $X$ to a functional $\widehat{F} : \langle X \to \mathbb{R} \rangle \to \mathbb{R}$ by letting $\widehat{F}(f) = F(f + a) - a$, for any $a \geq -\inf_{x \in X} f(x)$. (As before, $\langle X \to \mathbb{R} \rangle$ is the space of all bounded continuous maps from $X$ to $\mathbb{R}$, with the Scott topology of the pointwise ordering.) This is independent of $a$, because $F$ is normalized. It is easy to see that $\widehat{F}$ is monotonic (if $f \leq f'$ then $\widehat{F}(f) \leq \widehat{F}(f')$), positively homogeneous (if $r \geq 0$ then $\widehat{F}(rf) = r\widehat{F}(f)$), normalized, and lower, resp. upper, resp. linear, resp. Scott-continuous when $F$ is.

Solving the second problem is harder. We will have to approximate functions $-h$ with $h \in \langle X \to \mathbb{R}^+ \rangle$ by functions $g$ not from $X$, but from the *de Groot dual* $X^{\mathsf{d}}$ of $X$, to $\mathbb{R}^+$. This is defined (when $X$ is stably compact) as $X$, only with the so-called *cocompact topology*, whose opens are the cocompacts, i.e., subsets of the form $X \setminus Q$, $Q$ compact saturated subset of $X$. Observe that well-filternedness, coherence, and compactness imply that this is indeed a topology. Then $X^{\mathsf{d}}$ is again stably compact, and $X^{\mathsf{dd}} = X$ (see [9] for more background material on this).

A function $f : X \to \mathbb{R}$ is a *step function* if and only if it is of the form $\sum_{i=0}^{n} a_i \chi_{U_i}$, where $X = U_0 \supseteq U_1 \supseteq \ldots \supseteq U_n$ is a sequence of opens, and $a_0 \in \mathbb{R}$, $a_1, \ldots, a_n \in \mathbb{R}^+$. It is well-known (see e.g., [15]) that any element $f$ of $\langle X \to \mathbb{R} \rangle$ is the sup of a directed family of step functions, namely $f_K = a + \frac{1}{2^K} \sum_{k=1}^{\lfloor (b-a)2^K \rfloor} \chi_{f^{-1}(a + \frac{k}{2^K}, +\infty)}$, $K \in \mathbb{N}$, where $a$ is any lower bound for $f$ and $b$ is any upper bound for $f$.

**Definition 1** **($F^\perp$).** *Let $X$ be a stably compact space, $F$ a normalized prevision on $X$. The* dual *$F^\perp$ of $F$ is the map from $\langle X^{\mathsf{d}} \to \mathbb{R}^+ \rangle$ to $\mathbb{R}^+$ defined by $F^\perp(g) = -\inf_{\substack{f \text{ step function} \\ f \geq -g}} \widehat{F}(f)$.*

We sum up the main properties of this construction. The proof is omitted for lack of space, but can be found in [6, Appendix B].

**Theorem 4.** *Let $X$ be a stably compact space. For every normalized prevision $F$ on $X$, $F^\perp$ is a normalized prevision on $X^{\mathsf{d}}$. Moreover: (1) $F^\perp$ is continuous; (2) if $F$ is*

*lower, then $F^\perp$ is upper; (3) if $F$ is upper, then $F^\perp$ is lower; (4) if $F$ is linear, then so is $F^\perp$; (5) if $F$ is continuous, then $F^{\perp\perp} = F$; (6) if $F \leq F'$ then $F'^\perp \leq F^\perp$.*

The main step is to show that, when $g$ is a step function, $F^\perp(g)$ can be defined alternatively as $-\inf_{f \sqsupseteq^{\mathsf{d}} -g} \widehat{F}(f)$, where $\sqsupseteq^{\mathsf{d}}$ is a more constrained relation: $f \sqsupseteq^{\mathsf{d}} -g$ iff one can write $f$ as $-\sum_{i=0}^n a_i \chi_{X \setminus U_i}$ ($U_i$ opens, $\emptyset = U_0 \subseteq U_1 \subseteq \ldots \subseteq U_n$, $a_0 \in \mathbb{R}$, $a_1, \ldots, a_n \in \mathbb{R}^+$), $g$ as $\sum_{i=0}^n a_i \chi_{X \setminus Q_i}$ ($Q_i$ compact saturated subsets, $\emptyset = Q_0 \subseteq Q_1 \subseteq \ldots \subseteq Q_n$), with the *same* coefficients $a_i$, and $Q_i \subseteq U_i$ for each $i$. While the proofs require $\sqsupseteq^d$, we observe that Definition 1 can be simplified by eliminating the recourse to step functions [6, Appendix C].

**Lemma 1.** *Let $X$ be a stably compact space, and $F$ a normalized continuous prevision on $X$. For every $g \in \langle X^{\mathsf{d}} \to \mathbb{R} \rangle$, $F^\perp(g) = -\inf_{\substack{f \in \langle X \to \mathbb{R} \rangle \\ f \geq -g}} \widehat{F}(f)$.*

For every continuous game $\nu$ in the sense of [4] (in particular, a continuous valuation) on a stably compact space, we may define $\nu^\perp$ as $\gamma_{\mathcal{C}}(F^\perp)$ where $F = \alpha_{\mathcal{C}}(\nu)$. One may check that $\nu^\perp(X \setminus Q) = 1 - \nu^\dagger(Q)$, where $\nu^\dagger(Q) = \inf_{U \supseteq Q} \nu(U)$ [6, Claim X, Appendix B]. In the case where $\nu$ is a continuous valuation, the $\nu^\dagger$ construction was already studied by Tix [15, Satz 3.4].

**Proposition 5.** *Let $X$ be stably compact. The map $p \mapsto p^\perp$ defines an isomorphism between $\mathbf{V}_{1\,wk}(X)^{\mathsf{d}}$ and $\mathbf{V}_{1\,wk}(X^{\mathsf{d}})$. The map $F \mapsto F^\perp$ defines an isomorphism between $\mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}$ and $\mathbf{P}^{\triangle}_{1\,wk}(X^{\mathsf{d}})$.*

*Proof.* The second claim follows from the first through the isomorphism $\alpha_{\mathcal{C}}$, $\gamma_{\mathcal{C}}$. For any compact saturated subset $Q$ of $X$, for any real $r$, let $\langle Q < r \rangle = \{p \in \mathbf{V}_1(X) | p^\dagger(Q) < r\}$. By [9, Concluding remarks], the sets $\langle Q < r \rangle$ form a subbasis of the cocompact topology on $\mathbf{V}_{1\,wk}(X)$, provided $X$ is stably compact. (This is stated in terms of sets written $[K \geq r]$, which are the complements of $\langle K < r \rangle$. See [7, Section 6.4] for a proof.) So $p \mapsto p^\perp$ is continuous. Then apply Theorem 4 (6). □

For lower previsions, we used probability valuations $\mathcal{P}$ supported on a compact saturated subsets $\mathcal{Q}$ of $\mathbf{P}^{\triangle}_{1\,wk}(X)$. Upper previsions require us to use *co*supports instead. Say that $\mathcal{P}$ is *co-supported* on $\mathcal{F} \subseteq \mathbf{P}^{\triangle}_{1\,wk}(X)$ iff $\mathcal{P}(\mathcal{U}) = 0$, where $\mathcal{U}$ is the complement of the closure $cl(\mathcal{F})$ of $\mathcal{F}$. It is easy to see that $\mathcal{P}$ is supported on the compact saturated subset $\mathcal{Q}$ of $\mathbf{P}^{\triangle}_{1\,wk}(X)$ iff $\mathcal{P}^\perp$ is co-supported on the closed subset $\mathcal{Q}$ of $\mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}$.

Say that a subset $\mathcal{F}$ of $\mathbf{P}^{\triangle}_{1\,wk}(X)$ is *co-strongly convex* iff $Bary(\mathcal{P}) \in \mathcal{F}$ for every $\mathcal{P} \in \mathbf{V}_1(\mathbf{P}^{\triangle}_{1\,wk}(X))$ that is co-supported on $\mathcal{F}$. When $\mathcal{P}$ is simple, say $\mathcal{P} = \sum_{i=0}^n a_i \delta_{G_i}$, then $\mathcal{P}$ is co-supported on $\mathcal{F}$ (when $\mathcal{F}$ is downward-closed) iff every $G_i$ such that $a_i \neq 0$ is in $\mathcal{F}$; so every co-strongly convex downward-closed set is convex. We shall use this notion when $\mathcal{F}$ is a closed subset of $\mathbf{P}^{\triangle}_{1\,wk}(X)$, in which case $\mathcal{F}$ will always be downward-closed. By an argument similar to that of Proposition 1 [6, Appendix D]:

**Proposition 6.** *Let $X$ be stably compact, and $F \in \bigtriangleup \mathbf{P}_1(X)$. Then $CPeau_1(F)$ is co-strongly convex.*

The key argument for the converse is the following proposition, which states how $Bary$ behaves w.r.t. the dualizing operation $\_^\perp$. For any continuous map $f : Y \to Z$, and

every $p \in \mathbf{V}_{1\,wk}(Y)$, the *image*, a.k.a. *push-forward* continuous valuation $f[p] \in \mathbf{V}_{1\,wk}(Z)$ is that which sends each $V \in \mathcal{O}(Z)$ to $p(f^{-1}(V))$. The change-of-variables formula for Choquet integration states that $\oint_{z\in Z} g(z)df[p] = \oint_{y\in Y} g(f(y))dp$ (an easy consequence of the definition). We use the notation $\_^{\perp}[\mathcal{P}']$ below (with $\mathcal{P}' = \mathcal{P}^{\perp}$), where $\_^{\perp} : \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}} \to \mathbf{P}^{\triangle}_{1\,wk}(X^{\mathsf{d}})$.

**Proposition 7.** *Let $X$ be stably compact. For any continuous probability valuation $\mathcal{P}$ on $\mathbf{P}^{\triangle}_{1\,wk}(X)$, $(Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp} = Bary(\mathcal{P})$.*

*Proof.* Using the change-of-variables formula, $Bary(\_^{\perp}[\mathcal{P}^{\perp}]) = \lambda g \in \langle X^{\mathsf{d}} \to \mathbb{R}^+\rangle \cdot \oint_{G'\in \mathbf{P}^{\triangle}_{1\,wk}(X^{\mathsf{d}})} G'(g)d\_^{\perp}[\mathcal{P}^{\perp}] = \lambda g \in \langle X^{\mathsf{d}} \to \mathbb{R}^+\rangle \cdot \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}} G^{\perp}(g)d\mathcal{P}^{\perp} = \lambda g \in \langle X^{\mathsf{d}} \to \mathbb{R}^+\rangle \cdot -\inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot -G^{\perp}(g)}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P}$ (using the definition of $\mathcal{P}^{\perp}$ as $\alpha_{\mathcal{C}}(\mathcal{P}^{\perp}) = \alpha_{\mathcal{C}}(\mathcal{P})^{\perp}$, and Lemma 1).

For each $g \in \langle X^{\mathsf{d}} \to \mathbb{R}\rangle$, and each constant $a \geq -\inf_{x\in X} g(x)$, then, $\widehat{Bary(\_^{\perp}[\mathcal{P}^{\perp}])}(g) = -\inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot -G^{\perp}(g+a)}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P} - a$. So, for all $h \in \langle X \to \mathbb{R}^+\rangle$, with $a \geq \sup_{x\in X} h(x)$, we obtain:

$$(Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp}(h) = \sup_{\substack{g\in\langle X^{\mathsf{d}}\to\mathbb{R}\rangle \\ g\geq -h}} \inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot -G^{\perp}(g+a)}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P} + a \quad (1)$$

using Lemma 1. For every open $U$ in $X$, we claim that $(Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp}(\chi_U) \leq Bary(\mathcal{P})(\chi_U)$. For every $g \in \langle X^{\mathsf{d}} \to \mathbb{R}\rangle$ with $g \geq -\chi_U$, we have $\chi_U \geq -g$, so (since $G^{\perp}$ is normalized) $G^{\perp}(g+a) = G^{\perp}(g) + a = \sup_{f\geq -g} -G(f) + a \geq -G(\chi_U) + a$. For every $\varphi \in \langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}\rangle$ with $\varphi \geq \lambda G \in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}} \cdot G(\chi_U) - a$, therefore, $\varphi \geq \lambda G \in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}} \cdot -G^{\perp}(g+a)$. So $\inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot G(\chi_U)-a}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P} \geq \inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot -G^{\perp}(g+a)}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P}$. The left-hand side is exactly $Bary(\mathcal{P})(\chi_U) - a$: take $\varphi = \lambda G \in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}} \cdot G(\chi_U) - a$, and use the fact that $Bary(\mathcal{P})$ is normalized. (The key point is that this $\varphi$ is indeed in $\langle \mathbf{P}^{\triangle}_{1\,wk}(X) \to \mathbb{R}\rangle$, in particular continuous, as the inverse image of $(t, +\infty)$ is the open $[\chi_U > a + t]$.) So, using (1), $(Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp}(\chi_U) \leq \sup_{\substack{g\in\langle X^{\mathsf{d}}\to\mathbb{R}\rangle \\ g\geq -\chi_U}} Bary(\mathcal{P})(\chi_U) - a + a = Bary(\mathcal{P})(\chi_U)$.

Conversely, let us show that, for any two opens $U$ and $V$ of $X$ with $V \Subset U$, $Bary(\mathcal{P})(\chi_V) \leq (Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp}(\chi_U)$. The relation $\Subset$ is the way-below relation on $\mathcal{O}(X)$, ordered by inclusion; on every locally compact space, $U \Subset V$ iff $U \subseteq Q \subseteq V$ for some compact saturated set $Q$, and $\mathcal{O}(X)$ is then a continuous cpo [3]. Let $Q$ be a compact saturated subset such that $V \subseteq Q \subseteq U$. Then $g = -\chi_Q$ satisfies $g \geq -\chi_U$, so $(Bary(\_^{\perp}[\mathcal{P}^{\perp}]))^{\perp}(\chi_U) \geq \inf_{\substack{\varphi\in\langle \mathbf{P}^{\triangle}_{1\,wk}(X)\to\mathbb{R}\rangle \\ \varphi\geq \lambda G\in \mathbf{P}^{\triangle}_{1\,wk}(X)^{\mathsf{d}}\cdot -G^{\perp}(a-\chi_Q)}} \oint_{G\in \mathbf{P}^{\triangle}_{1\,wk}(X)} \varphi(G)d\mathcal{P} + a$.

Let us estimate $-G^\perp(a-\chi_Q) = \inf_{\substack{f\in\langle X\to\mathbb{R}^+\rangle \\ f\geq -(a-\chi_Q)}} G(f)$ (using Lemma 1). Whenever $f \geq -(a-\chi_Q)$, we have $f \geq \chi_Q - a \geq \chi_V - a$, so $-G^\perp(a-\chi_Q) \geq \inf_{\substack{f\in\langle X\to\mathbb{R}^+\rangle \\ f\geq \chi_V - a}} G(f) \geq G(\chi_V - a) = G(\chi_V) - a$, since $G$ is normalized. Using (1), $(Bary(\_^\perp[\mathcal{P}^\perp]))^\perp(\chi_U) \geq \inf_{\varphi\in\langle \mathbf{P}^\triangle_{1\,wk}(X)\to\mathbb{R}\rangle} \oint_{G\in\mathbf{P}^\triangle_{1\,wk}(X)} \varphi(G)d\mathcal{P} + a = \inf_{\varphi\geq\lambda G\in\mathbf{P}^\triangle_{1\,wk}(X)^{\mathsf{d}}\cdot G(\chi_V)-a} \oint_{G\in\mathbf{P}^\triangle_{1\,wk}(X)} G(\chi_V)d\mathcal{P} - a + a = Bary(\mathcal{P})(\chi_V)$.

Let $p = \gamma_{\mathcal{C}}(Bary(\mathcal{P}))$, $p' = \gamma_{\mathcal{C}}((Bary(\_^\perp[\mathcal{P}^\perp]))^\perp)$. We have shown that $p'(U) \leq p(U)$ for every open $U$ of $X$, and that $p'(U) \geq p(V)$ whenever $V \Subset U$. Since $\mathcal{O}(X)$ is a continuous cpo, and $p$ is continuous, $p(U) = \sup_{V\Subset U} p(V) \leq p'(U)$. So $p = p'$. Since $\alpha_{\mathcal{C}}, \gamma_{\mathcal{C}}$ form an isomorphism, $Bary(\mathcal{P}) = (Bary(\_^\perp[\mathcal{P}^\perp]))^\perp$. □

Let $Conv^*(\mathcal{F})$, the *co-strong convex closure* of $\mathcal{F}$, be $\{Bary(\mathcal{P}) \mid \mathcal{P}$ co-supported on $\mathcal{F}\}$, and $\downarrow$ be the downward-closure operator. Similarly to Section 3, we show that $CPeau_1(\bigsqcup\mathcal{F}) = \downarrow Conv^*(\mathcal{F})$ for every $\mathcal{F} \in \mathcal{H}(\mathbf{P}^\triangle_{1\,wk}(X))$. It is easy to see that (on stably compact $X$), for any normalized continuous upper prevision $F$ on $X$, $CPeau_1(F)^\perp = CCoeur_1(F^\perp)$; this is because $\_^\perp$ is antitone (Theorem 4 (6)). For every non-empty closed subset $\mathcal{F}$ of $\mathbf{P}^\triangle_{1\,wk}(X)$, $CPeau_1(\bigsqcup\mathcal{F})^\perp = CCoeur_1((\bigsqcup\mathcal{F})^\perp) = CCoeur_1(\bigsqcap\mathcal{F}^\perp)$ (because $\_^\perp$ is antitone) $= \uparrow Conv(\mathcal{F}^\perp)$ (see Section 3). Apply $\_^\perp$, using Theorem 4 (5) to get $CPeau_1(\bigsqcup\mathcal{F}) = \uparrow Conv(\mathcal{F}^\perp)^\perp = \downarrow Conv(\mathcal{F}^\perp)^\perp = \downarrow Conv^*(\mathcal{F})$. The latter equality is a straightforward exercise, using Proposition 7, and the easily proved facts that, for any continuous function $f : Y \to Z$, for any continuous probability valuation $p$ on $Y$: (a) if $p$ is supported on some subset $A$ of $X$, then $f[p]$ is supported on the direct image $f(A)$; (b) if $f[p]$ is co-supported on some closed subset $F$ of $Y$, then $p$ is co-supported on $f^{-1}(F)$ [6, Appendix E].

**Theorem 5 (Isomorphism).** *Let $X$ be stably compact. Then $CPeau_1$ and $\bigsqcup$ define an isomorphism between $\bigwedge\mathbf{P}_1(X)$ and the space $\mathcal{H}^{Cvx^*}(\mathbf{P}^\triangle_{1\,wk}(X))$ of co-strongly convex non-empty closed subsets of $\mathbf{P}^\triangle_{1\,wk}(X)$, ordered by $\subseteq$.*

The case where $X$ is a cpo is much simpler than for the demonic case (Section 4).

**Lemma 2.** *Let $X$ be a continuous pointed cpo. Every convex closed subset of $\mathbf{P}^\triangle_1(X)$ is co-strongly convex.*

*Proof.* Let $Z = \mathbf{P}^\triangle_1(X)$, $\mathcal{F}$ a convex closed subset of $Z$, and $\mathcal{P}$ a continuous probability valuation on $Z$, co-supported on $\mathcal{F}$: $\mathcal{P}(Z \setminus \mathcal{F}) = 0$. Since $Z \cong \mathbf{V}_1(X)$ is a continuous pointed cpo, $\mathbf{V}_1(Z)$ is one, too, with a basis of simple probability valuations [2]. So write $\mathcal{P}$ as the sup of a directed family $(\mathcal{P}_i)_{i\in I}$, with $\mathcal{P}_i \leq \mathcal{P}$. In particular, $\mathcal{P}_i(Z \setminus \mathcal{F}) = 0$, so $\mathcal{P}_i$ is co-supported on $\mathcal{F}$. Write $\mathcal{P}_i$ as $\sum_{j=1}^n a_j\delta_{G_j}$, where each $G_j$ is in $\mathcal{F}$, and $a_1 + \ldots + a_n = 1$. $\mathcal{F}$ is convex so $Bary(\mathcal{P}_i) = \sum_{j=1}^n a_jG_j$ is in $\mathcal{F}$. Now $Bary$ is continuous, so $Bary(\mathcal{P}) = \sup_{i\in I} Bary(\mathcal{P}_i)$. As $\mathcal{F}$ is Scott-closed, $Bary(\mathcal{P}) \in \mathcal{F}$. □

Writing $\mathcal{H}^{cvx}(Y)$ the subset of $\mathcal{H}(Y)$ consisting of convex subsets, it follows:

**Corollary 2 (Isomorphism).** *Let $X$ be a continuous, coherent pointed cpo. $CPeau_1$ and $\bigsqcup$ define an isomorphism between $\bigtriangleup \mathbf{P}_1(X)$ and $\mathcal{H}^{cvx}(\mathbf{P}^{\triangle}_{1\,wk}(X)) = \mathcal{H}^{cvx}(\mathbf{P}^{\triangle}_{1}(X)) \cong \mathcal{H}^{cvx}(\mathbf{V}_1(X))$.*

One may also show the following [6, Appendix F]. This depends crucially on the fact that $CPeau_1(\bigsqcup \mathcal{F}) = \downarrow Conv^*(\mathcal{F})$.

**Proposition 8.** *Let $X$ be a continuous, coherent, pointed cpo. Then $CPeau_1$ is a continuous map from $\bigtriangleup \mathbf{P}_1(X)$ to $\mathcal{H}(\mathbf{P}^{\triangle}_{1\,wk}(X))$.*

Recall that, in this case, $Y = \mathbf{P}^{\triangle}_{1\,wk}(X) = \mathbf{P}^{\triangle}_{1}(X)$ has a basis of simple normalized linear previsions. For any continuous cpo $Y$, $\mathcal{H}(Y)$ is continuous cpo too, with basis given by the *finitary closed subsets* $\downarrow \mathcal{E}$, $\mathcal{E}$ a finite subset of $Y$ [1]. As for Theorem 3, we can therefore conclude:

**Theorem 6.** *For any continuous, coherent pointed cpo $X$, $\bigtriangleup \mathbf{P}_1(X)$ is a continuous, coherent pointed cpo. A basis is given by previsions of the form $\lambda h \in \langle X \to \mathbb{R}^+ \rangle \cdot \max_{i=1}^{m} \sum_{j=1}^{n} a_{ij} h(x_j)$, where $a_{ij} \in \mathbb{R}^+$ and $\sum_{j=1}^{n} a_{ij} = 1$ for each $i$.*

So we can also approximate from below any continuous normalized upper prevision by one that is computable, using only finitely $\max$, $+$, and $\cdot$ operations.

## 6 Chaotic Non-determinism + Probabilistic Choice

In chaotic non-determinism we replace $\mathcal{Q}(Y)$ or $\mathcal{H}(Y)$ by the *Plotkin powerdomain* $\mathcal{P}\ell(Y)$. This is the set of all *lenses* $L$, which are non-empty intersections of a compact saturated subset $Q$ of $Y$ and a closed subset $F$ of $Y$. A canonical way of writing $L$ as $Q \cap F$ is then to take $Q = \uparrow L$, $F = cl(L)$. We order $\mathcal{P}\ell(Y)$ by the topological Egli-Milner ordering $\sqsubseteq_{\mathrm{EM}}$ defined by $L \sqsubseteq_{\mathrm{EM}} L'$ iff $\uparrow L \supseteq \uparrow L'$ and $cl(L) \subseteq cl(L')$. If $Y$ is a continuous, coherent pointed cpo, then $\mathcal{P}\ell(Y)$ is one, too. (See [1, Section 6.2.3].) Among all lenses, call *strong* those that obey the stronger property $F = \downarrow L$.

Let $\mathbf{F}_1(X)$ be the space of all normalized forks on $X$. Then [5, Proposition 6] there is a map $CCorps_1 : \mathbf{F}_1(X) \to \mathcal{P}\ell(\mathbf{P}^{\triangle}_{1\,wk}(X))$, defined by $CCorps_1(F^-, F^+) = CCoeur_1(F^-) \cap CPeau_1(F^+)$. Moreover, $CCoeur_1(F^-) = \uparrow CCorps_1(F)$ and $CPeau_1(F^+) = \downarrow CCorps_1(F)$. (So $CCorps_1(F^-, F^+)$ is a strong lens.) It is clear from our results on $CCoeur_1$ and $CPeau_1$ that $\bigsqcap\bigsqcup \circ CCorps_1 = \mathrm{id}$; the map $\bigsqcap\bigsqcup : \mathcal{P}\ell(\mathbf{P}^{\triangle}_{1\,wk}(X)) \to \mathbf{F}_1(X)$ is defined by $\bigsqcap\bigsqcup \mathcal{L} = (\bigsqcap \mathcal{Q}, \bigsqcup \mathcal{C})$, where $\mathcal{Q} = \uparrow \mathcal{L}$ and $\mathcal{C} = cl(\mathcal{L})$. The following is an immediate consequence of the results of previous sections:

**Proposition 9.** *A subset $\mathcal{A}$ of $\mathbf{P}^{\triangle}_{1\,wk}(X)$ is* bi-strongly convex *iff for every continuous probability valuation $\mathcal{P}$ supported on $\mathcal{A}$, $Bary(\mathcal{P})$ is in $\uparrow \mathcal{A}$, and for every continuous probability valuation $\mathcal{P}$ co-supported on $\mathcal{A}$, $Bary(\mathcal{P})$ is in $\downarrow \mathcal{A}$.*

*Let $X$ be stably compact. For every lens $\mathcal{L}$ on $\mathbf{P}^{\triangle}_{1\,wk}(X)$, $CCorps_1(\bigsqcap\bigsqcup \mathcal{L}) = \uparrow Conv(\mathcal{L}) \cap \downarrow Conv^*(\mathcal{L})$. $CCorps_1$ and $\bigsqcap\bigsqcup$ define an isomorphism between $\mathbf{F}_1(X)$ and the space $\mathcal{P}L^{biCvx}(\mathbf{P}^{\triangle}_{1\,wk}(X))$ of all strong bi-strongly convex lenses on $\mathbf{P}^{\triangle}_{1\,wk}(X)$.*

Let now $X$ be a continuous, coherent pointed cpo. It is an easy consequence of Theorem 2 and Lemma 2 that any strong lens $\mathfrak{L}$ on $\mathbf{P}^{\triangle}_{1\,wk}(X)$ is bi-strongly convex. Also, every lens is in fact strong [1, Lemma 6.2.20]. Write $\mathcal{P}\ell^{cvx}(Y)$ the subspace of convex lenses in $\mathcal{P}\ell(X)$:

**Theorem 7.** *Let $X$ be a continuous, coherent pointed cpo. $CCorps_1$ and $\bigsqcap\bigsqcup$ define an isomorphism between $\mathbf{F}_1(X)$ and $\mathcal{P}\ell^{cvx}(\mathbf{P}^{\triangle}_{1\,wk}(X)) = \mathcal{P}\ell^{cvx}(\mathbf{P}^{\triangle}_{1}(X)) \cong \mathcal{P}\ell^{cvx}(\mathbf{V}_1(X))$.*

## 7 Conclusion

We have solved the problem of relating domains of continuous previsions and forks à la [5], and convex powercones à la [17]: in standard cases, they are isomorphic. This question was raised under this form at the end of [5], while Keimel and Plotkin [10] show similar results for (unbounded) valuations instead of normalized valuations. They justify working on unbounded valuations because "the mathematics seems to be more natural if we take all the valuations, since one can then work with notions of linearity rather than convexity". I hope to have convinced the reader that the mathematics of the normalized case, once generalized to the topological case, is both beautiful and deep. Note in particular that the notion of (generalized) barycenters $Bary(\mathcal{P})$, and above all the convex-concave duality work naturally at the topological level, not directly on cpos.

**Acknowledgements.** Thanks to the anonymous referees, and to Roberto Segala for their careful rereading.

## References

1. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, OUP, vol. 3, pp. 1–168 (1994)
2. Edalat, A.: Domain theory and integration. Theor. Comp. Sci. 151, 163–193 (1995)
3. Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M., Scott, D.S.: Continuous lattices and domains. In: Encycl. Mathematics and its Applications, CUP, vol. 93 (2003)
4. Goubault-Larrecq, J.: Continuous capacities on continuous state spaces. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 764–776. Springer, Heidelberg (2007)
5. Goubault-Larrecq, J.: Continuous previsions. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 542–557. Springer, Heidelberg (2007)
6. Goubault-Larrecq, J.: Prevision domains and convex powercones. Research Report LSV-07-33, Laboratoire Spécification et Vérification, ENS Cachan, France, 34 pages (October 2007)
7. Goubault-Larrecq, J.: Une introduction aux capacités, aux jeux et aux prévisions (June 2007), http://www.lsv.ens-cachan.fr/~goubault/ProNobis/pp_1_8.pdf
8. Jung, A.: Cartesian Closed Categories of Domains. PhD thesis, T.H. Darmstadt (1998)
9. Jung, A.: Stably compact spaces and the probabilistic powerspace construction. In: Desharnais, J., Panangaden, P. (eds.) CADE 1980. Electronic Notes in Theoretical Computer Science, vol. 87, Elsevier, Amsterdam (2004)
10. Keimel, K., Plotkin, G.: Predicate transformers for convex powerdomains. Mathematical Structures in Computer Science, pages 42 (submitted, 2007)

11. Kirch, O.: Bereiche und Bewertungen. Diplom, T.H. Darmstadt (1993)
12. Mislove, M.: Topology, domain theory and theoretical computer science. Topology and Its Applications 89, 3–59 (1998)
13. Mislove, M.: Nondeterminism and probabilistic choice: Obeying the law. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 350–364. Springer, Heidelberg (2000)
14. Roth, W.: Hahn-Banach type theorems for locally convex cones. J. Australian Math. Soc. 68(1), 104–125 (2000)
15. Tix, R.: Stetige Bewertungen auf topologischen Räumen. Diplom, T.H. Darmstadt (June 1995)
16. Tix, R.: Continuous D-Cones: Convexity and Powerdomain Constructions. PhD thesis, T.U. Darmstadt (1999)
17. Tix, R., Keimel, K., Plotkin, G.: Semantic domains for combining probability and non-determinism. Electronic Notes in Theor. Comp. Sci. 129, 1–104 (2005)

# RPO, Second-Order Contexts, and λ-Calculus*

Pietro Di Gianantonio, Furio Honsell, and Marina Lenisa

Dipartimento di Matematica e Informatica, Università di Udine
via delle Scienze 206, 33100 Udine, Italy
{digianantonio,honsell,lenisa}@dimi.uniud.it

**Abstract.** We apply Leifer-Milner *RPO approach* to the λ-calculus, endowed with *lazy* and *call by value reduction strategies*. We show that, contrary to process calculi, one can deal directly with the λ-calculus syntax and apply Leifer-Milner technique to a category of contexts, provided that we work in the framework of *weak bisimilarities*. However, even in the case of the transition system with minimal contexts, the resulting bisimilarity is *infinitely branching*, due to the fact that, in standard context categories, parametric rules such as $\beta$ can be represented only by infinitely many ground rules. To overcome this problem, we introduce the general notion of *second-order context category*. We show that, by carrying out the RPO construction in this setting, the *lazy (call by value) observational equivalence* can be captured as a *weak bisimilarity equivalence* on a *finitely branching* transition system. This result is achieved by considering an encoding of λ-calculus in Combinatory Logic.

## 1 Introduction

Recently, much attention has been devoted to derive *labelled transition systems* and *bisimilarity congruences* from *reactive systems*, in the context of process languages and graph rewriting, [Sew02, LM00, SS03, GM05, BGK06, BKM06, EK06]. In the theory of process algebras, the operational semantics of CCS was originally given via a labelled transition system (lts), while more recent process calculi have been presented via reactive systems plus structural rules. Reactive systems naturally induce behavioral equivalences which are congruences w.r.t. contexts, while lts's naturally induce bisimilarity equivalences with coinductive characterizations. However, such equivalences are not congruences in general, or else it is an heavy, ad-hoc task to prove that they are congruences.

Generalizing [Sew02], Leifer and Milner [LM00] presented a general categorical method for deriving a transition system from a reactive system, in such a way that the induced bisimilarity is a congruence. The labels in Leifer-Milner's transition system are those contexts which are *minimal* for a given reaction to fire. Minimal contexts are identified via the categorical notion of *relative pushout (RPO)*. Leifer-Milner's central result guaranties that, under a suitable categorical condition, the induced bisimilarity is a *congruence* w.r.t. all contexts.

* Work supported by ART PRIN Project prot. 2005015824 (funded by MIUR).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 334–349, 2008.

In the literature, some case studies have been carried out in the setting of process calculi, for testing the expressivity of Leifer-Milner's approach. Some difficulties have arisen in applying the approach directly to such languages, viewed as Lawvere theories, because of structural rules. Thus more complex categorical constructions have been introduced in [Lei01], and by Sassone and Sobocinski in [SS03,SS05]. Moreover, often intermediate encodings have been considered, in graph theory, for which the approach of "borrowed contexts" has been developed [EK06], and in Milner's bigraph theory [Mil07].

In this paper, we focus on the prototypical example of reactive system given by the λ-calculus, endowed with lazy and call by value (cbv) reduction strategies. We show that, in principle, contrary to process calculi, one could deal directly with the λ-calculus syntax and apply Leifer-Milner technique to the category of term contexts induced by the λ-terms, provided that we work in the setting of *weak bisimilarities*. However, even in the case of the transition system with minimal contexts, the lts and the induced bisimilarity turn out to be infinitely branching. This is mainly due to the fact that, in the category of contexts, the β-rule cannot be described parametrically, but it needs to be described extensionally using an infinite set of pairs of ground terms. In order to overcome this problem, we consider the combinatory logic and we introduce the general notion of *category of second-order term contexts*. Our main result amounts to the fact that, by carrying out Leifer-Milner's construction in this setting, the *lazy (cbv) contextual equivalence* can be captured as a *weak bisimilarity equivalence* on a (finitely branching) transition system. Technically, this result is achieved by considering an encoding of the lazy (cbv) λ-calculus in KS Combinatory Logic (CL), endowed with a lazy (cbv) reduction strategy, and by showing that the lazy (cbv) contextual equivalence on λ-calculus can be recovered as a lazy (cbv) equivalence on CL. It is necessary to consider such encoding, since the approach of second-order context categories proposed in this paper works for reaction rules which are "local", that is the reaction does not act on the whole term, but only locally. But the substitution operation on λ-calculus is not local.

Finally, the second-order approach carried out in this paper for the λ-calculus suggests a new general technique for dealing with any calculus with parametric rules, alternative to the one of luxes in [KSS05]. Moreover, the correspondence results obtained in this paper about the observational equivalences on λ-calculus and CL are interesting *per se* and, although natural and ultimately elementary, had not appeared previously in the literature.

**Summary.** In Section 2, we summarize the theory of reactive systems of [LM00]. In Section 3, we present the λ-calculus together with lazy and cbv reduction strategies and observational equivalences, and we discuss the RPO approach applied to the λ-calculus endowed with a structure of context category. In Section 4, we focus on Combinatory Logic (CL), we show how to recover on CL the lazy and cbv strategies and observational equivalences, and we discuss the RPO approach applied to CL, viewed as a context category. In Section 5, we introduce the notion of second-order context category, and we apply the RPO approach to CL viewed as a second-order rewriting system, thus obtaining

characterizations of lazy and cbv observational equivalences as weak bisimilarities on finitely branching lts's. Final remarks and directions for future work appear in Section 6. For lack of space, proofs are omitted in this paper, however they are available in [DHL08].

## 2 The Theory of Reactive Systems

In this section, we summarize the theory of reactive systems proposed in [LM00] to derive lts's and bisimulation congruences from a given reduction semantics. Moreover, we discuss weak variants of Leifer-Milner's bisimilarity equivalence.

The theory of [LM00] is based on a categorical formulation of the notion of *reactive system*, whereby *contexts* are modeled as arrows of a category, *terms* are arrows having as domain 0 (a special object which denotes no holes), and reaction rules are pairs of terms.

**Definition 1 (Reactive System).** *A* reactive system $\mathbf{C}$ *consists of:*
– *a category* $\mathcal{C}$*;*
– *a distinguished object* $0 \in |\mathcal{C}|$*;*
– *a composition-reflecting subcategory* $\mathcal{D}$ *of* reactive contexts*;*
– *a set of pairs* $\mathbf{R} \subseteq \bigcup_{I \in |\mathcal{C}|} \mathcal{C}[0, I] \times \mathcal{C}[0, I]$ *of* reaction rules.

Reactive systems on term languages can be viewed as a special case of reactive systems in the sense of Leifer-Milner by instantiating $\mathcal{C}$ as a suitable category of term and contexts, also called the (free) Lawvere category, [LM00].

Given a reaction system with reactive contexts $\mathcal{D}$ and reaction rules $\mathbf{R}$, the *reaction relation* $\rightarrow$ is defined by: $t \rightarrow u$ iff $t = dl,\ u = dr$ for some $d \in \mathcal{D}$ and $\langle l, r \rangle \in \mathbf{R}$.

The behavior of a reactive system is expressed as an unlabelled transition system. On the other hand, many useful behavioral equivalences are only defined for lts's. The passage from reactive systems to lts's is obtained as follows.

**Definition 2 (Context Label Transition System).** *Given a reactive system* $\mathbf{C}$*, the associated context lts is defined as follows:*
– *states: arrows* $t : 0 \rightarrow I$ *in* $\mathcal{C}$*, for any* $I$*;*
– *transitions:* $t \xrightarrow{c}_C u$ *iff* $c \in \mathcal{C}$ *and* $ct \rightarrow u$*.*

In the case of a reactive system defined on a category of contexts, a state is a term $t$, and an associated label is a context $c$ such that $ct$ reduces. In the following, we will consider also lts's obtained by reducing the set of transitions of the context lts. In the sequel, we will use the word lts to refer to any such lts obtained from a context lts. In the standard way, any lts induces a bisimilarity relation. In [LM00], the authors proposed a categorical criterion for identifying the "smallest context allowing a reaction".

**Definition 3 (RPO/IPO)**
*i) Let* $\mathcal{C}$ *be a category and let us consider the commutative diagram in Fig. 1(i). Any tuple* $\langle I_5, e, f, g \rangle$ *which makes diagram in Fig. 1(ii) commute is called a*

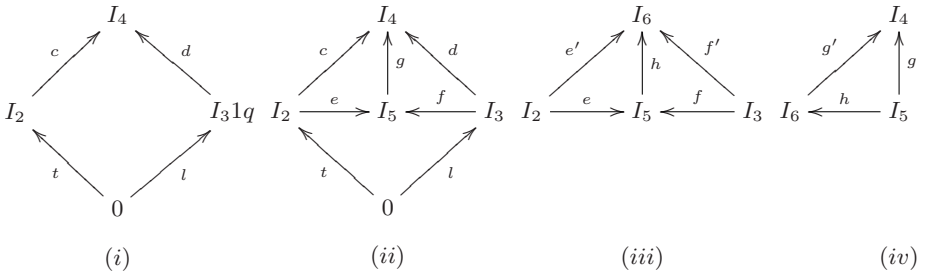



**Fig. 1.** Redex Square and Relative Pushout

candidate *for (i). A* relative pushout *is the smallest such candidate, i.e. it satisfies the universal property that given any other candidate* $\langle I, e', f', g' \rangle$*, there exists a* unique *mediating morphism* $h : I_5 \to I_6$ *such that both diagrams in Fig. 1(iii) and Fig. 1(iv) commute.*
*ii) A commuting square such as diagram in Fig 1(i) is an* idem pushout *if* $\langle I_4, c, d, id_{I_4} \rangle$ *is its RPO.*

**Definition 4 (IPO Transition System)**

- *States: arrows* $t : 0 \to I$ *in* $\mathcal{C}$*, for any* $I$*;*
- *transitions:* $t \xrightarrow{c}_I dr$ *iff* $d \in \mathcal{D}$*,* $\langle l, r \rangle \in \mathbf{R}$ *and the diagram in Fig. 1(i) is an IPO.*

That is, if inserting $t$ into the context $c$ matches $dl$, and $c$ is the "smallest" such context (IPO condition), then $t$ transforms to $dr$ with label $c$, where $r$ is the reduct of $l$. Let $\sim_I$ denote the bisimilarity induced by the IPO lts.

**Definition 5 (Redex Square).** *Let* $\mathbf{C}$ *be a reactive system and* $t : 0 \to I_2$ *an arrow in* $\mathcal{C}$*. A* redex square *(see Fig. 1(i)) consists of a left-hand side* $l : 0 \to I_3$ *of a reaction rule* $\langle l : 0 \to I_3, r : 0 \to I_3 \rangle \in \mathbf{R}$*, a context* $c : I_2 \to I_4$ *and a reactive context* $d : I_3 \to I_4$ *such that* $ct = dl$*.*

*A reactive system is said to* have redex RPOs *if every redex square has an RPO.*

The following is Leifer-Milner's central result:

**Theorem 1 ([LM00]).** *Let* $\mathbf{C}$ *be a reactive system having redex RPOs. Then the IPO bisimilarity* $\sim_I$ *is a congruence w.r.t. all contexts, i.e. if* $a \sim_I b$ *then for all* $c$ *of the appropriate type,* $ca \sim_I cb$*.*

### 2.1 Weak Bisimilarity

For dealing with the λ-calculus, it will be useful to consider the weak versions of the context and IPO lts's defined above, together with the corresponding notions of *weak bisimilarities.*

One can proceed in general, by defining a weak lts from a given lts:

**Definition 6 (Weak lts and Bisimilarity).** *Let $\xrightarrow{\alpha}$ be a lts, and let $\tau$ be a label (identifying an unobservable action).*

*i) We define the* weak lts $\stackrel{\alpha}{\Longrightarrow}$ *by*

$$t \stackrel{\alpha}{\Longrightarrow} u \quad \textit{iff} \quad \begin{cases} t \xrightarrow{\tau}{}^{*} u & \textit{if } \alpha = \tau \\ t \xrightarrow{\tau}{}^{*} t' \xrightarrow{\alpha} u' \xrightarrow{\tau}{}^{*} u & \textit{otherwise .} \end{cases}$$

*where $\xrightarrow{\tau}{}^{*}$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$.*
*ii) Let us call* weak bisimilarity *the bisimilarity induced by the weak lts.*

The above definition differs from the one proposed in [LM00]. We cannot use that in [LM00], since it discriminates $\lambda$-terms which are equivalent in the usual semantics. The following easy lemma gives a useful characterization of the weak bisimilarity, whereby any $\xrightarrow{\alpha}$-transition is mimicked by a $\stackrel{\alpha}{\Longrightarrow}$-transition:

**Lemma 1.** *Let $\xrightarrow{\alpha}$ be a lts and let $\stackrel{\alpha}{\Longrightarrow}$ be the corresponding weak lts. The induced weak bisimilarity is the greatest symmetric relation $\mathcal{R}$ s.t.:*

$$\langle a, b\rangle \in \mathcal{R} \ \wedge \ a \xrightarrow{f} a' \implies \exists b'.\ b \stackrel{f}{\Longrightarrow} b' \ \wedge \ \langle a', b'\rangle \in \mathcal{R} \ .$$

For dealing with the $\lambda$-calculus, we will consider a notion of *weak IPO bisimilarity*, where the identity context is unobservable. Such notions of weak IPO bisimilarities are not congruences w.r.t. all contexts, in general, however, as observed in [LM00] (end of Section 5), they are congruences at least w.r.t. reactive contexts:

**Theorem 2.** *Let* **C** *be a reactive system having redex RPOs. Then the weak IPO bisimilarity $\approx_I$, where the identity context is unobservable, is a congruence w.r.t. reactive contexts.*

## 3 The Lambda Calculus

First, we recall the $\lambda$-calculus syntax together with *lazy* and *cbv* reduction strategies and observational equivalences. Then, we show how to apply the RPO technique to $\lambda$-calculus, viewed as a context category, and we discuss some problematic issues.

The set of *$\lambda$-terms* $\Lambda$ is defined by $(\Lambda \ni)\ M ::= \ x \mid MM \mid \lambda x.M$ , where $x \in \mathit{Var}$ is an infinite set of variables. Let $FV(M)$ denote the set of free variables in $M$, and let us denote by $\Lambda^0$ the set of *closed* $\lambda$-terms.

As usual, $\lambda$-terms are taken up-to $\alpha$-conversion, and application associates to the left. We consider the standard notions of $\beta$-rule $(\lambda x.M)N \to_\beta M[N/x]$ and $\beta_V$-rule $(\lambda x.M)N \to_{\beta_V} M[N/x]$. We denote by $=_\beta$ and $=_{\beta_V}$ the corresponding conversions.

A *reduction strategy* on the $\lambda$-calculus determines, for each term which is not a value, a suitable $\beta$-redex appearing in it to be contracted. The lazy and cbv reduction strategies are defined on closed $\lambda$-terms as follows:

**Definition 7 (Reduction Strategies)**

*i) The* lazy strategy $\rightarrow_l \subseteq \Lambda^0 \times \Lambda^0$ *reduces the leftmost $\beta$-redex, not appearing within a $\lambda$-abstraction. Formally, $\rightarrow_l$ is defined by the axiom:*

$$\frac{}{(\lambda x.M)N \rightarrow_l M[N/x]} \qquad \frac{N \rightarrow_l N'}{NP \rightarrow_l N'P}$$

*ii) The* call by value strategy $\rightarrow_v \subseteq \Lambda^0 \times \Lambda^0$ *reduces the leftmost $\beta_V$-redex, not appearing within a $\lambda$-abstraction. Formally, $\rightarrow_v$ is defined by the following rules:*

$$\frac{}{(\lambda x.M)V \rightarrow_v M[V/x]} \qquad \frac{N \rightarrow_v N'}{NP \rightarrow_v N'P} \qquad \frac{N \rightarrow_v N'}{(\lambda x.M)N \rightarrow_v (\lambda x.M)N'}$$

*where $V$ is a $\lambda$-abstraction.*

We denote by $\rightarrow_\sigma^*$ the reflexive and transitive closure of a strategy $\rightarrow_\sigma$, for $\sigma \in \{l, v\}$, by $Val_\sigma$ the set of values, i.e. the set of terms on which the reduction strategy halts (which coincides with the set of $\lambda$-abstractions in both cases), and by $M \Downarrow_\sigma$ the fact that there exists $V \in Val_\sigma$ such that $M \rightarrow_\sigma^* V$.

As we will see in Section 3.1 below, each strategy defines a reactive system on $\lambda$-terms in the sense of Definition 1. To this aim, it is useful to notice that the above reduction strategies can be alternatively determined by specifying suitable sets of *reactive contexts*, which are subsets of the following (closed) *unary contexts*, i.e. contexts with a single hole: $C[\,] ::= [\,] \mid PC[\,] \mid C[\,]P$ .

*Remark 1.* i) The lazy strategy $\rightarrow_l$ is the closure of the $\beta$-rule under the reactive contexts, corresponding to the (closed) applicative contexts: $D[\,] ::= [\,] \mid D[\,]P$ , ii) The cbv strategy $\rightarrow_v$ is the closure of the $\beta_V$-rule under the following (closed) reactive contexts: $D[\,] ::= [\,] \mid D[\,]P \mid (\lambda x.M)D[\,]$ .

Each strategy induces an *observational (contextual) equivalence* à la Morris on closed terms, when we consider programs as *black boxes* and only observe their "halting properties".

**Definition 8 ($\sigma$-observational Equivalence).** *Let $\rightarrow_\sigma$ be a reduction strategy and let $M, N \in \Lambda^0$. The* observational equivalence $\approx_\sigma$ *is defined by $M \approx_\sigma N$ iff for any unary context $C[\,]$, $C[M] \Downarrow_\sigma \Leftrightarrow C[N] \Downarrow_\sigma$ .*

The definition of $\approx_\sigma$ can be extended to open terms by considering closing substitutions, i.e. for $M, N \in \Lambda$ s.t. $FV(M, N) \subseteq \{x_1, \ldots, x_n\}$, we define: $M \widehat{\approx}_\sigma N$ iff for all closing substitutions $\boldsymbol{P}$, $M[\boldsymbol{P}/\boldsymbol{x}] \approx_\sigma N[\boldsymbol{P}/\boldsymbol{x}]$.

*Remark 2.* Notice that the definition of unary contexts does not include $\lambda$-abstraction contexts, i.e. contexts where the hole appears under the scope of a $\lambda$-abstraction. Namely, such contexts are not relevant, since $\approx_\sigma$ is defined on closed terms. Moreover, often in the literature, the observational equivalence is defined by considering multi-holed contexts. However, it is easy to see that the two notions of observational equivalences, obtained by considering just unary or all multi-holed contexts, coincide.

The problem of reducing the set of contexts in which we need to check the behavior of two terms has been widely studied in the literature. In particular, for both strategies in Definition 7 above, a *Context Lemma* holds, which allows us to restrict ourselves to *applicative contexts* of the shape $[\,]\boldsymbol{P}$, where $\boldsymbol{P}$ denotes a list of terms. Let us denote by $\approx_\sigma^{app}$ the observational equivalence which checks the behavior of terms only in applicative contexts. This admits a coinductive characterization as follows:

**Definition 9 (Applicative $\sigma$-bisimilarity)**
*i) A relation* $\mathcal{R} \subseteq \Lambda^0 \times \Lambda^0$ *is an* applicative $\sigma$-bisimulation *if the following holds:* $\langle M, N\rangle \in \mathcal{R} \implies (M \Downarrow_\sigma \Leftrightarrow N \Downarrow_\sigma) \wedge \forall P \in \Lambda^0.\ \langle MP, NP\rangle \in \mathcal{R}$.
*ii) The applicative equivalence* $\approx_\sigma^{app}$ *is the largest applicative bisimulation.*

The *C*ontext Lemma, a well-known resut [AO93,EHR92], states $\approx_\sigma = \approx_\sigma^{app}$. By the Context Lemma, the class of contexts in which we have to check the behavior of terms is smaller, however it is still infinite, thus the applicative bisimilarity is infinitely branching. In the following, we will study alternative coinductive characterizations of the observational equivalences, arising from the application of Leifer-Milner technique.

### 3.1 Lambda Calculus as a Reactive System

Both lazy and cbv $\lambda$-calculus can be endowed with a structure of reactive system in the sense of Definition 1, by considering a suitable variant of context category.

**Definition 10 (Lazy, cbv $\lambda$-reactive Systems).** $\mathbf{C}_\sigma^\lambda$, *for* $\sigma \in \{l, v\}$, *consists of*

- *the category whose objects are* $0, 1$, *where the morphisms from 0 to 1 are the closed terms (up-to* $\alpha$*-equivalence), the morphisms from 1 to 1 are the unary contexts (up-to* $\alpha$*-equivalence), and composition is context insertion;*
- *the subcategory of reactive contexts is determined by the reactive contexts for the lazy and cbv strategy, respectively, presented in Remark 7;*
- *the (infinitely many) reaction rules are* $(\lambda x.M)N \rightarrow_{\beta_\sigma} M[N/x]$, *for all* $M, N$.

The above definition is well-posed, in particular the subcategory of reactive contexts is composition-reflecting.

One can easily check that the system $\mathbf{C}_\sigma^\lambda$ admits RPOs; since there are no abstraction contexts, this fact can be proved by repeating the corresponding proof for the category of term contexts, [Sew02].

**Lemma 2.** *The reactive system* $\mathbf{C}_\sigma^\lambda$, *for* $\sigma \in \{l, v\}$, *has redex RPOs.*

The IPO contexts of a closed term for the lazy and cbv reactive systems are summarized in the first two tables of Fig. 2. The applicative IPO contexts appear in the third table. This class of contexts is interesting, since it is sufficient for determining the observational equivalence (see Theorem 3 below).

| **Lazy lts** | | **Cbv lts** | | **Lazy/cbv appl. lts's** | |
|---|---|---|---|---|---|
| term | IPO contexts | term | IPO contexts | term | IPO cont. |
| $\lambda x.M$ | $[\,]P$, $PC[\,]$ | $\lambda x.M$ | $[\,]P$, $RC[\,]$, $(\lambda x.Q)[\,]$ | $\lambda x.M$ | $[\,]P$ |
| $(\lambda x.M)N\boldsymbol{P}$ | $[\,]$, $PC[\,]$ | $(\lambda x.M)N\boldsymbol{P}$ | $[\,]$, $RC[\,]$ | $(\lambda x.M)N\boldsymbol{P}$ | $[\,]$ |

where $R$ is not a cbv value.

**Fig. 2.** IPO contexts for the lazy/cbv lts's and for their applicative restrictions

The strong versions of context and IPO bisimilarities are too fine, since they take into account reaction steps, and tell apart $\beta$-convertible terms. Trivially, $I$ and $II$, where $I = \lambda x.x$, are equivalent neither in the context bisimilarity nor in the IPO bisimilarity, since $I \overset{[\,]}{\not\rightarrow}$, while $II \overset{[\,]}{\rightarrow}$ (both in the lazy and cbv case). On the other hand, one can easily check that the weak context bisimilarity, where the identity context $[\,]$ is unobservable, equates all closed terms.

The main result of this section, whose proof can be found in [DHL08], is the following:

**Theorem 3.** *Both for lazy and cbv strategies, the observational equivalence, the weak IPO bisimilarity (where the identity context is unobservable), and the applicative weak IPO bisimilarity (where only applicative contexts are considered) coincide.*

Theorem 3 above gives us interesting characterizations of lazy and cbv observational equivalences, in terms of lts's where the labels are significantly reduced. However, such lts's (and bisimilarities) are still infinitely branching, e.g. $\lambda x.M \overset{P}{\rightarrow}_I$, for all $P \in \Lambda^0$. This is due to the fact that the context categories underlying the reactive systems $\mathbf{C}_l^\lambda$ and $\mathbf{C}_v^\lambda$ allow only for a ground representation of the $\beta$-rule through infinitely many ground rules. In order to overcome this problem, one should look for alternative categories which allow for a parametric representation of the $\beta$-rule as $(\lambda x.X)Y \rightarrow X[Y/x]$, where $X, Y$ are parameters. To this aim, we introduce the category of *second-order term contexts* (see Section 5 below). However, as we will see, this approach works only if the reaction rules are "local", that is they do not act on the whole term, but only locally. In particular, the operation of substitution on the $\lambda$-calculus is not local and thus it is not describable by a finite set of reaction rules. To avoid this problem, in the following section we consider encodings of the $\lambda$-calculus into Combinatory Logic (CL) endowed with suitable strategies and equivalences, which turn out to correspond to lazy and cbv equivalences.

## 4 Combinatory Logic

In this section, we focus on *Combinatory Logic* [HS86] with Curry's combinators $\mathbf{K}, \mathbf{S}$, and we study its relationships with the $\lambda$-calculus endowed with lazy and cbv reduction strategies. An interesting result that we prove is that we can define

suitable reduction strategies on CL-terms, inducing observational equivalences which correspond to lazy and cbv equivalences on $\lambda$-calculus. As a consequence, we can safely shift our attention from the reactive system of $\lambda$-calculus to the simpler reactive system of CL. In this section, we apply Leifer-Milner construction to CL viewed as a (standard) context category, and we study weak versions of context and IPO bisimilarities. Our main result is that we can recover lazy and cbv observational equivalences as weak IPO equivalences on CL$^*$, a variant of standard CL. Here the approach is first-order, thus the IPO equivalences are still infinitely branching. However, the results in this section are both interesting in themselves, and useful for our subsequent investigation of Section 5, where CL is viewed as a second-order rewriting system, and characterizations of the observational equivalences as finitely branching IPO bisimilarities are given.

**Definition 11 (Combinatory Terms).** *The set of combinatory terms is defined by:* $(CL \ni)\ M ::= x \mid \mathbf{K} \mid \mathbf{S} \mid MM$ *, where* $\mathbf{K}$, $\mathbf{S}$ *are combinators. The set of combinatory terms is endowed with the following reaction rules:*

$$\mathbf{K}MN \to M \qquad \mathbf{S}MNP \to (MP)(NP)$$

*Let* $CL^0$ *denote the set of* closed *CL-terms.*

### 4.1 Correspondence with the $\lambda$-Calculus

Let $\Lambda(\mathbf{K},\mathbf{S})$ denote the set of $\lambda$-terms built over constants $\mathbf{K},\mathbf{S}$. The following is a well-known encoding, [HS86]:

**Definition 12 ($\lambda$-encoding).** *Let* $\mathcal{T} : \Lambda(\mathbf{K},\mathbf{S}) \to CL$ *be the transformation defined as follows:*

$$\begin{array}{ll} \mathcal{T}(x) = x & \mathcal{T}(C) = C \quad \text{if } C \in \{\mathbf{K},\mathbf{S}\} \\ \mathcal{T}(MN) = \mathcal{T}(M)\mathcal{T}(N) & \mathcal{T}(\lambda x.MN) = \mathbf{S}\mathcal{T}(\lambda x.M)\mathcal{T}(\lambda x.N) \\ \mathcal{T}(\lambda x.x) = \mathbf{SKK} & \mathcal{T}(\lambda x.\lambda y.M) = \mathcal{T}(\lambda x.\mathcal{T}(\lambda y.M)) \\ \mathcal{T}(\lambda x.y) = \mathbf{K}y & \mathcal{T}(\lambda x.C) = \mathbf{K}\mathcal{T}(C) \quad \text{if } C \in \{\mathbf{K},\mathbf{S}\} \end{array}$$

*In particular, if we restrict the domain of* $\mathcal{T}$ *to* $\Lambda$*, we get an encoding of the* $\lambda$*-calculus into CL.*

*Vice versa, there is a natural embedding of CL into the* $\lambda$*-calculus* $\mathcal{E} : CL \to \Lambda$*:*

$$\mathcal{E}(\mathbf{K}) = \lambda xy.x \qquad \mathcal{E}(\mathbf{S}) = \lambda xyz.(xz)(yz) \qquad \mathcal{E}(x) = x \qquad \mathcal{E}(MN) = \mathcal{E}(M)\mathcal{E}(N)$$

**Definition 13 (Lazy/cbv Reduction Strategy on CL)**

*i) The* lazy reduction strategy $\to_l \subseteq CL^0 \times CL^0$ *reduces the leftmost outermost CL-redex. Formally:*

$$\frac{}{\mathbf{S}M_1M_2M_3 \to_l (M_1M_3)(M_2M_3)} \qquad \frac{}{\mathbf{K}M_1M_2 \to_l M_1} \qquad \frac{M \to_l M'}{MP \to_l M'P}$$

*ii) The* cbv strategy $\to_v \subseteq CL^0 \times CL^0$ *is defined by*

$$\frac{}{\mathbf{S}V_1V_2V_3 \to_v (V_1V_3)(V_2V_3)} \qquad \frac{}{\mathbf{K}V_1V_2 \to_v V_1} \qquad \frac{M_1 \to_v M_1'}{\mathbf{K}M_1 \to_v \mathbf{K}M_1'}$$

$$\frac{M_2 \to_v M_2'}{\mathbf{K}V_1M_2 \to_v \mathbf{K}V_1M_2'} \qquad \frac{M_1 \to_v M_1'}{\mathbf{S}M_1 \to_v \mathbf{S}M_1'} \qquad \frac{M_2 \to_v M_2'}{\mathbf{S}V_1M_2 \to_v \mathbf{S}V_1M_2'}$$

$$\frac{M_3 \to_v M_3'}{\mathbf{S}V_1V_2M_3 \to_v \mathbf{S}V_1V_2M_3'} \qquad \frac{M \to_v M'}{MP \to_v M'P}$$

*where $V_1, V_2, V_3$ are values, i.e. non $\to_v$-reducible CL-terms:*

$$V \ ::= \ \mathbf{K} \mid \mathbf{S} \mid \mathbf{K}V \mid \mathbf{S}V \mid \mathbf{S}VV.$$

Alternatively we could define the lazy strategy $\to_l$ as the closure of CL-reaction rules under the following reactive contexts (which coincide with the applicative ones): $D[\,] \ ::= \ [\,] \mid D[\,]P$.

Similarly, by considering the restriction to values of the reaction rules of Definition 11, we could define the cbv strategy $\to_v$ as the closure of CL-reaction rules under the following reactive contexts:

$$D[\,] \ ::= \ [\,] \mid D[\,]P \mid \mathbf{K}D[\,] \mid \mathbf{K}VD[\,] \mid \mathbf{S}D[\,] \mid \mathbf{S}VD[\,] \mid \mathbf{S}V_1V_2D[\,].$$

Let $\downarrow_\sigma$ denote the convergence relation on CL, for $\sigma \in \{l, v\}$.

**Definition 14 (Lazy/cbv Equivalence on CL)**

*i) A relation $\mathcal{R} \subseteq CL^0 \times CL^0$ is a* CL lazy/cbv bisimulation *if:*
$\langle M, N\rangle \in \mathcal{R} \implies (M \downarrow_\sigma \Leftrightarrow N \downarrow_\sigma) \ \wedge \ \forall P \in \Lambda^0.\ \langle MP, NP\rangle \in \mathcal{R}$.
*ii) Let $\simeq_\sigma \subseteq CL^0 \times CL^0$ be the largest CL lazy/cbv bisimulation.*
*iii) Let $\widehat{\simeq}_\sigma \subseteq CL \times CL$ denote the extension of $\simeq_\sigma$ to open terms defined by: for $M, N \in CL$ s.t. $FV(M, N) \subseteq \{x_1, \ldots, x_n\}$, $M\widehat{\simeq}_\sigma N$ iff for all closing substitutions $\boldsymbol{P}$, $M[\boldsymbol{P}/\boldsymbol{x}] \simeq_\sigma N[\boldsymbol{P}/\boldsymbol{x}]$.*

The following theorem, whose proof is in [DHL08], is interesting *per se*:

**Theorem 4.** *For all $M, N \in \Lambda$, $M\widehat{\approx}_\sigma N \iff \mathcal{T}(M)\widehat{\simeq}_\sigma\mathcal{T}(N)$.*

### 4.2 The First-Order Approach: CL as a Context Category

In the lazy case, where the reactive contexts coincide with the applicative ones, we can endow CL with a structure of reactive system in the sense of [LM00], by considering the smaller context category consisting of just applicative contexts. This allows us to obtain directly an lts with only applicative labels. In the cbv case, where the set of reactive contexts is larger, one can reduce the labels of the IPO bisimilarity only a posteriori, see [DHL08] for more details. For lack of space and in order to focus on other important aspects, we work out in detail only the lazy case.

**Definition 15 (Lazy CL Reactive System).** *The* lazy CL reactive system $\mathbf{C}_l^1$ *consists of:*

- *the context category whose objects are* $0, 1$*, where the morphisms from 0 to 1 are the closed terms, the morphisms from 1 to 1 are the closed applicative contexts, and composition is context substitution;*
- *the reactive contexts are all the closed applicative contexts;*
- *the reaction rules are* $\mathbf{K}M_1M_2 \to M_1$ *and* $\mathbf{S}M_1M_2M_3 \to (M_1M_2)(M_1M_3)$*, for all* $M_1, M_2, M_3$*.*

It is easy to prove that the reactive system $\mathbf{C}_l^1$ has redex RPOs. One can easily check that the minimal contexts are of the shape $[\,]\boldsymbol{P}$, where $\boldsymbol{P}$ has the minimal length for the top-level reaction to fire.

The strong versions of context and IPO bisimilarities are too fine, since, as in the $\lambda$-calculus case, they take into account reduction steps, and tell apart $\beta$-convertible terms. Thus we consider weak variants of such equivalences, where the identity context $[\,]$ is unobservable. Weak context bisimilarity is too coarse, since it equates all terms. However, we will prove that the weak IPO bisimilarity "almost" coincides with the lazy equivalence. Moreover, we will show how to recover the exact correspondence by considering a suitable variant of CL.

First of all, let $\simeq_{lI}$ denote the *lazy weak IPO bisimilarity* obtained by considering the identity context as unobservable. By Theorem 2, $\simeq_{lI}$ is a congruence w.r.t. reactive contexts, i.e.:

**Proposition 1.** *For all* $M, N, P \in CL^0$*,* $M \simeq_{lI} N \implies MP \simeq_{lI} NP$*.*

The rest of this section is devoted to compare the lazy weak IPO bisimilarity $\simeq_{lI}$ with the lazy equivalence on CL $\simeq_l$ defined in Definition 14.

Using coinduction and Proposition 1 one can easily prove that $\simeq_{lI} \subseteq \simeq_l$.

However, the converse inclusion, i.e. $\simeq_l \subseteq \simeq_{lI}$, does not hold, since e.g. $\mathbf{K} \simeq_l \mathbf{S}(\mathbf{KK})(\mathbf{SKK})$, while $\mathbf{K} \not\simeq_{lI} \mathbf{S}(\mathbf{KK})(\mathbf{SKK})$. Namely $\mathbf{S}(\mathbf{KK})(\mathbf{SKK}) \xrightarrow{[\,]P}_I$, while $\mathbf{K} \not\xrightarrow{[\,]P}_I$. The problem arises since the equivalence $\simeq_{lI}$ tells apart terms whose top-level combinators expect a different number of arguments to reduce. In order to overcome this problem, we consider an extended calculus, CL$^*$, where the combinators $\mathbf{K}$ and $\mathbf{S}$ become unary, at the price of adding new intermediate combinators and intermediate reductions (the reactive contexts are the ones in Definition 15).

**Definition 16.** *Let CL$^*$ be the combinatory calculus defined by*

- *Terms:* $M ::= x \mid \mathbf{K} \mid \mathbf{S} \mid \mathbf{K}'M \mid \mathbf{S}'M \mid \mathbf{S}''MN \mid MN$
  *where* $\mathbf{K}$*,* $\mathbf{K}'$*,* $\mathbf{S}$*,* $\mathbf{S}'$*,* $\mathbf{S}''$ *are combinators.*
- *Rules:* $\mathbf{K}M \to \mathbf{K}'M \qquad \mathbf{K}'MN \to M$
  $\mathbf{S}M \to \mathbf{S}'M \qquad \mathbf{S}'MN \to \mathbf{S}''MN \qquad \mathbf{S}''MNP \to (MP)(NP)$

Notice that the calculus in the above definition is well-defined, since the set of terms is closed under the reaction rules.

Now let use denote by $\simeq_{lI}^*$ the weak IPO bisimilarity obtained by considering the lazy reactive system over CL$^*$. Then, we have $\mathbf{K} \simeq_{lI}^* \mathbf{S}(\mathbf{KK})(\mathbf{SKK})$. More in general, by considering the reactive system over CL$^*$, the induced weak IPO bisimilarity $\simeq_{lI}^*$ coincides with the lazy equivalence on CL:

**Theorem 5.** *For all* $M, N \in CL^0$, $\simeq^*_{lI} = \simeq_l$.

As a consequence of Theorem 4 and Theorem 5 above, we can recover the lazy observational equivalence on $\lambda$-terms as weak IPO bisimilarity on $\mathrm{CL}^*$.

**Proposition 2.** *For all* $M, N \in \Lambda^0$, $M \approx_l N \iff \mathcal{T}(M) \simeq^*_{lI} \mathcal{T}(N)$.

However, such notion of weak IPO bisimilarity still suffers of the problem of being infinitely branching, since IPO contexts are either $[\,]$ or $[\,]P$, for all $P \in (CL^*)^0$. This problem will be solved in Section 5.1, where $\mathrm{CL}^*$ is endowed with a structure of second-order context category.

## 5 Second-Order Term Contexts

The definition of term context category [LM00] can be generalized to a definition of second-order term context. The generalization is obtained by extending the term syntax with function (second-order) variables, that is variables not standing for terms but instead for functions on terms. The formal definition is the following

**Definition 17 (Category of Second-order Term Contexts).** *Let* $\Sigma$ *be a signature for a term language. The category of* second-order term contexts *over* $\Sigma$ *is defined by: objects are finite lists of naturals* $\langle n_1, \ldots, n_k\rangle$*, an arrow* $\langle m_1, \ldots, m_h\rangle \to \langle n_1, \ldots, n_k\rangle$ *is a k-tuple* $\langle t_1, \ldots, t_k\rangle$*, where the term* $t_i$ *is defined over the signature* $\Sigma \cup \{F_1^{m_1}, \ldots, F_h^{m_h}\} \cup \{X_{i,1} \ldots, X_{i,n_i}\}$*, where* $F_i^{m_i}$ *is a function variable of arity* $m_i$*,* $X_{i,j}$ *is a ground variable. The category of* second-order linear term context*,* $T_2^*(\Sigma)$*, is the subcategory whose arrows are n-tuples of terms, satisfying the condition that the n-tuples have to contain exactly one use of each function variable* $F_i^{m_i}$.

One can check that the standard category of term contexts over $\Sigma$ coincides with the subcategory whose objects are the lists containing only copies of the natural number 0; in fact this subcategory uses function variables with no arguments and the ground variables do not appear.

Since the simplest way to define composition in category $T_2^*(\Sigma)$, and more generally in the category of second-order term context, is in terms of $\beta$-reduction, it is useful to represent morphisms, i.e. terms on the signature $\Sigma \cup \{F_1^{m_1}, \ldots, F_h^{m_h}\} \cup \{X_{i,1}, \ldots, X_{i,n_i}\}$, using a $\lambda$-notation for binding variables, that is, instead of writing a term with free variables, we write its lambda closure. To avoid ambiguities we use a different symbol $\mathbb{\lambda}$ for this form of lambda abstraction. With this notation a term $t$ on the signature $\Sigma \cup \{F_1^{m_1}, \ldots, F_h^{m_h}\} \cup \{X_1, \ldots, X_n\}$ is written as: $\mathbb{\lambda}F_1^{m_1} \ldots F_h^{m_h}.\mathbb{\lambda}X_1 \ldots X_n.t$, or as $\mathbb{\lambda}\boldsymbol{F}.\mathbb{\lambda}\boldsymbol{X}.t$ for brevity. In general a second-order context $\langle t_1, \ldots, t_k\rangle : \langle m_1, \ldots, m_h\rangle \to \langle n_1, \ldots, n_k\rangle$ is written as $\mathbb{\lambda}\boldsymbol{F}.\langle\, \mathbb{\lambda}\boldsymbol{X}_1.t_1, \ldots,\ \mathbb{\lambda}\boldsymbol{X}_k.t_k\rangle$.

(i) The identity on $\langle n_1, \ldots, n_k\rangle$ is: $\mathbb{\lambda}\boldsymbol{F}.\langle\, \mathbb{\lambda}\boldsymbol{X_1}.F_1^{n_1}(\boldsymbol{X}_1), \ldots,\ \mathbb{\lambda}\boldsymbol{X}_k.F_k^{n_k}(\boldsymbol{X}_k)\rangle$.
(ii) The composition between the morphisms $\mathbb{\lambda}\boldsymbol{F}.\langle\, \mathbb{\lambda}\boldsymbol{X}_1.s_1, \ldots,\ \mathbb{\lambda}\boldsymbol{X}_k.s_k\rangle : \langle l_1, \ldots, l_h\rangle \to \langle m_1, \ldots, m_k\rangle$ and $\mathbb{\lambda}\boldsymbol{G}.\langle\, \mathbb{\lambda}\boldsymbol{Y}_1.t_1, \ldots,\ \mathbb{\lambda}\boldsymbol{Y}_j.t_j\rangle : \langle m_1, \ldots, m_k\rangle \to$

$\langle n_1, \ldots, n_j \rangle$ is the $\beta$-normal form of the expression $\lambda\!\!\lambda \boldsymbol{F}.(\lambda\!\!\lambda \boldsymbol{G}.\langle \lambda\!\!\lambda \boldsymbol{Y}_1.t_1, \ldots, \lambda\!\!\lambda \boldsymbol{Y}_j.t_j \rangle)\ (\lambda\!\!\lambda \boldsymbol{X}_1.s_1, \ldots, \lambda\!\!\lambda \boldsymbol{X}_k.s_k) : \langle l_1, \ldots, l_h \rangle \to \langle n_1, \ldots, n_j \rangle$. Informally, the composition is given by a $j$-tuple of expressions $t_i$ in which every function variable $G_l$ is substituted by the corresponding expression $s_l$, with the ground variables of $s_l$ substituted by the corresponding parameters of $G_l$ in $t_i$. For example, considering the signature for natural numbers $\{0, S, +\}$, the composition between $\lambda\!\!\lambda F.\ \lambda\!\!\lambda X_1.F(X_1, 0) : \langle 2 \rangle \to \langle 1 \rangle$ and $\lambda\!\!\lambda G.\ \lambda\!\!\lambda Y_1 Y_2.(G(S(Y_1)) + Y_2) : \langle 1 \rangle \to \langle 2 \rangle$ is the second order context: $\lambda\!\!\lambda F.\ \lambda\!\!\lambda Y_1 Y_2.F(S(Y_1), 0) + Y_2 : \langle 2 \rangle \to \langle 2 \rangle$.

Note that the identity morphism is defined as a $\lambda$-term implementing the identity function, while composition on morphisms is defined by the function composition in the $\lambda$-setting. Given this correspondence, it is easy to prove that the categorical properties for the identity hold, while the associativity of composition essentially follows from the unicity of the normal form.

The main general result on second-order term contexts is the following:

**Proposition 3.** *For any signature $\Sigma$, the category of second-order linear term contexts over $\Sigma$ admits RPOs.*

### 5.1 CL as Second-Order Rewriting System

In this section, we consider the second-order context category for the combinatory calculus CL$^*$ and we show that the weak IPO bisimilarity thus obtained coincides with the observational equivalence on $\lambda$-calculus. Interestingly, the second-order open bisimilarity gives a uniform characterization also on open terms. For simplicity, we work out in detail only the lazy case. However, the main result holds also in the cbv case.

Note that the terms of CL are defined by signature $\Sigma_{CL} = \{K, S, \mathtt{app}\}$, where $\mathtt{app}$ is the binary operation of application that is usually omitted. So the term $\mathbf{SKK}$ actually stands for $\mathtt{app}(\mathtt{app}(\mathbf{S}, \mathbf{K}), \mathbf{K})$.

**Definition 18 (Lazy CL$^*$ Second-order Reactive System).** *The* lazy second-order Reactive system $\mathbf{C}_l^{2*}$ *consists of:*

- *the category whose objects are the lists with at most one element, and whose arrows $\epsilon \to \langle n \rangle$ are the terms of CL$^*$ with, at most, n (first order) variables, and the whose $\langle m \rangle \to \langle n \rangle$ are the second-order applicative contexts of the shape $F(M_1, \ldots, M_m)N_1 \ldots N_k$, with, at most, n (first order) variables;*
- *the reactive contexts are all the second-order applicative contexts;*
- *the reaction rules are:*

$$\mathbf{K}X_1 \to \mathbf{K}'X_1 \qquad \mathbf{K}'X_1X_2 \to X_1$$
$$\mathbf{S}X_1 \to \mathbf{S}'X_1 \qquad \mathbf{S}'X_1X_2 \to \mathbf{S}''X_1X_2 \qquad \mathbf{S}''X_1X_2X_3 \to (X_1X_2)(X_1X_3).$$

To maintain the notation for contexts used in Sections 3, 4, in the sequel a second-order applicative context $F(M_1, \ldots, M_m)N_1 \ldots N_k : \langle m \rangle \to \langle n \rangle$ will be more conveniently written as $[\,]_\theta N_1 \ldots N_k$, where $\theta$ is a substitution s.t. $\theta(X_i) = M_i$ for all $i = 1, \ldots, m$, moreover we write $M \overset{[\,]_\theta \boldsymbol{P}}{\to} M'$ iff $(M\theta)\boldsymbol{P} \to M'$. Given Proposition 3, and the underlined RPOs construction, we have:

**Corollary 1.** *The reactive system* $\mathbf{C}_l^{2*}$ *has redex RPOs.*

Using Lemma 1, one can check that the lazy weak IPO bisimilarity for $\mathbf{C}_l^{2*}$, obtained by considering the identity context as unobservable, can be characterized as follows:

**Lemma 3 (Second-order Lazy Weak IPO bisimilarity)**

*i) A symmetric relation* $\mathcal{R}$ *on terms of* $CL^*$ *is a* second-order lazy weak IPO bisimulation *if the following holds* $\langle M, N\rangle \in \mathcal{R}$ *and* $M \overset{[\,]_\theta \boldsymbol{P}}{\longrightarrow}_I M'$ *then there exists* $N'$ *such that* $N \overset{[\,]_\theta \boldsymbol{P}}{\Longrightarrow}_I N'$ *and* $\langle M', N'\rangle \in \mathcal{R}$.
*ii) The* second-order lazy weak IPO bisimilarity $\simeq_{lI}^{2*}$ *is the greatest second-order lazy weak IPO bisimulation.*

Notice that, for a given term, there could be infinitely many second-order IPO reductions, where the redex is entirely contained in the context. E.g. for the term $XM_1$, $XM_1 \overset{[\,]_{\{\mathbf{K}Y/X\}}}{\longrightarrow} \mathbf{K}'YM_1$ and $XM_1 \overset{[\,]_{\{\mathbf{K}Y_1Y_2/X\}}}{\longrightarrow} \mathbf{K}'Y_1Y_2M_1$ are both IPO reductions of this shape. However, there are only finitely many IPO contexts s.t. the redex is not entirely contained in the context. One can show that such IPO reductions are sufficient for determining the weak IPO bisimilarity, thus getting a finitely branching characterization.

*Example:* Let $M = XM_1$. The "relevant" IPO reductions of $M$ (i.e. those which are not entirely contained in the context) are the following:

$XM_1 \overset{[\,]_{\{\mathbf{K}/X\}}}{\longrightarrow} \mathbf{K}'M_1$; $XM_1 \overset{[\,]_{\{\mathbf{K}'Y/X\}}}{\longrightarrow} Y$; $XM_1 \overset{[\,]_{\{\mathbf{K}'/X\}}Y}{\longrightarrow} M_1$; $XM_1 \overset{[\,]_{\{\mathbf{S}/X\}}}{\longrightarrow} \mathbf{S}'M_1$; $XM_1 \overset{[\,]_{\{\mathbf{S}'Y/X\}}}{\longrightarrow} \mathbf{S}''YM_1$; $XM_1 \overset{[\,]_{\{\mathbf{S}'/X\}}Y}{\longrightarrow} \mathbf{S}''M_1Y$; $XM_1 \overset{[\,]_{\{\mathbf{S}''YZ/X\}}}{\longrightarrow} (YZ)(YM_1)$; $XM_1 \overset{[\,]_{\{\mathbf{S}''Y/X\}}Z}{\longrightarrow} (YM_1)(YZ)$; $XM_1 \overset{[\,]_{\{\mathbf{S}''/X\}}YZ}{\longrightarrow} (M_1Y)(M_1Z)$.

In general, the relevant IPO contexts are summarized in the following table:

| term | IPO contexts |
|---|---|
| $X$ | $[\,]_{\{\overline{\mathbf{C}}Y/X\}}$, $[\,]_{\{\overline{\mathbf{C}}/X\}}Y$ |
| $XN\boldsymbol{P}$ | $[\,]_{\{\overline{\mathbf{C}}/X\}}$ |
| $\mathbf{C}$ | $[\,]_\emptyset X$ |
| $\mathbf{C}N\boldsymbol{P}$ | $[\,]_\emptyset$ |

In order to prove that the IPO contexts in the table are sufficient for determining the weak IPO bisimilarity, one can show (by "coinduction up-to") that the bisimilarity obtained by considering only such contexts is a congruence w.r.t. substitutions.

$\mathbf{C} \in \{\mathbf{K}, \mathbf{S}, \mathbf{K}'M, \mathbf{S}'M, \mathbf{S}''MN \mid M, N \in CL\}$
$\overline{\mathbf{C}} \in \{\mathbf{K}, \mathbf{S}, \mathbf{K}'Z, \mathbf{S}'Z, \mathbf{S}''Z_1Z_2 \mid Z, Z_1, Z_2 \text{ fresh variables}\}$.

More in general, by Theorem 2, $\simeq_{lI}^{2*}$ is a congruence w.r.t. reactive contexts:

**Proposition 4.** *For all terms of* $CL^*$ $M, N$, *for all substitutions* $\theta$, *for all* $CL^*$*-terms* $P_1, \ldots, P_n$, $M \simeq_{lI}^{2*} N \implies (M\theta)\boldsymbol{P} \simeq_{lI}^{2*} (N\theta)\boldsymbol{P}$.

The following is the main result of this section:

**Proposition 5.** *For all* $M, N \in \Lambda$, $M \widehat{\approx}_l N \iff \mathcal{T}(M) \simeq_{lI}^{2*} \mathcal{T}(N)$.

The proof of the above proposition, which appears in [DHL08], proceeds by showing that $\simeq_{lI}^{2*}$ coincides with the natural extension of $\simeq_{lI}^{*}$ to open terms.

Notice that Proposition 5 gives a uniform finitely branching characterization also on open terms.

## 6 Final Remarks and Directions for Future Work

– There are several other attempts to deal with parametric rules in the literature. In particular, in [KSS05], the authors introduce the notion of *luxes* to generalize the RPO approach to cases where the rewriting rules are given by pairs of arrows having a domain different from 0. When instantiated to the category of contexts, the luxes approach allows to express rewriting rules not formed by pairs of ground terms but, instead formed by pairs of contexts (open terms), and so allowing parametricity. Compared to our approach, based on the notion of second-order context, the approach of luxes is more abstract and can be applied to a wider range of cases (categories). However, if we compare the two approaches in the particular case of context categories, we find that luxes approach has a more restricted way to instantiate a given parametric rule. This restriction results in a not completely satisfactory treatment of the $\lambda$-calculus. It remains the open question whether substituting the notion of second-order contexts with a more abstract one, that should look like a general second-order Lawvere theory. In this way, it should be possible to recover the extra generality of luxes.

– A possible alternative approach for dealing with the $\lambda$-calculus in Leifer-Milner's RPO setting, it that of using suitable encodings in the (bi)graph framework [Mil07]. However, we feel that our term solution based on second-order context categories and CL is simpler and more direct. Alternatively, in place of CL, one could also consider a $\lambda$-calculus with explicit substitutions, in order to obtain a convenient encoding of the $\beta$-rule, allowing for a representation as a second-order reactive system. This is an experiment to be done. Here we have choosen CL, since it is simpler; moreover, the correspondence between the standard $\lambda$-calculus and the one with explicit substitutions deserves further study.

– We have considered lazy and cbv strategies, however also other strategies, e.g. *head* and *normalizing* could be dealt with, possibly at the price of some complications due to the fact that such strategies are usually defined on open terms. It would be also interesting to explore non-deterministic strategies on $\lambda$-calculus.

## References

AO93. Abramsky, S., Ong, L.: Full Abstraction in the Lazy Lambda Calculus. Information and Computation 105(2), 159–267 (1993)

BKM06. Bonchi, F., Konig, B., Montanari, U.: Saturated Semantics for Reactive Systems. In: LICS (2006)

BGK06. Bonchi, F., Gadducci, F., Konig, B.: Process Bisimulation via a Graphical Encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 168–183. Springer, Heidelberg (2006)

DHL08. Di Gianantonio, P., Honsell, F., Lenisa, M.: RPO, Second-order Contexts, and λ-calculus, Extended version (2008), www.dimi.uniud.it/pietro/papers/socl.pdf

EHR92. Egidi, L., Honsell, F., Ronchi Della Rocca, S.: Operational, Denotational and Logical Descriptions: A Case Study. Fundamenta Inf. 16(2), 149–169 (1992)

EK06. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. Mathematical Structure in Computer Science 16(6), 1133–1163 (2006)

GM05. Gadducci, F., Montanari, U.: Observing Reductions in Nominal Calculi via a Graphical Encoding of Processes. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 106–126. Springer, Heidelberg (2005)

HS86. Hindley, R., Seldin, J.: Introduction to combinators and l-calculus. Cambridge University Press, Cambridge (1986)

KSS05. Klin, B., Sassone, V., Sobocinski, P.: Labels from reductions: Towards a general theory. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 30–50. Springer, Heidelberg (2005)

Lei01. Leifer, J.: Operational congruences for reactive systems, PhD thesis, University of Cambridge Computer Laboratory (2001)

LM00. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)

Mil07. Milner, R.: Local bigraphs and confluence: Two conjectures. In: Proc. Express 2006. ENTCS, vol. 175, pp. 65–73 (2007)

SS03. Sassone, V., Sobocinski, P.: Deriving bisimulation congruences: 2-categories vs precategories. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 409–424. Springer, Heidelberg (2003)

SS05. Sassone, V., Sobocinski, P.: Reactive systems over cospans. In: LICS 2005, pp. 311–320. IEEE, Los Alamitos (2005)

Sew02. Sewell, P.: From rewrite rules to bisimulation congruences. Theoretical Computer Science 274(1–2), 183–230 (2002)

Sob04. Sobocinski, P.: Deriving process congruences from reduction rules, PhD thesis, University of Aarhus (2004)

# Erasure and Polymorphism in Pure Type Systems

Nathan Mishra-Linger and Tim Sheard

Portland State University
{rlinger,sheard}@cs.pdx.edu

**Abstract.** We introduce *Erasure Pure Type Systems*, an extension to Pure Type Systems with an erasure semantics centered around a type constructor $\forall$ indicating parametric polymorphism. The erasure phase is guided by lightweight program annotations. The typing rules guarantee that well-typed programs obey a phase distinction between erasable (compile-time) and non-erasable (run-time) terms.

The erasability of an expression depends only on how its value is *used* in the rest of the program. Despite this simple observation, most languages treat erasability as an intrinsic property of expressions, leading to code duplication problems. Our approach overcomes this deficiency by treating erasability extrinsically.

Because the execution model of EPTS generalizes the familiar notions of type erasure and parametric polymorphism, we believe functional programmers will find it quite natural to program in such a setting.

## 1 Background and Motivation

Statically typed programming languages have evolved ever more expressive type systems. The drive towards increased expressiveness inevitably leads to dependent types, a proven technology for verifying strong correctness properties of real programs [15,9,5,8]. For this reason, researchers have long sought practical ways to include dependent types in programming languages.

Heavy use of the expressive power of dependent types involves the embedding of proofs of program properties into the program text itself. Often, such proofs play no essential part in the execution of the program. They are necessary only as evidence used by the type-checker. Because these proofs can be quite large, an erasure semantics is critical for practical implementation of a dependently typed programming language.

However, proofs are not the only erasable parts of a program. Any time a program exhibits *parametric polymorphism* (whether it be polymorphism over proofs, types, numbers, or any other type of value) there are portions of the program that should be erased. One thesis of this paper is that parametric polymorphism can be understood entirely in terms of erasability.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 350–364, 2008.

### 1.1 Erasability Is Extrinsic Rather than Intrinsic

Which parts of a program[1] may be erased in an erasure-semantics? Our investigation of erasure semantics is grounded in a simple observation: Erasability of a program expression is not a property of the expression itself but rather a property of the context in which we find it. Erasability of a program expression is determined not by what it *is*, but by how it is *used*. In other words, erasability is an *extrinsic* rather than an *intrinsic* property. We give several examples demonstrating this principle.

**Type annotations.** The domain annotation $A$ in the $\beta$ rule,

$$(\lambda x{:}A.\, M)\ N \to_\beta M[N/x]$$

is simply discarded. For this reason, we may safely erase the domain annotations of all $\lambda$-abstractions in a program without changing their computational behavior. In this case, the context in which $A$ appears determines its erasability.

**Dummy $\lambda$-binders.** The erasure of domain annotations may cause some $\lambda$-binders to become superfluous. Consider the term $(\lambda\alpha{:}\mathsf{Type}.\, \lambda x{:}\alpha.\, x)\ Nat\ 5$. After erasing type annotations, we are left with $(\underline{\lambda\alpha.}\, \lambda x.\, x)\ \underline{Nat}\ 5$, in which the binder $\lambda\alpha$ is superfluous because $\alpha$ no longer appears anywhere in its scope. For any such dummy binder $\lambda x$, the resulting specialized $\beta$ rule

$$(\lambda x.\, M)\ N \to_\beta M \qquad \text{if } x \notin FV(M)$$

discards both the dummy binder and the argument which it would otherwise bind to $x$. Therefore we may erase both the binding site $\lambda x$ and any argument $N$ at an application site to which this $\lambda$-abstraction may flow during program execution. By this reasoning, we may erase the underlined portions of our previous example term, resulting in $(\lambda x.\, x)\ 5$.

However, other $\lambda$-abstractions may flow to some of those same application sites. We should not erase the argument $N$ at an application site[2] $M@N$ unless every $\lambda$-abstraction that may flow to be the value of $M$ has a dummy binder. In general, the "may-flow-to" relation induces a bipartite graph (see Figure 1). In order to decide if a given $\lambda$-binder or @-argument may be safely erased, we must analyze its entire connected component (CC) in this $\lambda/@$ graph.

In this type of erasure step, the *usage* of a term determines its erasability. The (local) erasability of a binder $\lambda x$ depends on how $x$ is (or is not) used in its scope. The erasability of an argument $N$ depends on its context — whether the function that is applied to it always ends up being a $\lambda$-abstraction with a dummy binder.

[1] For our purposes here, a program is a term in a typed $\lambda$-calculus.

[2] We sometimes write @ for application in order to have a more tangible notation than mere juxtaposition.

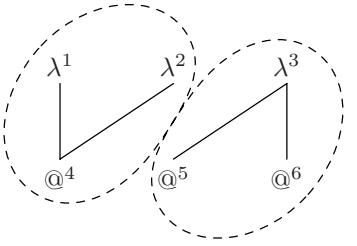



**Fig. 1.** The $\lambda/@$ graph induced by the "may-flow-to" relation of a simple program. The use of $y$ in $\lambda^2$'s body prevents erasure of both the $@^4$-argument and the $\lambda^1$-binder.

**Cascading Erasure.** Erasure of @-arguments may make other $\lambda$-binders into dummies, thereby enabling erasure in other CCs of the $\lambda/@$ graph. Consider the following family of identity functions.

$$\begin{array}{l} \mathsf{let}\ id_0 = \lambda a{:}s.\, \lambda x{:}a.\, x\ \mathsf{in} \\ \mathsf{let}\ id_1 = \lambda a{:}s.\, \lambda x{:}a.\, id_0\ a\ x\ \mathsf{in} \\ \mathsf{let}\ id_2 = \lambda a{:}s.\, \lambda x{:}a.\, id_1\ a\ x\ \mathsf{in} \\ \mathsf{let}\ id_3 = \lambda a{:}s.\, \lambda x{:}a.\, id_2\ a\ x\ \mathsf{in} \\ \ldots \end{array} \quad \Rightarrow \quad \begin{array}{l} \mathsf{let}\ id_0 = \lambda a.\, \lambda x.\, x\ \mathsf{in} \\ \mathsf{let}\ id_1 = \lambda a.\, \lambda x.\, id_0\ a\ x\ \mathsf{in} \\ \mathsf{let}\ id_2 = \lambda a.\, \lambda x.\, id_1\ a\ x\ \mathsf{in} \\ \mathsf{let}\ id_3 = \lambda a.\, \lambda x.\, id_2\ a\ x\ \mathsf{in} \\ \ldots \end{array}$$

After the initial erasure of domain annotations, a cascading sequence of $\lambda/@$ erasure steps is possible in this program. (Consider the $\lambda a$ binders).

### 1.2 Intrinsic Notions of Erasability Beget Code Duplication

Most prior attempts to combine dependent types and erasure semantics treat erasability as an intrinsic property. These attempts may be divided into two categories: erasure first and dependent types first.

**Erasure first.** Languages in this category start with a commitment to erasure semantics in the form of a syntactic phase distinction whereby types and program values may not depend on each other computationally. Singleton types are then used to simulate dependently typed programming. Examples of this approach include Dependent ML [22], $\Omega$mega [20,19], Applied Type Systems [7], and Haskell with generalized algebraic datatypes [16].

Singleton types are type families $T{:}I \to \mathsf{Type}$ for which each type index $i{:}I$ uniquely determines the one value of type $T\,i$. For example, the declarations

$$\begin{array}{l} \mathsf{datakind}\ Nat{\uparrow}\ :\mathsf{kind} \\ \quad \mathsf{where}\ Zero{\uparrow} : Nat{\uparrow} \\ \qquad\quad Succ{\uparrow} : Nat{\uparrow} \to Nat{\uparrow} \end{array} \qquad \begin{array}{l} \mathsf{datatype}\ Nat!\ \ :Nat{\uparrow} \to \mathsf{Type} \\ \quad \mathsf{where}\ Zero! : Nat!\ Zero{\uparrow} \\ \qquad\quad Succ! : Nat!\ n \to Nat!\ (Succ{\uparrow}\ n) \end{array}$$

introduce a singleton type family for the naturals. The datakind declaration defines a *copy* of the natural numbers at the type-level. The singleton type $Nat!$ then connects the type-level version $Nat{\uparrow}$ to the level of run-time expressions.

A singleton type acts as a proxy between run-time and compile-time notions of the same datatype: natural numbers in this case. Whenever a program does case analysis on the value of a singleton type, the type-checker benefits from the same case analysis at the type-level. In this way, dependence of types on values is simulated.

**Dependent types first.** Languages in this category start with full dependent types. An erasure phase then strips out parts of the program that are irrelevant to its run-time execution. Examples of this approach include Cayenne [3], Coq [1], and Epigram [12,6].

In Cayenne and Coq, the erasability of a subterm depends on its type. All types (subterms of type Type) are erased in Cayenne and Epigram[3]. Coq's program extraction mechanism supports erasure of *proofs* as well as types. A proof is distinguished by having a proposition as its type, and propositions are distinguished as terms of type Prop. In contrast to the universe Prop, Coq has another universe Set that is the type of the types of all non-erasable program terms.

**Code duplication.** Because languages in both categories treat erasability as an intrinsic property of an expression, usually determined by its type, users of these languages are sometimes forced to duplicate definitions of datatypes and functions over them in order to achieve a desired erasure behavior. In the erasure-first approach, programming with singleton types requires duplication of datatype definitions at the "type" and "kind" levels of the type hierarchy[4], as well as duplication of functions that operate on them. In the dependent-types-first approach, duplication of datatypes is also required if we want values of a particular type to be erased in one part of a program but not in another.

### 1.3 Methodology and Outline

We treat erasability as a property not of a term itself, but of the context in which it is used. In $\lambda$-calculus, functions reify such contexts, so we track erasability as a property of functions by distinguishing between functions that do not depend computationally on their arguments (of type $\forall x{:}A.\, B$) and those that might (of type $\Pi\, x{:}A.\, B$).

Note that the same $A$ is used in both cases, because erasability is no longer an intrinsic property of $x$, but rather a property of the (functional) contexts making use of $x$. In this way we avoid the code duplication problem. We have *one* type $A$ and therefore functions over $A$ can be written *once*.

*Pure Type Systems* (PTS) are a well-known family of typed $\lambda$-calculi that encompass a wide variety of type systems [4]. Most dependently typed languages

[3] Some work on representations of inductive types in Epigram notes that values of type families need not store certain indices that, regardless of their type, are uniquely determined by the value's data constructor.

[4] The singleton type itself may be thought of as a maximally informative copy of the original datatype.

have a PTS at their core. Therefore PTS is a good setting for studying features of dependently types languages. Section 2 briefly reviews the basics of PTS.

Section 3 introduces a conservative extension to PTS called Erasure Pure Type Systems (EPTS) supporting the $\forall$ type described above and rules for checking that programs using this type satisfy a *phase distinction*.

Section 4 introduces another PTS variant called Implicit Pure Type Systems (IPTS) that serves as the target language of the erasure phase. This language is very closely related to Miquel's Implicit Calculus of Constructions [13].

Section 5 introduces our erasure translation from EPTS to IPTS. This operation is the basis for the erasure semantics of EPTS (Section 6). We prove that erasure exhibits properties one would expect: It respects the static and dynamic semantics of programs and *eliminates portions of the source program that do not affect its final value*.

In Sections 7, 8, and 9 we discuss implementation issues, future work, and our conclusions.

## 2 Pure Type Systems

Pure Type Systems bring organization to type theory [4]. They generalize Barendregt's $\lambda$-cube, which includes such familiar systems as the simply typed $\lambda$-calculus, Systems F and $F^\omega$, the Edinburgh Logical Framework, and the Calculus of Constructions.

Pure Type Systems are a family of typed lambda calculi. Each PTS has a specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ consisting of a set $\mathcal{S}$ of *sorts* (a.k.a. *universes*), a set $\mathcal{A} \subseteq \mathcal{S}^2$ of *axioms*, and a set $\mathcal{R} \subseteq \mathcal{S}^3$ of *rules*. We assume a fixed specification throughout the development. The syntax of PTS is as follows:

$$M, N, A, B ::= x \mid \lambda x{:}A.\, M \mid M\ N \mid \Pi x{:}A.\, B \mid s$$

Note that there is a single syntactic category for types and terms. The metavariable $s$ is used to denote sorts. The typing rules of PTS are parameterized by $\mathcal{A}$ and $\mathcal{R}$ and can be obtained from those of EPTS (which we will discuss shortly) by simply ignoring all erasure annotations.

## 3 Erasure Pure Type Systems

This section introduces Erasure Pure Type Systems (EPTS), an extension of Pure Type Systems (PTS) with annotations indicating erasable parts of a program. The EPTS type system checks the erasability of the parts so annotated.

**Syntax.** The syntax of EPTS is that of PTS with erasure annotations added.

$$\begin{array}{lll} \textit{(terms)} & M, N, A, B ::= x \mid \lambda^\tau x{:}A.\, M \mid M@^\tau N \mid \Pi^\tau x{:}A.\, B \mid s \\ \textit{(contexts)} & \Gamma, \Delta ::= \varepsilon \mid \Gamma, x{:}^\tau A \\ \textit{(times)} & \tau ::= \mathsf{r} \mid \mathsf{c} \end{array}$$

The metavariable $\tau$ ranges over erasure annotations. The annotation $\mathsf{r}$ means "run-time". Syntax with this annotation behaves just as it would in PTS without any annotation. The annotation $\mathsf{c}$ means "compile-time" and indicates erasable portions of a program.

All $\Pi$s, $\lambda$s, and @s are annotated. Annotations on $\Pi$s distinguish between computational dependence ($\Pi^{\mathsf{r}}$) and polymorphism ($\Pi^{\mathsf{c}}$). In concrete syntax, we would simply write $\Pi$ for $\Pi^{\mathsf{r}}$ and $\forall$ for $\Pi^{\mathsf{c}}$, but this choice of abstract syntax affords us economy of presentation. These annotations guide the erasure operation to be defined in Section 5.

**Type System.** Figure 2 contains typing rules for EPTS. There are two forms of judgment, $\Gamma \vdash M :^{\mathsf{c}} A$ and $\Gamma \vdash M :^{\mathsf{r}} A$. The judgment $\Gamma \vdash M :^{\mathsf{r}} A$ says that $M$ is a well-formed run-time entity, while $\Gamma \vdash M :^{\mathsf{c}} A$ says that $M$ is a well-formed compile-time (erasable) entity.

The type system needs to check that all $\lambda$s and @s marked $\mathsf{c}$ are erasable. Recall from Section 1.1 that erasability of $\lambda$s and @s in the $\lambda$/@ graph must be considered one connected component (CC) at a time. The flow analysis implicit in the typing rules ensures that every $\lambda$ and @ in the same CC are annotated with the same $\tau$. Therefore, if every $\lambda^{\mathsf{c}}$-binder is erasable, then so is every $@^{\mathsf{c}}$-argument. So we need only verify that each $\lambda^{\mathsf{c}}$ is erasable — for each $\lambda^{\mathsf{c}} x{:}A.\, M$ in the program, all free occurrences of $x$ in $M$ must appear either inside a type annotation or inside an $@^{\mathsf{c}}$ argument.

The typing rules enforce this invariant using the following technique, due to Pfenning [17]. Each $\lambda^{\mathsf{c}}$-bound variable $x$ is marked with $\mathsf{c}$ when it is added to the typing context. This mark is then locally switched off (reset) whenever we check a type annotation or $@^{\mathsf{c}}$ argument. We then require that the mark $\mathsf{c}$ has been switched off by the time we reach any occurrence of $x$. For economy of presentation, an "off" mark in the typing context is represented as an $\mathsf{r}$ mark. Passing this mark/reset/check test guarantees that each $\lambda^{\mathsf{c}}$ is actually erasable.

**Definition (Context Reset Operation)** $\boxed{\Gamma^\circ}$

$$\varepsilon^\circ = \varepsilon \qquad\qquad (\Gamma, x{:}^\tau A)^\circ = \Gamma^\circ, x{:}^{\mathsf{r}} A$$

The key steps of the mark/reset/check test are found in the typing rules $\Pi$-Intro (mark), $\Pi$-Elim and Reset (reset), and Var (check). In particular, notice how rule $\Pi$-Intro marks context entries and $\Pi$-Elim checks function arguments for both $\tau = \mathsf{r}$ and $\tau = \mathsf{c}$.

Rules Var, Weak, $\Pi$-Intro, and Conv each have a premise of the form $\Gamma \vdash A :^{\mathsf{c}} s$. The purpose of these rules is to check that $A$ is well-formed *as a type*. Because these rules deal explicitly with types, they use the compile-time typing judgment. In particular, domain annotations are considered as compile-time entities in the $\Pi$-Intro rule.

The $\Pi$-Form rule may seem counter-intuitive at first. Because $\Pi$ is a type former, one might expect this rule to use $\mathsf{c}$-judgments rather than $\mathsf{r}$ ones. However, in a dependently typed language, terms may compute (at run-time) to

$\boxed{\Gamma \vdash M :^{\tau} A}$

$$\textsc{Axiom}\ \frac{(s_1, s_2) \in \mathcal{A}}{\vdash s_1 :^{\mathsf{r}} s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash A :^{\mathsf{c}} s}{\Gamma, x{:}^{\mathsf{r}} A \vdash x :^{\mathsf{r}} A} \qquad \textsc{Weak}\ \frac{\Gamma \vdash A :^{\mathsf{c}} s \qquad \Gamma \vdash M :^{\mathsf{r}} B}{\Gamma, x{:}^{\tau} A \vdash M :^{\mathsf{r}} B}$$

$$\Pi\text{-}\textsc{Form}\ \frac{(s_1, s_2, s_3) \in \mathcal{R} \qquad \Gamma \vdash A :^{\mathsf{r}} s_1 \qquad \Gamma, x{:}^{\mathsf{r}} A \vdash B :^{\mathsf{r}} s_2}{\Gamma \vdash \Pi^{\tau} x{:}A.\, B :^{\mathsf{r}} s_3}$$

$$\Pi\text{-}\textsc{Intro}\ \frac{\Gamma \vdash \Pi^{\tau} x{:}A.\, B :^{\mathsf{c}} s \qquad \Gamma, x{:}^{\tau} A \vdash M :^{\mathsf{r}} B}{\Gamma \vdash \lambda^{\tau} x{:}A.\, M :^{\mathsf{r}} \Pi^{\tau} x{:}A.\, B} \qquad \Pi\text{-}\textsc{Elim}\ \frac{\Gamma \vdash M :^{\mathsf{r}} \Pi^{\tau} x{:}A.\, B \qquad \Gamma \vdash N :^{\tau} A}{\Gamma \vdash M @^{\tau} N :^{\mathsf{r}} B[N/x]}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash M :^{\mathsf{r}} A \qquad \Gamma \vdash B :^{\mathsf{c}} s \qquad A =_{\beta} B}{\Gamma \vdash M :^{\mathsf{r}} B} \qquad \textsc{Reset}\ \frac{\Gamma^{\circ} \vdash M :^{\mathsf{r}} A}{\Gamma \vdash M :^{\mathsf{c}} A}$$

**Fig. 2.** Typing rules for EPTS. (The typing rules for PTS may be obtained from these by ignoring all erasure annotations and removing the then useless RESET rule).

types, so the $\mathsf{r}$ is appropriate. Another possible surprise is that $x$ is marked with $\mathsf{r}$ rather than $\tau$ in the typing context of $B$. This is because the binding site of the $x$ will never be erased: The only purpose of the context mark $\mathsf{c}$ is to check erasability of a $\lambda^{\mathsf{c}}$.

If we ignore erasure annotations, these typing rules are exactly those of PTS. The extra restrictions on erasure annotations ensure the following sort of phase distinction: evaluation of any well-typed term never depends on its compile-time portions. We formalize and prove this in Section 5.

**Semantics.** The default operational semantics of EPTS is simply $\beta$-reduction. We do not commit to any particular evaluation order, so the single-step reduction relation is non-deterministic.

Actually this is only one of *two* different operational semantics for EPTS. The remainder of this paper introduces an erasure semantics with potential for more efficient execution.

**Meta-theory.** The top half of Figure 4 depicts the meta-theory of EPTS. Each box in that figure contains a particular result of the meta-theory. As the development follows closely that of Pure Type Systems, we focus on the changes due to introducing erasure annotations.

First we investigate properties of the context reset operation $\Gamma^{\circ}$. It is idempotent (Lemma 1) and weakens the strength of the typing assumptions (Lemma 2). An admissible phase-weakening rule (Corollary 3) follows immediately from Lemma 2. *Proofs:* Lemma 1 is easily proved by induction on $\Gamma$. Lemma 2 is proved by structural induction on the typing derivation. The interesting cases

$\boxed{\Gamma \vdash M : A}$

$$\textsc{Axiom}\ \frac{(s_1, s_2) \in \mathcal{A}}{\vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \qquad \textsc{Weak}\ \frac{\Gamma \vdash A : s \qquad \Gamma \vdash M : B}{\Gamma, x{:}A \vdash M : B}$$

$$\Pi\text{-}\textsc{Form}\ \frac{(s_1, s_2, s_3) \in \mathcal{R} \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.\, B : s_3} \qquad \forall\text{-}\textsc{Form}\ \frac{(s_1, s_2, s_3) \in \mathcal{R} \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \forall x{:}A.\, B : s_3}$$

$$\Pi\text{-}\textsc{Intro}\ \frac{\Gamma \vdash \Pi x{:}A.\, B : s \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.\, M : \Pi x{:}A.\, B} \qquad \forall\text{-}\textsc{Intro}\ \frac{x \notin FV(M) \qquad \Gamma \vdash \forall x{:}A.\, B : s \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash M : \forall x{:}A.\, B}$$

$$\Pi\text{-}\textsc{Elim}\ \frac{\Gamma \vdash M : \Pi x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[N/x]} \qquad \forall\text{-}\textsc{Elim}\ \frac{\Gamma \vdash M : \forall x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M : B[N/x]}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s \qquad A =_\beta B}{\Gamma \vdash M : B}$$

**Fig. 3.** Typing rules for IPTS. Note that ∀-Intro and ∀-Elim are not syntax-directed.

are Reset, where Lemma 1 is used, and Var and Weak, which case split on whether $\Delta$ is empty. Corollary 3 is an immediate consequence of Lemma 2.

Next, we prove the Substitution Lemma (4). Note that the mode $\tau_1$ of the typing judgment for the term $N$ to be substituted must match the context entry mark of the variable $x$ for which it will be substituted. *Proof*: By induction on the typing derivation. The interesting cases are Reset (requiring Corollary 3) and Var and Weak (each proceeding by cases on whether $\Delta = \varepsilon$ or not).

The Coherence Lemma (5) says that our type system is internally coherent in the following way — If it can prove that $M$ has type $A$, then it can also prove that $A$ is a type. *Proof:* By structural induction on the typing derivation. The interesting cases are Reset, which uses Lemma 1, and $\Pi$-Elim, which makes use of Corollary 3 and Lemma 4.

Finally, Subject Reduction (Lemma 6) tells us that evaluation preserves types. Note that the mode $\tau$ of the typing judgment is preserved as well as the type. *Proof:* By structural induction on the typing derivation. The most interesting case is $\Pi$-Elim in which we use Lemma 4.

## 4 Implicit Pure Type Systems

IPTS, the target language of the erasure translation, is an implicitly typed (Curry-style) calculus with both explicit and implicit dependent products. This

calculus is modeled after Miquel's Implicit Calculus of Constructions (ICC) [14,13], a Curry-style variant of Luo's Extended Calculus of Constructions [10]. ICC has a rich notion of subtyping that orders Church encodings at various levels of type refinement in a natural way [13].

IPTS is both more and less general than ICC. It is more general because IPTS is defined in terms of an arbitrary PTS specification. It is less general because ICC (1) uses $\beta\eta$-conversion instead of $\beta$-conversion in determining type equality, (2) supports a notion of universe subtyping called *cumulativity* as in Luo's Extended Calculus of Constructions [10], and (3) contains extra typing rules ensuring $\eta$ subject reduction and strengthening.

The syntax of IPTS is as follows:

$$\begin{array}{ll} \textit{(terms)} & M, N, A, B ::= x \mid \lambda x.\, M \mid M\ N \mid \Pi x{:}A.\, B \mid \forall x{:}A.\, B \mid s \\ \textit{(contexts)} & \Gamma, \Delta ::= \varepsilon \mid \Gamma, x{:}A \end{array}$$

Note the distinction between $\Pi x{:}A.\, B$ and $\forall x{:}A.\, B$ as well as the omission of domain labels from $\lambda$-abstractions.

The difference between explicit and implicit products shows up in the type system (Figure 3). Whereas the explicit product is introduced by functional abstraction (rule $\Pi$-Intro) and eliminated by function application (rule $\Pi$-Elim), no syntactic cues indicate introduction or elimination of the implicit product (rules $\forall$-Intro and $\forall$-Elim). So $\Pi$ indicates functional abstraction and $\forall$ indicates *polymorphism*.

## 5 Erasure

We now define erasure as a translation from EPTS to IPTS.

**Definition (Erasure).** $\boxed{\Gamma^\bullet}$ and $\boxed{M^\bullet}$

$$\varepsilon^\bullet = \varepsilon \qquad (\Gamma, x{:}^\tau A)^\bullet = \Gamma^\bullet, x{:}A^\bullet \qquad x^\bullet = x \qquad s^\bullet = s$$

$$(\Pi^{\mathsf{r}} x{:}A.\, B)^\bullet = \Pi x{:}A^\bullet.\, B^\bullet \qquad (\lambda^{\mathsf{r}} x{:}A.\, M)^\bullet = \lambda x.\, M^\bullet \qquad (M @^{\mathsf{r}} N)^\bullet = M^\bullet\ N^\bullet$$

$$(\Pi^{\mathsf{c}} x{:}A.\, B)^\bullet = \forall x{:}A^\bullet.\, B^\bullet \qquad (\lambda^{\mathsf{c}} x{:}A.\, M)^\bullet = M^\bullet \qquad (M @^{\mathsf{c}} N)^\bullet = M^\bullet$$

The bottom half of Figure 4 sketches out the meta-theory of erasure. We now discuss the significance of the results listed there.

A pair of key lemmas (7 and 8) characterize which variable occurrences in a term may survive erasure: those which are tagged with $\mathsf{r}$ in the typing context. *Proofs:* Lemmas 7 and 8 must be proved simultaneously by structural induction on typing derivations. The Weak case of the proof of Lemma 7 requires Lemma 8 and the $\Pi$-Intro case of the proof of Lemma 8 requires Lemma 7.

**Preservation of Reductions.** Since computation happens by substitution, we first show that erasure commutes with substitution (Lemma 9). We then

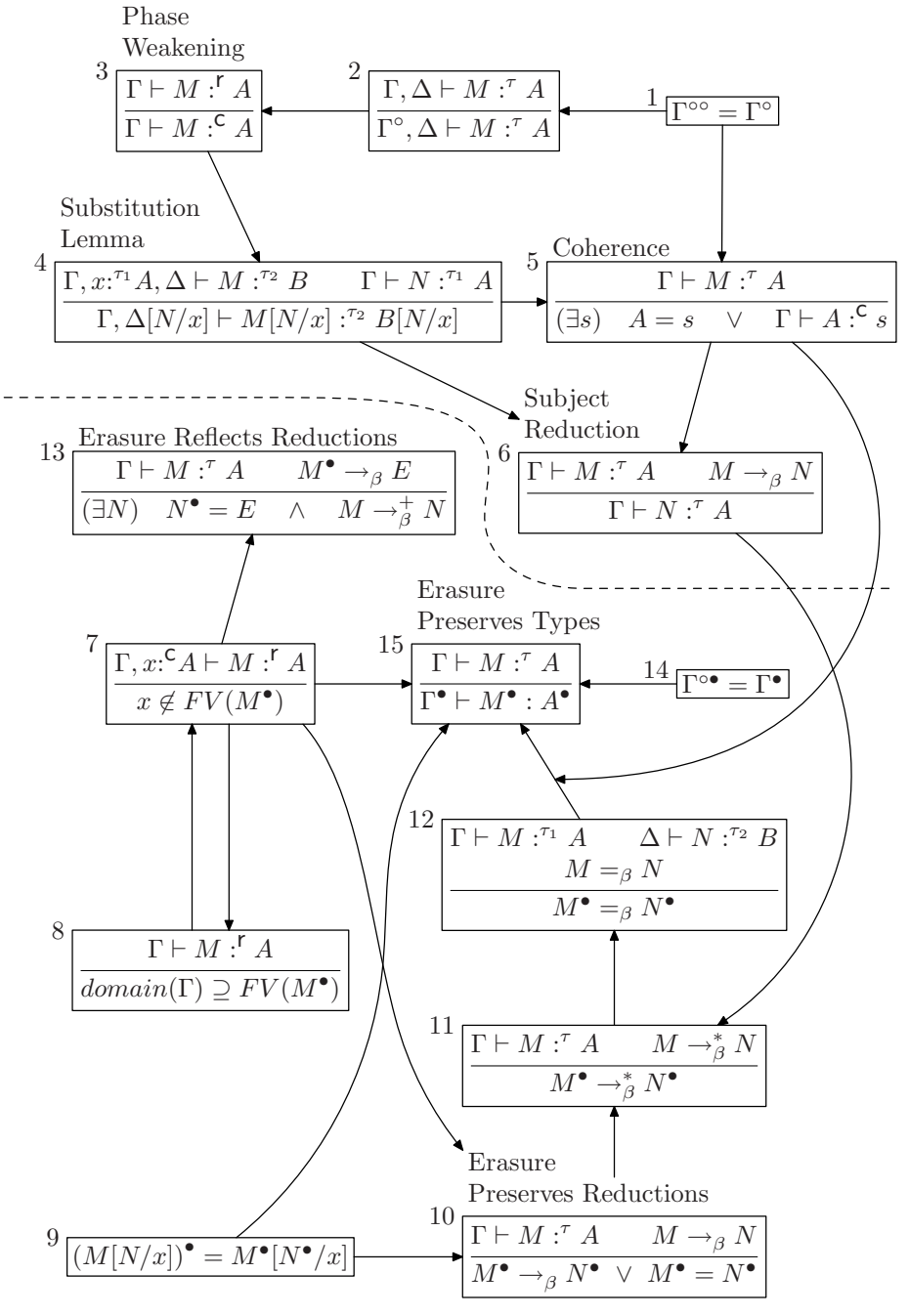



**Fig. 4.** Identities and admissible rules in the meta-theory of EPTS (above the dotted line) and erasure (below it). Arrows indicate proof dependencies.

show that erasure respects reduction in the following sense: Each reduction step of a well-formed term in EPTS maps to either one *or zero* reduction steps in IPTS (Theorem 10). *Proof:* Lemma 9 is proved by straightforward induction on $M$. Theorem 10 is by straightforward induction over the typing derivation. The interesting cases are $\Pi$-Intro and $\Pi$-Elim, where we split by cases on $\tau$. In the $\Pi$-Elim case when the reduction step is $\beta$, the proof depends on Lemma 9 in the case where $\tau = \mathsf{r}$ and on Lemma 7 in the case where $\tau = \mathsf{c}$.

The proof of Theorem 10 shows that some EPTS reductions in fact do no work when viewed through the lens of erasure. This is precisely why we want an erasure semantics — to eliminate the work associated with run-time-irrelevant portions of a program. Examination of the proof shows *where* erasure eliminates work. As expected, the eliminated work includes erased redices (terms of the form $(\lambda^{\mathsf{c}} x{:}A.\, M)@^{\mathsf{c}} N$, which erase to just $M^\bullet$) as well as unnecessary reduction steps inside domain-annotations and erased arguments.

Corollaries 11 and 12 follow immediately from Theorem 10. The proof of Corollary 12 also requires the Church-Rosser Theorem.

**Preservation of Typing.** Again, we first investigate the properties of the context reset operation. The erasure operation annihilates it (Lemma 14). *Proof:* By induction on $\Gamma$.

Then we prove that erasure preserves well-typedness (Theorem 15). *Proof:* We prove Theorem 15 by structural induction on the typing derivation. The interesting cases are: Reset, in which Lemma 14 is used to simplify $\Gamma^{\circ\bullet}$; $\Pi$-Intro, in which Lemma 7 is used to ensure the premise $x \notin FV(M^\bullet)$ of the $\forall$-Intro rule of IPTS; $\Pi$-Elim, in which Lemma 9 is used to simplify the type of the application; and Conv, in which Coherence and Corollary 12 are used to establish the premise $A^\bullet =_\beta B^\bullet$ of the IPTS Conv rule.

**Reflection of Reductions.** Next we show that a reduction of a post-erasure IPTS term can be reflected back into one *or more* EPTS reductions (Theorem 13). *Proof:* By structural induction on the typing derivation. The interesting case is $\Pi$-Elim when the @-annotation is $\tau = \mathsf{r}$ and the reduction is a $\beta$-step $(\lambda x.\, P^\bullet)\ {N_0}^\bullet \to_\beta P^\bullet[{N_0}^\bullet/x]$. In this case, $M = M_0 @^{\mathsf{r}} N_0$ and ${M_0}^\bullet = \lambda x.\, P^\bullet$ and $E = P^\bullet[{N_0}^\bullet/x]$. The only way $M^\bullet$ can be $\lambda x.\, P^\bullet$ is if $M_0$ is a $\lambda^{\mathsf{r}} x{:}B.\, P$ nested under some (perhaps zero) "frames" of the form $\lambda^{\mathsf{c}} y{:}C.\, [\,]$ or $[\,]@^{\mathsf{c}} L$. Because the type of $M_0$ is $\Pi^{\mathsf{r}} x{:}A.\, B$, we know the top-most frame cannot be a $\lambda^{\mathsf{c}}$. Similarly, for typing reasons, the bottom-most frame cannot be a $@^{\mathsf{c}}$, because it is applied to a $\lambda^{\mathsf{r}}$. Therefore, if there are any frames at all on top of $\lambda^{\mathsf{r}} x{:}B.\, P$, then there are at least two, and at some point there is a $\lambda^{\mathsf{c}}$ frame just underneath a $@^{\mathsf{c}}$ one, forming a redex. If we reduce this redex, the rest of the frame structure remains intact. We repeat this process until no intermediate frames are left. Then $M_0 \to_\beta^* \lambda^{\mathsf{r}} x{:}B[\theta].\, P[\theta]$ where $\theta$ is the simultaneous substitution effected by the sequence of reductions. Because $\theta$ is comprised solely of substitutions for $\lambda^{\mathsf{c}}$-bound variables, Lemma 7 tells us there will be no occurrences of these variables inside $P^\bullet$. Therefore $P[\theta]^\bullet = P^\bullet[\theta^\bullet] = P^\bullet$. Let $N = P[\theta][N_0/x]$. Then

$$N^\bullet = P[\theta][N_0/x]^\bullet = P[\theta]^\bullet[{N_0}^\bullet/x] = P^\bullet[{N_0}^\bullet/x] = E$$

and $M \rightarrow_\beta^+ N$ because

$$M = M_0@^{\mathsf{r}}N_0 \rightarrow_\beta^* (\lambda^{\mathsf{r}}x{:}B[\theta].\, P[\theta])@^{\mathsf{r}}N_0 \rightarrow_\beta P[\theta][N_0/x] = N,$$

thereby completing this case of the proof.

The proof of Theorem 13 shows that certain reduction steps in IPTS (of post-erasure EPTS terms) require additional reductions in the original EPTS term before the reduction corresponding to that in IPTS can take place in EPTS. This means that some of the work that erasure avoids is unavoidable, in general, without erasure.

Theorem 13 says that any post-erasure reduction corresponds to some potential pre-erasure reductions. In other words, the erasure of a well-formed EPTS term cannot reduce in IPTS in a strange way that was not possible in EPTS.

## 6 Erasure Semantics

The erasure semantics for EPTS is simply this: First erase and then execute in IPTS. The meta-theory supports the claim that this is a good erasure semantics.

Theorem 10 : erasure eliminates some old work
Theorem 13 : erasure does not introduce any new work
Theorem 15 : erasure preserves the meanings (types) of programs

One final result supports the validity of our erasure semantics for EPTS. We would not want a PTS program to compute to a value while some annotation of it diverges under the erasure semantics. Thankfully, this cannot happen.

**Theorem (Erasure Preserves Strong Normalization)**
*For a strongly normalizing PTS, any well-typed term in the corresponding EPTS erases to a strongly normalizing IPTS term.*

*Proof:* Suppose there is an infinite reduction sequence in IPTS starting with the erasure of a well-typed term $M$ in EPTS. By Theorem 13 and Lemma 6, this reflects back into EPTS as an infinite reduction sequence starting with $M$. Because $\flat$ (the erasure-annotation-forgetting map from EPTS to PTS) preserves both reduction steps and typing judgments, we obtain an infinite reduction sequence in the underlying PTS starting with the well-typed term $M^\flat$. But this contradicts our assumption that the underlying PTS is strongly-normalizing. □

## 7 Implementation

It should be easy to extend an existing type-checker to handle $\forall$-types. One must add $\tau$ annotations to the abstract syntax and some extra logic to the type-checker to handle these annotations properly. The only potential increase in the time complexity of type-checking comes from the context reset operation $\Gamma^\circ$.

However, a clever representation of typing contexts renders reset a constant-time operation. The new representation of typing contexts is as follows:

$$\Gamma ::= (\hat{\Gamma}\,;i) \qquad\qquad \hat{\Gamma} ::= \hat{\varepsilon} \mid \hat{\Gamma}, x{:}^{i} A$$

(where $i$ denotes an integer). The context operations then become

$$\varepsilon = (\hat{\varepsilon}\,;0) \qquad \begin{array}{l}(\hat{\Gamma}\,;i), x{:}^{\mathsf{c}} A = (\hat{\Gamma}, x{:}^{i+1} A\,;\, i)\\ (\hat{\Gamma}\,;i), x{:}^{\mathsf{r}} A = (\hat{\Gamma}, x{:}^{i} A\,;\quad i)\end{array} \qquad (\hat{\Gamma}\,;i)^{\circ} = (\hat{\Gamma}\,;i+1)$$

$$x{:}^{\mathsf{r}} A \in (\hat{\Gamma}\,;i) \text{ iff } x{:}^{j} A \in \hat{\Gamma} \text{ and } j \leq i$$

The top-level $i$ in $\Gamma = (\hat{\Gamma}\,;i)$ counts how many times prefixes of $\Gamma$ have been reset. For any binding $x{:}^{j} A \in \hat{\Gamma}$ originally introduced with the mark $\tau$, we have $j > i$ iff (1) $\tau = \mathsf{c}$ and (2) there have been no resets since $x$ was introduced — exactly the condition in which the binding for $x$ would be marked with $\mathsf{c}$ in the original implementation. The implementation of $x{:}^{\mathsf{r}} A \in \Gamma$ is therefore correct.

## 8 Future Work

### 8.1 Proof Irrelevance

In a dependently typed language, the conversion typing rule reflects the semantics of a language back into its type system. In EPTS, however, there are two notions of operational semantics. The Conv rule of EPTS reflects the default semantics rather than the erasure semantics. We may attempt to remedy this by modifying the Conv rule as follows:

$$\frac{\Gamma \vdash M :^{\mathsf{r}} A \qquad \Gamma \vdash B :^{\mathsf{c}} s \qquad A^{\bullet} =_{\beta} B^{\bullet}}{\Gamma \vdash M :^{\mathsf{r}} B}\ \textsc{Conv}^{\bullet}$$

This variant of EPTS is interesting because the Conv$^{\bullet}$ rule seems to yield a generalized form of *irrelevance*, including *proof irrelevance* as a special case. Proof irrelevance in a conversion rule means that two proofs are considered equal if they prove the same proposition, regardless of how they each prove it. The Conv$^{\bullet}$ rule only requires the run-time portions of $A$ and $B$ to be equal — compile-time portions of $A$ and $B$ (including proofs and perhaps other terms) are considered irrelevant.

Pfenning's modal variant of LF with built-in notions of intensional code and proof irrelevance [17] provided inspiration for EPTS. The conversion rule of that system seems to us quite similar to Conv$^{\bullet}$.

### 8.2 Parametricity

Languages such as Haskell and ML make heavy use of parametric polymorphism centered around a $\forall$ type constructor. *Parametricity* is a property of such languages enabling one to derive "free theorems" about polymorphic terms based

solely on their type [21]. We conjecture that the ∀-types of EPTS satisfy parametricity properties similar to those of System F.

Many studies of parametricity for System F are based on denotational semantics. It seems impossible to develop a semantic model for an arbitrary EPTS. We think a proof-theoretic approach is necessary, somehow generalizing existing work on System F [18,2,11].

## 9 Conclusions

Languages combining dependent types with erasure semantics sometimes require users to maintain more than one copy of a datatype in order to ensure erasure of some of its values but not others. This problem stems from the treatment of erasability as an intrinsic property of data, rather than a property of the way that data is used.

By treating erasability extrinsically — distinguishing functions that don't depend computationally on their arguments from those that do — we overcome the code duplication problem and arrive at a general form of polymorphism over arbitrary sorts of entities (types, proofs, numbers, etcetera).

This change of perspective leads to a notion of erasure generalizing both type-erasure and proof-erasure (program-extraction). We hope the resulting notion of computational irrelevance similarly generalizes both proof-irrelevance and parametricity-style notions of representation independence.

**Acknowledgments.** Thanks to Andrew Tolmach, Mark Jones, Jim Hook, Tom Harke, Chuan-Kai Lin, Ki-Yung Ahn, Andrew McCreight, Dan Brown, and John McCall for their comments on this work. This work is supported by the National Science Foundation (Grants Nos. CCF-0541447 and CCF-0613969).

## References

1. The Coq proof assistant, http://coq.inria.fr
2. Abadi, M., Cardelli, L., Curien, P.-L.: Formal parametric polymorphism. Theoretical Computer Science 121(1–2), 9–58 (1993)
3. Augustsson, L.: Cayenne – A language with dependent types. In: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming, pp. 239–250 (1998)
4. Barendregt, H.P.: Lambda calculi with types. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 2, Oxford University Press, Oxford (1992)
5. Blazy, S., Dargaye, Z., Leroy, X.: Formal verification of a C compiler front-end. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 460–475. Springer, Heidelberg (2006)
6. Brady, E.: Practical Implementation of a Dependently Typed Functional Programming Language. PhD thesis, University of Durham (2005)
7. Chen, C., Xi, H.: Combining programming with theorem proving. In: Proceedings of the Tenth ACM SIGPLAN International Conference on Functional Programming, pp. 66–77 (2005)

8. Leroy, X.: Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In: Proceedings of the 33rd ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 42–54 (2006)
9. Lin, C., McCreight, A., Shao, Z., Chen, Y., Guo, Y.: Foundational typed assembly language with certified garbage collection. In: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, pp. 326–338. IEEE Computer Society Press, Los Alamitos (2007)
10. Luo, Z.: Computation and reasoning: A type theory for computer science. Oxford University Press, New York, USA (1994)
11. Mairson, H.G.: Outline of a proof theory of parametricity. In: Hughes, J. (ed.) FPCA 1991. LNCS, vol. 523, pp. 313–327. Springer, Heidelberg (1991)
12. McBride, C., McKinna, J.: The view from the left. Journal of Functional Programming 14(1), 69–111 (2004)
13. Miquel, A.: The implicit calculus of constructions. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 344–359. Springer, Heidelberg (2001)
14. Miquel, A.: Le Calcul des Constructions Implicite: Syntaxe et Sémantique. PhD thesis, Université Paris 7 (2001)
15. Necula, G.C.: Proof-carrying code. In: Proceedings of the 24th ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 106–119 (1997)
16. Peyton-Jones, S., Vytiniotis, D., Weirich, S., Washburn, G.: Simple unification-based type inference for GADTs. In: Proceedings of the Eleventh ACM SIGPLAN International Conference on Functional Programming (2006)
17. Pfenning, F.: Intensionality, extensionality, and proof irrelevance in modal type theory. In: LICS 2001: Proceedings of the 16th Annual Symposium on Logic in Computer Science, pp. 221–230. IEEE Computer Society Press, Los Alamitos (2001)
18. Plotkin, G.D., Abadi, M.: A logic for parametric polymorphism. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 361–375. Springer, Heidelberg (1993)
19. Sheard, T.: Languages of the future. In: Proceedings of the Nineteenth ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA Companion Volume, pp. 116–119 (2004)
20. Sheard, T., Pašalić, E.: Meta-programming with built-in type equality. In: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages (LFM 2004), pp. 106–124 (2004), http://cs-www.cs.yale.edu/homes/carsten/lfm04/
21. Wadler, P.: Theorems for free! In: Functional Programming Languages and Computer Architecture, pp. 347–359. ACM Press, New York (1989)
22. Xi, H., Pfenning, F.: Dependent types in practical programming. In: Proceedings of the 26th ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 214–227 (1999)

# The Implicit Calculus of Constructions as a Programming Language with Dependent Types

Bruno Barras and Bruno Bernardo

INRIA Futurs and Ecole polytechnique, France
{Bruno.Barras,Bruno.Bernardo}@lix.polytechnique.fr

**Abstract.** In this paper, we show how Miquel's Implicit Calculus of Constructions (ICC) can be used as a programming language featuring dependent types. Since this system has an undecidable type-checking, we introduce a more verbose variant, called ICC* which fixes this issue. Datatypes and program specifications are enriched with logical assertions (such as preconditions, postconditions, invariants) and programs are decorated with proofs of those assertions. The point of using ICC* rather than the Calculus of Constructions (the core formalism of the Coq proof assistant) is that all of the static information (types and proof objects) is transparent, in the sense that it does not affect the computational behavior. This is concretized by a built-in extraction procedure that removes this static information. We also illustrate the main features of ICC* on classical examples of dependently typed programs.

## 1 Introduction

In software verification, typing disciplines have shown to be a decisive step towards safer programs. The success of strongly typed functional languages of the ML family is an evidence of that claim. Still, in those systems, typing is not expressive enough to address problems such as array bound checks.

Such issue can be alleviated by using dependent types. Dependent ML [19] is an extension of SML implementing a restricted form of dependent types. The idea is to annotate datatype specifications and program types with expressions in a given constraint domain. Type-checking generate constraints whose satisfiability is checked automatically. But it is limited to decidable constraint domains since the programmer is not allowed to help the type-checker by providing the proof of the satifiability of constraints. The main point of having restricted dependent types is that it applies to programming languages with non-pure features (side-effects, input/output,...).

The system ATS [4] is an evolution of DML that integrates theorem proving in the LF style in case the automatic solver fails. It lets the programmer prove simple invariants (there is very little support for proof construction). Proofs systems (let us name a small number of them: Epigram [8], Agda [13], NuPRL [5] and Coq [17]) provide better tools for proof automation, but in most of them, the distinction between statics (logical and typing arguments) and dynamics (actual code) raises problems.

To illustrate this claim, let us recall one typical example in programming with dependent types: vectors. In order to statically check that programs never access a vector out of its bounds, its type is decorated with an integer representing its length. This can lead

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 365–379, 2008.

to more efficient programs since no runtime check is necessary. It is also safer since it is known statically that such program never raises an exception nor returns dummy values.

In the Calculus of Constructions (CC, the core formalism of Coq), one defines a type `vect` parameterized by a natural number and two constructors `nil` and `cons`. $\texttt{vect}\ n$ is the type of vectors of length $n$ ($A$ is the type of the elements).

$$\begin{aligned}\texttt{vect} &: \texttt{nat} \to \mathsf{Set}\\ \texttt{nil} &: \texttt{vect}\ 0\\ \texttt{cons} &: \Pi(n{:}\texttt{nat}).\, A \to \texttt{vect}\ n \to \texttt{vect}\ (S\ n)\end{aligned}$$

For instance the list $[x_1; x_2; x_3]$ is represented as $(\texttt{cons}\ 2\ x_1\ (\texttt{cons}\ 1\ x_2\ (\texttt{cons}\ 0\ x_3\ \mathit{nil})))$. In fact, the first argument of `cons` is not intended to be part of the data structure: it is used only for type-checking purposes. The safe access function can be specified as

$$\texttt{get} : \Pi(n{:}\texttt{nat})\,(i{:}\texttt{nat}).\, \texttt{vect}\ n \to (i < n) \to A,$$

That is, `get` is a function taking as argument a vector size $n$, the accessed index $i$, a vector of the specified size and a proof that $i$ is within the bounds of the vector. Here again, we have two arguments ($n$ and the proof) that do not participate in computing the result, but merely help type-checking.

We can see that programming in the Calculus of Constructions is quite coarse: programs have arguments that are indeed only static information (type decorations, proof objects, dependencies). There exists a procedure called extraction (described in [6]) that produces source code for a number of functional languages from intuitionistic proofs. It tries to remove this static information. The decision of keeping or removing an argument is made by the user when he defines a new type. For this purpose, Coq considers two sorts (the types of types) Prop and Set that are almost identical regarding typing, but types of Prop are intended to be logical propositions (like $i < n$), while types of Set are the actual datatypes (like `nat` and `vect`). In the example, the proof argument of `get` would be erased, but the length $n$ would not. This issue could be alleviated by doing a dead-code analysis [15], but it would not allow the user to specify *a priori* arguments that shall not be used in the algorithmic part of the proof. Another drawback of the extraction approach is that it is external to the system. This means that within the logic, the programmer deals with the fully decorated term. In situations where datatypes carry proofs to guarantee invariants, two datastructures may be equal, but containing different proofs. Since there is no proof-irrelevance, such objects cannot be proven equal in spite of having the same runtime counterparts.

The Implicit Calculus of Constructions (ICC, see [10] and [11]) offers a more satisfying alternative to the distinction between Prop and Set. It is a Curry-style presentation of the Calculus of Constructions.[1] It features a so-called implicit product that corresponds to an intersection type rather than a function type. The main drawback of this system is the undecidability of type-checking. This is mainly because terms do not carry the arbitrarily complex proofs. This means that programs do not carry enough information to recheck them, which is a problem from an implementation point of view. Proving

[1] Type Assignment Systems are also type systems in Curry style, but the usage of the polymorphic quantification is too restrictive for our purposes.

a skeptical third party that a program is correctly typed requires communicating the full derivation.

The main idea of this paper is to introduce a more verbose variant of ICC such that typing is decidable. This is made by decorating terms with the implicit information. The challenge is to ensure that this information does not get in the way, despite the fact that it must be maintained consistent when evaluating a term.

The paper is organized as follows: we define a new calculus (ICC$^*$), and show its main metatheoretical properties. Consistency is established thanks to a sound and complete translation towards a subset of ICC called ICC$^-$. We also introduce a reduction on decorated terms that enjoys subject-reduction, meaning that it maintains decorations consistent. Then we revisit some classical examples and show that a significative part of the expressiveness of ICC is captured. We also discuss several extensions that would make the system more akin to be a pratical programming language.

## 2 A Decidable Implicit Calculus of Constructions

### 2.1 Syntax

Its syntax (see Figure 1) is the same that in the standard Calculus of Constructions in Church style, except that we duplicate each operation (product, abstraction and application) into an explicit one and an implicit one. As often in Type Theory, terms and types belong to the same syntactic class, and special constants called sorts represent the types of types.

Sorts $(\mathsf{Type}_i)_{i\in\mathbb{N}}$ denote the usual predicative universe hierarchy of the Extended Calculus of Constructions [7]. There is only one impredicative sort, $\mathsf{Prop}$, because the distinction between propositional types and data types will be made by defining a term as being explicit (data types) or implicit (propositional types) instead of giving it a type of type $\mathsf{Set}$ or $\mathsf{Prop}$.

| | |
|---|---|
| **Sorts** | $s ::= \mathsf{Prop} \mid \mathsf{Type}_i \ (i \in \mathbb{N})$ |
| **Terms** | $M ::= x \mid s$ |
| | $\mid \Pi(x:M_1).\, M_2 \mid \lambda(x:M_1).\, M_2 \mid M_1\ M_2 \ \ (explicit)$ |
| | $\mid \Pi[x:M_1].\, M_2 \mid \lambda[x:M_1].\, M_2 \mid M_1[M_2] \ \ (implicit)$ |
| **Contexts** | $\Gamma ::= [] \mid \Gamma; x : M$ |

**Fig. 1.** Syntax of ICC$^*$

As usual we consider terms up to $\alpha$-conversion. The set of free variables of term $t$ is written $\mathsf{FV}(t)$. Arrow types are explicit non-dependent products (if $x \notin \mathsf{FV}(U)$, we write $T \to U$ for $\Pi(x{:}T).\, U$). Substitution of the free occurrences of variable $x$ by $N$ in term $M$ is noted $M\{x/N\}$. We write $\mathsf{DV}(\Gamma)$ the set of variables $x$ that are declared in $\Gamma$, *i.e.* such that $(x : T) \in \Gamma$ for some term $T$.

### 2.2 Extraction

We define inductively (see Fig.2) an *extraction* function $M \mapsto M^*$ that associates a term of ICC to every term of our calculus. This function removes the static information: domains of abstractions, implicit arguments and implicit abstractions. Beware that extraction does not preserve $\alpha$-conversion for any term. So, many properties of extraction will hold only for well-typed terms[2].

**ICC Terms** $M ::= x \mid s \mid \Pi(x{:}M_1).\,M_2 \mid \forall(x{:}M_1).M_2 \mid \lambda x.M \mid M_1\,M_2$
*(ICC has the same set of sorts $\mathcal{S}$ as ICC$^*$)*

**Extraction**

$$\begin{array}{rlrl} s^* &= s & x^* &= x \\ (\Pi(x{:}T).\,U)^* &= \Pi(x{:}T^*).\,U^* & (\Pi[x{:}T].\,U)^* &= \forall(x{:}T^*).U^* \\ (\lambda(x{:}T).\,U)^* &= \lambda x.U^* & (\lambda[x{:}T].\,U)^* &= U^* \\ (MN)^* &= M^*N^* & (M[N])^* &= M^* \end{array}$$

**Fig. 2.** ICC terms and Extraction

### 2.3 Typing Rules

For any relation $R$ in ICC or in ICC$^*$, we will use the symbols $\rhd_R$, $\rightarrow_R$, $\rightarrow_R^*$, $\rightarrow_R^+$ and $\cong_R$ to denote the relation itself, its congruence closure, the reflexive and transitive closure of $\rightarrow_R$, the transitive closure of $\rightarrow_R$ and the reflexive, symmetric and transitive closure of $\rightarrow_R$. $\rightarrow_R^h$ will denote the head reduction of $\rhd_R$ - reduction occurs in the left subterm of applications (implicit or explicit applications in the case of decorated terms).

As in the traditional presentation of Pure Type Systems [1], we define two sets **Axiom** $\subset \mathcal{S}^2$ and **Rule** $\subset \mathcal{S}^3$ by

$$\begin{array}{rl} \textbf{Axiom} = & \{(\mathsf{Prop}, \mathsf{Type}_0);\ (\mathsf{Type}_i, \mathsf{Type}_{i+1}) \mid i \in \mathbb{N}\} \\ \textbf{Rule} = & \{(\mathsf{Prop}, s, s);\ (s, \mathsf{Prop}, \mathsf{Prop}) \mid s \in \mathcal{S}\} \\ & \cup\{(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{max(i,j)}) \mid i, j, \in \mathbb{N}\} \end{array}$$

We will also consider these two judgements:

- the judgement $\Gamma \vdash$ that means "the context $\Gamma$ is well-formed"
- the judgement $\Gamma \vdash M : T$ that means "under the context $\Gamma$, the term $M$ has type $T$". By convention, we will implicitly $\alpha$-convert $M$ in order that $\mathsf{DV}(\Gamma)$ and the set of bound variables of $M$ are disjoint.

**Definition 1 (Typing judgements).** *They are defined in Fig. 3*

They are very similar to the rules of a standard Calculus of Constructions where product, abstraction and application are duplicated into explicit and implicit ones. There are though two important differences:

[2] For instance, $(\lambda[x{:}T].\,x)^*$ depends on the name of the binder.

$$\frac{}{[] \vdash} (\textsc{Wf-E}) \qquad \frac{\Gamma \vdash T : s \quad x \notin \mathsf{DV}(\Gamma)}{\Gamma; x : T \vdash} (\textsc{Wf-S})$$

$$\frac{\Gamma \vdash \quad (s_1, s_2) \in \textbf{Axiom}}{\Gamma \vdash s_1 : s_2} (\textsc{Sort}) \qquad \frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T} (\textsc{Var})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi(x{:}T).\, U : s_3} (\textsc{E-Prod})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi[x{:}T].\, U : s_3} (\textsc{I-Prod})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \Pi(x{:}T).\, U : s}{\Gamma \vdash \lambda(x{:}T).\, M : \Pi(x{:}T).\, U} (\textsc{E-Lam})$$

$$\frac{\Gamma; x : T \vdash M : U \quad \Gamma \vdash \Pi[x{:}T].\, U : s \quad x \notin \mathsf{FV}(M^*)}{\Gamma \vdash \lambda[x{:}T].\, M : \Pi[x{:}T].\, U} (\textbf{I-Lam})$$

$$\frac{\Gamma \vdash M : \Pi(x{:}T).\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M\, N : U\{x/N\}} (\textsc{E-App}) \qquad \frac{\Gamma \vdash M : \Pi[x{:}T].\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M[N] : U\{x/N\}} (\textsc{I-App})$$

$$\frac{\Gamma \vdash M : T \quad \Gamma \vdash T' : s \quad T^* \cong_{\beta\eta} T'^*}{\Gamma \vdash M : T'} (\textsc{Conv})$$

**Fig. 3.** Typing rules of ICC*

- in the **(I-Lam)** rule, we add the condition $x \notin \mathsf{FV}(M^*)$ (variables introduced by an implicit abstraction cannot appear in the extraction of the body), so $x$ is not used during the computation. This excludes meaningless terms like $\lambda[x{:}T].\, x$.
- In the **(Conv)** rule we replace the usual conversion by the conversion of extracted terms. This makes it clear that the denotation of objects do not depend on implicit information.

This last modification completely changes the semantics of the formalism. Despite of being apparently very close to the Calculus of Constructions, it is in fact semantically much closer to ICC (usual models of CC do not validate the (Conv) rule).

Before developing the metatheory of our system, we shall make the subset of ICC we are targeting more precise.

**Definition 2 (Typing rules of ICC$^-$).** *See Fig. 4.*

In comparison to ICC as presented in [10], we made the following restrictions:

- we removed the rules related to subtyping (rules Cum and Ext), to make things simpler, but we consider extending our system with these rules (or equivalent ones);
- we also removed the context strengthening rule (Str) for quite different reasons. In our formalism, non-dependent implicit products are intended to encode preconditions of a program: a proof of $\Pi[\_ : P].\, Q$ yields a program with specification

$$\frac{}{[] \vdash_{\text{ICC}^-}} (\text{WF-E}) \qquad \frac{\Gamma \vdash_{\text{ICC}^-} T : s \quad x \notin \mathsf{DV}(\Gamma)}{\Gamma; x{:}T \vdash_{\text{ICC}^-}} (\text{WF-S})$$

$$\frac{\Gamma \vdash_{\text{ICC}^-} \quad (s_1, s_2) \in \textbf{Axiom}}{\Gamma \vdash_{\text{ICC}^-} s_1 : s_2} (\text{Sort}) \qquad \frac{\Gamma \vdash_{\text{ICC}^-} \quad (x : T) \in \Gamma}{\Gamma \vdash_{\text{ICC}^-} x : T} (\text{Var})$$

$$\frac{\Gamma \vdash_{\text{ICC}^-} T : s_1 \quad \Gamma; x{:}T \vdash_{\text{ICC}^-} U : s_2 \quad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash_{\text{ICC}^-} \Pi(x{:}T).U : s_3 \qquad \Gamma \vdash_{\text{ICC}^-} \forall(x{:}T).U : s_3} (\text{ExpProd})\&(\text{ImpProd})$$

$$\frac{\Gamma; x{:}T \vdash_{\text{ICC}^-} M : U \quad \Gamma \vdash_{\text{ICC}^-} \Pi(x{:}T).U : s}{\Gamma \vdash_{\text{ICC}^-} \lambda x.M : \Pi(x{:}T).U} (\text{Lam})$$

$$\frac{\Gamma \vdash_{\text{ICC}^-} M : \Pi(x{:}T).U \quad \Gamma \vdash_{\text{ICC}^-} N : T}{\Gamma \vdash_{\text{ICC}^-} M\,N : U\{x/N\}} (\text{App})$$

$$\frac{\Gamma; x{:}T \vdash_{\text{ICC}^-} M : U \quad \Gamma \vdash_{\text{ICC}^-} \forall(x{:}T).U : s \quad x \notin \mathsf{FV}(M)}{\Gamma \vdash_{\text{ICC}^-} M : \forall(x{:}T).U} (\text{Gen})$$

$$\frac{\Gamma \vdash_{\text{ICC}^-} M : \forall(x{:}T).U \quad \Gamma \vdash_{\text{ICC}^-} N : T}{\Gamma \vdash_{\text{ICC}^-} M : U\{x/N\}} (\text{Inst})$$

$$\frac{\Gamma \vdash_{\text{ICC}^-} M : T \quad \Gamma \vdash_{\text{ICC}^-} T' : s \quad T \cong_{\beta\eta} T'}{\Gamma \vdash_{\text{ICC}^-} M : T'} (\text{Conv})$$

**Fig. 4.** Typing rules of $\text{ICC}^-$

$Q$ provided you can produce a proof of $P$; but the strengthening rule makes this type equivalent to $Q$. So this rule would not require a proof of $P$ prior to using a program with specification $Q$.

## 3 Metatheory

Unlike ICC, the basic metatheory can be proven just like for PTSs (see for instance [1]). This is due to the fact that it relies heavily on the form of the typing rules, but very little on the nature of conversion. We first prove inversion lemmas. They allow us to characterize the type $R$ of judgment $\Gamma \vdash M : R$ according to the nature of the term $M$. Other important properties are substitutivity and context conversion (if $\Gamma \vdash M : T$, $\Delta \vdash$ and $\Gamma^* \cong_{\beta\eta} \Delta^*$ hold, then we have $\Delta \vdash M : T$).

### 3.1 Preservation of the Theory and Consistency

In this section, we prove that $\text{ICC}^*$ is consistent and that it is equivalent to $\text{ICC}^-$ (any derivation in $\text{ICC}^*$ has a counterpart in $\text{ICC}^-$ and vice versa). First, we prove by mutual structural induction that any derivation of $\text{ICC}^*$ can be mapped into a derivation in $\text{ICC}^-$:

**Proposition 1 (Soundness of extraction)**

$$\begin{array}{ll} (i) & \Gamma \vdash \quad \Rightarrow \quad \Gamma^* \vdash_{ICC^-} \\ (ii) & \Gamma \vdash M : T \Rightarrow \Gamma^* \vdash_{ICC^-} M^* : T^* \end{array}$$

Consistency of ICC$^*$ is an easy consequence of consistency of ICC$^-$ [11] and of Proposition 1.

**Proposition 2 (Consistency).** *There is no proof of the absurd proposition. There exists no term $M$ such that the following judgment is derivable:*

$$[] \vdash M : \Pi(A{:}\mathsf{Prop}).\, A.$$

If we cannot prove the completeness results for the full ICC (subtyping and strengthening are not derivable), we can still prove it for the restricted system ICC$^-$ using mutual structural induction and context conversion.

**Proposition 3 (Completeness of extraction)**

1. *For any judgement $\Gamma \vdash_{ICC^-}$ there exists a context $\Delta$ in ICC$^*$ such that $\Delta^* = \Gamma \ \wedge \ \Delta \vdash$.*
2. *For any judgement $\Gamma \vdash_{ICC^-} M : T$ there exists $\Delta$, $N$ and $U$ such that $\Delta \vdash N : U \ \wedge \ \Delta^* = \Gamma \ \wedge \ N^* = M \ \wedge \ U^* = T$*

Note that since we have completeness for a subset of ICC rules and not for ICC itself, we cannot deduce properties of ICC$^*$ from properties of ICC.

### 3.2 Decidability of Type Inference

As usual, decidability of type-checking requires the decidability of type inference. In our case, we can consider two kinds of type inference: inferring a decorated term or a term of ICC$^-$. The latter is enough for decidability of type-checking, but for implementation purposes, it is desirable to have the former. We want then to prove the following:

**Proposition 4 (Decidability of type inference).** *There exists a sound and complete algorithm that receives as an input a well-founded context $\Gamma$ and a term $M$ and returns a decorated term $T$ such that $\Gamma \vdash M : T$ if there exists one and* `false` *otherwise.*

*Proof.* The cases of variable and sort are trivial. For products and abstractions we can conclude easily because ICC is strongly normalizing (see [11]) and because we can infer a sort $s$ from $s^*(= s)$.

For applications it is much trickier because we need precise information to infer the type. If we consider *e.g.* an implicit application $M[N]$, infering the type of $M$ and $N$ is not enough. If we know that $M$ has type $T'$, we have to reach, if it exists, to the *annotated* term $U$ such that $T'^* \cong_{\beta\eta} \forall(x : T).U^*$. The problem is that we only have access to the extracted term $U^*$.

In order to solve this we need to introduce some reduction rules to ICC$^*$ terms. The key point is that we only need to do head reduction : if $T'$ is reduced to a product, this product can let us infer the type of the application. Since for well-typed terms, we

cannot have $(\lambda(x:T).\,Mx) \rhd_\eta M$ with $M$ being a product, $\eta$-reduction rules are not useful and thus we only introduce $\beta$-reduction rules.

These rules and two basic properties are presented in Fig.5. Note that these rules are not necessary for the definition of our calculus. We only need them to infer types.

We also have a completeness result saying that, given $M$ a well-typed term of ICC$^*$ and $N'$ a term of ICC such that $M^* \to_\beta N'$, there exists a term $N$ of ICC$^*$ such that $N^* = N'$ and $M \to^+_{\beta_{ie}} N$. It is proved as in Lemma 3.4 of [9] except that we use the fact that every well-typed term has a $\beta_i$-weak head normal form (WHNF) - instead of a $\beta_i$-normal form in [9].

The last result that we need is the existence of a $\beta_{ie}$-WHNF for every well-typed term of ICC$^*$. This is a consequence of the soundness and completeness of $\beta_{ie}$-reduction and of the existence of a $\beta_i$-WHNF for every well-typed term of ICC$^*$.

We can now infer the type of an application $MN$ or $M[N]$. We compute the $\beta_{ie}$-WHNF of the type of $M$. If it is a product then $M$ is typed by this product (soundness and subject reduction of $\beta_{ie}$) and we can infer the type of the application. If not, completeness informs us that the application is not well-typed.

| | | | | |
|---|---|---|---|---|
| **Definitions** | $(\lambda(x:T).\,M)\,N$ | $\rhd_{\beta_e}$ | $M\{x/N\}$ | **(explicit $\beta$-reduction)** |
| | $(\lambda[x:T].\,M)\,[N]$ | $\rhd_{\beta_i}$ | $M\{x/N\}$ | **(implicit $\beta$-reduction)** |
| | | $\rhd_{\beta_{ie}}$ $=$ | $\rhd_{\beta_e} \cup \rhd_{\beta_i}$ | **($\beta_{ie}$-reduction)** |
| **Properties** | $(\Gamma \vdash M : T) \wedge (M \to_{\beta_{ie}} N)$ | $\Rightarrow$ | $M^* \to^*_\beta N^*$ | **(Soundness)** |
| | $(\Gamma \vdash M : T) \wedge (M \to^*_{\beta_{ie}} M')$ | $\Rightarrow$ | $\Gamma \vdash M' : T$ | **(Subject Reduction)** |

**Fig. 5.** $\beta_{ie}$-reduction in ICC$^*$

## 4 Implementation and Inductive Types

Our long term goal is to design a formalism dedicated to software verification. To experiment on examples, we have developped a clone of Coq that implements ICC$^*$.[3] Despite the fact that the formalism is quite different from the one implemented by Coq (the underlying models are completely different), the cost of its implementation was amazingly low: the type of terms has been slightly changed (to duplicate product, abstraction and application into implicit/explicit pairs). But then the code is adapted straightforwardly since the type-checking algorithm is modular w.r.t. conversion.

Our prototype inherits inductive types from Coq, but this is considered an experimental feature. We proceeded by analogy with their impredicative encoding. Of course, inductive types are more expressive: strong and dependent elimination schemes cannot be derived with the impredicative encodings. The most challenging task part of justifying our implementation of inductive types is probably to introduce an implicit sigma type $\Sigma[x:A].\,B_x$ that should be interpreted as a union of the $B_x$ family of types. It is not clear how to define union of types in Miquel's model.

[3] This implementation is available as a *Darcs* repository at
`http://www.lix.polytechnique.fr/Labo/Bruno.Barras/coq-implicit`

In the following examples, we will rather use impredicative encodings and axioms that correspond to derivable properties of inductive types. Among those are:

- the strong elimination of absurdity $\texttt{False\_elim} : \Pi[P:\mathsf{Type}].\,[\texttt{False}] \to P$,[4] which is used to mark parts of a program as dead code,
- the strong elimination of equality proofs $\texttt{eq\_elim} : \Pi[A]\,[x]\,[y]\,[P:A \to \mathsf{Type}].\,[x = y] \to P\,x \to P\,y$, which is used to "cast" an expression of type $P\,x$ with type $P\,y$ whenever we can prove $x = y$,
- and a strong elimination of accessibility proofs that would allow building recursive functions whose termination is justified by a logical (implicit) proof of well-foundedness, but this is not used in this work.

The former is easy to validate since the denotation of `False` is empty, $[\texttt{False}] \to P$ is the full interpretation domain. `eq_elim` can be interpreted by the denotation of $\lambda x.\,x$: if $x$ and $y$ are equal (in $A$), then a predicate (of domain $A$) cannot distinguish them, so $P\,x$ and $P\,y$ are the same type, and finally the identity is actually a mapping from $P\,x$ to $P\,y$.[5]

In the implementation, the above properties result from two principles: (dependent) pattern-matching on the proof object (which would require this proof to be present at runtime), and the following axiom:

$$\texttt{impl\_PI} : \Pi[P:\mathsf{Prop}].\,[P] \to P.$$

Informally it says that there exists an object that belongs to any provable proposition, which holds in the model. This is weaker than the usual axiom of proof irrelevence which says that there exists only one proof object. Thanks to that axiom, logical arguments of programs can always be made implicit (as for `False_elim` and `eq_elim`) and thus never compared. Note that this was an important motivation for considering a proof irrelevant Calculus of Constructions [18].

One might want not just axiom `eq_elim`, but also a reduction associated to it (witnessing the fact that it can be interpreted by the identity). Unfortunately we see no way to do this since this axiom should decide if reduction is possible without any information about $x$, $y$ or the proof of $x = y$ (they are all implicit objects). Nonetheless, it is possible to add a variant of Streicher's K axiom (justified by the model):

$$\Pi[A]\,[x]\,[P]\,[e:x = x][a:P\,x].\,\texttt{eq\_elim}\,[A]\,[x]\,[x]\,[P]\,[e]\,a = a.$$

## 5 Examples

In this section, we develop several examples that illustrate the features of our formalism. We first show how preconditions and postconditions can be encoded in ICC$^*$ by defining a simple division algorithm. Then, we define concatenation and head of vectors, the latter illustrate how to deal with absurd cases. Such examples are direct translations

[4] We do not to write type of variables when it can be easily inferred from the context, and notation $[A] \to B$ stands for $\Pi[\_:A].\,B$.

[5] Thanks to Alexandre Miquel for pointing out this fact.

of what could already be done in the Calculus of Constructions, but our claim is that ICC provides a better framework. Next examples show features that depart from CC: PVS' predicate subtyping can be encoded faithfully, and under some circumstances, datastructures carrying dependencies do not have to be copied.

### 5.1 Euclidean Division

This example illustrates how to encode programs with preconditions and postconditions. Euclidean division can be expressed as a primitive recursive function, following this informal algorithm:

$$\begin{array}{l}\mathtt{div}\ a\ b := \mathtt{if}\ a = 0\ \mathtt{then}\ (0,0)\ \mathtt{else}\\ \qquad\qquad \mathtt{let}\ (q,r) := \mathtt{div}\ (a-1)\ b \mathtt{in}\\ \qquad\qquad \mathtt{if}\ r = b-1\ \mathtt{then}\ (q+1,0)\ \mathtt{else}\ (q,r+1)\end{array}$$

$$\begin{array}{rl}\mathtt{nat\_elim}: & \Pi(n{:}\mathtt{nat})\,[P{:}\mathtt{nat}\to \mathsf{Prop}].\,P\,0 \to (\Pi k.\,P\,k \to P\,(S\,k)) \to P\,n\\ \mathtt{eq\_nat\_elim}: & \Pi m\,n\,[P{:}\mathsf{Prop}].\,(\Pi[H{:}m=n].\,P) \to (\Pi[H{:}m\neq n].\,P) \to P\\ \mathtt{diveucl}: & \mathtt{nat}\to\mathtt{nat}\to\mathsf{Prop}\\ \mathtt{div\_intro}: & \Pi[a]\,[b]\,q\,r\,[H{:}a=bq+r\wedge r<b].\,\mathtt{diveucl}\,a\,b\\ \mathtt{div\_elim}: & \Pi[a]\,[b]\,[P{:}\mathsf{Prop}].\,\mathtt{diveucl}\,a\,b \to (\Pi q\,r\,[H{:}a=bq+r\wedge r<b].\,P) \to P\end{array}$$

$$\begin{array}{l}\mathtt{div} := \lambda(a\,b{:}\mathtt{nat})\,[H{:}b<>0].\\ \quad \mathtt{nat\_elim}\,a\,[\lambda a.\,\mathtt{diveucl}\,a\,b]\\ \quad (\mathtt{div\_intro}\,[0]\,[b]\,0\,0\,[\pi_1\,H])\\ \quad (\lambda k\,(div_k{:}\mathtt{diveucl}\,k\,b).\\ \quad\ \mathtt{div\_elim}\,[k]\,[b]\,div_k\,[\mathtt{diveucl}\,(S\,k)\,b]\\ \quad\ (\lambda q\,r\,[H_0].\\ \quad\ \mathtt{eq\_nat\_elim}\,r\,(b-1)\,[\mathtt{diveucl}\,(S\,k)\,b]\\ \quad\ (\lambda[H_1].\,\mathtt{div\_intro}\,[S\,k]\,[b]\,(S\,q)\,0\,[\pi_2\,H_0\,H_1])\\ \quad\ (\lambda[H_1].\,\mathtt{div\_intro}\,[S\,k]\,[b]\,q\,(S\,r)\,[\pi_3\,H_0\,H_1])))\end{array}$$

where:

$$\begin{array}{rl}\pi_1: & b\neq 0 \to 0 = b.0 + 0 \wedge 0 < b\\ \pi_2: & k = bq+r\wedge r<b \to r = b-1 \to Sk = b(q+1)+0\wedge 0<b\\ \pi_3: & k = bq+r\wedge r<b \to r\neq b-1 \to Sk = bq+(Sr)\wedge Sr<b\end{array}$$

**Fig. 6.** Euclidean division

One would like to specifiy that if $b$ is not 0 (precondition), then $\mathtt{div}\,a\,b$ returns a pair $(q,r)$ such that $a = bq + r \wedge r < b$ $(\alpha)$ (postcondition). To express the result, we define a type `diveucl`, parameterized by $a$ and $b$ that encodes pairs $(q,r)$ that satisfy $(\alpha)$. More precisely, it is a triple made of 2 explicit components of type `nat` and an implicit proof of $(\alpha)$ (this is an instance of the predicate subtyping scheme, section 5.3). See Fig.6 for the types of the introduction and elimination rules. We can see that the introduction rule has only 2 explicit arguments, so it will behave (w.r.t. conversion) as a pair of numbers. Then, the division program can be specified by type

$$\Pi a\,b\,[H{:}b\neq 0].\,\mathtt{diveucl}\,a\,b.$$

The program can then be written without difficulty (Fig.6) by adapting the proof made in the Calculus of Constructions, assuming `nat_elim` to define programs by recursion on a natural number and `eq_nat_elim` to decide if two numbers are equal.

The point of using ICC$^*$ is that `div` actually behaves as the informal algorithm. Within CC, `div` is a function of arity 3 returning a triple. So if we have two distinct proofs $P_1$ and $P_2$ of $b \neq 0$, $(\texttt{div}\, a\, b\, P_1)$ and $(\texttt{div}\, a\, b\, P_2)$ reduce to triples $(q, r, f(P_1))$ and $(q, r, f(P2))$ respectively, for some $f$. The two proofs $f(P_1)$ and $f(P_2)$ of the postcondition $(\alpha)$ have no particular reason to be equal.

Not only this is solved by ICC$^*$, but furthermore, $(\texttt{div}\ 17\ 5\ [P_3])$ is convertible to $(\texttt{div}\ 11\ 3\ [P_4])$ since the quotient and remainder of both divisions are equal. This is not the case in CC (even assuming proof-irrelevance) since `div_intro` also depends on the inputs $a$ and $b$.

### 5.2 Vectors

Miquel showed how lists and vectors can be encoded in ICC [10]. We adapt his example to ICC$^*$ (see Fig.7). It consists in merging definitions of vectors in ICC and CC: the definitions of CC have to be decorated with implicit/explicit flags as in ICC. Note that `vect` and $P$ are explicit functions since we want to distinguish vectors of different lengths. But $P$ itself is implicit since it is only used to type the other two arguments, which correspond to constructors.

$$
\begin{aligned}
\texttt{vect} &:= \lambda m.\, \Pi[P{:}\texttt{nat} \to \mathsf{Prop}].\, P\ 0 \to (\Pi[n].\, A \to P\ n \to P\ (\texttt{S}\ n)) \to P\ m \\
\texttt{nil} &:= \lambda[P]\ f\ g.\, f \\
\texttt{cons}\ [n]\ x\ v &:= \lambda[P]\ f\ g.\, g\ [n]\ x\ (v\ [P]\ f\ g)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{append}: &\quad \Pi[n_1]\,[n_2].\, \texttt{vect}\ n_1 \to \texttt{vect}\ n_2 \to \texttt{vect}\ (n_1 + n_2) \\
&:= \lambda[n_1]\,[n_2]\, v_1\, v_2.\, v_1\ [\lambda n'.\, \texttt{vect}\ (n' + n_2)]\ v_2\ (\lambda[n']\, x\, v'.\, \texttt{cons}\ [n' + n_2]\, x\, v') \\
\texttt{head}: &\quad \Pi[n].\, \texttt{vect}\ (\texttt{S}\ n) \to A \\
&:= \lambda[n]\, v.\, v\ [\lambda k.\, [k = \texttt{S}\ n] \to A] \\
&\qquad (\lambda[H{:}0 = \texttt{S}\ n].\, \texttt{False\_elim}\ [A]\ [\texttt{discr}\ n\ H]) \\
&\qquad (\lambda[k]\, x\, y\, [\_{:}\texttt{S}\ k = \texttt{S}\ n].\, x) \\
&\qquad [\texttt{refl}\ [\texttt{S}\ n]]
\end{aligned}
$$

**Fig. 7.** Vectors

The reader can check that by extraction towards ICC, these definitions becomes strictly those for the untyped $\lambda$-calculus:

$$\texttt{nil}^* = \lambda f\ g.\, f \qquad \texttt{cons}^* = \lambda x\, v\, f\, g.\, g\ x\ (v\ f\ g)$$

Here, $\texttt{cons}^*$ has arity 2 (if we consider it returns vectors), and there is no extra argument $P$.

Concatenation of two vectors can be expressed easily if we assume that addition satisfies $0 + n \cong_\beta n$ and $(S\, n) + m \cong_\beta S\, (n + m)$, which is the case for the usual impredicative encoding of natural numbers.

In the Calculus of Constructions, computing the length of a vector is useless since a vector always comes along with its length (either as an extra argument or because it

is fixed). In ICC*, when we decide to make the length argument implicit, it cannot be used in the explicit part of the program. For instance, it is illegal to define the length of a vector as $\lambda[n]\,(v{:}\mathtt{vect}\,n).\,n$. It has to be defined as recursive function that examines the full vector.

We hope we made clear that in many situations, the Implicit Calculus of Constructions allows to have the safety of a dependently typed $\lambda$-calculus, but all the typing information (here $P$ and the vector size) does not get in the way, since objects are compared modulo extraction.

### 5.3 Predicate Subtyping a la PVS

One key feature of PVS [14] is predicate subtyping, which corresponds to the comprehension axiom in set theory. For instance, one can define the type of even number as a subtype of the natural numbers satisfying the appropriate predicate. When a number is claimed to have this type, it has to be proven it is actually even, and a type-checking condition (TCC) is generated.

In the Calculus of Constructions, this can be encoded as a dependent pair formed by a natural number and a proof object that it is actually even. The coercion from even numbers to numbers is simply the first projection. This encoding is not faithfull since it might happen that there exists two distinct proofs[6] (let us call them $\pi_1$ and $\pi_2$) of the fact that, say, $4$ is even. Then $(4,\pi_1)$ and $(4,\pi_2)$ are distinct inhabitants of the type of even numbers while they represent the same number. In ICC*, this issue can be avoided by making the second component of the pair implicit.[7] Let us define `Even` as:

$$\Pi[P{:}\mathsf{Prop}].\,(\Pi(n{:}\mathtt{nat})\,[H{:}\mathtt{even}\,n].\,P) \to P$$

Coercion can then be defined as the first projection:

$$\lambda(x{:}\mathtt{Even}).\,x\,[\mathtt{nat}]\,(\lambda n\,[H].\,n) : \mathtt{Even} \to \mathtt{nat}$$

and equalities such as $(4,\pi_1) = (4,\pi_2)$ are proven by reflexivity.

These facts generalize to any predicate over any type since no particular property of natural numbers has been used here.

In [16], Sozeau introduces a feature of Coq similar to the predicate subtyping of PVS, including the possibility to prove claimed invariants by generating proof obligations. Subset types are coded by a pair of an object and a proof of the claimed invariants (e.g. being even). Proof obligations are metavariables that the user must instantiate. This method relies on the fact that programs shall never try to access the proof object, to avoid the issue above mentioned. Combining ICC and his method seems to solve the problem.

### 5.4 Subtyping Coercions

In ICC, vectors are subtypes of lists. Here, we have no subtyping but we can easily write a coercion from vectors to lists (defined as $\Pi[P{:}\mathsf{Prop}].\,P \to (A \to P \to P) \to P$):

[6] Proof-irrelevance is not provable in CC, nor in Coq.

[7] Note that this does not work in ICC because of the strengthening rule.

$$\lambda(v{:}\mathtt{vect}\ n)\,[P]\,f\,g.\,v\,[\lambda\_.\,P]\,f\,\lambda[n]\,x\,v.\,g\,x\,v \quad : \quad \mathtt{vect}\ n \to \mathtt{list}$$

Remark that the extraction of this coercion is $\lambda v\ f\ g.v\ f\ (\lambda x\,v.g\,x\,v)$, which $\eta$-reduces to the identity.

Another illustration of the expressiveness of ICC is the example of terms indexed by the set of free variables (`var` denotes the type used to represent variables):

$$\begin{aligned}\mathtt{term} := {} & \lambda(s{:}\mathtt{var} \to \mathsf{Prop}).\,\Pi[P{:}\mathsf{Prop}].\\ & (\Pi(x{:}\mathtt{var}).\,[s\,x] \to P) \to\\ & (P \to P \to P) \to P\end{aligned}$$

It corresponds to a type with two constructors:

$$\begin{aligned}&\mathtt{Var} : \Pi[s{:}\mathtt{var} \to \mathsf{Prop}]\,(x{:}\mathtt{var}).\,[s\,x] \to \mathtt{term}\,s\\ &\mathtt{App} : \Pi[s{:}\mathtt{var} \to \mathsf{Prop}].\,\mathtt{term}\,s \to \mathtt{term}\,s \to \mathtt{term}\,s.\end{aligned}$$

If set $s$ is included in set $s'$ (i.e. there is a proof $h$ of $\Pi x.\,s\,x \to s'\,x$), then any term of $\mathtt{term}\,s$ can be seen as a term of $\mathtt{term}\,s'$. This can be done by the following function that recursively goes through the term to update the proofs:

$$\begin{aligned}\mathtt{lift} : \ & \mathtt{term}\,s \to \mathtt{term}\,s'\\ := {} & \lambda t\,[P]\,f\,g.\,t\,[P]\,(\lambda x\,[H].\,f\,x\,[h\,x\,H])\,(\lambda t_1\,t_2.\,g\,t_1\,t_2)\end{aligned}$$

We can notice that, as previously, the extraction of this term $\eta$-reduces to the identity.

It seems natural to introduce a notion of *coercion*: terms which extraction reduces to the identity. In cases where we manipulate extracted terms (for instance in the conversion test), then coercions can be dropped. On the other hand, for reduction of decorated terms, coercions can be used to update the implicit subterms of its argument. This duality between annotated and extracted terms forms the most striking feature of ICC$^*$. The current implementation does not optimize coercions yet.

## 6 Related Works

**Epigram.** In comparison with our system, Epigram focuses on implementing inductive types in a more powerful way, and also provides more natural properties for equality. On the other hand, the theory behind is not so different from usual Type Theories (in the tradition of Martin Löf theories). A number of techniques are developed in order to optimize the evaluation:

- in the case of vectors, the length argument can be removed from the constructor [3]. Unfortunately, only information that is uniquely recoverable can be erased. For instance, in the example of terms with their set of variables, the proof that variables belong to the set cannot be made implicit.
- A notion of compilation stages is introduced and an erasure function removes parts that belong to the most "static" stages [2]. However, this process faces the same problems that the extraction of Coq: the conversion rule applies on the fully decorated term, so the erased parts are still compared.

We emphasize on the fact that ICC (and ICC$^*$) have a very powerful conversion rule where logical information is actually irrelevant.

**Proof-irrelevant Calculus of Constructions.** Werner [18] introduces a variant of the Calculus of Constructions where objects of the Prop kind can be erased. His idea is very similar to ours since he modifies conversion so that proofs of a given proposition are always convertible. On the one hand, this does not require a complicated model since proof-irrelevance is a valid property in the classical model of the Calculus of Constructions [12]. On the other hand, this approach does not address the problem of other extra arguments (types, domains of abstractions, dependencies belonging to Set).

## 7 Future Work

We have shown that the Implicit Calculus of Constructions provides a simple yet powerful way to write dependently typed programs and proof-carrying programs where specifications and proofs do not interfere with the computational content. However there are still aspects that are not completely satisfactory.

**Inductive Types.** The key one is to extend the model of ICC to inductive types. As already mentionned, the difficult part is understanding how to support union types in the setting of coherent spaces.

Once the theory is set up, it is very desirable to have a powerful case-analysis operator as in Epigram or Agda. This can save awkward manipulations of equality proofs as in the vector `head` example.

**Other programming paradigms.** ATS can deal with a large variety of aspects: side-effects, non-termination, memory allocation, and more. This is definitely a must have if we expect dependent types to reach a larger audience.

## 8 Conclusion

We have shown how a restriction of the Implicit Calculus of Constructions can be turned into an implementable system, and we developed several typical examples. We shall stress on the fact that some problems seem less difficult to deal with than in most type systems implemented so far. Moreover, from the Coq user point of view, there are only a few changes, and the new features can be learnt quickly.

## References

1. Barendregt, H.: Lambda Calculi with Types. Technical Report 91-19, Catholic University Nijmegen. In: Handbook of Logic in Computer Science, vol. II (1991)
2. Brady, E.: Practical Implementation of a Dependently Typed Functional Programming Language. PhD thesis, Durham University (2005)
3. Brady, E., McBride, C., McKinna, J.: Inductive families need not store their indices. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 115–129. Springer, Heidelberg (2004)
4. Chen, C., Xi, H.: Combining Programming with Theorem Proving. In: Proceedings of the tenth ACM SIGPLAN ICFP, Tallinn, Estonia, pp. 66–77 (September 2005)

5. Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the Nuprl Development System. Prentice-Hall, NJ (1986)
6. Letouzey, P.: Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq. PhD thesis, Université Paris-Sud (July 2004)
7. Luo, Z.: An Extended Calculus of Constructions. PhD thesis, University of Edinburgh (1990)
8. McBride, C., McKinna, J.: The view from the left. Journal of Functional Programming 14(1), 69–111 (2004)
9. Miquel, A.: Arguments implicites dans le calcul des constructions: étude d'un formalisme à la curry. Master's thesis, University Paris 7 (1998)
10. Miquel, A.: The implicit calculus of constructions. Extending pure type systems with an intersection type binder and subtyping. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, Springer, Heidelberg (2001)
11. Miquel, A.: Le Calcul des Constructions implicite: syntaxe et sémantique. PhD thesis, Université Paris 7 (December 2001)
12. Miquel, A., Werner, B.: The not so simple proof-irrelevent model of CC. In: Geuvers, H., Wiedijk, F. (eds.) TYPES 2002. LNCS, vol. 2646, Springer, Heidelberg (2003)
13. Norell, U.: Towards a practical programming language based on dependent type theory. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (September 2007)
14. Owre, S., Shankar, N.: The formal semantics of PVS. Technical Report SRI-CSL-97-2, Menlo Park, CA (1997)
15. Prost, F.: Interprétation de l'analyse statique en théorie des types. PhD thesis, École Normale Supérieure de Lyon (December 1999)
16. Sozeau, M.: Subset coercions in Coq. In: Altenkirch, T., McBride, C. (eds.) TYPES 2006. LNCS, vol. 4502, Springer, Heidelberg (2007)
17. The Coq development team. The coq proof assistant reference manual v8.0. Technical report, INRIA, France, mars (2004), http://coq.inria.fr/doc/main.html
18. Werner, B.: On the strength of proof-irrelevant type theories. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 604–618. Springer, Heidelberg (2006)
19. Xi, H., Pfenning, F.: Dependent types in practical programming. In: Proceedings of the 26th ACM SIGPLAN Symposium on Principles of Programming Languages, San Antonio, pp. 214–227 (January 1999)

# Strong Normalisation of Cut-Elimination That Simulates $\beta$-Reduction

Kentaro Kikuchi[1] and Stéphane Lengrand[2]

[1] RIEC, Tohoku University, Japan
[2] CNRS, Laboratoire d'Informatique de l'Ecole Polytechnique, France

**Abstract.** This paper is concerned with strong normalisation of cut-elimination for a standard intuitionistic sequent calculus. The cut-elimination procedure is based on a rewrite system for proof-terms with cut-permutation rules allowing the simulation of $\beta$-reduction. Strong normalisation of the typed terms is inferred from that of the simply-typed $\lambda$-calculus, using the notions of *safe* and *minimal* reductions as well as a simulation in Nederpelt-Klop's $\lambda I$-calculus. It is also shown that the type-free terms enjoy the preservation of strong normalisation (PSN) property with respect to $\beta$-reduction in an isomorphic image of the type-free $\lambda$-calculus.

## 1 Introduction

It is now established that cut-elimination procedures in sequent calculus have a computational meaning (see e.g. [12,7,32,26]), in the same sense as that of proof transformations in natural deduction. The paradigm of the Curry-Howard correspondence is then illustrated not only by (intuitionistic implicational) natural deduction and the simply-typed $\lambda$-calculus [13], but also by a typed higher-order calculus corresponding to the (intuitionistic implicational) sequent calculus.

In [16], the first author identified through a Prawitz-style translation a subset of proofs in a standard sequent calculus that correspond to simply-typed $\lambda$-terms, and defined a reduction relation on those proofs that precisely corresponds to $\beta$-reduction of the simply-typed $\lambda$-calculus. The reduction relation was shown to be simulated by a cut-elimination procedure, so the system of proof-terms for the sequent calculus is a conservative extension of the $\lambda$-calculus in both term-structure and reduction. Since the correspondence holds also for the type-free case, the rewrite system in [16] can simulate $\beta$-reduction of the type-free $\lambda$-calculus, which means that it is strong enough to represent all computable functions. It was also shown in [17] that a restriction of the rewrite system in [16], which is still strong enough to simulate $\beta$-reduction, is confluent.

The present paper presents the first proof of strong normalisation of the cut-elimination procedure in [17]. Since the cut-elimination procedure can simulate $\beta$-reduction of the simply-typed $\lambda$-calculus, its strong normalisation is at least as hard as that of the latter. In fact, the proof we develop in this paper relies on strong normalisation of the simply-typed $\lambda$-calculus. However, a naive simulation of the cut-elimination procedure by $\beta$-reduction fails, so we refine

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 380–394, 2008.

the approach by using two techniques formalised in [21,22] as the *Safeness & Minimality technique* and the *simulation in the $\lambda I$-calculus* of [19] through a non-deterministic encoding. A by-product of our method is a proof of strong normalisation of the type-free terms that encode strongly normalising type-free $\lambda$-terms through the Prawitz-style translation. This is known as the property of *Preservation of Strong Normalisation* (PSN) in the field of calculi with explicit substitutions.

Strong normalisation of cut-elimination has been studied by a number of authors. Here we mention some of them that treat cut-elimination procedures consisting of (Gentzen-style) local proof transformations in a standard sequent calculus. The first is Dragalin's cut-elimination procedure and simple proof of its strong normalisation, which can be found in Chapter 7 of [28]. However, the cut-elimination procedure does not allow any permutation of cuts, so cannot simulate $\beta$-reduction. A cut-elimination procedure that simulates $\beta$-reduction can be found in [31] (for the classical sequent calculus), with a proof of strong normalisation. However, the cut-permutation rules involve extra kinds of cuts that are allowed to pass over usual cuts; therefore it is not clear how the proof of strong normalisation could be adapted to our case, which leaves the simple syntax of the calculus untouched. Recently, [23] introduced another cut-elimination procedure and proved its strong normalisation. The cut-elimination procedure is a modification of the one in [16], but differs from the one in the present paper. The proof technique for strong normalisation in [23] does not work for our system, and our proof in this paper solves a related problem that was explicitly given in Section 6 of [23]. Finally, [29] presents a proof of strong normalisation for a cut-elimination system that is not intended (and is unlikely) to simulate $\beta$-reduction. However, their technique is also inspired by Nederpelt and Klop's works on $\lambda I$ and how it compares to ours, different though the cut-elimination systems are, remains to be investigated.

The structure of the paper is as follows. In Section 2 we introduce a term assignment system for a standard sequent calculus and a rewrite system for a cut-elimination procedure in the sequent calculus. In Section 3 we explain our proof techniques and apply them to showing strong normalisation for the typed terms and the PSN property for the type-free terms. In Section 4 we discuss related work and conclude in Section 5.

To save space we omit some of the details in proofs, but a longer paper [18] is available at `http://www.lix.polytechnique.fr/~lengrand/Work/`.

## 2 A Rewrite System for Cut-Elimination

### 2.1 Term Assignment for Sequent Calculus

We start by giving a proof-term assignment system for a standard intuitionistic sequent calculus, following [16]. The syntax of proof-terms can be found in various textbooks (e.g. [30,28]) and papers (e.g. [10]) with notational variants. Here we call the proof-terms $\lambda$G3-*terms*. Our cut-elimination procedure is represented as reduction rules for typed $\lambda$G3-terms.

First, the set of raw $\lambda$G3-terms is defined by the grammar:

$$M ::= x \mid \lambda x.M \mid \langle xM/x\rangle M \mid [M/x]M$$

where $x$ ranges over a denumerable set of variables. $\langle \_\,\_/\_\rangle\_$ and $[\_/\_]\_$ are term constructors similar to explicit substitutions ($[\_/\_]\_$ is called the cut-constructor). We use letters $x, y, z, w$ for variables and $M, N, P, Q$ for $\lambda$G3-terms. To denote that $M$ is a strict subterm of $N$, we write $M \sqsubset N$ or $N \sqsupset M$. The notions of free and bound variables are defined as usual, with an additional clause that the variable $x$ in $\langle yM/x\rangle N$ or $[M/x]N$ binds the free occurrences of $x$ in $N$. The set of free variables of a $\lambda$G3-term $M$ is denoted by $\mathsf{FV}(M)$. We often use the notation $\langle \underline{x}M/y\rangle N$ to denote $\langle xM/y\rangle N$ if $x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N)$. The symbol $\equiv$ denotes syntactical equality modulo $\alpha$-conversion; so for example $\langle zP/x\rangle\langle \underline{x}M/y\rangle N \equiv \langle zP/w\rangle\langle \underline{w}M/y\rangle N$.

The proof-term assignment system for a standard intuitionistic sequent calculus is given in Figure 1. We define a typing context, ranged over by $\Gamma$, as a finite set of pairs $\{x_1 : A_1, \ldots, x_n : A_n\}$ where the variables are pairwise distinct. $\Gamma, x : A$ denotes the union $\Gamma \cup \{x : A\}$, and $x \notin \Gamma$ means that $x$ does not appear in $\Gamma$. For precise representation of proofs by terms, we should specify formulas on binders, but we will omit them for brevity. If $x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N)$ in the $\lambda$G3-term $\langle xM/y\rangle N$, we assume $x \notin \Gamma$ in the rule $\mathsf{L}\supset$, which means the formula $A \supset B$ is introduced without implicit contraction.

$$\mathsf{Ax}\ \frac{}{\Gamma, x : A \vdash x : A} \qquad \mathsf{L}\supset\ \frac{\Gamma \vdash M : A \quad \Gamma, y : B \vdash N : C}{\Gamma, x : A \supset B \vdash \langle xM/y\rangle N : C}\ y \notin \Gamma$$

$$\mathsf{R}\supset\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ x \notin \Gamma \qquad \mathsf{Cut}\ \frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash [M/x]N : B}\ x \notin \Gamma$$

**Fig. 1.** Proof-term assignment for sequent calculus

We write $\Gamma \vdash_{\lambda\mathsf{G3}} M : A$ if the sequent is derivable with the inference rules of Figure 1. We also write $\Gamma \vdash_{\lambda} t : A$ if it is derivable with the standard inference rules of the simply-typed $\lambda$-calculus.

In order to understand the semantics of $\lambda$G3, we can re-express Gentzen's translation of proofs in sequent calculus to those in natural deduction using $\lambda$G3-terms and $\lambda$-terms ($\{-/\_\}\_$ means usual implicit substitution).

$$\begin{array}{lcl}
\mathcal{G}^1(x) & := & x \\
\mathcal{G}^1(\lambda x.M) & := & \lambda x.\mathcal{G}^1(M) \\
\mathcal{G}^1(\langle xM/y\rangle N) & := & \left\{{}^{x\ \mathcal{G}^1(M)}/_{y}\right\}\mathcal{G}^1(N) \\
\mathcal{G}^1([M/x]N) & := & \left\{{}^{\mathcal{G}^1(M)}/_{x}\right\}\mathcal{G}^1(N)
\end{array}$$

Notice that terms of $\lambda\mathsf{G3}^{\mathsf{cf}}$ (i.e. $\lambda\mathsf{G3}$-terms without the cut-constructor) are always mapped to $\lambda$-terms in normal form.

We can also give a backward translation from natural deduction to sequent calculus:

$$\begin{array}{lcl}\mathcal{G}^2(x) & := & x\\ \mathcal{G}^2(\lambda x.t) & := & \lambda x.\mathcal{G}^2(t)\\ \mathcal{G}^2(t\ s) & := & [\mathcal{G}^2(t)/x]\langle \underline{x}\ \mathcal{G}^2(s)/y\rangle y\end{array}$$

In the above translation, normal forms of $\lambda$-calculus are not necessarily mapped to $\lambda\mathsf{G3}^{\mathsf{cf}}$-terms. This is fixed by a Prawitz-style encoding:

$$\begin{array}{lcl}\mathcal{P}r(x) & := & x\\ \mathcal{P}r(\lambda x.t) & := & \lambda x.\mathcal{P}r(t)\\ \mathcal{P}r(t\ s) & := & \mathcal{P}r_{x.x}(t\ s)\\ \hline \mathcal{P}r_{x.M}(y\ s) & := & \langle y\mathcal{P}r(s)/x\rangle M\\ \mathcal{P}r_{x.M}((\lambda y.t)\ s) & := & [\lambda y.\mathcal{P}r(t)/z]\langle \underline{z}\mathcal{P}r(s)/x\rangle M\\ \mathcal{P}r_{x.M}(t_1\ t_2\ s) & := & \mathcal{P}r_{z.\langle \underline{z}\mathcal{P}r(s)/x\rangle M}(t_1\ t_2)\end{array}$$

**Theorem 1 (Preservation of typing)**

- *If* $\Gamma \vdash_{\lambda G3} M{:}A$ *then* $\Gamma \vdash_{\lambda} \mathcal{G}^1(M){:}A$.
- *If* $\Gamma \vdash_{\lambda} t{:}A$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{G}^2(t){:}A$.
- *If* $\Gamma \vdash_{\lambda} t\ s{:}A$ *and* $\Gamma, x{:}A \vdash_{\lambda G3} M{:}B$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{P}r_{x.M}(t\ s){:}B$.
- *If* $\Gamma \vdash_{\lambda} t{:}A$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{P}r(t){:}A$.

The following theorems can be found in the literature, some of them in [10].

**Theorem 2 (Properties of the encodings)**

- $\mathcal{G}^1$ *is surjective, its restriction* $\mathcal{G}^1{}_{|\lambda G3^{\mathsf{cf}}}$ *to cut-free terms is surjective on normal* $\lambda$*-terms, and neither is injective.*
- $\mathcal{G}^2$ *and* $\mathcal{P}r$ *are both injective but not surjective on* $\lambda G3$.
- $\mathcal{G}^1 \circ \mathcal{G}^2 = \mathsf{Id}_{\lambda}$ *and* $\mathcal{G}^1 \circ \mathcal{P}r = \mathsf{Id}_{\lambda}$.
- *Neither* $\mathcal{G}^2 \circ \mathcal{G}^1 \neq \mathsf{Id}_{\lambda G3}$ *nor* $\mathcal{P}r \circ \mathcal{G}^1 \neq \mathsf{Id}_{\lambda G3}$.
- $\mathcal{G}^2 \circ \mathcal{G}^1$ *does not leave* $\lambda G3^{\mathsf{cf}}$ *stable*[1] *but* $\mathcal{P}r \circ \mathcal{G}^1$ *does.*

## 2.2 The Cut-Elimination Procedure

Our cut-elimination procedure is based on a rewrite system for $\lambda\mathsf{G3}$-terms. The system is the same as the one in [17], which is a confluent restriction of the system in [16]. (Although confluence is not used in this paper, the system in [16] so far seems to resist the present technique.)

[1] (i.e. if $M$ is cut-free, $\mathcal{G}^2(\mathcal{G}^1(M))$ might not be).

Figure 2 shows the reduction rules of the rewrite system. Each of these reduction rules corresponds to a local cut-elimination step (cf. [18]). The reduction rules (1) through (5) correspond to cut-elimination steps that permute a cut upwards through its right subproof. The rules (6) and (7′) correspond to steps permuting a cut upwards through its left subproof. The rule (B) corresponds to the key-case which breaks a cut on an implication into two cuts on its subformulas. The rules ($\mathsf{Perm}_1$) and ($\mathsf{Perm}_2$) permute two cuts with some restrictions. In ($\mathsf{Perm}_1$), the left rule over the lower cut is another cut, and the right rules over both cuts must be $\mathsf{L}\supset$ that introduces the cut-formula without implicit contraction. In ($\mathsf{Perm}_2$), the right rule over the lower cut is another cut, which must construct a proof corresponding to a redex of the rule (B).

| | |
|---|---|
| (1) | $[M/x]y \to y \quad (y \not\equiv x)$ |
| (2) | $[M/x]x \to M$ |
| (3) | $[N/x](\lambda y.M) \to \lambda y.[N/x]M$ |
| (4) | $[P/z]\langle xM/y\rangle N \to \langle x([P/z]M)/y\rangle[P/z]N \quad (x \not\equiv z)$ |
| (5) | $[P/x]\langle xM/y\rangle N \to [P/x]\langle \underline{x}([P/x]M)/y\rangle[P/x]N \quad$ if $x \in \mathsf{FV}(M) \cup \mathsf{FV}(N)$ |
| (6) | $[z/x]\langle \underline{x}M/y\rangle N \to \langle zM/y\rangle N$ |
| (7′) | $[\langle xM/y\rangle N/z]\langle \underline{z}M'/w\rangle N' \to \langle xM/y\rangle[N/z]\langle \underline{z}M'/w\rangle N'$ |
| (B) | $[\lambda z.P/x]\langle \underline{x}M/y\rangle N \to [[M/z]P/y]N$ |
| ($\mathsf{Perm}_1$) | $[[P/x]\langle \underline{x}M/y\rangle N/z]\langle \underline{z}M'/w\rangle N' \to [P/x][\langle \underline{x}M/y\rangle N/z]\langle \underline{z}M'/w\rangle N'$ |
| ($\mathsf{Perm}_2$) | $[Q/w][\lambda z.P/x]\langle \underline{x}M/y\rangle N \to [[Q/w](\lambda z.P)/x][Q/w]\langle \underline{x}M/y\rangle N$ |

**Fig. 2.** Rewrite system for cut-elimination

The reduction relation $\longrightarrow_{\mathsf{cut}}$ is defined by the contextual closures of the reduction rules in Figure 2. We use $\longrightarrow^{+}_{\mathsf{cut}}$ for its transitive closure, and $\longrightarrow^{*}_{\mathsf{cut}}$ for its reflexive transitive closure. The set of $\lambda$G3-terms that are strongly normalising with respect to $\longrightarrow_{\mathsf{cut}}$ is denoted by $\mathsf{SN}^{\mathsf{cut}}$. These kinds of notations are also used for the notions of other reductions in this paper.

The rewrite system without the rule (B) is called x. It was shown in [17] that the system x is strongly normalising and confluent.

The original rewrite system in [16] has instead of (7′) the rule (7) which is obtained by replacing $\langle \underline{z}M'/w\rangle N'$ in (7′) by a general term $P$. However then the system becomes non-confluent (e.g. the critical pair $w \leftarrow [\langle xM/y\rangle N/z]w \to \langle xM/y\rangle[N/z]w$ is not joinable). We study in this paper the system with (7′), which was shown to be confluent in [17] and which is still strong enough to simulate $\beta$-reduction.

**Theorem 3 (Simulation of $\beta$-reduction)**
*$\longrightarrow_{cut}$ strongly simulates $\longrightarrow_{\beta}$ through the translation $\mathcal{P}r$, i.e. if $M \longrightarrow_{\beta} M'$ then $\mathcal{P}r(M) \longrightarrow^{+}_{cut} \mathcal{P}r(M')$.*

*Proof.* This is a minor variant of the proof in [16]. The proof is by induction on the derivation of the reduction step, using various lemmas. □

## 3 Strong Normalisation

In this section we prove strong normalisation of $\longrightarrow_{\mathsf{cut}}$ on (typed) $\lambda$G3-terms. Since this reduction relation can simulate $\beta$-reduction in $\lambda$-calculus, its strong normalisation is at least as hard as that of the simply-typed $\lambda$-calculus. In fact, our proof relies on the latter.

A by-product of our method is a proof of strong normalisation of those $\lambda$G3-terms that encode strongly normalising type-free $\lambda$-terms through the Prawitz-style translation. This is known as the property of *Preservation of Strong Normalisation* (PSN) [3]. In other words, the reduction relation of $\lambda$G3 is big enough to simulate $\beta$-reduction through the Prawitz-style translation, but small enough to be strongly normalising.

The basic idea in proving that a term $M$ of $\lambda$G3 is SN is to simulate the reduction steps from $M$ by $\beta$-reduction steps from a strongly normalising $\lambda$-term $\mathcal{G}^1(M)$. Indeed, this would be relevant for PSN since $\mathcal{G}^1(\mathcal{P}r(t)) = t$, as well as for the strong normalisation of a typed $\lambda$G3-term $M$, since $\mathcal{G}^1(M)$ is a simply-typed $\lambda$-term. The idea of simulating cut-elimination by $\beta$-reductions has been investigated in [35,25].

Unfortunately, Gentzen's encoding into $\lambda$-calculus, which allows the simulation, needs to interpret cut-constructors (and constructors for $\mathsf{L}\supset$) as implicit substitutions such as $\{{}^{u}/_{x}\}t$. Should $x$ not be free in $t$, all reduction steps taking place within the term of which $u$ is the encoding would not induce any $\beta$-reduction in $\{{}^{u}/_{x}\}t$. Therefore, the reduction relation that is only weakly simulated, i.e. the one consisting of all the reductions that are not necessarily simulated by at least one $\beta$-reduction, is too big to be proved terminating (in fact, it is not).

In order to overcome the aforementioned problem, we combine two techniques formalised in [21,22] as the *Safeness & Minimality technique* and the *simulation in the $\lambda I$-calculus* of [19] through a non-deterministic encoding.

### 3.1 Safeness and Minimality

We first define safe and minimal reductions for the rewrite system of $\longrightarrow_{\mathsf{cut}}$ on (some class of) $\lambda$G3-terms.

The intuitive idea is that a reduction step is *minimal* if all the (strict) subterms of the redex are in $\mathsf{SN}^{\mathsf{cut}}$. Theorem 4(1) says that in order to prove that $\longrightarrow_{\mathsf{cut}}$ is terminating, we can restrict our attention to minimal reductions only, without loss of generality.[2] Similarly, a reduction step is *safe* if the redex itself is in $\mathsf{SN}^{\mathsf{cut}}$, which is a stronger requirement than minimality. Theorem 4(2) says that

[2] Note that a perpetual strategy, in the sense of [33], can be defined so that only minimal reductions are performed. Also, the technique seems close to that of dependency pairs (see e.g. [1]) and formal connections should be studied.

safe reductions always terminate. Those ideas are made precise in the following definition:

**Definition 1 (Safe and minimal reduction).** *Given a subsystem $\mathsf{h}$ of our cut-elimination system, we define the following rules:*

$$\mathsf{minh} \quad M \longrightarrow N \quad \text{if } M \longrightarrow_{\mathsf{h}} N \text{ and for all } P \sqsubset M,\ P \in SN^{cut}$$
$$\mathsf{safeh} \quad M \longrightarrow N \quad \text{if } M \longrightarrow_{\mathsf{h}} N \text{ and } M \in SN^{cut}$$

*and denote their contextual closures by $\longrightarrow_{\mathsf{minh}}$ and $\longrightarrow_{\mathsf{safeh}}$ respectively.*

*We say that a reduction step $M \longrightarrow_{\mathsf{h}} N$ is safe (resp. minimal) if $M \longrightarrow_{\mathsf{safeh}} N$ (resp. $M \longrightarrow_{\mathsf{minh}} N$) and that it is unsafe if not.*[3]

*Remark 1.* We shall constantly use the following facts:

1. $\longrightarrow_{\mathsf{min(safeh)}} = \longrightarrow_{\mathsf{safe(minh)}} = \longrightarrow_{\mathsf{safeh}}$
2. $\longrightarrow_{\mathsf{safe(h,h')}} = \longrightarrow_{\mathsf{safeh,safeh'}}$
3. $\longrightarrow_{\mathsf{min(h,h')}} = \longrightarrow_{\mathsf{minh,minh'}}$

We have the following theorems (proofs can be found in [21,22]):

**Theorem 4.** *1. $SN^{mincut} = SN^{cut}$.*
*2. For every $\lambda G3$-term $M$, $M \in SN^{safecut}$.*

In other words, safe reductions terminate, and in order to prove that a term is strongly normalising, it suffices to prove that it is strongly normalising for minimal reductions only.

This leads directly to the following corollary:

**Theorem 5 (Safeness & minimality theorem).** *Given a rewrite system $\mathsf{h}$ satisfying $\longrightarrow_{\mathsf{safecut}} \subseteq \longrightarrow_{\mathsf{h}} \subseteq \longrightarrow_{\mathsf{mincut}}$, suppose that we have:*

- *the strong simulation of $\longrightarrow_{\mathsf{mincut}} \setminus \longrightarrow_{\mathsf{h}}$ in a strongly normalising calculus, through a total relation $\mathcal{H}$*
- *the weak simulation of $\longrightarrow_{\mathsf{h}}$ through $\mathcal{H}$*
- *the strong normalisation of $\longrightarrow_{\mathsf{h}}$.*

*Then $\longrightarrow_{\mathsf{cut}}$ is strongly normalising.*

A naive attempt would be to take $\mathsf{h} = \mathsf{safecut}$, which terminates by Theorem 4(2). Unfortunately, this situation is too coarse, that is to say, the relation $\longrightarrow_{\mathsf{h}}$ is too small so that $\longrightarrow_{\mathsf{mincut}} \setminus \longrightarrow_{\mathsf{h}}$ is too big to be strongly simulated. Hence, in order to define $\mathsf{h}$, we use the safeness criterion in a more subtle way, that is, we define $\mathsf{h} = \mathsf{safeB}, \mathsf{minx}$.

Among the conditions to apply Theorem 5, we first prove the third one, i.e. the strong normalisation of $\mathsf{safeB}, \mathsf{minx}$. For this we give a technical definition. The idea is to distinguish a class of terms with cut-constructors, reflecting the restrictions on permutations of two cuts in the rules ($\mathsf{Perm}_1$) and ($\mathsf{Perm}_2$).

[3] In both rules we could require $M \longrightarrow_{\mathsf{h}} N$ to be a root reduction so that $M$ *is* the redex, but $\longrightarrow_{\mathsf{safeh}}$ and $\longrightarrow_{\mathsf{minh}}$ would be the same as they are.

**Definition 2 (Application term).** *A $\lambda G3$-term of the form $[M/x]N$ is called an* application term *if $N$ is one of the forms: $[P/w]\langle \underline{x}M'/y\rangle N'$, $\langle \underline{x}M'/y\rangle N'$ and $[\langle \underline{x}M'/y\rangle N'/z]\langle \underline{z}M''/w\rangle N''$ where $x$ occurs only once in $N$.*

**Lemma 1.** *If $[M/x]N$ is an application term and $N \longrightarrow_{cut} N'$, then $[M/x]N'$ is also an application term.*

*Proof.* It suffices to check each case. □

Next we briefly recall the lexicographic path ordering. For a more detailed description and proofs, the reader is referred to, e.g. [14,2].

**Definition 3 (Lexicographic path ordering).** *Let $\succ$ be a transitive and irreflexive ordering on the set of function symbols in a first-order signature, and let $s \equiv f(s_1, \ldots, s_m)$ and $t \equiv g(t_1, \ldots, t_n)$ be terms over the signature. Then $s >_{\mathsf{lpo}} t$, if one of the following holds:*

1. *$s_i \equiv t$ or $s_i >_{\mathsf{lpo}} t$ for some $i = 1, \ldots, m$,*
2. *$f \succ g$ and $s >_{\mathsf{lpo}} t_j$ for all $j = 1, \ldots, n$,*
3. *$f \equiv g$, $s >_{\mathsf{lpo}} t_j$ for all $j = 1, \ldots, n$, and $s_1 \equiv t_1, \ldots, s_{i-1} \equiv t_{i-1}, s_i >_{\mathsf{lpo}} t_i$ for some $i = 1, \ldots, m$.*

**Theorem 6.** *$>_{\mathsf{lpo}}$ is well-founded if and only if $\succ$ is well-founded.*

Now we encode $\lambda$G3-terms into a first-order syntax given by the following ordered infinite signature:

$$\mathsf{sub}(\_,\_) \succ \mathsf{app}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \mathsf{c}^M$$

where for every $M \in \mathsf{SN}^{\mathsf{cut}}$ there is a constant $\mathsf{c}^M$. Those constants are all below $\mathsf{i}(\_)$, and the precedence between them is given by $\mathsf{c}^M \succ \mathsf{c}^N$ if $M \longrightarrow^+_{\mathsf{cut}} N$ or $M \sqsupset N$. Then the precedence relation is well-founded, and so $>_{\mathsf{lpo}}$ induced on the first-order terms is also well-founded. The encoding aforementioned is given in Figure 3.

| | | | |
|---|---|---|---|
| $\overline{M}$ | $:=$ | $\mathsf{c}^M$ | if $M \in \mathsf{SN}^{\mathsf{cut}}$ |
| otherwise | | | |
| $\overline{\lambda x.M}$ | $:=$ | $\mathsf{i}(\overline{M})$ | |
| $\overline{\langle xM/y\rangle N}$ | $:=$ | $\mathsf{ii}(\overline{M}, \overline{N})$ | |
| $\overline{[M/x]N}$ | $:=$ | $\mathsf{app}(\overline{M}, \overline{N})$ | if $[M/x]N$ is an application term |
| $\overline{[M/x]N}$ | $:=$ | $\mathsf{sub}(\overline{M}, \overline{N})$ | otherwise |

**Fig. 3.** Encoding of $\lambda$G3 into a first-order syntax

**Lemma 2.** *If $M \longrightarrow_{safeB,minx} M'$ then $\overline{M} >_{\mathsf{lpo}} \overline{M'}$. Hence, $\longrightarrow_{safeB,minx}$ is strongly normalising.*

*Proof.* By induction on the derivation of the reduction step. If $M \equiv [P/x]N$ is an application term and $N \longrightarrow_{\mathsf{safeB,minx}} N'$, then we use Lemma 1. (A detailed proof can be found in [18].) □

### 3.2 Simulation in $\lambda I$

Now we have to find a strongly normalising calculus and a total relation $\mathcal{H}$ to strongly simulate $\longrightarrow_{\mathsf{mincut}} \setminus \longrightarrow_{\mathsf{h}}$ therein. Since a simple simulation in $\lambda$-calculus fails we use instead the $\lambda I$-calculus of [19], based on earlier work by [6,24]. For instance, the technique works for proving $\mathsf{PSN}$ of the explicit substitution calculus $\lambda\mathsf{lxr}$ of [15]. We refer the reader to [27,34] for a survey on different techniques based on the $\lambda I$-calculus to infer normalisation properties.

On the one hand, $\lambda I$ extends the syntax of $\lambda$-calculus with a "memory operator" so that, instead of being thrown away, a term $U$ can be retained and carried along in a construct $[-, U]$. With this operator, those bodies of substitutions are encoded that would otherwise disappear, as explained above. On the other hand, $\lambda I$ restricts $\lambda$-abstractions to variables that have at least one free occurrence, so that $\beta$-reduction never erases its argument.

**Definition 4 (Grammar of $\lambda I$).** *The set $\Lambda I$ of terms of the $\lambda I$-calculus of [19] is defined by the following grammar:*

$$T, U ::= x \mid \lambda x.T \mid T\ U \mid [T, U]$$

*with the additional restriction that every abstraction $\lambda x.T$ satisfies $x \in \mathsf{FV}(T)$.*

The following property is straightforward by induction on terms.

**Lemma 3 (Stability under substitution [19]).**
*If $T, U \in \Lambda I$, then $\left\{{}^{U}/_{x}\right\}T \in \Lambda I$.*

**Definition 5 (Reduction system of $\lambda I$).** *The reduction rules are:*

$$\begin{array}{ll} (\beta) & (\lambda x.T)\ U \to \left\{{}^{U}/_{x}\right\}T \\ (\pi) & [T, U]\ T' \to [T\ T', U] \end{array}$$

The following remark is straightforward [19]:

*Remark 2.* If $T \longrightarrow_{\beta,\pi} T'$ then $\mathsf{FV}(T) = \mathsf{FV}(T')$ and $\left\{{}^{T}/_{x}\right\}U \longrightarrow^{+}_{\beta,\pi} \left\{{}^{T'}/_{x}\right\}U$ provided that $x \in \mathsf{FV}(U)$.

Performing a simulation in $\lambda I$ requires the encoding to be non-deterministic, i.e. we define a relation $\mathcal{H}$ between $\lambda\mathsf{G3}$ and $\lambda I$, and the reason for this is that, since the reductions in $\lambda I$ are non-erasing reductions, we need to add this memory operator at random places in the encoding, using such a rule:

$$\frac{M\ \mathcal{H}\ T}{M\ \mathcal{H}\ [T, U]}\ U \in \Lambda I$$

The reduction relation of $\lambda\mathsf{G3}$ must then satisfy the hypotheses of Theorem 5. Namely, $\longrightarrow_{\mathsf{mincut}} \setminus \longrightarrow_{\mathsf{h}}$ should be strongly simulated by $\longrightarrow_{\beta,\pi}$ through $\mathcal{H}$, and $\mathsf{safeB}, \mathsf{minx}$ should be weakly simulated by $\longrightarrow_{\beta,\pi}$ through $\mathcal{H}$.

The relation $\mathcal{H}$ between $\lambda\mathsf{G3}$-terms and $\lambda I$-terms is inductively defined in Figure 4.

$$\frac{}{x \mathrel{\mathcal{H}} x} \qquad \frac{M \mathrel{\mathcal{H}} T}{\lambda x.M \mathrel{\mathcal{H}} \lambda x.T}\, x \in \mathsf{FV}(T) \qquad \frac{M \mathrel{\mathcal{H}} U \quad N \mathrel{\mathcal{H}} T}{\langle xM/y\rangle N \mathrel{\mathcal{H}} \left\{{}^{x\ U}/_{y}\right\} T}\, y \in \mathsf{FV}(T)$$

$$\frac{M \mathrel{\mathcal{H}} T}{M \mathrel{\mathcal{H}} [T,U]}\, U \in \Lambda I \qquad \frac{M \mathrel{\mathcal{H}} U \quad N \mathrel{\mathcal{H}} T}{[M/x]N \mathrel{\mathcal{H}} \left\{{}^{U}/_{x}\right\} T}\, x \in \mathsf{FV}(T) \vee M \in \mathsf{SN}^{\mathsf{cut}}$$

**Fig. 4.** Relation between $\lambda$G3 & $\lambda I$

It satisfies the following properties:

**Lemma 4.** *If $M \mathrel{\mathcal{H}} T$, then*

1. $\mathsf{FV}(M) \subseteq \mathsf{FV}(T)$
2. $T \in \Lambda I$
3. $x \notin \mathsf{FV}(M)$ *and* $U \in \Lambda I$ *implies* $M \mathrel{\mathcal{H}} \left\{{}^{U}/_{x}\right\} T$
4. $\{{}^{y}/_{x}\} M \mathrel{\mathcal{H}} \{{}^{y}/_{x}\} T$.

**Theorem 7 (Simulation in $\lambda I$).** *Suppose $M \mathrel{\mathcal{H}} T$.*

1. *If $M \longrightarrow_{\mathsf{minB}} N$ is unsafe then there exists $U$ such that $N \mathrel{\mathcal{H}} U$ and $T \longrightarrow^{+}_{\beta\pi} U$.*
2. *If $M \longrightarrow_{\mathsf{minB}} N$ is safe then there exists $U$ such that $N \mathrel{\mathcal{H}} U$ and $T \longrightarrow^{*}_{\beta\pi} U$.*
3. *If $M \longrightarrow_{\mathsf{minx}} N$ then there exists $U$ such that $N \mathrel{\mathcal{H}} U$ and $T \longrightarrow^{*}_{\pi} U$.*

*Proof.* By induction on the derivation of the reduction step, by case analysis for root reduction. Indeed, for root-reduction, remember that we only simulate minimal reductions. Hence, when reducing a redex, all its subterms are in $\mathsf{SN}^{\mathsf{cut}}$, so the side-condition in the encoding of the cut-constructor is thus satisfied.

For the contextual closure, we have to ensure that, in the first of the above three points, the one reduction step that must take place is preserved through the inductive argument. This comes from the assumption that the reduction is unsafe, which ensures that, in the side-condition $x \in \mathsf{FV}(T) \vee M \in \mathsf{SN}^{\mathsf{cut}}$, it must be true that $x \in \mathsf{FV}(T)$.

A more detailed proof can be found in [18]. □

### 3.3 Concluding the Proof

Finally, we need the fact that every term $M$ of $\lambda$G3 that we wish to prove strongly normalising can be encoded into a strongly normalising term of $\lambda I$, to start off the simulations. The following method works:

1. Encode the term $M$ as a strongly normalising $\lambda$-term $t$, such that no subterm is lost, i.e. *not* using implicit substitutions.
2. Using a translation $\mathsf{i}$ from $\lambda$-calculus to $\lambda I$, introduced in this section, prove that $\mathsf{i}(t)$ reduces to one of the non-deterministic encodings of $M$ in $\lambda I$, that is, that there is a term $T$ such that $M \mathrel{\mathcal{H}} T$ and $\mathsf{i}(t) \longrightarrow^{*}_{\beta,\pi} T$.

The technique is summarised in Figure 5.

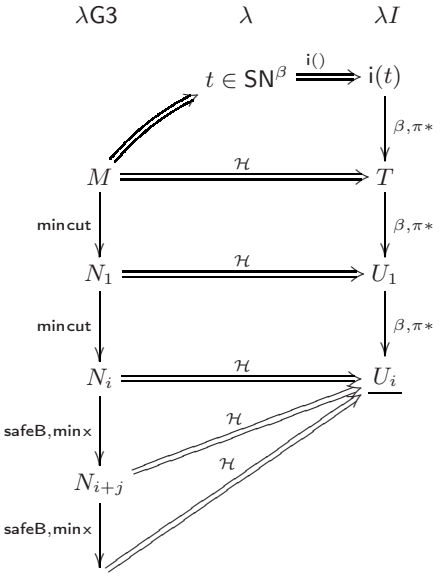



**Fig. 5.** The general technique to prove that $M \in \mathsf{SN}$

**Definition 6 (Encoding of λG3 into λ-calculus).** *We encode the λG3 into λ-calculus by slightly refining Gentzen's encoding as follows:*

$$
\begin{array}{lcll}
\mathcal{G}^{1\uparrow}(x) & := & x & \\
\mathcal{G}^{1\uparrow}(\lambda x.M) & := & \lambda x.\mathcal{G}^{1\uparrow}(M) & \\
\mathcal{G}^{1\uparrow}(\langle xM/y\rangle N) & := & \left\{{}^{x\ \mathcal{G}^{1\uparrow}(M)}\!/_{y}\right\}\mathcal{G}^{1\uparrow}(N) & \text{if } y \in \mathsf{FV}(N) \\
\mathcal{G}^{1\uparrow}(\langle xM/y\rangle N) & := & (\lambda y.\mathcal{G}^{1\uparrow}(N))\ (x\ \mathcal{G}^{1\uparrow}(M)) & \text{if } y \notin \mathsf{FV}(N) \\
\mathcal{G}^{1\uparrow}([M/x]N) & := & \left\{{}^{\mathcal{G}^{1\uparrow}(M)}\!/_{x}\right\}\mathcal{G}^{1\uparrow}(N) & \text{if } x \in \mathsf{FV}(N) \\
\mathcal{G}^{1\uparrow}([M/x]N) & := & (\lambda x.\mathcal{G}^{1\uparrow}(N))\ \mathcal{G}^{1\uparrow}(M) & \text{if } x \notin \mathsf{FV}(N)
\end{array}
$$

The reason why the above encoding is interesting for strong normalisation of some λG3-terms lies in two facts:

**Lemma 5**

1. *For the strong normalisation of typed terms:*
   *If* $\Gamma \vdash_{\lambda G3} M : A$ *then* $\Gamma \vdash_{\lambda} \mathcal{G}^{1\uparrow}(M) : A$
2. *For proving PSN:*
   $\mathcal{G}^{1\uparrow}(\mathcal{P}r(t)) = t$.

*Proof.* Straightforward inductions on $M$ and $t$. □

Now we recall from [21,22] an encoding of $\lambda$-calculus into $\lambda I$[4]:

**Definition 7 (Encoding of $\lambda$-calculus into $\lambda I$).** *We encode the $\lambda$-calculus into $\lambda I$ as follows:*

$$\begin{array}{lcll} i(x) & := & x & \\ i(\lambda x.t) & := & \lambda x.i(t) & \text{if } x \in \mathsf{FV}(t) \\ i(\lambda x.t) & := & \lambda x.[i(t), x] & \text{if } x \notin \mathsf{FV}(t) \\ i(t\ u) & := & i(t)\ i(u) & \end{array}$$

The above encodings satisfy the following properties:

**Lemma 6.** *For any $\lambda G3$-term $M$, there is a $\lambda I$-term $T$ such that $M \mathcal{H} T$ and $i(\mathcal{G}^{1\uparrow}(M)) \longrightarrow^*_{\beta,\pi} T$.*

*Proof.* By induction on $M$. □

**Theorem 8 ([21,22]).** *For any $\lambda$-term $t$, if $t \in \mathsf{SN}^\beta$, then $i(t) \in \mathsf{SN}^{\beta,\pi}$.*

Hence we get

**Corollary 1.** *If $\mathcal{G}^{1\uparrow}(M) \in \mathsf{SN}^\beta$, then $M \in \mathsf{SN}^{cut}$.*

*Proof.* Suppose $\mathcal{G}^{1\uparrow}(M) \in \mathsf{SN}^\beta$. Then by Theorem 8, $i(\mathcal{G}^{1\uparrow}(M)) \in \mathsf{SN}^{\beta,\pi}$, and so by Lemma 6, there is a $\lambda I$-term $T$ such that $M \mathcal{H} T$ and $T \in \mathsf{SN}^{\beta,\pi}$. Now apply Theorem 5 with Theorem 7 and Lemma 2. □

Finally this gives the two strong normalisation results:

**Theorem 9 (Strong normalisation and PSN)**
*If $\Gamma \vdash_{\lambda G3} M : A$, or if $M = \mathcal{P}r(t)$ with $t \in \mathsf{SN}^\beta$, then $M \in \mathsf{SN}^{cut}$.*

*Proof.* It suffices to combine Lemma 5 and Corollary 1. □

## 4 Related Work

In this section we discuss related work on strong normalisation of cut-elimination procedures. We focus on those cut-elimination procedures that have the ability to simulate $\beta$-reduction of the simply-typed $\lambda$-calculus.

Danos et al. [8,9] introduced strongly normalising cut-elimination procedures in sequent calculi for classical logic. The cut-elimination procedures include global proof transformations analogous to proof transformations in natural deduction. In rewrite systems for proof-terms, such cut-elimination procedures are implemented by reduction rules that use meta-operations like implicit

[4] Note that a similar encoding (without the case distinction for abstractions) can be found in [19]; unfortunately we have found it necessary to twist it to prove Theorem 8, which we have not found in the literature.

substitution in the $\lambda$-calculus. Urban [31] described a cut-elimination procedure for the classical sequent calculus in such a rewrite system. Many strong normalisation results of cut-elimination procedures with global proof transformations in the literature can be derived from Urban's result, in both classical and intuitionistic cases, including those for systems in the style of $\overline{\lambda}\mu\tilde{\mu}$ [7] (cf. e.g. [20]).

On the other hand, strong normalisation of cut-elimination procedures consisting of Gentzen-style local proof transformations requires us to use techniques from the field of calculi with explicit substitutions. Urban [31] proved strong normalisation of such a cut-elimination procedure using the technique of [5] and the strong normalisation result of his procedure with global proof transformations mentioned above. The cut-elimination procedure involves *labelled* cut, which are allowed to pass over usual cuts. In the present paper, cut-elimination uses only one kind of cut, and does not seem to be directly simulated by Urban's cut-elimination procedure. For example, the rule ($\mathsf{Perm}_1$) corresponds to a permutation of labelled cuts, which is not included in Urban's reduction rules.

Another example of a cut-elimination procedure that consists of local proof transformations is the one by Dyckhoff and Urban [11] for Herbelin's sequent calculus [12]. Our method of proving strong normalisation works also for their system without using a simulation in $\lambda I$. For the details, see [21,22].

Recently, Nakazawa [23] introduced a cut-elimination procedure for a standard intuitionistic sequent calculus, which is close to ours. The main difference between the two cut-elimination procedures is as follows. In his cut-elimination procedure, the redex $[\lambda z.P/x]\langle \underline{x}M/y\rangle N$ of the rule ($\mathsf{B}$) is reduced to $[M/z][P/y]N$, while it is reduced to $[[M/z]P/y]N$ in our cut-elimination procedure. This difference corresponds to the order of applications of cuts in the resulting proofs. Strong normalisation of his cut-elimination procedure was proved by an inductive method as in [4], but it does not work for our rule ($\mathsf{B}$) as explained in Section 6 of [23]. Another difference is that his cut-elimination procedure does not entirely follow Gentzen-style local steps; the cut-permutation rules of his cut-elimination procedure can be decomposed into two steps of ours (cf. Notes 3 and 4 of [23]).

## 5 Conclusion

We have proved strong normalisation of a cut-elimination procedure for a standard intuitionistic sequent calculus, by using the safeness and minimality technique and a simulation in $\lambda I$, both of which are formalised in [21,22]. We have also established the $\mathsf{PSN}$ property of the type-free terms with respect to $\beta$-reduction through a Prawitz-style translation from the type-free $\lambda$-terms.

We consider our cut-elimination procedure for the intuitionistic sequent calculus as a canonical one, since it is strong normalising and confluent, consists of completely local steps (without an extra kind of cut), and can simulate $\beta$-reduction. For future work, it will be interesting to show strong normalisation of more liberal cut-elimination procedures such as the one in [16].

## References


1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoret. Comput. Sci. 236(1–2), 133–178 (2000)
2. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, Cambridge (1998)
3. Benaissa, Z., Briaud, D., Lescanne, P., Rouyer-Degli, J.: $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. J. Funct. Programming 6(5), 699–722 (1996)
4. Bloo, R.: Preservation of Termination for Explicit Substitution. PhD thesis, Technische Universiteit Eindhoven, IPA Dissertation Series 1997-05 (1997)
5. Bloo, R., Geuvers, H.: Explicit substitution: On the edge of strong normalization. Theoret. Comput. Sci. 211(1–2), 375–395 (1999)
6. Church, A.: The Calculi of Lambda Conversion. Princeton University Press, Princeton (1941)
7. Curien, P.-L., Herbelin, H.: The duality of computation. In: Proc. of the 5th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP 2000), pp. 233–243. ACM Press, New York (2000)
8. Danos, V., Joinet, J.-B., Schellinx, H.: LKQ and LKT: Sequent calculi for second order logic based upon dual linear decompositions of classical implication. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Proc. of the Work. on Advances in Linear Logic. London Math. Soc. Lecture Note Ser., vol. 222, pp. 211–224. Cambridge University Press, Cambridge (1995)
9. Danos, V., Joinet, J.-B., Schellinx, H.: A new deconstructive logic: Linear logic. J. of Symbolic Logic 62(3), 755–807 (1997)
10. Dyckhoff, R., Pinto, L.: Permutability of proofs in intuitionistic sequent calculi. Theoret. Comput. Sci. 212(1–2), 141–155 (1999)
11. Dyckhoff, R., Urban, C.: Strong normalization of Herbelin's explicit substitution calculus with substitution propagation. J. Logic Comput. 13(5), 689–706 (2003)
12. Herbelin, H.: A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 61–75. Springer, Heidelberg (1995)
13. Howard, W.A.: The formulae-as-types notion of construction. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, pp. 479–490. Academic Press, London (1980) (reprint of a manuscript written 1969)
14. Kamin, S., Lévy, J.-J.: Attempts for generalizing the recursive path orderings. Handwritten paper. University of Illinois (1980)
15. Kesner, D., Lengrand, S.: Resource operators for $\lambda$-calculus. Inform. and Comput. 205(4), 419–473 (2007)
16. Kikuchi, K.: On a local-step cut-elimination procedure for the intuitionistic sequent calculus. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 120–134. Springer, Heidelberg (2006)
17. Kikuchi, K.: Confluence of cut-elimination procedures for the intuitionistic sequent calculus. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 398–407. Springer, Heidelberg (2007)
18. Kikuchi, K., Lengrand, S.: Strong normalisation of cut-elimination that simulates $\beta$-reduction - long version, `http://www.lix.polytechnique.fr/~lengrand/Work/`

19. Klop, J.-W.: Combinatory Reduction Systems, Mathematical Centre Tracts, PhD Thesis, vol. 127, CWI, Amsterdam (1980)
20. Lengrand, S.: Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In: Gramlich, B., Lucas, S. (eds.) Post-proc. of the 3rd Int. Work. on Reduction Strategies in Rewriting and Programming (WRS 2003). ENTCS, vol. 86, Elsevier, Amsterdam (2003)
21. Lengrand, S.: Induction principles as the foundation of the theory of normalisation: concepts and techniques. Technical report, Université Paris 7 (March 2005), http://hal.ccsd.cnrs.fr/ccsd-00004358
22. Lengrand, S.: Normalisation & Equivalence in Proof Theory & Type Theory. PhD thesis, Université Paris 7 & University of St. Andrews (2006)
23. Nakazawa, K.: An isomorphism between cut-elimination procedure and proof reduction. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 336–350. Springer, Heidelberg (2007)
24. Nederpelt, R.: Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types. PhD thesis, Eindhoven University of Technology (1973)
25. Pottinger, G.: Normalization as a homomorphic image of cut-elimination. Ann. of Math. Logic 12, 323–357 (1977)
26. Santo, J.E.: Revisiting the correspondence between cut elimination and normalisation. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 600–611. Springer, Heidelberg (2000)
27. Sørensen, M.H.B.: Strong normalization from weak normalization in typed lambda-calculi. Inform. and Comput. 37, 35–71 (1997)
28. Sørensen, M.H.B., Urzyczyn, P.: Lectures on the Curry-Howard Isomorphism. Studies in Logic and the Foundations of Mathematics, vol. 149. Elsevier, Amsterdam (2006)
29. Sørensen, M.H.B., Urzyczyn, P.: Strong cut-elimination in sequent calculus using Klop's $\iota$-translation and perpetual reduction (available from the authors) (submitted for publication, 2007)
30. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, 2nd edn. Cambridge Tracts in Theoret. Comput. Sci., vol. 43. Cambridge University Press, Cambridge (2000)
31. Urban, C.: Classical Logic and Computation. PhD thesis, University of Cambridge (2000)
32. Urban, C., Bierman, G.M.: Strong normalisation of cut-elimination in classical logic. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 365–380. Springer, Heidelberg (1999)
33. van Raamsdonk, F., Severi, P., Sørensen, M.H.B., Xi, H.: Perpetual reductions in $\lambda$-calculus. Inform. and Comput. 149(2), 173–225 (1999)
34. Xi, H.: Weak and strong beta normalisations in typed lambda-calculi. In: de Groote, P., Hindley, J.R. (eds.) TLCA 1997. LNCS, vol. 1210, pp. 390–404. Springer, Heidelberg (1997)
35. Zucker, J.: The correspondence between cut-elimination and normalization. Ann. of Math. Logic 7, 1–156 (1974)

# Symbolic Semantics Revisited*

Filippo Bonchi and Ugo Montanari

Dipartimento di Informatica, Università di Pisa

**Abstract.** *Symbolic bisimulations* were introduced as a mean to define value-passing process calculi using smaller, possibly finite labelled transition systems, equipped with symbolic actions. Similar ideas have been used for modeling with fewer transitions the input behavior of open and asynchronous $\pi$-calculus. In this paper we generalize the symbolic technique and apply the resulting theory to these two cases, re-deriving existing results. We also apply our approach to a new setting, i.e. *open Petri nets*, with the usual result of reducing input transitions. Our theory generalizes Leifer and Milner reactive systems by adding observations.

## 1 Introduction

A compositional interactive system is usually defined as a labelled transition system (LTS) where states are equipped with an algebraic structure. Behavioural equivalence is often defined as bisimilarity, namely the largest bisimulation. Then a key property is that bisimilarity be a congruence, i.e. that abstract semantics respects the algebraic operations.

When this is not the case for some operations, the obvious fix is to define the abstract semantics as the largest bisimulation which is closed for those operations. An equivalent approach is to introduce additional moves of the form $p \stackrel{c,a}{\longrightarrow} q$, for every context $c$ built with the faulty operations, whenever $c(p) \stackrel{a}{\longrightarrow} q$ is a transition in the original LTS. If we call *saturated* the resulting LTS, we have that ordinary bisimilarity on the saturated LTS coincides with the largest bisimulation (on the original LTS) which is closed for the faulty operations. By analogy we call the latter *saturated bisimilarity*.

This idea was originally introduced by the second author and Sassone in [18]. They define *dynamic bisimilarity* in order to make weak bisimilarity of CCS [14] a congruence w.r.t. non-deterministic choices: before any transition, the observer inserts the processes into all possible choice contexts. Analogously, since late and early bisimilarity of $\pi$-calculus [16] are not preserved under substitution (and thus under input prefixes), in [21] Sangiorgi introduces *open bisimilarity* ($\sim^o$) as the largest bisimulation on $\pi$-calculus agents which is closed under substitutions.

Another example of saturated bisimilarity is $\sim^1$ [1] for the asynchronous $\pi$-calculus [1,9]. Here the basic bisimilarity, namely $o\tau$-bisimilarity, is not a congruence under parallel outputs, and thus at any step of definition of $\sim^1$ the observer

* Research partially supported by the IST 2004-16004 SENSORIA, and the MIUR PRIN 2005015824 ART.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 395–412, 2008.

inserts the process in parallel with all possible outputs. In the same way, $\sim^N$ has been defined in [3] amongst open Petri nets [10,2,12] that are an interactive version of P/T Petri nets.

Here we introduce a general model encompassing the three examples above.

The definition of saturated bisimilarity as ordinary bisimulation on the saturated LTS, while in principle operational, often makes infinite state the portion of LTS reachable by any nontrivial agent, and in any case is very inefficient, since it introduces a large number of additional transitions.

Inspired by Hennessy and Lin [8], who introduced a *symbolic* semantics of value passing calculi, Sangiorgi defines in [21] a symbolic transition system and a new notion of bisimilarity that coincides with $\sim^O$. Analogously in [1], Amadio et al. defined asynchronous bisimilarity that coincides with $\sim^1$.

We define symbolic transition system and symbolic bisimilarity and we show that the latter coincides with saturated bisimilarity. We also show that the results by Sangiorgi and Amadio et al. are special cases. Concerning Petri nets, no symbolic semantics exists. Here our framework produces a new symbolic semantics (equivalent to the standard $\sim^N$) by reducing input transitions.

Our construction employs some general knowledge about the modeled formalism. For instance, we know that in $\pi$-calculus (without mismatch):

$$\text{“}\forall \text{ process } p \text{ and substitution } \sigma, \text{ if } p \xrightarrow{\mu} q \text{ then } \sigma(p) \xrightarrow{\sigma(\mu)} \sigma(q) \text{ ”} \quad [16].$$

Thus, if in the saturated LTS, $p \xrightarrow{\phi,\mu} p'$ (meaning that $\phi(p) \xrightarrow{\mu} p'$), then surely also $p \xrightarrow{\psi(\phi),\psi(\mu)} \psi(p')$. The second transition is to some extent *redundant*, i.e., we can ignore it without changing the saturated bisimilarity. For any formalism, we identify a set of rules (given in a fixed format) expressing how contexts modify transitions and we prune the saturated LTS by employing these rules.

Our results have been inspired also by the theory of reactive systems by Leifer and Milner [11]. Their aim is to take a transition system (expressed through reaction rules) without any observations and to (automatically) derive a labeled transition system in such a way that bisimilarity is a congruence. The idea is to take as labels the *minimal* contexts that enable a reaction.

Reactive system theory has been applied to several interesting formalisms, but only rarely the canonical abstract semantics have been retrieved (amongst these, CCS in [4] and Petri nets [15,22]). In our opinion, labels cannot represent both *interactions* and *observations*, because these two concepts are different, like for example, in the asynchronous calculi where the input interaction is not observable. Thus we believe that some notion of observation, either on transitions or on states (e.g. barbs [17,20]), is necessary. In this sense we can say that our theory generalizes reactive systems by adding observations. The special case of reactive systems can be retrieved from our approach by starting from an unlabeled transition system with the rule:

$$\text{“}\forall \text{ process } p \text{ and reactive context } d, \text{ if } p \longrightarrow q \text{ then } d(p) \longrightarrow d(q)\text{”}.$$

In Sec. 2, we recall open and asynchronous bisimilarities of $\pi$-calculus. In Sec. 3, we introduce our theory and, in Sec. 4, we apply it to $\pi$-calculus. In Sec. 5, we

introduce open Petri nets and, applying our approach, we get a new symbolic semantics. In Sec. 6 we show how our theory generalizes reactive systems and, in Sec. 7, we outline conclusions and future works.

## 2 Background on $\pi$-Calculus

Let $\mathcal{N}$ be a set of *names* (ranged over by $a, b, c \ldots$) with $\tau \notin \mathcal{N}$. The set of $\pi$-processes is defined by the following grammar:

$$p ::= \mathbf{0},\ \ \alpha.p,\ \ [a=b]p,\ \ p_1 \mid p_2,\ \ p_1 + p_2,\ \ \nu a.p,\ \ !p, \qquad \alpha ::= a(b),\ \ \overline{a}b,\ \ \tau$$

Considering $a(b).p$ and $\nu b.p$, the occurrences of $b$ in $p$ are bound. An occurrence of a name in a process is *free*, if it is not bound. The set of *free names* of $p$ (denoted by $\mathrm{fn}(p)$) is the set of names that have a free occurrence in the process $p$. The process $p$ is $\alpha$-equivalent to $q$ (written $p \equiv_\alpha q$), if they are equivalent up to $\alpha$-renaming of bound occurrences of names. The operational semantics of $\pi$-calculus is a transition system labeled on actions $Act = \{a(b), \overline{a}b, \overline{a}(b), \tau \mid a, b \in \mathcal{N}\}$ (ranged over by $\mu$) where $b$ is a *bound name* (written $b \in \mathrm{bn}(\mu)$) in $a(b)$ and $\overline{a}(b)$. In all the other cases $a$ and $b$ are free in $\mu$ ($a, b \in \mathrm{fn}(\mu)$). By $\mathrm{nm}(\mu)$ we denote the set of both free and bound names of $\mu$. The same notation will be used later for match sequences, distinctions and substitutions.

**Table 1.** Late operational semantics of $\pi$-calculus

$$\text{(PRE)}\ \alpha.p \xrightarrow{\alpha} p \qquad \text{(COM)}\ \frac{p \xrightarrow{\overline{a}b} p' \quad q \xrightarrow{a(x)} q'}{p \mid q \xrightarrow{\tau} p' \mid q'\{^b/_x\}} \qquad \text{(PAR)}\ \frac{p \xrightarrow{\mu} p'}{p \mid q \xrightarrow{\mu} p' \mid q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(q) = \varnothing$$

$$\text{(SUM)}\ \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \text{(OPN)}\ \frac{p \xrightarrow{\overline{a}b} p'}{\nu b.p \xrightarrow{\overline{a}(b)} p'}\ b \neq a \qquad \text{(RES)}\ \frac{p \xrightarrow{\mu} p'}{\nu b.p \xrightarrow{\mu} \nu b.p'}\ b \notin \mathrm{nm}(\mu)$$

$$\text{(REP)}\ \frac{p \mid p! \xrightarrow{\mu} q}{p! \xrightarrow{\mu} q} \qquad \text{(MAT)}\ \frac{p \xrightarrow{\mu} p'}{[a=a]p \xrightarrow{\mu} p'} \qquad \text{(CLS)}\ \frac{p \xrightarrow{\overline{a}(x)} p' \quad q \xrightarrow{a(x)} q'}{p \mid q \xrightarrow{\tau} \nu x.p' \mid q'}$$

The (late) labeled transition system is inductively defined by the rules in Table 1, where we have omitted the symmetric version of the rules SUM, PAR, COM and CLS and where we consider processes up to $\alpha$-equivalence.

**Open Bisimilarity.** In [16], the authors introduce late and early bisimilarities. These are congruences w.r.t. parallel composition, but they are not preserved by the input prefixes. Consider the processes $p = \overline{a}b \mid c(x)$ and $q = \overline{a}b.c(x) + c(y).\overline{a}b$ (here and in the following we abbreviate $\alpha.\mathbf{0}$ with $\alpha$). These are (late and early) bisimilar, but whenever we put them into the context $d(a).-$, they are not anymore. Indeed if this prefix receives $c$, then $a = c$, and thus $p$ can perform a $\tau$ action (synchronizing the two parallel components), while $q$ cannot.

Sangiorgi in [21] introduces open bisimilarity ($\sim^O$) that is a congruence with respect to all the operators. In $\sim^O$, name instantiation is part of the coinductive

definition of bisimilarity. At any step of the bisimulation game, names can be identified by a substitution $\sigma$. Thus, $[a = b]\tau$ and $\mathbf{0}$ are not considered bisimilar anymore, because, $\sigma([a = b]\tau)$ perform a $\tau$ transition if $\sigma$ identifies $a$ and $b$. Now consider $\nu a.[a = b]\tau$. It will never perform a $\tau$ transition, because $a$ is restricted and then it cannot be identified with $b$. Thus in the bisimulation game, we have to avoid those substitutions that identify $a$ and $b$. In order to properly handle the restriction operator, we have to introduce *distinctions*, i.e. relations that express permanent inequalities on names.

**Definition 1 (Distinction).** *A* distinction $D$ *is a finite symmetric and irreflexive relation on names. A substitution* $\sigma$ respects $D$ *iff* $aDb$ *implies* $\sigma(a) \neq \sigma(b)$.

In the following we will use $\mathcal{D}$ to mean the set of all distinctions and $\sigma(D)$ to mean the distinction $\{(\sigma(a), \sigma(b)) \mid (a, b) \in D\}$. Sometimes, in the expressions defining distinctions we shall avoid to give all the symmetric pairs; for instance, we might define $D = \{(a, b)\}$ without recalling that also $(b, a) \in D$. In the following definitions, a name declared *fresh* is supposed to be different from all the others appearing in the definition.

**Definition 2.** *Let* $R = \{R_D \mid D \in \mathcal{D}\}$ *be a* $\mathcal{D}$ *sorted family of symmetric relations.* $R$ *is an* open bisimulation *iff* $\forall D \in \mathcal{D}$ *and* $\forall \sigma$ *respecting* $D$*, whenever* $pR_Dq$*:*

- *if* $\sigma(p) \xrightarrow{\alpha} p'$ *with* $\mathrm{bn}(\alpha)$ *fresh, then* $\sigma(q) \xrightarrow{\alpha} q'$ *and* $p'R_{\sigma(D)}q'$,
- *if* $\sigma(p) \xrightarrow{\overline{a}(b)} p'$ *with* $b$ *fresh, then* $\sigma(q) \xrightarrow{\overline{a}(b)}$ *and* $p'R_{D^*}q'$

*where* $D^* = \sigma(D) \cup \{(b, i), \ \forall i \in \mathrm{fn}(\sigma(p) \cup \sigma(q))\}$.

*We write* $p \sim_D^O q$*, if there is an open bisimulation* $R$ *such that* $pR_Dq$.

The intuitive meaning of the last clause, is that $b$ is different from all the other free names appearing in $\sigma(p)$ and $\sigma(q)$ since it has been generated by some restriction $\nu b$. Thus we have to check that the arriving states $p'$ and $q'$ are bisimilar when considering $b$ distinct form all the other names.

The definition of $\sim^O$ involves at each step a quantification over all substitutions. In [21], the author introduces a more efficient characterization of $\sim^O$, by defining a symbolic transition system. Labels on this LTS are pairs $(M, \mu)$ where $M$ is a *match sequence* and $\mu$ is an action. A match sequence (ranged over by $M$, $N$) is a sequence of equalities of names of the form $[a = b]$. We will write $MN$ to denote the concatenation of $M$ and $N$ and $M \rhd N$ if $M$ implies $N$, i.e., whenever $M$ holds, also $N$ holds. Every matching sequence $M$ defines an equivalence relation $E_M$. We denote by $\sigma^M$ a special substitution that chooses a representative for each equivalence class of $E_M$, and maps every name in the representative of its class. Note that there may exists more than one $\sigma^M$, we just choose one of them.

The symbolic transition system is presented in Table 2. In the transition $p \xrightarrow{M,\mu}_e p'$, $M$ represents the minimal substitution $\sigma^M$ that allows $p$ to perform

**Table 2.** Symbolic transition system for open $\pi$-calculus

$$(\text{PRE})\ \alpha.p \xrightarrow{\varnothing,\alpha}_e p \qquad (\text{CLS})\ \frac{p \xrightarrow{M,\overline{a}(x)}_e p' \quad q \xrightarrow{N,b(x)}_e q'}{p \mid q \xrightarrow{MN[a=b],\tau}_e \nu x.p' \mid q'} \qquad (\text{SUM})\ \frac{p \xrightarrow{M,\mu}_e p'}{p+q \xrightarrow{M,\mu}_e p'}$$

$$(\text{PAR})\ \frac{p \xrightarrow{M,\mu}_e p'}{p \mid q \xrightarrow{M,\mu}_e p' \mid q}\ \text{bn}(\mu) \cap \text{fn}(q) = \varnothing \qquad (\text{COM})\ \frac{p \xrightarrow{M,\overline{a}b}_e p' \quad q \xrightarrow{N,c(d)}_e q'}{p \mid q \xrightarrow{MN[a=c],\tau}_e p' \mid q'\{^b/_d\}}$$

$$(\text{REP})\ \frac{p \mid p! \xrightarrow{M,\mu}_e q}{p! \xrightarrow{M,\mu}_e q} \qquad (\text{MAT})\ \frac{p \xrightarrow{M,\mu}_e p'}{[a=b]p \xrightarrow{M[a=b],\mu}_e p'}$$

$$(\text{RES})\ \frac{p \xrightarrow{M,\mu}_e p'}{\nu b.p \xrightarrow{M,\mu}_e \nu b.p'}\ b \notin \text{nm}(M \cup \mu) \qquad (\text{OPN})\ \frac{p \xrightarrow{M,\overline{a}b}_e p'}{\nu b.p \xrightarrow{M,\overline{a}(b)}_e p'}\ b \notin \text{nm}(m) \cup \{a\}$$

$\mu$. As an example consider the processes $p = [a=b]\tau$ and $q = p + [c=d][a=b]\tau$. The process $q$ can perform the transitions $q \xrightarrow{[a=b],\tau}_e \mathbf{0}$ and $q \xrightarrow{[a=b][c=d],\tau}_e \mathbf{0}$, while $p$ performs only the former transition. However $p \sim^O q$.

**Definition 3.** *Let $R = \{R_D \mid D \in \mathcal{D}\}$ be a $\mathcal{D}$ sorted family of symmetric relations. $R$ is an* efficient open bisimulation *iff $\forall D \in \mathcal{D}$, whenever $pR_Dq$:*

- *if $p \xrightarrow{M,\alpha}_e p'$ with $\text{bn}(\alpha)$ fresh and $M$ respects $D$, then $q \xrightarrow{N,\alpha'}_e q'$ such that $M \rhd N$, $\sigma^M(\alpha) \equiv_\alpha \sigma^M(\alpha')$ and $\sigma^M(p')R_{\sigma^M(D)}\sigma^M(q')$,*
- *if $p \xrightarrow{M,\overline{a}(b)}_e p'$ with $b$ fresh and $M$ respects $D$, then $q \xrightarrow{N,\overline{c}(b)}_e q'$ such that $M \rhd N$, $\sigma^M(\overline{a}(b)) \equiv_\alpha \sigma^M(\overline{c}(b))$ and $\sigma^M(p')R_{D^*_M}\sigma^M(q')$*

*where $D^*_M = \sigma^M(D) \cup \{(b,i),\ \forall i \in \text{fn}(\sigma^M(p) \cup \sigma^M(q))\}$.*

*We write $p \asymp_D q$, if there is an efficient open bisimulation $R$ such that $pR_Dq$.*

Intuitively the above clauses ensure that in the ordinary transition system, the move $\sigma^M(p) \xrightarrow{\sigma^M(\alpha)} \sigma^M(p')$ is matched by $\sigma^M(q) \xrightarrow{\sigma^M(\alpha')} \sigma^M(q')$. In [21], it is proved that $\asymp$ and $\sim^O$ coincide, but the former is more efficient than the latter, since $\asymp$ forces only those fusions of names which are strictly necessary to ensure the equivalence, while $\sim^O$ forces all the fusions.

**The asynchronous fragment.** The asynchronous $\pi$-calculus [9,1] is defined as a subset of $\pi$, without output prefixes and outputs in choice points. Formally:

$$p ::= \overline{a}b,\ \ p_1 \mid p_2,\ \ \nu a.p,\ \ !g,\ \ g \qquad g ::= \mathbf{0},\ \ \alpha.p,\ \ g_1 + g_2 \qquad \alpha ::= a(b),\ \ \tau$$

The operational semantics is obtained by replacing the rules (SUM) and (REP) of Table 1 with the three rules of Table 3. The main difference with standard $\pi$ is in the notion of observation. Indeed in the asynchronous case, input are not observable. Intuitively an observer that sends a message to a system, cannot know if the system has received it. Thus bisimulation ignores input transitions.

**Table 3.** Operational semantics of the asynchronous $\pi$-calculus

$$(\text{OUT})\ \overline{a}b \xrightarrow{\overline{a}b} \mathbf{0} \qquad (\text{SUM})\ \frac{g \xrightarrow{\mu} p}{g + g' \xrightarrow{\mu} p'} \qquad (\text{REP})\ \frac{g \xrightarrow{\mu} p'}{!g \xrightarrow{\mu} p' | !g}$$

**Definition 4 ($o\tau$-bisimilarity).** *A symmetric relation $R$ is an $o\tau$-bisimulation iff, whenever $pRq$, if $p \xrightarrow{\mu} p'$ where $\mu$ is not an input action and $\mathrm{bn}(\alpha)$ is fresh, then $q \xrightarrow{\mu} q'$ and $p'Rq'$. Let $\sim^{o\tau}$ be the largest $o\tau$-bisimulation.*

Note that $a(x).\overline{c}x \sim^{o\tau} a(x).\overline{d}x$, even if the two processes are really different when they are put in parallel with a process $\overline{a}b$. In order to obtain an abstract semantics preserved under parallel composition, we proceed analogously to open bisimilarity, i.e. at any step of the bisimulation we put the process in parallel with all possible outputs (in the open we apply all possible substitutions).

**Definition 5 (1-bisimilarity).** *An 1-bisimulation is an $o\tau$-bisimulation $R$ such that $pRq$ implies $\forall \overline{a}b$, $(\overline{a}b \mid p)\, R\, (\overline{a}b \mid q)$. Let $\sim^1$ be the largest 1-bisimulation.*

**Definition 6 (asynchronous bisimilarity).** *A symmetric relation $R$ is an asynchronous bisimulation iff it is an $o\tau$-bisimulation and whenever $pRq$, if $p \xrightarrow{a(b)} p'$, then either $q \xrightarrow{a(b)} q'$ and $p'Rq'$, or $q \xrightarrow{\tau} q'$ and $p'R(q' \mid \overline{a}b)$. Let $\sim^a$ be the largest asynchronous bisimulation*[1]*.*

In [1], it is proved that $\sim^1 = \sim^a$. Our theory will formally show that $\sim^1 = \sim^a$ and $\sim^O = \asymp$ are two instances of the same general concept. The abstract semantics $\sim^1$ and $\sim^O$ are saturated, i.e., obtained by closing w.r.t. all contexts, while $\sim^a$ and $\asymp$ are symbolic, i.e., efficient characterizations of the saturated ones.

## 3 Saturated and Symbolic Semantics

A *closed many-sorted unary signature* $(S, \Sigma)$ consists of a set of sorts $S$, and an $S \times S$ sorted family $\Sigma = \{\Sigma_{s,t} \mid s, t \in S\}$ of sets of operation symbols which are closed under composition, that is if $f \in \Sigma_{s,t}$ and $g \in \Sigma_{t,u}$, then $g \circ f \in \Sigma_{s,u}$. Given $f \in \Sigma_{u,v}, g \in \Sigma_{t,u}, h \in \Sigma_{s,t}$, $f \circ (g \circ h) = (f \circ g) \circ h$ and moreover $\forall s \in S$, $\exists id_s \in \Sigma_{s,s}$ such that $\forall f \in \Sigma_{s,t}$, $id_t \circ f = f$ and $f \circ id_s = f$. A $(S, \Sigma)$-algebra $\mathbb{A}$ consists of an $S$ sorted family $|\mathbb{A}| = \{A_s \mid s \in S\}$ of sets and a function $f_{\mathbb{A}} : A_s \to A_t$ for all $f \in \Sigma_{s,t}$ such that $(g \circ f)_{\mathbb{A}} = g_{\mathbb{A}}(f_{\mathbb{A}}(-))$ and $id_{s\mathbb{A}}$ is the identity function on $A_s$[2]. When $\mathbb{A}$ is clear from the context, we will write $f$ to mean $f_{\mathbb{A}}$, and we will write $A_s$ to mean the set of sort $s$ of the family $|\mathbb{A}|$. Given $f \in \Sigma_{s,t}$, we call $s$ the *source* of $f$, and $t$ the *target*.

[1] In [1], $\sim^a$ was originally defined in the early LTS. The above definition, however coincides with the original $\sim^a$ because $\sim^a = \sim^g$ (Corollary 1, [1]).

[2] A closed many-sorted unary signature $(S, \Sigma)$ is a category $\mathbf{C}$ and a $(S, \Sigma)$-algebra is a presheaf on $\mathbf{C}$. We adopt the above notation to be accessible to a wider audience.

In order to develop a general theory of bisimulation, we introduce *context interactive systems*. In our theory, an interactive system is a state-machine that can interact with the environment (contexts) through an evolving interface.

**Definition 7 (Context Interactive System).** *A* context interactive system *$\mathcal{I}$ is a quadruple $\langle (S, \Sigma), \mathbb{A}, O, tr \rangle$ where:*

- *$(S, \Sigma)$ is a closed many-sorted unary signature,*
- *$\mathbb{A}$ is a $(S, \Sigma)$-algebra,*
- *$O$ is a set of observations,*
- *$tr \subseteq |\mathbb{A}| \times O \times |\mathbb{A}|$ is a labeled transition relation ($p \xrightarrow{o} p'$ means $(p, o, p') \in tr$).*

Roughly speaking sorts are interfaces of the system, while operators of $\Sigma$ are contexts. Every state $p$ with interface $s$ (i.e. $p \in A_s$) can be inserted into the context $c \in \Sigma_{s,t}$, obtaining $c_{\mathbb{A}}(p)$ that has interface $t$. Every state can evolve into a new state (possibly with different interface) producing an observation $o \in O$.

The abstract semantics of interactive systems is usually defined through behavioural equivalences. In this paper we propose a general notion of bisimilarity. The idea is that two states of a system are equivalent if they are indistinguishable from an external observer that, in any moment of their execution, can insert them into some environment and then observe some transition.

**Definition 8 (Saturated Bisimilarity).** *Let $\mathcal{I} = \langle (S, \Sigma), \mathbb{A}, O, tr \rangle$ be a context interactive system. Let $R = \{R_s \subseteq A_s \times A_s \mid s \in S\}$ be an $S$ sorted family of symmetric relations. $R$ is a* saturated bisimulation *iff, $\forall s, t \in S$, $\forall c \in \Sigma_{s,t}$, whenever $pR_sq$, if $c_{\mathbb{A}}(p) \xrightarrow{o} p'$, then $c_{\mathbb{A}}(q) \xrightarrow{o} q'$ and $p'Rq'$.*

*We write $p \sim_s^S q$ iff there is a saturated bisimulation $R$ such that $pR_sq$.*

An alternative but equivalent definition can be given by defining the *saturated transition system* (SATTS) as follows: $p \xrightarrow{c;o}_S q$ if and only if $c(p) \xrightarrow{o} q$. Trivially the standard bisimilarity over SATTS coincides with $\sim^S$.

**Proposition 1.** *$\sim^S$ is the coarsest bisimulation congruence.*

Saturated bisimulation is a good notion of equivalence but it is hard to check, since it involves a quantification over all contexts. A solution out of the impasse is suggested by $\asymp$. We can define a symbolic transition system where transitions are labeled both with the usual observation and also with the minimal context (substitutions, in the case of open $\pi$) that allows the transition.

**Definition 9 (Symbolic Context Transition System).** *A* symbolic context transition system *(SCTS for short) for a system $\mathcal{I} = \langle (S, \Sigma), \mathbb{A}, O, tr \rangle$ is a transition system $\beta \subseteq |\mathbb{A}| \times \Sigma \times O \times |\mathbb{A}|$.*

In SCTS, we have two labels with different tastes. The first label is a context that tells us when the transition can be performed. We call this label the *interaction*, while the second is the *observation* produced by the transition. It is worth to note that in some formalisms interactions and observations coincide and thus

only one label is necessary. However, in the general case, the two concepts are distinct as it is the case of asynchronous formalisms, where the input interaction cannot be observed. In the asynchronous $\pi$, the transition $a(x) \xrightarrow{a(x)} \mathbf{0}$, can be seen as $a(x) \xrightarrow{-|\overline{a}b,\tau}_{\beta} \mathbf{0}$, where $- \mid \overline{a}b$ is the interaction (i.e., the minimal context that allows the transition) and $\tau$ is the observation.

The intuitive meaning of such a transition is that for all larger contexts $- \mid \overline{a}b \mid Q$, we have that $a(x) \mid \overline{a}b \mid Q \xrightarrow{\tau} \mathbf{0} \mid Q$. The same happens in the symbolic LTS of $\sim^O$, where the transition $[a = b]\overline{c}d \xrightarrow{[a=b],\overline{c}d}_{e} \mathbf{0}$ roughly means that for all substitution $\sigma$ that equate $a$ and $b$, $\sigma([a = b]\overline{c}d) \xrightarrow{\sigma(\overline{c}d)} \sigma(\mathbf{0})$.

It is worth to note that there is a difference between the asynchronous and the open case. In the former, the insertion inside the context does not modify the observation, while in the open, the substitution is applied also to the observation. In general, contexts can modify in many different ways the execution of a transition. For example in the $\pi$-calculus, the prefix contexts $\alpha.-$ inhibits all the transitions (i.e. if $p \xrightarrow{\mu} p'$, then $\alpha.p \not\xrightarrow{\mu}$), while the context $\nu a.-$ inhibits only the transitions with subject $a$. For this reason we need a set of rules that formally describes how contexts modify transitions. Hereafter we fix the following format.

$$\frac{p_s \xrightarrow{o} q_t}{c(p_s) \xrightarrow{o'} d(q_t)}$$

This rule states that all processes with sort $s$ that perform a transition with observation $o$ going into $q_t$, when inserted into the context $c \in \Sigma_{s,s'}$ can perform a transition with the observation $o'$ going into $d(q_t)$ for some context $d \in \Sigma_{t,t'}$.

In the following, we write $c \xrightarrow[o']{o} d$ to mean a rule like the above. The rules $c \xrightarrow[o']{o} c'$ and $d \xrightarrow[o'']{o'} d'$ *derive* the rule $d \circ c \xrightarrow[o'']{o} d' \circ c'$ if $d \circ c$ and $d' \circ c'$ are defined. Given a set of rules $\mathcal{R}$, $\Phi(\mathcal{R})$ is the set of all the rules derivable from $\mathcal{R}$ together with the identities rules ($\forall o \in O$ and $\forall s, t \in S$, $id_s \xrightarrow[o]{o} id_t$).

**Definition 10 (Saturation).** *Let $\mathcal{I} = \langle (S, \Sigma), \mathbb{A}, O, tr \rangle$ be a context interactive system, $\beta$ an* SCTS *and $\mathcal{R}$ a set of rules of the format described above. The* saturation *of $\beta$ w.r.t. $\mathcal{R}$ (denoted by $\mathcal{R}(\beta)$) is the transition system described as:*

$$\frac{p \xrightarrow{c,o}_{\beta} p' \quad d \xrightarrow[o']{o} e \in \Phi(\mathcal{R})}{p \xrightarrow{d\circ c,o'}_{\mathcal{R}(\beta)} e(p')}.$$

*We say that $\beta$ and $\mathcal{R}$ are* sound and complete *w.r.t. $\mathcal{I}$ if the saturation of $\beta$ w.r.t. $\mathcal{R}$ coincides with* SATTS*, i.e., $p \xrightarrow{c,o}_{\mathcal{R}(\beta)} p'$ iff $p \xrightarrow{c,o}_{S} p'$ (i.e., iff $c(p) \xrightarrow{o} p'$). A transition $p \xrightarrow{c,o} q$* dominates *$p \xrightarrow{c',o'} q'$ if there exists $d \xrightarrow[o']{o} e \in \Phi(\mathcal{R})$ such that $c' = d \circ c$ and $q' = q(e)$.*

A sound and complete SCTS could be considerably smaller than the saturated transition system, but still containing all the information needed to recover $\sim^S$. Note that the standard bisimilarity over SCTS is usually stricter than $\sim^S$. Consider a process $p$ that performs only the transitions $p \xrightarrow{c,o}_\beta p_1$ and $p \xrightarrow{c',o'}_\beta p_2$ such that $c' = d \circ c$, $d \xrightarrow[o']{o} e$, $e(p_1) \sim^S p_2$. Now take a process $q$ that performs only $q \xrightarrow{c,o}_\beta q_1$ such that $p_1 \sim^S q_1$. Clearly $p$ and $q$ are not bisimilar on SCTS, because $p \xrightarrow{c',o'}_\beta p'$ while $q$ cannot. However $p \sim^S q$, because $q \xrightarrow{c',o'}_{\mathcal{R}(\beta)} e(q_1)$ (i.e., $c'(q) \xrightarrow{o'} e(q_1)$) and, since $q_1 \sim^S p_1$, then $e(q_1) \sim^S e(p_1) \sim^S p_2$.

**Definition 11 (Symbolic and Semi-Saturated Bisimilarity).** *Let $\mathcal{I} = \langle (S, \Sigma), \mathbb{A}, O, tr \rangle$ be an interactive system, $\mathcal{R}$ be a set of rules and $\beta$ be a symbolic transition system. Let $R = \{R_s \subseteq A_s \times A_s \mid s \in S\}$ be an $S$ sorted family of symmetric relations. $R$ is a* symbolic bisimulation *iff $\forall s \in S$, whenever $pR_sq$:*

- *if $p \xrightarrow{c',o'}_\beta p'_1$, then $\exists d \xrightarrow[o']{o} d' \in \Phi(\mathcal{R})$, $d \circ c = c'$, $q \xrightarrow{c,o}_\beta q_1$ and $p'_1 R d'(q_1)$.*

*We write $p \sim_s^{SYM} q$ iff there exists a symbolic bisimulation $R$ such that $pR_sq$.* Semi-saturated bisimilarity *($\sim^{SS}$) is obtained replacing the above condition with:*

- *if $p \xrightarrow{c',o'}_\beta p'_1$, then $c'(q) \xrightarrow{o'} q'_1$ and $p'_1 R q'_1$.*

**Theorem 1.** *Let $\mathcal{I}$ be a context interactive system, $\beta$ an* SCTS *and $\mathcal{R}$ a set of rules. If $\beta$ and $\mathcal{R}$ are sound and complete w.r.t. $\mathcal{I}$, then $\sim^{SYM} = \sim^{SS} = \sim^S$.*

## 4 Context Interactive Systems for π-Calculus

In this section we present context interactive systems for asynchronous (4.1) and open (4.2) $\pi$-calculus. In the former, contexts are parallel output processes, saturated bisimilarity coincides with $\sim^1$, while symbolic bisimilarity coincides with $\sim^a$. In the latter, contexts are fusions of names that respect distinctions, saturated bisimilarity coincides with $\sim^O$ and symbolic bisimilarity with $\asymp$.

### 4.1 Asynchronous

We assume the set of names $\mathcal{N}$ to be totally ordered . With $n$ we mean both the nth names and the set of names smaller or equal than $n$. The context interactive system for asynchronous $\pi$-calculus is $\mathcal{A} = \langle (S^{\mathcal{A}}, \Sigma^{\mathcal{A}}), \mathbb{A}, O_{\mathcal{A}}, tr_{\mathcal{A}} \rangle$.

The many-sorted signature $(S^{\mathcal{A}}, \Sigma^{\mathcal{A}})$ is formally defined as

- $S^{\mathcal{A}} = \omega$ (the set of natural numbers);
- $\Sigma^{\mathcal{A}}_{n,m}$ with $m \geq n$ is the set of contexts $c$ generated by $c ::= -,\ - \mid \overline{a}b$, where $a \in n$ and $b \in m$.[3]

[3] $\forall n \in \omega$, $id_n$ is $- \in \Sigma^{\mathcal{A}}_{n,n}$, while $\circ$ is the syntactic composition of contexts.

Let us define the $(S^{\mathcal{A}}, \Sigma^{\mathcal{A}})$-algebra $\mathbb{A}$. For every sort $n$, $A_n$ is the set of asynchronous $\pi$-processes $p$ such that $n \geq max\ \mathrm{fn}(p)$. Then $\forall p \in A_n$ and $\forall c \in \Sigma^{\mathcal{A}}_{n,m}$, $c_{\mathbb{A}}(p)$ is the process of sort $m$ obtained by syntactically inserting $p$ into $c$. In this system, interfaces are sets of names. A process with interface $n$ uses only names in $n$ (not all, just a part) and can be put in parallel with outputs sending messages over $n$. Given a process $p$ and a natural number $n \geq max\ \mathrm{fn}(p)$, we denote with $p_n$ the process $p$ with interface $n$.

The set of observations is $O_{\mathcal{A}} = \{\overline{a}b, \overline{a}(), \tau \mid a, b \in \mathcal{N}\}$. Note that the input action is not an observation, since in the asynchronous case it is not observable. Moreover note that in the bound output, the sent name does not appear. This is because, any process with sort $n$ will send as bound output the name $n+1$.

The following rules define the transition structure $tr_{\mathcal{A}}$ (denoted by $\rightarrow_{\mathcal{A}}$).

$$\frac{p \xrightarrow{\overline{a}b} p'}{p_n \xrightarrow{\overline{a}b}_{\mathcal{A}} p'_n} \qquad \frac{p \xrightarrow{\tau} p'}{p_n \xrightarrow{\tau}_{\mathcal{A}} p'_n} \qquad \frac{p \xrightarrow{\overline{a}(n+1)} p'}{p_n \xrightarrow{\overline{a}()}_{\mathcal{A}} p'_{n+1}}$$

**Proposition 2.** *Let $p, q$ be asynchronous $\pi$-processes, and let $n \geq max\ \mathrm{fn}(p \cup q)$. Then $p \sim^1 q$ iff $p_n \sim^S_n q_n$.*

The above result states that $\sim^1$ is an instance of the more general concept of saturated bisimilarity. In the rest of this subsection, we will show that $\sim^a$ is an instance of symbolic bisimilarity. The SCTS for the asynchronous $\pi$ is defined by the following rule, where $- \in \Sigma^{\mathcal{A}}_{n,n}$ and $- \mid \overline{a}m \in \Sigma^{\mathcal{A}}_{n,x}$.

$$\frac{p \xrightarrow{\overline{a}b} p'}{p_n \xrightarrow{-,\overline{a}b}_{\alpha} p'_n} \qquad \frac{p \xrightarrow{\overline{a}(n+1)} p'}{p_n \xrightarrow{-,\overline{a}()}_{\alpha} p'_{n+1}} \qquad \frac{p \xrightarrow{\tau} p'}{p_n \xrightarrow{-,\tau}_{\alpha} p'_n} \qquad \frac{p \xrightarrow{a(m)} p' \quad x = max\{m,n\}}{p_n \xrightarrow{-\mid \overline{a}m,\tau}_{\alpha} p'_x}$$

Note that the only non standard rule is the fourth. If, in the standard transition system a process can perform an input, in the SCTS the same process can perform a $\tau$, provided that there is an output processes in parallel. Note that the sort of the arriving state depends on the name received $m$: if it is smaller than $n$, then the arriving sort is $n$, otherwise it is $m$.

Now we have to define a set of rules $\mathcal{R}_{\mathcal{A}}$ that describes how contexts transforms transitions. Since our contexts are just parallel outputs, all the contexts preserve transitions. This is expressed by the following (parametric) rules

$$\frac{P_n \xrightarrow{\tau}_{\mathcal{A}} P'_n}{c(P_n) \xrightarrow{\tau}_{\mathcal{A}} c(P'_n)} \qquad \frac{P_n \xrightarrow{\overline{a}b}_{\mathcal{A}} P'_n}{c(P_n) \xrightarrow{\overline{a}b}_{\mathcal{A}} c(P'_n)} \qquad \frac{P_n \xrightarrow{\overline{a}(b)}_{\mathcal{A}} P'_{n+1}}{c(P_n) \xrightarrow{\overline{a}(b)}_{\mathcal{A}} c^{+1}(P'_{n+1})}$$

where $c \in \Sigma^{\mathcal{A}}_{n,m}$ is a generic context, and $c^{+1} \in \Sigma^{\mathcal{A}}_{n+1,m+1}$ is the same syntactic context as $c$, but with different sorts. Instantiating the general definition of symbolic bisimulation to $\alpha$ and $\mathcal{R}_{\mathcal{A}}$, we retrieve the definition of asynchronous bisimulation. Indeed transitions of the form $p \xrightarrow{-,\mu}_{\alpha} p'$ (in the original LTS $\tau$ and output), can be matched only by transitions with the same label, while

transitions $p \xrightarrow{-|\overline{a}m,\tau}_{\alpha} p'$ (the input) can be matched either by $q \xrightarrow{-|\overline{a}m,\tau}_{\alpha} q'$ using the rule $id \xrightarrow[\tau]{} id$ or by $q \xrightarrow{-|,\tau}_{\alpha} q'$ using the rule $- \mid \overline{a}m \xrightarrow[\tau]{} - \mid \overline{a}m$.

**Proposition 3.** *Let $p, q$ be asynchronous $\pi$-processes, and let $n \geq max$ $\mathrm{fn}(p \cup q)$. Then $p \sim^a q$ iff $p_n \sim_n^{SYM} q_n$.*

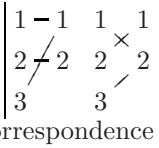
**Proposition 4.** *$\alpha$ and $\mathcal{R}_{\mathcal{A}}$ are sound and complete w.r.t. $\mathcal{A}$.*

**Corollary 1 (by Thm. 1).** *$\sim^1 = \sim^a$ as shown in [1].*

### 4.2 Open

In this section we will present $\mathcal{O} = \langle (S^{\mathcal{O}}, \Sigma^{\mathcal{O}}), \mathbb{O}, O_{\mathcal{O}}, tr_{\mathcal{O}} \rangle$ for open $\pi$-calculus. As in the asynchronous case, we assume the set of names $\mathcal{N}$ to be totally ordered[4], and we write $n$ to mean the set of names smaller or equal to $n$. A *fusion* $\sigma : n \to m$ is a surjective function where

$$\sigma(i) < \sigma(j) \Rightarrow \exists k \in \sigma^{-1}(\sigma(i)) \text{ such that } k < j.$$




The above condition guarantees that fusions are in one to one correspondence with the equivalence classes on names and thus with matching sequences. For example consider the two functions depicted above on the right. Both represents the matching $[1 = 3]$, but only the leftmost is a fusion.

The multi-sorted signature $(S^{\mathcal{O}}, \Sigma^{\mathcal{O}})$ is formally defined as

- $\Sigma^{\mathcal{O}} = \{(n, D)$ for $n \in \omega$ and $D \in \mathcal{D}$ such that $\mathrm{nm}(D) \subseteq n\}$;
- $\Sigma^{\mathcal{O}}_{(n,D),(n',D')}$ is the set of fusions $\sigma : n \to n'$ such that:
  1. Respect distinction, i.e., $iDj \Rightarrow \sigma(i) \neq \sigma(j)$,
  2. Preserve distinction, i.e., $iDj \Rightarrow \sigma(i)D'\sigma(j)$.[5]

Let us define the $(S^{\mathcal{O}}, \Sigma^{\mathcal{O}})$-algebra $\mathbb{O}$. For every sort $(n, D)$, $O_{n,D}$ is the set of processes $p$ such that $n \geq max\{\mathrm{fn}(p)\}$. Then $\forall p \in O_{n,D}$ and $\forall \sigma \in \Sigma^{\mathcal{O}}_{(n,D),(n',D')}$, $\sigma_{\mathbb{O}}(p)$ is the process of sort $(n', D')$ obtained by replacing in $p$ all the occurrences of $a \in \mathrm{fn}(p)$ with $\sigma(a)$. In this system, interfaces are pairs $(n, D)$ where $n$ is a set of names (as in the asynchronous case) and $D$ is a distinction. A process with interface $(n, D)$, can be inserted only in those fusions that respect $D$. Given a process $p$, a natural number $n \geq max$ $\mathrm{fn}(p)$ and $D$ such that $\mathrm{nm}(D) \subseteq n$, we denote with $p_{n,D}$ the process $p$ with interface $(n, D)$.

The set of observations is $O_{\mathcal{O}} = \{a(), \overline{a}b, \overline{a}(), \tau \mid a, b \in \mathcal{N}\}$. Differently from the asynchronous case, here input is observable. However note that the received name does not appear. This is because any process with sort $(n, D)$ will receive the name $n + 1$ (that could be later fused with other names).

The following rules define the transition structure $tr_{\mathcal{O}}$ (denoted by $\rightarrow_{\mathcal{O}}$).

[4] We can work with not ordered $\mathcal{N}$ by taking as signature the category **D** of [13].

[5] $\forall (n, D) \in \Sigma^{\mathcal{O}}$, $id_{n,D}$ is the identity fusion, while $\circ$ is composition of substitutions.

$$\frac{p \xrightarrow{\overline{a}b} p'}{p_{n,D} \xrightarrow{\overline{a}b}_{\mathcal{O}} p'_{n,D}} \quad \frac{p \xrightarrow{\tau} p'}{p_{n,D} \xrightarrow{\tau}_{\mathcal{O}} p'_{n,D}} \quad \frac{p \xrightarrow{a(n+1)} p'}{p_{n,D} \xrightarrow{a()}_{\mathcal{O}} p'_{n+1,D}} \quad \frac{p \xrightarrow{\overline{a}(n+1)} p'}{p_{n,D} \xrightarrow{\overline{a}()}_{\mathcal{O}} p'_{n+1,\bar{D}}}$$

where $\bar{D} = D \cup \{(n+1, i), \quad \forall i < n+1\}$. The only non-standard transition is the bound output, where in the arriving state the distinction $\overline{D}$ is forced.

**Proposition 5.** *Let $p, q \in O$ be $\pi$-processes, and let $n \geq max\ \mathrm{fn}(p \cup q)$ and $\mathrm{nm}(D) \subseteq n$. Then $p \sim_D^O q$ iff $p_{n,D} \sim_{n,D}^S q_{n,D}$.*

In order to define the symbolic transition system we have to fix some notations. For any $\sigma \in \Sigma^{\mathcal{O}}_{(n,D),(m,D')}$, we denote by $\sigma^{+1} \in \Sigma^{\mathcal{O}}_{(n+1,D),(m+1,D')}$ the fusion that maps $n+1$ into $m+1$ and all the $i \leq n$ into $\sigma(i)$, while we use $\overline{\sigma^{+1}} \in \Sigma^{\mathcal{O}}_{(n+1,\overline{D}),(m+1\overline{D'})}$ to mean $\sigma^{+1}$ with the enforced distinction $\overline{D'}$. For any matching sequence $M$ that respects $D$, we denote by $\sigma^M \in \Sigma^{\mathcal{O}}_{(n,D),(m,\sigma^M(D))}$ the unique fusion corresponding to $M$.

The following rules define the SCTS by relying on the symbolic transition system presented in Sec. 2. In all the rules, we implicitly assume as premise that $\sigma^M$ respects $D$.

$$\frac{p \xrightarrow{M,\overline{a}b}_e p'}{p_{n,D} \xrightarrow{\sigma^M,\overline{\sigma^M(a)}\sigma^M(b)}_o \sigma^M(p'_{n,D})} \quad \frac{p \xrightarrow{M,\tau}_e p'}{p_{n,D} \xrightarrow{\sigma^M,\tau}_o \sigma^M(p'_{n,D})}$$

$$\frac{p \xrightarrow{M,a(n+1)}_e p'}{p_{n,D} \xrightarrow{\sigma^M,\sigma^M(a)()}_o \sigma^{M^{+1}}(p'_{n+1,D})} \quad \frac{p \xrightarrow{M,\overline{a}(n+1)}_e p'}{p_{n,D} \xrightarrow{\sigma^M,\overline{\sigma^M(a)}()}_o \overline{\sigma^{M^{+1}}}(p'_{n+1,\overline{D}})}$$

Our SCTS differs form the canonical symbolic transition system, because the substitution here is applied both to observations and arriving states. Now we have to fix a set of rules $\mathcal{R}_{\mathcal{O}}$ that describes how fusions transform transitions. It is well known from [16] that substitutions preserve all the transitions by applying the substitution also to the observation. This is expressed by the following parametric rules for every $\sigma \in \Sigma^{\mathcal{O}}_{(n,D),(n',D')}$.

$$\frac{P_{n,D} \xrightarrow{\tau} P'_{n,D}}{\sigma(P_{n,D}) \xrightarrow{\tau} \sigma(P'_{n,D})} \quad \frac{P_{n,D} \xrightarrow{\overline{a}b} P'_{n,D}}{\sigma(P_{n,D}) \xrightarrow{\overline{\sigma(a)}\sigma(b)} \sigma(P'_{n,D})}$$

$$\frac{P_{n,D} \xrightarrow{a()} P'_{n+1,D}}{\sigma(P_{n,D}) \xrightarrow{\sigma(a)()} \sigma^{+1}(P'_{n+1,D})} \quad \frac{P_{n,D} \xrightarrow{\overline{a}()} P'_{n+1,\overline{D}}}{\sigma(P_{n,D}) \xrightarrow{\overline{\sigma(a)}()} \overline{\sigma^{+1}}(P'_{n+1,\overline{D}})}$$

**Proposition 6.** *Let $p, q \in O$ be $\pi$-processes, and let $n \geq max\ \mathrm{fn}(p \cup q)$ and $\mathrm{nm}(D) \subseteq n$. Then $p \asymp_D q$ iff $p_{n,D} \sim_{n,D}^{SYM} q_{n,D}$.*

**Proposition 7.** *$o$ and $\mathcal{R}_{\mathcal{O}}$ are sound and complete w.r.t. $\mathcal{O}$.*

**Corollary 2 (by Thm. 1).** *$\sim^O = \asymp$ as shown in [21].*

## 5 Open Petri Nets

Differently from process calculi, Petri nets have not a widely known interactive behaviour. Indeed they model concurrent systems that are closed, in the sense that they do not interact with the environment. *Open nets* [10,2] are P/T Petri nets that can interact by exchanging tokens on *input* and *output places*.

Given a set $X$, we write $X^{\oplus}$ for the free commutative monoid over $X$. A multiset $m \in X^{\oplus}$ is a function from $X$ to $\omega$ (the set of natural number) that associates a multiplicity to every element of $X$. Given two multisets $m_1$ and $m_2$, $m_1 \oplus m_2$ is defined as $\forall x \in X,\ m_1 \oplus m_2(x) = m_1(x) + m_2(x)$. We write $m_1 \subseteq m_2$ if $\forall x \in X,\ m_1(x) \leq m_2(x)$. If $m_1 \subseteq m_2$, the multiset $m_2 \ominus m_1$ is defined as $\forall x \in X\ m_2 \ominus m_1(x) = m_2(x) - m_1(x)$. Given a set $Y \subseteq X$, and $m \in X^{\oplus}$, the multiset $m \restriction Y$ is defined as $m \restriction Y(x) = m(x)$ if $x \in Y$, 0 otherwise. We write $\varnothing$ to denote both the empty set and the empty multiset.

**Definition 12 (open net).** *An open net is a tuple $N = (S, T, pre, post, \lambda, I, O)$ where $S$ is the set of places, $T$ is the set of transitions (with $S \cap T = \varnothing$), $pre, post : T \to S^{\oplus}$ are functions mapping each transition to its pre- and post-set, $\lambda : T \to \Lambda$ is a labeling function ($\Lambda$ is a set of labels) and $I, O \subseteq S$ are the sets of input and output places (with $I \cap O = \varnothing$). A marked open net is pair $\langle N, m \rangle$ where $N$ is an open net and $m \in S^{\oplus}$ is a marking.*

Fig. 1 shows two open nets where, as usual, circles represents places and rectangles transitions (labeled with $\alpha, \beta, \chi$). Arrows from places to transitions represent *pre*, while arrows from transitions to places represent *post*. Input places are denoted by ingoing edges, while output places are denoted by outgoing edges. Thus in $N_1$, $x$ and $y$ are output places, while $z$ is the only input place. In $N_2$, it is the converse. The *parallel composition* of the two nets is defined by attaching them on their input and output places. As an example, we can compose $N_1$ and $N_2$ by attaching them through $x, y$ and $z$.

The operational semantics of marked open nets is expressed by the rules on Table 4 where, in order to make lighter the notation, we use ${}^\bullet t$ and $t^\bullet$ to denote $pre(t)$ and $post(t)$ and we avoid to put brackets around the marked net $\langle N, m \rangle$. The rule (TR) is the standard rule of P/T nets (seen as multisets rewriting), while the other two are specific of open nets. The rule (IN) states that in any moment a token can be inserted inside an input place and, for this reason, the LTS has always an infinite number of states. The rule (OUT) states that when a token is in an output place, it can be removed. Fig.1[A] shows part of the infinite transition system of $\langle N_2, a \rangle$.

The abstract semantics is defined in [3] as the standard bisimilarity (denoted by $\sim^N$) and it is a congruence under the parallel composition outlined above. This is due to the rules (IN) and (OUT), since they put a marked net in all the possible contexts. If we consider just the rule (TR), then bisimilarity fails to be a congruence. Thus also for open nets, the canonical definition of bisimulation consists of inserting the system in all the possible contexts and observing what

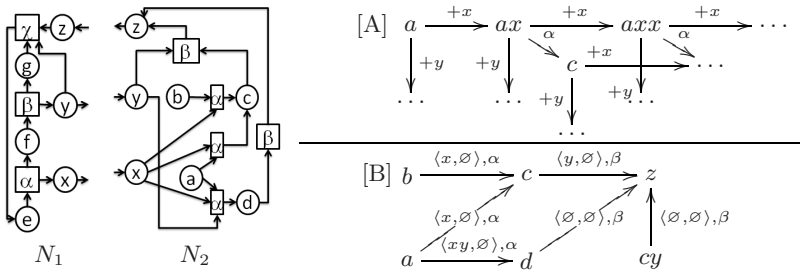



**Fig. 1.** $N_1$ and $N_2$ are two open Petri nets. [A] Part of the infinite transition system of $\langle N_2, a\rangle$. [B] The symbolic transition system of $\langle N_2, a\rangle$, $\langle N_2, b\rangle$ and $\langle N_2, cy\rangle$.

happens, but differently from open and asynchronous bisimilarity, a symbolic LTS and an efficient characterization of $\sim^N$ has never been given.

### 5.1 Context Interactive System for Open Nets

In this section we introduce the context interactive system for open nets $\mathcal{N} = \langle (S^{\mathcal{N}}, \Sigma^{\mathcal{N}}), \mathbb{N}, \Lambda, tr_{\mathcal{N}}\rangle$. Contexts are insertions and the deletions of tokens.

The many-sorted signature $(S^{\mathcal{N}}, \Sigma^{\mathcal{N}})$ is formally defined as:

- $S^{\mathcal{N}} = \{(I, O, m) \mid\ m \in O^{\oplus}\}$,
- $\Sigma^{\mathcal{N}}_{(I,O,m),\langle I,O,m'\rangle} = \{\langle i, o\rangle \mid\ i \in I^{\oplus},\ o \in O^{\oplus},\ o \subseteq m,\ m' = m \ominus o\}$.[6]

The sorts of the signature are triples $(I, O, m)$ where $I$ and $O$ are sets of input places and output places and $m \in O^{\oplus}$ is a marking on the output places.

Operators of $\Sigma^{\mathcal{N}}$ are pairs $\langle i, o\rangle \in \Sigma^{\mathcal{N}}_{(I,O,m),\langle I,O,m'\rangle}$ where $i \in I^{\oplus}$, $o \in O^{\oplus}$ are, respectively, multisets of tokens added in the input places and removed from the output places. Note that in the target sort, the set of input and output places are the same of the source (meaning that context cannot modify $I$ and $O$), while the marking $m' \in O^{\oplus}$ is equal to $m \ominus o$.

We say that an open net $N$ has interface $(I, O)$ if $I$ and $O$ are respectively its sets of input and output places. While a marked open net $\langle N, m\rangle$ has interface $(I, O, m')$ if $(I, O)$ is the interface of $N$ and moreover if $m' = m \restriction O$. This means that tokens in the output places are visible from the environment, while tokens in the input places are not. We can better understand this difference, by observing that the environment can remove tokens in the output places only if they are present, while it can always add tokens in the input places.

Let us define the $(S^{\mathcal{N}}, \Sigma^{\mathcal{N}})$-algebra $\mathbb{N}$. For any sort $(I, O, m)$, the carrier set $N_{I,O,m}$ contains all the marked open nets with interface $(I, O, m)$. Any operator $\langle i, o\rangle \in \Sigma_{(I,O,m),(I,O,m')}$ is defined as the function that maps $\langle N, m_1\rangle$ into $\langle N, m_1 \oplus i \ominus o\rangle$. The transition structure $tr_{\mathcal{N}}$ (denoted by $\rightarrow_{\mathcal{N}}$) associates to a state $\langle N, m\rangle$ the transitions obtained by using the rule (TR) of Table 4.

[6] $\forall (I, O, m) \in S^{\mathcal{N}}$, $id_{I,O,m}$ is $\langle \varnothing, \varnothing\rangle$, while $\langle i_1, o_1\rangle \circ \langle i_2, o_2\rangle = \langle i_1 \oplus i_2, o_1 \oplus o_2\rangle$.

**Table 4.** Operational Semantics of marked open nets

$$(\text{TR})\ \frac{t \in T \quad \lambda(t) = l \quad m = {}^\bullet t \oplus c}{N, m \xrightarrow{l} N, t^\bullet \oplus c} \qquad (\text{IN})\ \frac{i \in I_N}{N, m \xrightarrow{+i} N, m \oplus i} \qquad (\text{OUT})\ \frac{o \in O_N \quad o \in m}{N, m \xrightarrow{-o} N, m \ominus o}$$

**Proposition 8.** *Let* $\langle N_1, m_1\rangle$ *and* $\langle N_2, m_2\rangle$ *be two marked nets both with interface* $(I, O, m)$*. Thus* $\langle N_1, m_1\rangle \sim^N \langle N_2, m_2\rangle$ *iff* $\langle N_1, m_1\rangle \sim^S_{I,O,m} \langle N_2, m_2\rangle$*.*

### 5.2 A Symbolic Semantics for Open Nets

In the case of open and asynchronous $\pi$-calculus, we already knew the symbolic transition system by classical results in literature. In the case of open nets, no symbolic semantics does exists, and thus we have to define it. We use exactly the same intuition underlying the symbolic LTS of open and asynchronous, i.e., we consider the *minimal contexts* that allow a given system to perform a transition.

The SCTS for open nets, $\eta$ is defined by the following rule.

$$\frac{t \in T \quad \lambda(t) = l \quad m = (m \cap {}^\bullet t) \oplus c \quad i \subseteq I^\oplus \quad {}^\bullet t = (m \cap {}^\bullet t) \oplus i \quad o \subseteq c \upharpoonright O}{N, m \xrightarrow{\langle i,o\rangle, l}_\eta N, t^\bullet \oplus c \ominus o}$$

The marking $m \cap {}^\bullet t$ contains all the tokens of $m$ that are needed to perform $t$. The marking $c$ contains all the tokens of $m$ that are not useful for performing $t$, while the marking $i$ contains all the tokens that $m$ needs to reach ${}^\bullet t$. Note that $i$ is exactly the *smallest* multiset that is needed to perform the transition $t$. Indeed if we take $i_1$ strictly included into $i$, $m \oplus i_1$ cannot match ${}^\bullet t$.

As an example consider the net $N_1$ in Fig. 1 with marking $gxy$ and let $t$ be the only transition labeled with $\chi$. We have that $gxy \cap {}^\bullet t = gy$, $c = x$ and $i = z$. Thus $N_1, gxy \xrightarrow{\langle z,x\rangle,\chi}_\eta N_1, e$ and also $N_1, gxy \xrightarrow{\langle z,\varnothing\rangle,\chi}_\eta N_1, ex$. In the former transition we have taken $o$ equal to $x = c \upharpoonright O$, while in the latter $o = \varnothing$. The multiset $c \upharpoonright O$ is the largest that can be safely removed by $m$ without inhibiting the transition $t$. Differently than input, in the output we have to consider both the transitions (expressed by the premise $o \subseteq c \upharpoonright O$) because one cannot *dominate* (in the sense of Def. 10) the other. Indeed the former cannot dominate the latter because there are no contexts that add tokens in the output places, while the latter cannot dominate the former because in general, we cannot know if removing tokens from output places preserves a transition.

This is expressed by the set of rules $\mathcal{R}_\mathcal{N}$ that is defined by the following parametric rule.

$$\frac{N, m \xrightarrow{l}_\mathcal{N} N, m'}{\langle i, \varnothing\rangle(N, m) \xrightarrow{l}_\mathcal{N} \langle i, \varnothing\rangle(N, m')}$$

This rule states that the addition of tokens in the input places preserves transitions. While it does not state anything about the deletion of tokens. Indeed an output place could be in the precondition of some transition (e.g., $y$ in the net

$N_1$ in Fig. 1) and thus, the deletion of some tokens can inhibit the transition. Fig. 1[B] shows the SCTS of $\langle N_2, a\rangle$ and $\langle N_2, b\rangle$. The former perform a transition with $\langle xy, \varnothing\rangle$, while the latter cannot. However they are saturated bisimilar.

**Proposition 9.** *$\eta$ and $\mathcal{R}_{\mathcal{N}}$ are sound and complete w.r.t. $\mathcal{N}$.*

The above proposition together with Thm. 1 state that symbolic and semi-saturated bisimilarity coincide with $\sim^S$. In the following we instantiate their general definition to $\mathcal{N}$ and $\mathcal{R}_{\mathcal{N}}$.

**Definition 13 (Symbolic and semi-saturated bisimulation for nets).** *Let $R = \{R_{I,O,m} \subseteq N_{I,O,m} \times N_{I,O,m} \mid (I,O,m) \in S^{\mathcal{N}}\}$ be a $S^{\mathcal{N}}$ sorted family of symmetric relations. $R$ is a* symbolic bisimulation *iff $\forall (I,O,m) \in S^{\mathcal{N}}$, whenever $\langle N_1, m_1\rangle R_{I,O,m} \langle N_2, m_2\rangle$*

- *if $\langle N_1, m_1\rangle \xrightarrow{\langle i,o\rangle, l}_{\eta} \langle N_1, m_1'\rangle$ then $\exists i_1, x \in I^{\oplus}$ such that: $i = i_1 \oplus x$, $\langle N_2, m_2\rangle \xrightarrow{<i_1,o>,l}_{\eta} \langle N_2, m_2'\rangle$ and $\langle N_1, m_1'\rangle R \langle N_2, m_2 \oplus x\rangle$.*

*$R$ is a* semi-saturated *bisimulation iff whenever $\langle N_1, m_1\rangle R_{I,O,m} \langle N_2, m_2\rangle$*

- *if $\langle N_1, m_1\rangle \xrightarrow{\langle i,o\rangle, l}_{\eta} \langle N_1, m_1'\rangle$ then $\langle N_2, m_2 \oplus i \ominus o\rangle \xrightarrow{l} \langle N_2, m_2'\rangle$ and $\langle N_1, m_1'\rangle R \langle N_2, m_2'\rangle$.*

## 6 Leifer and Milner Reactive Systems

As stated in the introduction, our approach generalizes the theory of reactive system by Leifer and Milner [11]. They define the syntax of the formalism through a (Lawvere-like) category $\mathbf{C}$ whose arrows are contexts and terms are arrows having as source a special object 0. In our theory, $\mathbf{C}$ is the closed many-sorted unary signature $(S, \Sigma)$: objects are sorts and arrows are operators. Every term $p : 0 \to s$ is an element of the carrier-set $A_s$. Given a context $c : s \to t$, the composition $p; c$ is defined as $c_{\mathbb{A}}(p)$. They also define a subcategory $\mathbf{D}$ of reactive arrows. This is modeled in our formalism by adding for every arrow $d \in \mathbf{D}$ a rule as the following: $\dfrac{p \longrightarrow q}{d(p) \longrightarrow d(q)}$.

They define the reaction relation by closing some reaction rules under all reactive contexts. In the same way, we start with some labeled transitions (that generalize rewriting rules) and we generalizes w.r.t. all the rules as the above. Idem-PushOut (IPO) represents the minimal context that allows a reaction. The transition system labeled with IPOs (ITS) is an instance of our SCTS. Indeed the saturation of it, trough the above rules exactly coincides with SATTS as formally shown in [5]. However, IPO-bisimilarity (that is a congruence under restrictive condition) is stricter than $\sim^S$. In [5], we have provided a symbolic bisimilarity for ITS (Thm. 3) and proved that it coincides with $\sim^S$. This result is thus a special case of Thm. 1 presented here.

## 7 Conclusions

In this paper we have introduced saturated bisimilarity for context interactive systems and an efficient way to characterize it through symbolic bisimilarity. We have shown that our theory works for real formalism re-deriving well-known semantics, namely, the saturated and symbolic versions of asynchronous and open bisimilarities. Moreover we have applied our approach to open Petri nets with the result of a new (at our knowledge, the first) symbolic semantics that efficiently characterizes canonical bisimilarity. Leifer and Milner reactive systems have been applied to open Petri nets (without observations on transitions) in [15,22], but the derived LTS is infinite.

Our theory generalizes Leifer and Milner reactive systems by allowing observations. We think that observations are usually necessary, since one label cannot represent at the same time both interaction and observation.

As next step, we would like to give a coalgebraic semantics for symbolic bisimilarity by extending *normalized coalgebras* [6] and by exploiting the connections with coalgebras on presheafs [7]. The coalgebraic approach might yield a general minimization algorithm working directly on the symbolic transition systems in the style of [19].

## References

1. Amadio, R.M., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous $\pi$-calculus. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 147–162. Springer, Heidelberg (1996)
2. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. M.S.C.S 15(1), 1–35 (2005)
3. Baldan, P., Corradini, A., Ehrig, H., Heckel, R., König, B.: Bisimilarity and behaviour-preserving reconfiguration of open petri nets. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 126–142. Springer, Heidelberg (2007)
4. Bonchi, F., Gadducci, F., König, B.: Process bisimulation via a graphical encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 168–183. Springer, Heidelberg (2006)
5. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: LICS, pp. 69–80. IEEE, Los Alamitos (2006)
6. Bonchi, F., Montanari, U.: Coalgebraic models for reactive systems. In: ECML 2007. LNCS, vol. 4701, pp. 364–380. Springer, Heidelberg (2007)
7. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: LICS, pp. 93–104. IEEE, Los Alamitos (2001)
8. Hennessy, M., Lin, H.: Symbolic bisimulations. T.C.S. 138(2), 353–389 (1995)
9. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: America, P. (ed.) ECOOP 1991. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)
10. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)

11. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
12. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
13. Miculan, M., Yemane, K.: A unifying model of variables and names. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 170–186. Springer, Heidelberg (2005)
14. Milner, R.: Communicating and Mobile Systems: the $\pi$-Calculus. Cambridge University Press, Cambridge (1999)
15. Milner, R.: Bigraphs for petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 686–701. Springer, Heidelberg (2004)
16. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i and ii. Information and Computation 100(1), 1–77 (1992)
17. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
18. Montanari, U., Sassone, V.: Dynamic congruence vs. progressing bisimulation for ccs. Fundamenta Informaticae 16(1), 171–199 (1992)
19. Pistore, M., Sangiorgi, D.: A partition refinement algorithm for the $\pi$-calculus. Information and Computation 164(2), 264–321 (2001)
20. Rathke, J., Sassone, V., Sobocinski, P.: Semantic barbs and biorthogonality. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 302–316. Springer, Heidelberg (2007)
21. Sangiorgi, D.: A theory of bisimulation for the $\pi$-calculus. Acta Informatica 33(1), 69–97 (1996)
22. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Petri Nets and Graph Transformation. E.N.T.C.S, vol. 127, pp. 107–120. Elsevier, Amsterdam (2005)

# Deriving Bisimulation Congruences in the Presence of Negative Application Conditions*

Guilherme Rangel[1], Barbara König[2], and Hartmut Ehrig[1]

[1] Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, Germany
{rangel,ehrig}@cs.tu-berlin.de
[2] Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany
barbara_koenig@uni-due.de

**Abstract.** In recent years there have been several approaches for the automatic derivation of labels from an unlabeled reactive system. This can be done in such a way that the resulting bisimilarity is automatically a congruence. One important aspect that has not been studied so far is the treatment of reduction rules with negative application conditions. That is, a rule may only be applied if certain patterns are absent in the vicinity of a left-hand side. Our goal in this paper is to extend the borrowed context framework to label derivation with negative application conditions and to show that bisimilarity remains a congruence. An important application area is graph transformation and we will present a small example in order to illustrate the theory.

## 1 Introduction

Bisimilarity is an equivalence relation on states of transition systems, associating states that can match each other's moves. In this sense, bisimilar states can not be distinguished by an external observer. Bisimilarity provides a powerful proof technique to analyze the properties of systems and has been extensively studied in the field of process calculi since the early 80's. Especially for CCS [1] and the $\pi$-calculus [2,3] an extensive theory of bisimulation is now available.

Congruence is a very desirable property that a bisimilarity may have, since it allows the exchange of bisimilar systems in larger systems without effect on the observable behavior. Unfortunately, a bisimulation defined on unlabeled reaction rules is in general not a congruence. Hence, Leifer and Milner [4,5] proposed a method that uses so-called idem pushouts (IPOs) to derive a labeled transition system from unlabeled reaction rules such that the resulting bisimilarity is a congruence. Motivated by this work, two of the authors proposed in [6,7] an extension to the double pushout approach (DPO, for short) called DPO with borrowed contexts (DPO-BC), which provides the means to derive labeled transitions from rewriting rules in such a way that the bisimilarity is automatically

* Research partially supported by the DFG project SANDS and DAAD (German Academic Exchange Service).

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 413–427, 2008.

a congruence. This has turned out to be equivalent to a technique by Sassone and Sobociński [8,9] which derives labels via groupoidal idem pushouts. In all approaches the basic idea is the one suggested by Leifer and Milner: the labels should be the minimal contexts that an observer has to provide in order to trigger a reduction.

The DPO with borrowed contexts works with productions consisting of two arrows $L \leftarrow I \rightarrow R$ where the arrows are either graph morphisms, or—more generally—arrows in an adhesive category. Even though the generative power of the DPO approach is sufficient to generate any recursively enumerable set of graphs, very often extra application conditions are a required feature of nontrivial specifications. Negative application conditions (NACs) [10] for a graph production are conditions such as the non-existence of nodes, edges, or certain subgraphs in the graph $G$ being rewritten, as well as embedding restrictions concerning the match $L \rightarrow G$. Similar restrictions can also be achieved in Petri nets with inhibitor arcs, where these arcs impose an extra requirement to transition firing, i.e., a transition can only be fired if some specific places are currently unmarked.

Graph transformation systems, which are our main focus, are often used for specification purposes, where—in contrast to programming—it is quite convenient and often necessary to constrain the applicability of rules by negative application conditions. We believe that this is a general feature of specification languages, which means that the problem of deriving behavioural equivalences in the presence of NACs may occur in many different settings.

In this work we extend the borrowed context framework to handle productions with negative application conditions. The extension, which is carried out for adhesive categories, requires an enrichment of the labels which now do not only indicate the context that is provided by the observer, but also constrain further additional contexts that may satisfy the negative application condition. That is, we do not only specify what must be borrowed, but also what must not be borrowed. We prove that the main result of [7] (bisimilarity is a congruence) still holds for our extension. Moreover, we further develop an up-to context technique in order to cope with NACs and apply it to an example.

The current paper is structured as follows. Section 2 briefly reviews the DPO approach with borrowed contexts. In Section 3 we discuss the problems which arise due to productions with NACs and how they can be overcome in order to guarantee that the derived bisimilarities are congruences. Section 4 presents the up-to proof method for our extension and finally an example in terms of graph transformation is shown in Section 5.

An extended example and the full proof with all lemmas can be found in a technical report [11].

## 2 Double-Pushout with Borrowed Contexts

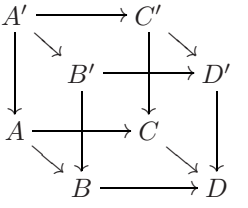
In this section we recall the DPO approach with borrowed contexts [6,7]. In standard DPO [12], productions rewrite graphs with no interaction with any other entity than the graph itself and the production. In the DPO with borrowed contexts [7] graphs have interfaces and may borrow missing parts of left-hand

sides from the environment via the interface. This leads to open systems which take into account interaction with the outside world.

The DPO-BC framework was originally defined for the category of graph structures, but, as already stated in [6,7], its results can be automatically lifted to adhesive categories since the corresponding proofs only use pushout and pullback constructions which are compliant with adhesive categories. In the following we present the DPO-BC setting for adhesive categories [13] to which we first give a short introduction.

**Definition 1 (Adhesive Category).** *A category* $\mathbf{C}$ *is called* adhesive *if*

1. $\mathbf{C}$ *has pushouts along monos;*
2. $\mathbf{C}$ *has pullbacks;*
3. *Given a cube diagram as shown on the right with: (i)* $A \to C$ *mono, (ii) the bottom square a pushout and (iii) the left and back squares pullbacks, we have that the top square is a pushout iff the front and right squares are pullbacks.*




Pullbacks preserve monos and pushouts preserve epis in any category. Furthermore, for adhesive categories it is known that monos are preserved by pushouts. For the DPO-BC extension to productions with negative application conditions, defined in Section 3, we need one further requirement, namely that pullbacks preserve epis. This means that if the square $(A', B', A, B)$ above is a pullback and $A \to B$ is epi, we can conclude that $A' \to B'$ is epi as well.

Our prototypical instance of an adhesive category, which will be used for the examples in the paper are the categories of node-labeled and edge-labeled graphs, where arrows are graph morphisms. In this category pullbacks preserve epis.

We will now define the notion of objects with interfaces and contexts, followed by the definition of a rewriting step with borrowed contexts as defined in [7] and extended in [9].

**Definition 2 (Objects with Interfaces and Contexts).** *An* object $G$ with interface $J$ *is an arrow* $J \to G$ *and a context consists of two arrows* $J \to E \leftarrow \overline{J}$. *The* embedding[1] *of* $J \to G$ *into a context* $J \to E \leftarrow \overline{J}$ *is an object with interface* $\overline{J} \to \overline{G}$ *which is obtained by constructing* $\overline{G}$ *as the pushout of* $J \to G$ *and* $J \to E$.

$$\begin{array}{ccc} J & \longrightarrow & E \longleftarrow \overline{J} \\ \downarrow & PO & \downarrow \\ G & \longrightarrow & \overline{G} \end{array}$$

**Definition 3 (Rewriting with Borrowed Contexts).** *Given an object with interface* $J \to G$ *and a production* $p\colon L \leftarrow I \to R$, *we say that* $J \to G$ *reduces to* $K \to H$ *with transition label*[2] $J \to F \leftarrow K$ *if there are objects* $D$, $G^+$, $C$ *and additional arrows such that the diagram below commutes and the squares are*

[1] The embedding is defined up to iso since the pushout object is unique up to iso. Embedding/insertion into a context and contextualization are used as synonyms.

[2] Transition labels, derived labels and labels are synonyms in this work.

*either pushouts (PO) or pullbacks (PB) with monos. In this case a* rewriting step with borrowed context *(BC step) is called feasible:* $(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$.

$$\begin{array}{ccccc} D & \rightarrowtail & L & \longleftarrow & I & \longrightarrow & R \\ \downarrow & PO & \downarrow & PO & \downarrow & PO & \downarrow \\ G & \rightarrowtail & G^+ & \longleftarrow & C & \longrightarrow & H \\ \uparrow & PO & \uparrow & PB & \uparrow & \nearrow & \\ J & \rightarrowtail & F & \longleftarrow & K & & \end{array}$$

In the diagram above the upper left-hand square merges $L$ and the object $G$ to be rewritten according to a partial match $G \leftarrow D \to L$. The resulting object $G^+$ contains a total match of $L$ and can be rewritten as in the standard DPO approach, producing the two remaining squares in the upper row. The pushout in the lower row gives us the borrowed (or minimal) context $F$, along with an arrow $J \to F$ indicating how $F$ should be pasted to $G$. Finally, we need an interface for the resulting object $H$, which can be obtained by "intersecting" the borrowed context $F$ and the object $C$ via a pullback. Note that the two pushout complements that are needed in Definition 3, namely $C$ and $F$, may not exist. In this case, the rewriting step is not feasible. The arrows depicted as $\to$ in the diagram above can also be non-mono (see [8]).

Note that with the procedure described above we may derive infinitely many labels of the form $J \to F \leftarrow K$. However, note that there are only finitely many up to iso and hence they can be represented in a finite way.

A bisimulation is an equivalence relation between states of transition systems, associating states which can simulate each other.

**Definition 4 (Bisimulation and Bisimilarity).** *Let $\mathcal{P}$ be a set of productions and $\mathcal{R}$ a symmetric relation containing pairs of objects with interfaces $(J \to G, J \to G')$. The relation $\mathcal{R}$ is called a* bisimulation *if, whenever we have $(J \to G)\,\mathcal{R}\,(J \to G')$ and a transition $(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$, then there exists an object with interface $K \to H'$ and a transition $(J \to G') \xrightarrow{J \to F \leftarrow K} (K \to H')$ such that $(K \to H)\,\mathcal{R}\,(K \to H')$.*

*We write $(J \to G) \sim (J \to G')$ whenever there exists a bisimulation $\mathcal{R}$ that relates the two objects with interface. The relation $\sim$ is called* bisimilarity.

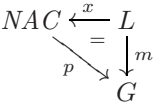
**Theorem 1 (Bisimilarity is a Congruence [7]).** *The bisimilarity relation $\sim$ is a congruence, i.e., it is preserved by contextualization as described in Definition 2.*

## 3 Borrowed Contexts with NACs

Here we will extend the DPO-BC framework of [7] to productions with negative application conditions. In order to simplify the theory and the presentation we will from now on require that productions and objects with interfaces consist of monos, which implies that all arrows in the diagram in Definition 3 are monos.

Prior to the extension we will investigate in Section 3.1 why such an extension is not trivial. It is worth emphasizing that the extension will be carried out for adhesive categories with an additional requirement that pullbacks preserve epis, but the examples will be given in the category of labeled directed graphs. First we define negative application conditions for productions.

**Definition 5 (Negative Application Condition).** *A negative application condition $NAC(x)$ on $L$ is a mono $x\colon L \to NAC$ . A mono $m\colon L \to G$ satisfies $NAC(x)$ on $L$ if and only if there is no mono $p\colon NAC \to G$ with $p \circ x = m$.*

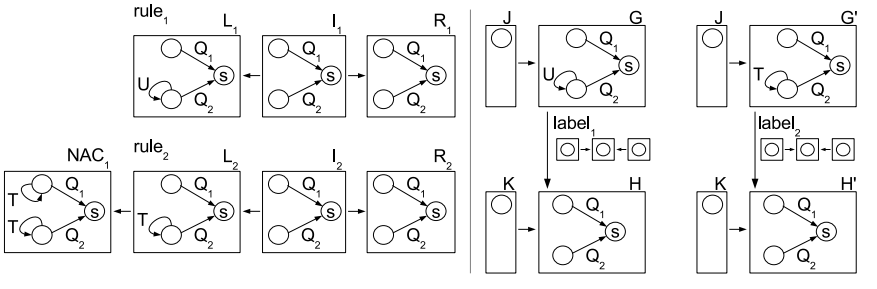



A rule $L \leftarrow I \rightarrow R$ with NACs is equipped with a finite set of negative application conditions $\{L \to NAC_y\}_{y \in Y}$ and is applicable to a match $m\colon L \to G$ only if all NACs are satisfied. If we add NACs to the rules in Definition 3, we have two ways to check their satisfiability: before (on $G$) or after the borrowing (on $G^+$), but the latter is more suitable since the first one does not take into account any borrowed structure.

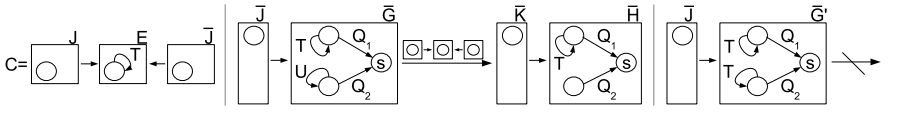
### 3.1 Bisimulation and NACs – Is Bisimilarity Still a Congruence?

Let us assume that borrowed context rewriting works as in Definition 3 (with monos) if the total match $L \to G^+$ satisfies all NACs of a production, i.e., $G^+$ does not contain any prohibited structure (specified by a NAC) at the match of $L$. With the following example in terms of labeled directed graphs we will show that such a definition is unsuitable.

Below on the right we depict two servers as graphs with interfaces: $\mathsf{J} \to \mathsf{G}$ and $\mathsf{J} \to \mathsf{G}'$. An s-node represents a server. Each server has two queues $\mathsf{Q}_1$ and $\mathsf{Q}_2$ where it receives tasks to be processed. Tasks are modelled as loops and may either be standard ($\mathsf{T}$) or urgent ($\mathsf{U}$). In real world applications, standard tasks may come from regular users while urgent ones come from administrators. On the left we depict how the servers work. $\mathsf{Rule}_1$ says that an urgent task in $\mathsf{Q}_2$ must be immediately executed, whereas $\mathsf{Rule}_2$ specifies how a standard task $\mathsf{T}$ in $\mathsf{Q}_2$ is executed. The negative application condition $\mathsf{NAC}_1$ allows $\mathsf{rule}_2$ to be fired only when there is no other $\mathsf{T}$-task waiting in the high priority queue $\mathsf{Q}_1$. We consider that a processed task is consumed by the server (see $\mathsf{R}_1$ and $\mathsf{R}_2$).

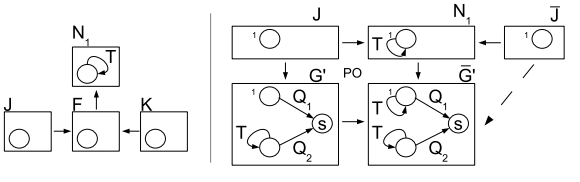


From the servers $J \to G$ and $J \to G'$ above we derive the labeled transition system (LTS) on the right w.r.t. $\mathsf{rule}_1$ and $\mathsf{rule}_2$. No further label can be derived from $K \to H$ and $K \to H'$ and the labels leading to these graphs are equal. By Definition 4 we can conclude that $(J \to G) \sim (J \to G')$. Since bisimilarity is a congruence (at least for rules without NACs), the insertion of $J \to G$ and $J \to G'$ into a context $C$, as in Definition 2, produces graphs $\overline{J} \to \overline{G}$ and $\overline{J} \to \overline{G}'$ respectively, which should be bisimilar. Below we show a context $C$ with a standard task, the resulting graphs $\overline{J} \to \overline{G}$ and $\overline{J} \to \overline{G}'$ which received the T-task in queue $Q_1$ via the interface $J$, and their LTS. The server $\overline{J} \to \overline{G}'$ cannot perform any transition since $\mathsf{NAC}_1$ of $\mathsf{rule}_2$ forbids the BC step, i.e., the T-task in $Q_2$ cannot be executed because there is another standard task in the high priority queue $Q_1$. However, $\overline{J} \to \overline{G}$ is still able to perform a transition and evolve to $\overline{K} \to \overline{H}$. Thus, bisimilarity is no longer a congruence when productions have NACs.




The LTS for $J \to G$ and $J \to G'$ shows that $\mathsf{label}_1$, which is derived from $\mathsf{rule}_1$ (without NAC) is matched by $\mathsf{label}_2$, which is generated by $\mathsf{rule}_2$ (with NAC). These matches between labels obtained from rules with and without NACs are the reason why the congruence property does no longer hold. In fact, the actual definitions of bisimulation and borrowed context step are too coarse to handle NACs.

Our idea is to enrich the transition labels $J \to F \leftarrow K$ with some information provided by the NACs in order to define a finer bisimulation based on these labels. A label must not only know which structures (borrowed context) are needed to perform it, but also which forbidden structures (defined by the NACs) cannot be additionally present in order to guarantee its execution. These forbidden structures will be called *negative borrowed contexts* and are represented by objects $N_i$ attached to the label via monomorphisms from the borrowed context $F$ (see example below). In our server example, $\mathsf{label}_1$ would remain without any negative borrowed context since $\mathsf{rule}_1$ has no NAC. However, $\mathsf{label}_2$ would be the label below on the left, where the negative borrowed context $F \to N_1$ specifies that if a T-task was in $Q_1$, then $\mathsf{NAC}_1$ would have forbidden the BC step of $J \to G'$ via $\mathsf{rule}_2$. That is, with the new form of labels the two graphs are no longer bisimilar and hence we no longer have a counterexample to the congruence property.



The intuition of negative borrowed contexts is the following: given $J \to G$, whenever it is possible to derive a label $J \to F \leftarrow K$ with negative borrowed context $F \to N_i$ via a production $p$ with NACs, then if $J \to G$ is inserted into a context[3] $J \to N_i \leftarrow J$ no further label can be derived from $J \to \overline{G}$ via $p$ since some of its NACs will forbid the rule application (see example above on the right). Put differently the label says that a transition can be executed if the environment "lends" $F$ as minimal context. Furthermore the environment can observe that a production is only executable under certain constraints on the context. Finally, it is not executable at all if the object $G^+$ with borrowed context already contains the NAC.

### 3.2 DPO with Borrowed Contexts – Extension to Rules with NACs

Now we are ready to extend the DPO-BC framework to deal with productions with NACs. First we define when a BC step is executable.

**Definition 6 (Executable Borrowed Context Step).** *Assume that all arrows are mono. Given $J \to G$, a production $L \leftarrow I \to R; \{x_y \colon L \to NAC_y\}_{y \in Y}$ and a partial match $G \leftarrow D \to L$, we say that the BC step is* executable *on $J \to G$ if for the pushout $G^+$ in the diagram below there is no $p_y \colon NAC_y \to G^+$ with $m = p_y \circ x_y$ for every $y \in Y$.*

$$\begin{array}{ccccc} D & \longrightarrow & L & \xrightarrow{x_y} & NAC_y \\ \downarrow & PO & \downarrow{\scriptstyle m} & = & \swarrow_{p_y} \\ J & \longrightarrow & G & \longrightarrow & G^+ \end{array}$$

In the following we need the concept of a pair of jointly epi arrows in order to "cover" an object with two other objects. That is needed to find possible overlaps between the NACs and the object $G^+$ which includes the borrowed context.

**Definition 7 (Jointly Epi Arrows).** *Two arrows $f \colon A \to B$ and $g \colon C \to B$ are* jointly epi *whenever for every pair of arrows $a, b \colon B \to D$ such that $a \circ f = b \circ f$ and $a \circ g = b \circ g$ it holds that $a = b$.*

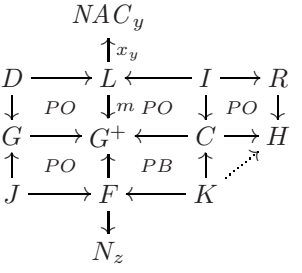
In a pushout square the generated arrows are always jointly epi. This is a straightforward consequence of the uniqueness of the mediating arrow.

**Definition 8 (Borrowed Context Rewriting for Rules with NACs).** *Given $J \to G$, a production $L \leftarrow I \to R; \{L \to NAC_y\}_{y \in Y}$ and a partial match $G \leftarrow D \to L$, we say that $J \to G$ reduces to $K \to H$ with transition label $J \to F \leftarrow K; \{F \to N_z\}_{z \in Z}$ if the following holds:*

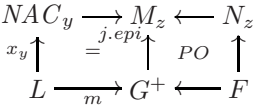
*(i) the BC step is executable (as in Definition 6);*
*(ii) there is an object $C$ and additional arrows such that Diagram (1) below commutes and the squares are either pushouts (PO) or pullbacks (PB) with monos;*

[3] $J \to N_i$ is the composition of $J \to F \to N_i$.

*(iii) the set* $\{F \to N_z\}_{z \in Z}$ *contains exactly the arrows constructed via Diagram (2) (where all arrows are mono). (That is, there exists an object* $M_z$ *such that all squares commute and are pushouts or arrows are jointly epi as indicated.)*




(1)




(2)

*In this case a* borrowed context step *is feasible and we write:* $(J \to G) \xrightarrow{J \to F \leftarrow K; \{F \to N_z\}_{z \in Z}} (K \to H)$.

Observe that Definition 8 coincides with Definition 3 when no NACs are present (cf. Condition (ii)). By taking NACs into account, a BC step can only be executed when $G^+$ contains no forbidden structure of any $NAC_y$ at the match of $L$ (Condition (i)). Additionally, enriched labels are generated (Condition (iii)).

In Condition (iii) the arrows $F \to N_z$ are also called *negative borrowed contexts* and each $N_z$ represents the structures that should not be in $G^+$ in order to enable the BC step. This extra information in the label is of fundamental importance for the bisimulation game with NACs (Definition 9), where two objects with interfaces must not only agree on the borrowed context which enables a transition but also on what should not be present in order to perform the transition. The negative borrowed contexts $F \to N_z$ are obtained from $NAC_y \overset{x_y}{\leftarrow} L \overset{m}{\to} G^+ \leftarrow F$ of Diagram (1) via Diagram (2), where we create all possible overlaps $M_z$ of $G^+$ and $NAC_y$ in order to check which structures the environment should not provide in order to guarantee the execution of a BC step. To consider all possible overlaps is necessary in order to take into account that parts of the NAC might already be present in the object which is being rewritten.

Whenever the pushout complement in Diagram (2) exists, the object $G^+$ with borrowed context can be extended to $M_z$ by attaching the negative borrowed context $N_z$ via $F$. When the pushout complement does not exist, some parts of $G^+$ which are needed to perform the extension are not visible from the environment and no negative borrowed context is generated.

Due to the non-uniqueness of the jointly-epi square one single negative application condition $NAC_y$ may produce more than one negative borrowed context. Furthermore, the set $\{F \to N_z\}_{z \in Z}$ is in general infinite, but if we consider finite objects $L$, $NAC_y$ and $G^+$ (i.e., objects which have only finitely many subobjects) there exist only finitely many overlaps $M_z$ up to iso. Hence the set $\{F \to N_z\}_{z \in Z}$ can be finitely represented by forming appropriate isomorphism classes of arrows.

A concrete instance of Diagram (2) is discussed in Section 5 in relation with our running example.

**Definition 9 (Bisimulation and Bisimilarity with NACs).** *Let $\mathcal{P}$ be a set of productions with NACs and $\mathcal{R}$ a symmetric relation containing pairs of objects with interfaces $(J \to G, J \to G')$. The relation $\mathcal{R}$ is called a* bisimulation *if, for every $(J \to G)\,\mathcal{R}\,(J \to G')$ and a transition $(J \to G) \xrightarrow{J \to F \leftarrow K;\{F \to N_z\}_{z \in Z}} (K \to H)$, there exists an object with interface $K \to H'$ and a transition $(J \to G') \xrightarrow{J \to F \leftarrow K;\{F \to N_z\}_{z \in Z}} (K \to H')$ such that $(K \to H)\,\mathcal{R}\,(K \to H')$.*

*We write $(J \to G) \sim (J \to G')$ whenever there exists a bisimulation $\mathcal{R}$ that relates the two objects with interface. The relation $\sim$ is called* bisimilarity.

The difference between the bisimilarity of Definition 4 and the one above is the transition label, which in the latter case is enriched with negative borrowed contexts. Thus, Definition 9 yields in general a finer bisimulation.

We are now ready to show the congruence result. Recall that we are working in the framework of adhesive categories. Our main result below needs one extra requirement, namely that pullbacks preserve epis. The full proof of the following theorem with all lemmas is contained in the technical report [11].

**Theorem 2 (Bisimilarity based on Productions with NACs is a Congruence).** *The bisimilarity $\sim$ of Definition 9 is a congruence, i.e., it is preserved by contextualization as in Definition 2.*

*Proof (Sketch).* In [7] it was shown for the category of graph structures that bisimilarity derived from graph productions of the form $L \leftarrow I \rightarrow R$ with monos is a congruence. The pushout and pullback properties employed in [7] also hold for any adhesive category. Here we will extend the proof of [7] to handle productions with NACs in adhesive categories. All constructions used in this current proof are compliant with adhesive categories, except for some steps which require that pullbacks preserve epis.

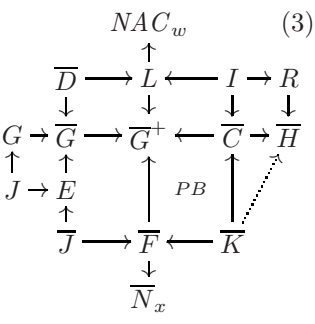
We will show that whenever $\mathcal{R}$ is a bisimulation, then $\hat{\mathcal{R}}$, which is the contextualization of $\mathcal{R}$ as in Definition 2, is also a bisimulation.

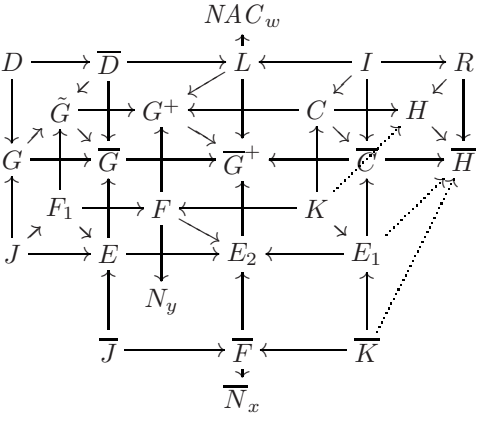
Let $\mathcal{R}$ be a bisimulation and let $(\overline{J} \to \overline{G})\ \hat{\mathcal{R}}\ (\overline{J} \to \overline{G}')$. That is, there is a pair $(J \to G)\ \mathcal{R}\ (J \to G')$ and a context $J \to E \leftarrow \overline{J}$ such that $\overline{J} \to \overline{G}$ and $\overline{J} \to \overline{G}'$ are obtained by inserting $J \to G$ and $J \to G'$ into this context.

Let us also assume that $(\overline{J} \to \overline{G}) \xrightarrow{\overline{J} \to \overline{F} \leftarrow \overline{K};\{\overline{F} \to \overline{N}_x\}_{x \in X}} (\overline{K} \to \overline{H})$. Our goal is to show that there exists a transition label $(\overline{J} \to \overline{G}') \xrightarrow{\overline{J} \to \overline{F} \leftarrow \overline{K};\{\overline{F} \to \overline{N}_x\}_{x \in X}} (\overline{K} \to \overline{H'})$ with $(\overline{K} \to \overline{H})\ \hat{\mathcal{R}}\ (\overline{K} \to \overline{H'})$, which implies that $\hat{\mathcal{R}}$ is a bisimulation. In *Step A* we construct a transition $(J \to G) \xrightarrow{J \to F \leftarrow K;\{F \to N_y\}_{y \in Y \cup Z}} (K \to H)$ which implies a transition $(J \to G') \xrightarrow{J \to F \leftarrow K;\{F \to N_y\}_{y \in Y \cup Z}} (K \to H')$ with $(K \to H)\ \mathcal{R}\ (K \to H')$, since $\mathcal{R}$ is a bisimulation. In *Step B* we extend the second transition to obtain the transition stated in our goal above. This argument is basically the same as in [7], except for the fact that here we are dealing

with a bisimulation definition involving transition labels with negative borrowed contexts.

**Step A:** From transition $(\overline{J} \to \overline{G}) \xrightarrow{\overline{J}\to\overline{F}\leftarrow\overline{K};\{\overline{F}\to\overline{N}_x\}_{x\in X}} (\overline{K} \to \overline{H})$ we can derive Diagram (3), where the decomposition of $\overline{J} \to \overline{G}$ is shown explicitly, all arrows are mono and all squares are pushouts, except for the indicated pullback.




From Diagram (3) we construct Diagram (4) according to [7], i.e., we project the borrowed context diagram of $\overline{J} \to \overline{G}$ to a borrowed context diagram of $J \to G$, first without taking into account NACs. The square $(K, H, E_1, \overline{H})$ is a pushout.

Observe that all negative borrowed contexts $\overline{N}_x$ of the transition are obtained via Diagram (7). It can be shown that such a diagram can be "decomposed"[4] into two Diagrams (5) and (6), where the former shows the derivation of negative borrowed contexts for $G^+$. That is, every negative borrowed context of the larger object $\overline{G^+}$ is associated with at least one negative borrowed context of the smaller object $G^+$. Note that the transformation of one negative borrowed context into the other via Diagram (6) is only dependent on the context $J \to E \leftarrow \overline{J}$, into which $J \to G$ is inserted, but not on $G$ itself, since $E_2$ is the pushout of $\overline{J} \to E$, $\overline{J} \to \overline{F}$. This independence of $G$ will allow us to use this construction for $J \to G'$ in *Step B*.

In addition there might be further negative borrowed contexts $F \to N_y$ with indices $y \in Z$, where $Y$ and $Z$ are disjoint index sets. These are exactly the negative borrowed contexts for which Diagram (6) can not be completed since the pushout complement does not exist. If we could complete Diagram (6) we would be able to reconstruct Diagram (7).

Hence we obtain a transition from $J \to G$ which satisfies Conditions (ii) and (iii) of Definition 8. We still have to show that the BC step for $G^+$ is executable

[4] We will use this "decomposition" result throughout the proof. Note that from Diagram (7) we can construct Diagrams (5) and (6) and vice versa. The proof of this result requires that pullbacks preserve epis.

(Condition (i)). By assumption, the BC step from $\overline{J} \to \overline{G}$ of Diagram (4) is executable. One can show that a transition is executable if and only if none of the derived negative borrowed contexts is an iso. This means that there does not exist any iso $\overline{F} \to \overline{N}_x$, which in turn implies that no $F \to N_y$, $y \in Y$ is an iso. Furthermore no $F \to N_y$ with $y \in Z$ can be an iso, since otherwise we could complete Diagram (6). Finally we conclude with the observation above that the BC step from $J \to G$ is executable.

Since all conditions of Definition 8 are satisfied, we can derive the transition $(J \to G) \xrightarrow{J \to F \leftarrow K; \{F \to N_y\}_{y \in Y \cup Z}} (K \to H)$ from Diagram (4) using Definition 9. Since $\mathcal{R}$ is a bisimulation, this implies $(J \to G') \xrightarrow{J \to F \leftarrow K; \{F \to N_y\}_{y \in Y \cup Z}} (K \to H')$ with $(K \to H)\ \mathcal{R}\ (K \to H')$. Additionally, we can infer from Diagram (4) that $\overline{K} \to \overline{H}$ is the insertion of $K \to H$ into the context $K \to E_1 \leftarrow \overline{K}$.

**Step B:** In *Step A* we have shown that $J \to G'$ can mimic $J \to G$ due to the bisimulation $\mathcal{R}$. Here we will show that $(\overline{J} \to \overline{G}')$ can also mimic $(\overline{J} \to \overline{G})$ since $\mathcal{R}$ is a bisimulation and both objects with interface are derived from the insertion of $J \to G$ and $J \to G'$ into the context $J \to E \leftarrow \overline{J}$.

We take the transition from $J \to G'$ to $K \to H'$ with $(K \to H)\ \mathcal{R}\ (K \to H')$ from *Step A* and construct a transition from $(\overline{J} \to \overline{G}')$ to $(\overline{K} \to \overline{H}')$ with $(\overline{K} \to \overline{H})\ \hat{\mathcal{R}}\ (\overline{K} \to \overline{H}')$. Recall that $\overline{J} \to \overline{G}'$ is $J \to G'$ in the context $J \to E \leftarrow \overline{J}$.

$$\begin{array}{ccccc} NAC_w & \to & M_y & \leftarrow & N_y \\ \uparrow & & \overset{j.epi}{=} & PO & \uparrow \\ L & \to & G^+ & \leftarrow & F \end{array} \quad (5)$$

$$\begin{array}{ccccc} N_y & \to & M'_x & \leftarrow & \overline{N}_x \\ \uparrow & & \overset{j.epi}{=} & PO & \uparrow \\ F & \to & E_2 & \leftarrow & \overline{F} \end{array} \quad (6)$$

$$\begin{array}{ccccc} NAC_w & \to & \overline{M}_x & \leftarrow & \overline{N}_x \\ \uparrow & & \overset{j.epi}{=} & PO & \uparrow \\ L & \to & \overline{G}^+ & \leftarrow & \overline{F} \end{array} \quad (7)$$

$$\begin{array}{ccccccccc} & & & & NAC_w & & & & \\ & & & & \uparrow & & & & \\ D' & \to & \overline{D}' & \to & L & \leftarrow & I & \to & R \\ \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow \\ G' & \to & \tilde{G}' & \to & G'^+ & \leftarrow & C' & \to & H' \\ \uparrow & \nearrow & \uparrow & \searrow & \uparrow & \searrow & \uparrow & \searrow & \downarrow \\ G' & \to & \overline{G}' & \to & \overline{G}'^+ & \leftarrow & \overline{C}' & \to & \overline{H}' \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \\ & & F_1 & \to & F & \leftarrow & K & & \\ & \nearrow & & \searrow & & \searrow & & & \\ J & \to & E & & \to & E_2 & \leftarrow & E_1 & \\ & & \downarrow & & & & & \uparrow & \\ & & N_y & & & & & & \\ & & \uparrow & & & & & & \\ & & \overline{J} & \to & \overline{F} & \leftarrow & \overline{K} & & \\ & & & & \downarrow & & & & \\ & & & & \overline{N}_x & & & & \end{array} \quad (8)$$

According to [7] we obtain Diagram (8), first without considering the NACs. The square $(K, H', E_1, \overline{H}')$ is a pushout. Then we construct $\{\overline{F} \to \overline{N}_x\}_{x \in X}$ as shown in Diagram (6). The arrows $F \to E_2 \leftarrow \overline{F}$ and $\{F \to N_y\}_{y \in Y}$ are already present in Diagram (8) and so we build $M'_x$ and $\overline{N}_x$ by considering all jointly epi squares. Each $\overline{F} \to \overline{N}_x$ constructed in this way can be also derived as a negative borrowed context with Diagram (7) (where $\overline{G}^+$ is replaced by $\overline{G}'^+$) due to the fact that we can construct Diagram (7) based on (5) and (6). Furthermore we will not derive additional negative borrowed contexts because the arrows $F \to N_y$ with $y \in Z$ can not be extended to negative borrowed contexts of the full object $\overline{G}'^+$

since an appropriate Diagram (6) does not exist. Hence we obtain a transition label from $\overline{J} \to \overline{G}'$ which satisfies Conditions (ii) and (iii) of Definition 8. We still have to show that the BC step for $\overline{G}'^{+}$ is executable (Condition (i)).

Observe that $F \to E_2 \leftarrow \overline{F}$ of Diagram (6) are equal in *Step A* and *Step B* and do not contain any information about $G$ or $G'$. Hence we can conclude that Diagram (6) generates the same negative borrowed contexts in both steps. Since in Diagram (4) there is no negative borrowed context which is an iso, the same holds for Diagram (8). By the observation concerning isos we conclude that the BC step from $\overline{J} \to \overline{G}'$ is also executable.

Finally, by Definition 9 we infer that $(\overline{J} \to \overline{G}') \xrightarrow{\overline{J} \to \overline{F} \leftarrow \overline{K}; \{\overline{F} \to \overline{N}_x\}_{x \in X}} (\overline{K} \to \overline{H}')$, and since the square $(K, H', E_1, \overline{H}')$ is a pushout, $\overline{K} \to \overline{H}'$ is $K \to H'$ inserted into the context $K \to E_1 \leftarrow \overline{K}$. From earlier considerations we know that $\overline{K} \to \overline{H}$ is obtained by inserting $K \to H$ into $K \to E_1 \leftarrow \overline{K}$. Hence, we can conclude that $(\overline{K} \to \overline{H})\ \hat{\mathcal{R}}\ (\overline{K} \to \overline{H}')$ and we have achieved our goal stated at the beginning of the proof, which implies that $\hat{\mathcal{R}}$ is a bisimulation and $\sim$ is a congruence.

## 4 Up-to Techniques for DPO-BC with NACs

Bisimulation proofs often need infinite relations. Up-to techniques [14] relieve the onerous task of bisimulation proofs by reducing the size of the relation needed to define a bisimulation. It is also possible to check bisimilarity with finite up-to relations in some cases where any bisimulation is infinite. We first need to define progression (see also [14]).

**Definition 10 (Progression with NACs).** *Let $\mathcal{R}$, $\mathcal{S}$ be relations containing pairs of objects with interfaces of the form $(J \to G, J \to G')$, where $\mathcal{R}$ is symmetric. We say that $\mathcal{R}$ progresses to $\mathcal{S}$, abbreviated by $\mathcal{R} \rightarrowtail \mathcal{S}$, if whenever $(J \to G)\,\mathcal{R}\,(J \to G')$ and $(J \to G) \xrightarrow{J \to F \leftarrow K; \{F \to N_z\}_{z \in Z}} (K \to H)$, there exists an object with interface $K \to H'$ such that $(J \to G') \xrightarrow{J \to F \leftarrow K; \{F \to N_z\}_{z \in Z}} (K \to H')$ with $(K \to H)\,\mathcal{S}\,(K \to H')$.*

According to Definition 9, a relation $\mathcal{R}$ is a bisimulation if and only if $\mathcal{R} \rightarrowtail \mathcal{R}$.

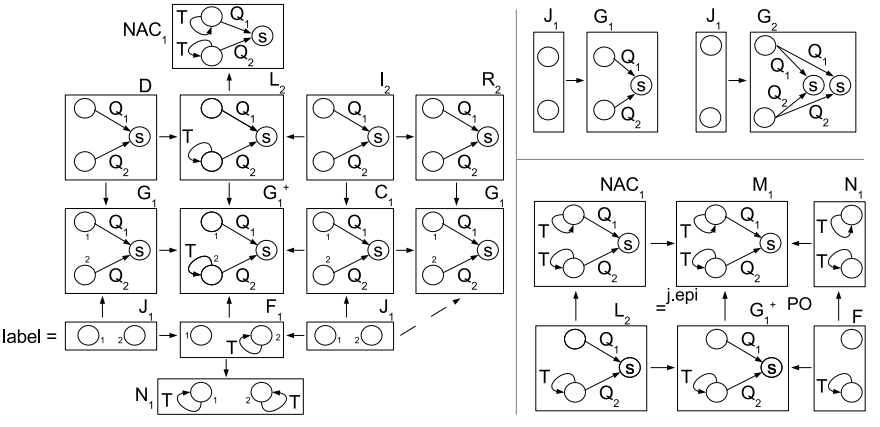
**Definition 11 (Bisimulation up to Context with NACs).** *Let $\mathcal{R}$ be a symmetric relation containing pairs of objects with interfaces of the form $(J \to G, J \to G')$. If $\mathcal{R} \rightarrowtail \hat{\mathcal{R}}$, where $\hat{\mathcal{R}}$ is the closure of $\mathcal{R}$ under contextualization, then $\mathcal{R}$ is called* bisimulation up to context.

**Proposition 1 (Bisimulation up to Context with NACs implies Bisimilarity).** *Let $\mathcal{R}$ be a bisimulation up to context. Then it holds that $\mathcal{R} \subseteq \sim$.*

*Proof.* Follows quite easily from the proof of Theorem 2 (see also [7]).

## 5 Example: Servers as Graphs with Interfaces

Here we apply the DPO-BC extension to NACs in order to check the bisimilarity of two graphs with interfaces $J_1 \to G_1$ and $J_1 \to G_2$ (shown below on the right) with respect to $\mathsf{rule}_1$ and $\mathsf{rule}_2$ of Section 3.1. Here $G_1$ contains only one server, whereas $G_2$ contains two severs which may work in parallel.




Above on the left we show a transition derivation for $J_1 \to G_1$ (which contains only one server) via $\mathsf{rule}_2$ according to Definition 8. There is no mono $\mathsf{NAC}_1 \to G_1^+$ forbidding the BC rewriting (Condition (i)) and the step is executable. The graph $C_1$ and additional monos lead to the BC step (Condition (ii)). The construction of the negative borrowed context $F_1 \to N_1$ from $\mathsf{NAC}_1 \leftarrow L_2 \to G_1^+ \leftarrow F_1$, as specified in Condition (iii), is shown on the right. Here the graph $M_1$ is the only possible overlap of $\mathsf{NAC}_1$ and $G_1^+$ such that the square with indicated jointly epi monos commutes. Since the pushout complement $F_1 \to N_1 \to M_1$ exists, $G_1^+$ can be indeed extended to $M_1$ by gluing $N_1$ via $F_1$. All three conditions of Definition 8 are satisfied and so the BC step above with $\mathsf{label} = J_1 \to F_1 \leftarrow J_1; \{F_1 \to N_1\}$ is feasible. This transition can be interpreted as follows: the environment provides $G_1$ with a T-task in $Q_2$ (see borrowed context $F_1$) in order to enable the BC step, but the rewriting is only possible if no T-task is waiting in queue $Q_1$ (see $N_1$).

Analogously we can derive other transitions from $J_1 \to G_1$ and $J_1 \to G_2$, where the labels generated via $\mathsf{rule}_1$ (without NAC) do not have any negative borrowed context. So $J_1 \to G_1$, $J_1 \to G_2$ and all their successors can be matched via a bisimulation and we conclude that $(J_1 \to G_1) \sim (J_1 \to G_2)$.

Note that in order to obtain an extended example, we could add a rule modeling the processing of tasks waiting in queue $Q_1$.

## 6 Conclusions and Future Work

We have shown how rules with NACs should be handled in the DPO with borrowed contexts and proved that the derived bisimilarity relation is a congruence.

This extension to NACs is relevant for the specification of several kinds of non-trivial systems, where complex conditions play a very important role. They are also frequently used when specifying model transformation, such as transformations of UML models. Behaviour preservation is an important issue for model transformation.

Here we have obtained a finer congruence than the usual one. Instead, if one would reduce the number of possible contexts (for instance by forbidding contexts that contain certain patterns or subobjects), we would obtain coarser congruences, i.e., more objects would be equivalent. Studying such congruences will be a direction of future work.

Furthermore, a natural question to ask is whether there are other extensions to the DPO approach that, when carried over to the DPO-BC framework, would require the modification of transition labels. One such candidate are generalized application conditions, so-called graph conditions [15], which are equivalent to first-order logic and of which NACs are a special case. Such conditions would lead to fairly complex labels.

Due to the fact that the bisimulation checking procedure is time consuming and error-prone when done by hand, we plan to extend the on-the-fly bisimulation checking algorithm, defined in [16,17], for productions with NACs. In order to do this efficiently we need further speed-up techniques such as additional up-to techniques and methods for downsizing the transition system, such as the elimination of independent labels. Preliminary investigations have already determined that the proof technique eliminating independent labels as in [6,7] (or non-engaged labels as they are called in [18]) does not carry over straightforwardly from the case without NACs.

Some open questions remain for the moment. First, in the categorical setting it would be good to know whether pullbacks always preserve epis in adhesive categories. This question is currently open, as far as we know. Second, it is unclear where the congruence is located in the lattice of congruences that respect rewriting steps with NACs. As for IPO bisimilarity it is probably not the coarsest such congruence, since saturated bisimilarity is in general coarser [19]. So it would be desirable to characterize such a congruence in terms of barbs [20].

Also, it is not clear to us at the moment how NACs could be integrated directly into reactive systems and how the corresponding notion of IPO would look like. In our opinion this would lead to fairly complex notions, for instance one would have to establish a concept similar to that of jointly epi arrows.

**Acknowledgements.** We would like to thank Tobias Heindel for helpful discussions on this topic.

## References

1. Milner, R.: Communication and concurrency. Prentice-Hall, Englewood Cliffs (1989)
2. Milner, R., Parrow, J.: A calculus for mobile process I. Information and Computation 100, 1–40 (1992)

3. Milner, R., Parrow, J., Walker, D.: A calculus for mobile process II. Information and Computation 100, 41–77 (1992)
4. Leifer, J.J.: Operational Congruences for Reactive Systems. PhD thesis, University of Cambridge Computer Laboratory (2001)
5. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
6. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 151–166. Springer, Heidelberg (2004)
7. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. Mathematical Structures in Computer Science 16(6), 1133–1163 (2006)
8. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: Proc. of LICS 2005, pp. 311–320. IEEE, Los Alamitos (2005)
9. Sobociński, P.: Deriving process congruences from reaction rules. PhD thesis, Department of Computer Science, University of Aarhus (2004)
10. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundam. Inf. 26(3–4), 287–313 (1996)
11. Rangel, G., König, B., Ehrig, H.: Deriving bisimulation congruences in the presence of negative application conditions. Technical Report 2008-1, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen (to appear, 2008)
12. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Loewe, M.: Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph transformation, Foundations, vol. 1, pp. 163–246. World Scientific, Singapore (1997)
13. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. RAIRO - Theoretical Informatics and Applications 39(2), 522–546 (2005)
14. Sangiorgi, D.: On the proof method for bisimulation. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 479–488. Springer, Heidelberg (1995)
15. Rensink, A.: Representing first-order logic using graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004)
16. Rangel, G., König, B., Ehrig, H.: Bisimulation verification for the DPO approach with borrowed contexts. In: Proc. of GT-VMT 2007. Electronic Communications of the EASST, vol. 6 (2007)
17. Hirschkoff, D.: Bisimulation verification using the up-to techniques. International Journal on Software Tools for Technology Transfer 3(3), 271–285 (2001)
18. Milner, R.: Pure bigraphs: structure and dynamics. Inf. Comput. 204(1), 60–122 (2006)
19. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS 2006, pp. 69–80. IEEE, Los Alamitos (2006)
20. Rathke, J., Sassone, V., Sobociński, P.: Semantic barbs and biorthogonality. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 302–316. Springer, Heidelberg (2007)

# Structural Operational Semantics for Stochastic Process Calculi

Bartek Klin[1] and Vladimiro Sassone[2]

[1] Warsaw University, University of Edinburgh
[2] ECS, University of Southampton

**Abstract.** A syntactic framework called SGSOS, for defining well-behaved Markovian stochastic transition systems, is introduced by analogy to the GSOS congruence format for nondeterministic processes. Stochastic bisimilarity is guaranteed a congruence for systems defined by SGSOS rules. Associativity of parallel composition in stochastic process algebras is also studied within the framework.

## 1 Introduction

Process algebras such as CCS [18] or CSP [5] are widely accepted as useful tools for compositional modeling of nondeterministic, communicating processes. Their semantics is usually described within the framework of Structural Operational Semantics (SOS) [19], where labelled nondeterministic transition systems (LTSs) are defined by induction on the syntactic structure of processes. Formalisms for SOS decriptions of nondeterministic systems have been widely studied and precisely defined (see [1] for a survey). In particular, several syntactic formats have been developed that guarantee certain desirable properties of the induced systems, most importantly that bisimulation is a congruence on them.

Stochastic process algebras have been deployed for applications in performance evaluation, and more recently in systems biology, where the underpinning of labelled continuous time Markov chains (CTMCs), and more generally stochastic processes, is required rather than simple LTSs. Examples of such algebras include TIPP [11], PEPA [15], EMPA [3], and stochastic $\pi$-calculus [20]. Semantics of these calculi have been given by variants of the SOS approach. However, in contrast with the case of nondeterministic processes, SOS formalisms used here are not based on any general framework for operational descriptions of stochastic processes, and indeed differ substantially from one another. This is unfortunate, as such a framework would make languages easier to understand, compare, and extend. Specifically, a format for SOS descriptions which guarantees the compositionality of stochastic bisimilarity, would make extending process algebras with new operators a much simpler task, liberating the designer from the challenging and time-consuming task of proving congruence results.

In this paper we define such a *congruence format*, which we call *SGSOS*. First we review existing approaches to the operational semantics of process algebras, concentrating on the examples of PEPA [15] and the stochastic $\pi$-calculus [20]. As the operational techniques used there seem hard to extend to a general format for well-behaved stochastic specifications, we resolve to adapt a general theory of well-behaved SOS, based on

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 428–442, 2008.

category theory and developed by Turi and Plotkin [24]. The inspiration for our approach comes directly from the work of F. Bartels [2], who used Turi and Plotkin's results to design a congruence format for probabilistic transition systems.

Standard operations of stochastic process algebras, as well as plenty of non-standard but potentially useful ones, fall within our format. Exceptions are recursive definitions and name-passing features of stochastic $\pi$-calculus, which we leave for future work.

Within the SGSOS framework, we also investigate the issue of *associativity of parallel composition* in stochastic process algebras, a design issue that, to our knowledge, has been overlooked in the literature. We notice in fact that in the original definition of stochastic $\pi$-calculus, parallel composition fails to be associative up to stochastic bisimilarity, and study conditions under which two forms of parallel composition, CSP-style synchronization and CCS-style communication, are associative.

The structure of the paper is as follows. In §2 we recall previously studied approaches to operational semantics of nondeterministic and stochastic systems. In §3 the bialgebraic theory of well-behaved SOS is recalled. In §4 we adapt the theory to obtain the SGSOS congruence format, with simple examples of GSOS specifications following in §5. The associativity of parallel composition is studied in §6, and in §7 we mention some directions of future work. Due to lack of space, all proofs are omitted in this extended abstract.

## 2 Transition Systems and Process Calculi

We begin our development by comparing two previously studied approaches to defining SOS for Markovian process algebras with the well-known world of SOS for nondeterministic systems such as CCS.

### 2.1 Nondeterministic Systems and GSOS

A *labelled transition system* (LTS) is a triple $(X, A, \longrightarrow)$, with $X$ a set of *states*, $A$ a set of *labels* and $\longrightarrow \subseteq X \times A \times X$ a labelled *transition relation*, typically written $x \xrightarrow{a} y$ for $(x, a, y) \in \longrightarrow$. An LTS is *image-finite* if for every $x \in X$ and $a \in A$ there are only finitely many $y \in X$ such that $x \xrightarrow{a} y$. In the context of Structural Operational Semantics (SOS), LTS states are terms, and transition relations are defined inductively, by means of inference rules. For example, in a fragment of CCS [18], processes are terms over the grammar $P ::= \mathtt{nil} \mid a.P \mid P + P \mid P \parallel P$, and the LTS is induced from the following rules:

$$\frac{}{a.\mathtt{x} \xrightarrow{a} \mathtt{x}} \qquad \frac{\mathtt{x}_1 \xrightarrow{a} \mathtt{y}}{\mathtt{x}_1+\mathtt{x}_2 \xrightarrow{a} \mathtt{y}} \qquad \frac{\mathtt{x}_2 \xrightarrow{a} \mathtt{y}}{\mathtt{x}_1+\mathtt{x}_2 \xrightarrow{a} \mathtt{y}} \tag{1}$$

$$\frac{\mathtt{x}_1 \xrightarrow{a} \mathtt{y}}{\mathtt{x}_1\|\mathtt{x}_2 \xrightarrow{a} \mathtt{y}\|\mathtt{x}_2} \qquad \frac{\mathtt{x}_2 \xrightarrow{a} \mathtt{y}}{\mathtt{x}_1\|\mathtt{x}_2 \xrightarrow{a} \mathtt{x}_1\|\mathtt{y}} \qquad \frac{\mathtt{x}_1 \xrightarrow{a} \mathtt{y}_1 \quad \mathtt{x}_2 \xrightarrow{\bar{a}} \mathtt{y}_2}{\mathtt{x}_1\|\mathtt{x}_2 \xrightarrow{\tau} \mathtt{y}_1\|\mathtt{y}_2}$$

Plenty of operators can be defined formally by rules like these. Indeed, the above specification is an instance of a general framework for SOS definitions of LTSs (see e.g., [1]), called *GSOS* and defined formally as follows.

An *algebraic signature* is a set $\Sigma \ni \mathtt{f}, \mathtt{g}, \ldots$ of *operation symbols* with an *arity* function $ar : \Sigma \to \mathbb{N}$, usually left implicit. The set of all terms over $\Sigma$ with variables from set $X$ is denoted $T_\Sigma X$. In particular, $T_\Sigma 0$ is the set of closed $\Sigma$-terms.

Fix a countably infinite set $\Xi \ni \mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots$ of variables. A *GSOS inference rule* [4] over a signature $\Sigma$ and a set of labels $A$ is an expression of the form

$$\frac{\left\{\mathtt{x}_{i_j} \xrightarrow{a_j} \mathtt{y}_j\right\}_{1\le j\le k} \quad \left\{\mathtt{x}_{i_l} \xrightarrow{b_l}\!\!\!\!\!\!\rightarrow\right\}_{1\le i\le m}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{c} \mathtt{t}} \tag{2}$$

where $\mathtt{f} \in \Sigma$, $n = ar(\mathtt{f})$, $k, m \in \mathbb{N}$, $i_j, i_l \in \{1, \ldots, n\}$, $a_j, b_l, c \in A$, $\mathtt{t} \in T_\Sigma \Xi$, $\mathtt{x}_i$ and $\mathtt{y}_j \in \Xi$ are all distinct and no other variables occur in the term $\mathtt{t}$. Expressions above the horizontal line in a GSOS rule are called its *premises*, and the expression below it is the *conclusion*. A *GSOS specification* is a set of GSOS rules; it is *image-finite* if it contains only finitely many rules for each $\mathtt{f}$ and $c$.

Every GSOS specification $\Lambda$ *induces* an LTS $(T_\Sigma 0, A, \longrightarrow)$, with the transition relation $\longrightarrow$ defined by induction of the syntactic structure of the source states. For a term $s = \mathtt{f}(s_1, \ldots, s_n) \in T_\Sigma 0$, one adds a transition $s \xrightarrow{c} t$ for each substitution $\sigma : \Xi \to T_\Sigma 0$ such that for some rule $r \in \Lambda$ as in (2), there is $\sigma\mathtt{x}_i = s_i$, $\sigma\mathtt{t} = t$, and $\sigma$ *satisfies* all premises of $r$, meaning that for each premise $\mathtt{x} \xrightarrow{a} \mathtt{y}$ there is $\sigma\mathtt{x} \xrightarrow{a} \sigma\mathtt{y}$, and for each premise $\mathtt{x} \xrightarrow{a}\!\!\!\!\!\!\rightarrow$ there is no $t \in T_\Sigma 0$ for which $\sigma\mathtt{x} \xrightarrow{a} t$.

An important property of the LTS induced by $\Lambda$ is that bisimilarity on it is guaranteed to be a congruence with respect to the syntactic structure of states. This means that GSOS is a *congruence format* for bisimilarity on LTSs. Moreover, it is easy to prove by induction that the LTS induced by an image-finite GSOS specification is image-finite.

### 2.2 Stochastic Systems

Just as nondeterministic process algebras are defined using labelled transition systems, the semantics of stochastic processes is often provided by labelled continuous time Markov chains (CTMCs). These are conveniently presented in terms of what we shall call *rated transition systems* (RTSs), i.e., triples $(X, A, \rho)$, where $X$ is a set of states, $A$ a set of labels and $\rho : X \times A \times X \to \mathbb{R}_0^+$ is a rate function, equivalently presented as an $A$-indexed family of $\mathbb{R}_0^+$-valued matrices. The number $\rho(x, a, y)$ is the parameter of an exponential probability distribution governing the duration of the transition of $x$ to $y$ with label $a$ (for more information and intuition on CTMCs and their presentation by transition rates see e.g. [12,15,20]). For the sake of readability we will write $\rho(x \xrightarrow{a} y)$ instead of $\rho(x, a, y)$, and $x \xrightarrow{a,r} y$ will indicate that $\rho(x \xrightarrow{a} y) = r$. The latter notation suggests that RTSs can be seen as a special kind of $A \times \mathbb{R}_0^+$-labelled nondeterministic transition systems; more specifically, exactly those that are "rate-deterministic," i.e., such that for each $x, y \in X$ and $a \in A$ there exists exactly one $r \in \mathbb{R}_0^+$ for which $x \xrightarrow{a,r} y$.

In the following we will consider *image-finite* processes, i.e. such that for each $x \in X$ and $a \in A$ there are only finitely many $y \in X$ such that $\rho(x, a, y) > 0$. For such processes, the sum

$$\rho_a(x) = \sum_{y \in X} \rho(x \xrightarrow{a} y) \tag{3}$$

exists for each $x \in X$ and $a \in A$; it will be called the *apparent rate* of label $a$ in state $x$. Further, $\rho(x \xrightarrow{a} y)/\rho_a(x)$ is called the *conditional probability* of the transition $x \xrightarrow{a} y$. It is the probability that $x$ makes the transition provided that it makes some $a$-transition.

Various equivalence relations on states in RTSs have been considered. Of those, the most significant is *stochastic bisimilarity* (called strong equivalence in [14], and inspired by the notion of probabilistic bisimilarity from [17]), defined as follows. Given an RTS with state space $X$, a *stochastic bisimulation* is an equivalence relation $R$ on $X$ such that whenever $x\,R\,y$ then for each $a \in A$, and for each equivalence class $C$ with respect to $R$,

$$\sum_{z \in C} \rho(x \xrightarrow{a} z) = \sum_{z \in C} \rho(y \xrightarrow{a} z).$$

Two states are *bisimilar* if they are related by some bisimulation. It is easy to check that bisimilarity is itself an equivalence relation and indeed the largest bisimulation.

Due to the additional rate component present in transitions, the traditional approach to SOS recalled in §2.1 is inadequate for modeling stochastic process calculi. Instead, other variants of SOS have been used for this purpose. For a comparison with the following development, we recall two of these variants: the multi-transition system approach used for the stochastic calculus PEPA [14,15], and the proved SOS approach of stochastic $\pi$-calculus [20,21,22].

In (a fragment of) PEPA, processes are terms over the grammar:

$$P ::= \mathtt{nil} \mid (a, r).P \mid P + P \mid P \underset{L}{\bowtie} P$$

where $a$ ranges over a fixed set $A$ of labels, $L$ over subsets of $A$, and $r$ over $\mathbb{R}^+$. Their semantics is defined by inference rules:

$$\frac{}{(a,r).\mathtt{x} \xrightarrow{a,r} \mathtt{x}} \qquad \frac{\mathtt{x}_1 \xrightarrow{a,r} \mathtt{y}}{\mathtt{x}_1 + \mathtt{x}_2 \xrightarrow{a,r} \mathtt{y}} \qquad \frac{\mathtt{x}_2 \xrightarrow{a,r} \mathtt{y}}{\mathtt{x}_1 + \mathtt{x}_2 \xrightarrow{a,r} \mathtt{y}}$$

$$\frac{\mathtt{x}_1 \xrightarrow{a,r} \mathtt{y}}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{a,r} \mathtt{y} \underset{L}{\bowtie} \mathtt{x}_2} \qquad \frac{\mathtt{x}_2 \xrightarrow{a,r} \mathtt{y}}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{a,r} \mathtt{x}_1 \underset{L}{\bowtie} \mathtt{y}} \quad (a \notin L) \tag{4}$$

$$\frac{\mathtt{x}_1 \xrightarrow{a,r_1} \mathtt{y}_1 \quad \mathtt{x}_2 \xrightarrow{a,r_2} \mathtt{y}_2}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{a,R} \mathtt{y}_1 \underset{L}{\bowtie} \mathtt{y}_2} \quad (a \in L)$$

where $a \in A$ and $r, r_1, r_2, R \in \mathbb{R}^+$ with $R$ depending on $r_1, r_2$ according to an application-specific formula (see below). Note that instead of a single parallel composition operator, PEPA provides a *cooperation operator* $\underset{L}{\bowtie}$ for each set $L$ of labels. These operators are based on CSP-style synchronisation [5] rather than CCS-style communication [18].

It turns out that the standard interpretation of the above rules as described in §2.1 would (among other things) contradict the intended meaning of the operator + as a *stochastic choice*, where a process $P + P$ can perform the same transitions as $P$, with twice the rates. In particular, the processes $P$ and $P + P$ should not be stochastic bisimilar. This is why the semantics of PEPA is given as a *multi-transition* system labeled

with pairs $(a, r) \in A \times \mathbb{R}^+$, which is a transition system whose transition relation is a *multi*set of triples $(x, (a, r), y)$. To define such a semantics for PEPA, the rules in (4) are interpreted similarly as the GSOS rules in §2.1, where the multiplicity of a transition is determined by counting all its different derivations. To obtain an RTS from the induced multi-transition system, one then discards multiplicities by summing up all their rates in single rated transitions. Thus, for example, the process $(a, 3).\mathtt{nil} + (a, 3).\mathtt{nil}$ in the induced multi-transition systems has two identical transitions to $\mathtt{nil}$ with label $(a, 3)$, whilst in the final RTS it can make a single transition to $\mathtt{nil}$ with label $a$ and rate 6. For more details of this construction, see [14].

The formula for calculating $R$ based on $r_1$ and $r_2$ in the last rule of (4) depends on the intended meaning of synchronisation. In applications to performance evaluation [14], the formula

$$R = \min(\rho_a(\mathtt{x}_1), \rho_a(\mathtt{x}_2)) \cdot \frac{r_1}{\rho_a(\mathtt{x}_1)} \cdot \frac{r_2}{\rho_a(\mathtt{x}_2)} \tag{5}$$

is a natural choice. We shall call it the *minimal rate law*, since in the resulting RTS, the apparent rate of $a$ in $P \underset{L}{\bowtie} Q$ (with $a \in L$) is the least of the apparent rates of $P$ and $Q$. For applications to systems biology, where rates model concentrations of molecules, a more convenient choice is

$$R = r_1 \cdot r_2, \tag{6}$$

which following [6] we call the *mass action law*. The apparent rate of $a$ in $P \underset{L}{\bowtie} Q$ (with $a \in L$) here is the product of the corresponding apparent rates of $P$ and $Q$. For an intuitive motivation for these and other similar formulae, see [13].

A different approach was used to define semantics of stochastic $\pi$-calculus [20]. Since stochastic features of the calculus are independent from its name-passing aspects, for simplicity we discuss it here on a fragment of the calculus that corresponds to a stochastic version of CCS (see §2.1). Thus we consider, as processes, terms over the grammar:

$$P ::= \mathtt{nil} \mid (a, r).P \mid P + P \mid P \| P$$

where $a$ ranges over a fixed set $A$ of labels, and $r$ over $\mathbb{R}^+$. For the semantics, the authors of [20] decided to avoid multi-transition systems and rely on the standard process of LTS induction from inference rules. For this, to model stochastic choice and communication accurately, they enriched transition labels substantially, equipping them with encodings of derivations that lead to them. In this *proved operational semantics*, our "stochastic CCS" fragment of stochastic $\pi$-calculus would be defined by:

$$\frac{}{(a,r).\mathtt{x} \xrightarrow{(a,r)} \mathtt{x}} \qquad \frac{\mathtt{x}_1 \xrightarrow{\theta} \mathtt{y}}{\mathtt{x}_1 + \mathtt{x}_2 \xrightarrow{+_1\theta} \mathtt{y}} \qquad \frac{\mathtt{x}_2 \xrightarrow{\theta} \mathtt{y}}{\mathtt{x}_1 + \mathtt{x}_2 \xrightarrow{+_2\theta} \mathtt{y}} \tag{7}$$

$$\frac{\mathtt{x}_1 \xrightarrow{\theta} \mathtt{y}}{\mathtt{x}_1 \| \mathtt{x}_2 \xrightarrow{\|_1\theta} \mathtt{y} \| \mathtt{x}_2} \qquad \frac{\mathtt{x}_2 \xrightarrow{\theta} \mathtt{y}}{\mathtt{x}_1 \| \mathtt{x}_2 \xrightarrow{\|_2\theta} \mathtt{x}_1 \| \mathtt{y}} \qquad \frac{\mathtt{x}_1 \xrightarrow{\theta_1(a,r_1)} \mathtt{y}_1 \quad \mathtt{x}_2 \xrightarrow{\theta_2(\bar{a},r_2)} \mathtt{y}_2}{\mathtt{x}_1 \| \mathtt{x}_2 \xrightarrow{\langle \|_1\theta_1(a,r_1), \|_1\theta_2(\bar{a},r_2)\rangle, R} \mathtt{y}_1 \| \mathtt{y}_2}$$

where $\theta$ ranges over *derivation proofs*, e.g. represented by terms of the grammar:

$$\theta = (a, r) \mid +_1\theta \mid +_2\theta \mid \|_1\theta \mid \|_2\theta \mid \langle \|_1\theta, \|_2\theta \rangle,$$

and where $R$ depends on $r_1$ and $r_2$ according to the minimal rate law [20] or the mass action law [22], as in PEPA.

These rules are then used to induce an LTS, which results in relatively complex labels. To obtain an RTS, one then extracts more familiar labels $a \in A$ from proofs in the obvious way, by adding up rates of identical transitions. Thus, for example, the process $P = (a, 3).\mathtt{nil} + (a, 3).\mathtt{nil}$ in the induced LTS can make two distinct transitions $P \xrightarrow{+_1(a,3)} \mathtt{nil}$ and $P \xrightarrow{+_2(a,3)} \mathtt{nil}$, and in the final RTS it can make a transition to $\mathtt{nil}$ with label $a$ and rate 6.

Although both the multi- and the proved-transition approaches work fine for the specific examples described above, it appears difficult to extend any of them to a general framework for defining operational semantics for stochastic transition systems. Consider for example the proved SOS approach of stochastic $\pi$-calculus. As in the case of GSOS for nondeterministic systems, a well-behaved semantic framework should guarantee that stochastic bisimilarity is a congruence for the induced RTS. This is the case for our CCS example above, but it is easy to write examples where it fails; for example, extend the CCS language with a unary operator $\mathtt{f}$ with semantics defined by a rule:

$$\frac{\mathtt{x} \xrightarrow{+_1\theta} \mathtt{y}}{\mathtt{f(x)} \xrightarrow{\mathtt{f}+_1\theta} \mathtt{y}}$$

and see that, although $(a, 2).\mathtt{nil} + \mathtt{nil}$ and $\mathtt{nil} + (a, 2).\mathtt{nil}$ are stochastic bisimilar, they are not so when put in context $\mathtt{f}(-)$, since only the former process can make a step in this context. Clearly, this is because the structure of a proof is inspected in the premise of the rule. However, it would be wrong to forbid such inspection altogether, as it is needed, e.g. in the communication rule for stochastic $\pi$-calculus.

The source of the problem is the richness of labels in the proved approach to SOS. In [8], it is claimed that proofs as transition labels carry almost all information about processes that is ever needed. Indeed, it appears they may sometimes carry excessive information; in a well-behaved SOS framework they should only carry as much data as required for the derivation of the intended semantics (here, an RTS), not a bit more.

The same criticism, though perhaps to a lesser extent, can be moved to the multi-transition systems approach used in the semantics of PEPA, where transition multiplicities are the superfluous data. In the process of multi-transition system induction, two identical transitions of rate 3 are distinguished from a single transition of rate 6. As a result, one can write specifications such as

$$\frac{\mathtt{x} \xrightarrow{a,r} \mathtt{y}}{\mathtt{f(x)} \xrightarrow{a,\max(r,5)} \mathtt{y}}$$

and see that, although processes $(a, 3).\mathtt{nil} + (a, 3).\mathtt{nil}$ and $(a, 6).\mathtt{nil}$ are stochastic bisimilar, they are not so in the context $\mathtt{f}(-)$. On the other hand, forbidding arbitrary dependency of transition rates on subprocesses rates is hard to contemplate, since that forms the very core of PEPA.

It may be possible to determine the exact range of constructs and formulas that must be forbidden in the proved- or in the multi-transition approach in order to guarantee that stochastic bisimilarity is compositional. Indeed, this approach has been used with

success in the related framework of probabilistic processes [16], where a well-behaved version of the proved semantics is developed. In this paper, however, we take a more principled approach and derive a formalism for stochastic operational semantics from an abstract theory of congruence formats developed in [24] and applied to the case of probabilistic transition systems in [2].

## 3 An Abstract Approach to SOS

Our approach to a stochastic counterpart of the GSOS framework of §2.1 is based on a categorical generalisation of GSOS, developed by Plotkin and Turi in [24]. In this section we briefly recall that work; in the rest of the paper we develop a syntactic format for stochastic SOS as an instance of the general framework.

### 3.1 Transition Systems as Coalgebras

The abstract study of well-behaved structural operational semantics is based on modeling the behaviour of processes via coalgebras, and their syntax via algebras. The original motivating example is that of LTSs: for a fixed set $A$ of labels, image-finite LTSs can be seen as functions $h : X \to (\mathcal{P}_\omega X)^A$ (here, $\mathcal{P}_\omega$ is the finite powerset construction), along the correspondence $y \in h(x)(a)$ if and only if $x \xrightarrow{a} y$. More generally, for any covariant functor $B$ on the category **Set** of sets and functions, a *B-coalgebra* is a set $X$ (the *carrier*) and a function $h : X \to BX$ (the *structure*). Thus image-finite LTSs are coalgebras for the functor $(\mathcal{P}_\omega -)^A$.

A $B$-coalgebra *morphism* from a $h : X \to BX$ to $g : Y \to BY$ is a function $f : X \to Y$ such that the equation $g \circ f = Bf \circ h$ holds. This notion provides a general coalgebraic treatment of process equivalences: a *bisimulation* on a coalgebra $h : X \to BX$ is a binary relation $Q \subseteq X \times X$ such that for some coalgebra structure $q : Q \to BQ$ the projections $\pi_1, \pi_2 : Q \to X$ extend to a span of coalgebra morphisms from $q$ to $h$. For example, for $B = (\mathcal{P}_\omega -)^A$, this *span bisimulation* specializes to the well-known notion of LTS bisimulation [18]. For more information about the coalgebraic approach to process theory, see [23].

We now show how to view RTSs as coalgebras for a suitable functor on **Set**. Call a function $f : X \to \mathbb{R}_0^+$ *finitely supported* if the set $\{x \in X \mid f(x) > 0\}$ is finite. For any set $X$, let $\mathcal{R}_\omega X$ be the set of all finitely supported functions from $X$ to $\mathbb{R}_0^+$. This extends to a functor $\mathcal{R}_\omega$ on **Set**, with the action $\mathcal{R}_\omega f$ on function $f : X \to Y$ defined by

$$\mathcal{R}_\omega f(g)(y) = \sum_{f(x)=y} g(x),$$

for $g \in \mathcal{R}_\omega X$ and $y \in Y$. Since $g$ is finitely supported the sum exists and $\mathcal{R}_\omega f(g)$ is finitely supported too. Functoriality of $\mathcal{R}_\omega$ is then easy to check.

Fix an arbitrary set $A$ of labels. Coalgebras for the functor

$$BX = (\mathcal{R}_\omega X)^A$$

are exactly image-finite rated transition systems as defined in §2.2. Indeed, a coalgebra $h : X \to BX$ is an image-finite RTS with states $X$ along the correspondence:

$$x \xrightarrow{a,r} y \quad \text{if and only if} \quad r = h(x)(a)(y).$$

This coalgebraic treatment of RTSs is justified by the following statement.

**Proposition 1.** Span bisimulations on $(\mathcal{R}_\omega -)^A$-coalgebras, when restricted to equivalence relations, are exactly stochastic bisimulations as defined in §2.2.

In the following, a technical property of the functor $(\mathcal{R}_\omega -)^A$ will be useful:

**Proposition 2.** $(\mathcal{R}_\omega -)^A$ preserves weak pullbacks.

To prove the above two results, proceed exactly as in [7] for the case of probabilistic bisimulation and the corresponding behaviour functor.

### 3.2 Process Syntax Via Algebras

In the context of SOS, processes typically are closed terms over some algebraic signature, i.e., a set $\Sigma \ni \mathtt{f}, \mathtt{g}, \ldots$ of operation symbols with an arity function $ar : \Sigma \to \mathbb{N}$. Such a signature corresponds to a functor $\Sigma X = \coprod_{\mathtt{f} \in \Sigma} X^{ar(\mathtt{f})}$ on **Set**, in the sense that a model for the signature is exactly an *algebra* for the functor, i.e., a set $X$ (the *carrier*) and a function $g : \Sigma X \to X$ (the *structure*).

The set of terms over a signature $\Sigma$ and a set $X$ of variables is denoted by $T_\Sigma X$; in particular, $T_\Sigma 0$ is the set of closed terms over $\Sigma$ and it admits an obvious algebra structure $a : \Sigma T_\Sigma 0 \to T_\Sigma 0$ for the *functor* $\Sigma$ corresponding to the signature. This is the *initial* $\Sigma$-algebra. The construction $T_\Sigma$ is also a functor, called the *free monad* over $\Sigma$.

### 3.3 SOS Rules, Distributive Laws, Bialgebras

In [24], Turi and Plotkin proposed an elegant treatment of well-behaved SOS at the level of algebras and coalgebras. Their main motivating application was GSOS (see §2.1). Turi and Plotkin observed (full proof provided later by Bartels [2]), that image finite GSOS specifications are in an essentially one-to-one correspondence with *distributive laws*, i.e., natural transformations of the type

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT_\Sigma \tag{8}$$

where $B = (\mathcal{P}_\omega -)^A$ is the behaviour functor used for modeling LTSs, $\Sigma$ is the functor corresponding to the given signature, and $T_\Sigma$ is the free monad over $\Sigma$. Informally, (8) says that 'structural' combinations ($\Sigma$) of behaviours ($B$) are mapped to the behaviour of terms ($BT_\Sigma$), which is the essence of a SOS rule, with Id accounting for subterms that stay idle in a transition. Moreover, any $\lambda$ as above gives rise to a $B$-coalgebra structure $h_\lambda$ on $T_\Sigma 0$, defined by a "structural recursion theorem" (see [24] for details) as the only function $h_\lambda : T_\Sigma 0 \to BT_\Sigma 0$ such that:

$$h_\lambda \circ a = Ba^\sharp \circ \lambda_X \circ \Sigma\langle \mathrm{id}, h_\lambda \rangle. \tag{9}$$

The fact that bisimilarity on LTSs induced from GSOS specifications is guaranteed to be a congruence, can be proved at the level of coalgebras and distributive laws:

**Theorem 1 ([24], Cor. 7.5).** If a functor $B$ on **Set** preserves weak pullbacks, then for any $\lambda$ as in (8), span bisimilarity on $h_\lambda : T_\Sigma 0 \to BT_\Sigma 0$ is a congruence on $T_\Sigma 0$.

This result, together with Propositions 1 and 2, is the basis of our search for a congruence format for stochastic systems.

## 4 Stochastic GSOS

We now proceed to the main technical contribution of this paper: a complete characterisation of distributive laws (8) for stochastic systems in terms of inference rules. To find the characterisation, we closely follow the technique used by Bartels [2] for the case of probabilistic transition systems.

**Definition 1.** An *SGSOS rule* for a signature $\Sigma$ and a set $A$ of labels is an expression of the form:

$$\frac{\left\{\mathtt{x}_i \xrightarrow{a@r_{ai}}\right\}_{a\in D_i, 1\le i\le n} \quad \left\{\mathtt{x}_{i_j} \xrightarrow{b_j} \mathtt{y}_j\right\}_{1\le j\le k}}{\mathtt{f}(\mathtt{x}_1,\ldots,\mathtt{x}_n) \xrightarrow{c@W} \mathtt{t}} \tag{10}$$

where

- $\mathtt{f}\in\Sigma$ and $ar(\mathtt{f}) = n$, with $n, k \in \mathbb{N}$, and $\{i_1,\ldots,i_k\} \subseteq \{1,\ldots,n\}$;
- $\mathtt{x}_i$ and $\mathtt{y}_j$ are all distinct variables and no other variables appear in $\mathtt{t} \in T_\Sigma\Xi$; moreover, all variables $\mathtt{y}_j$ appear in $\mathtt{t}$;
- $D_i \subseteq A$, $c \in A$ and $b_j \in D_{i_j}$,
- $W \in \mathbb{R}^+$, $r_{ai} \in \mathbb{R}^+_0$, and moreover $r_{b_j i_j} > 0$, for $j = 1,\ldots,k$.

A rule is *triggered* by a tuple of real values $(v_{ai})_{a\in A, 1\le i\le n}$ if $v_{ai} = r_{ai}$ for all $1 \le i \le n$ and all $a \in D_i$. A collection of rules is called an *SGSOS specification* if for every $\mathtt{f}\in\Sigma$, $c \in A$, every tuple $(v_{ai})$ triggers only finitely many rules with $\mathtt{f}$ and $c$ in the conclusion.

In order to complete the definition of SGSOS we need describe how to derive an RTS from an SGSOS specification. Intuitively, a rule as in (10) contributes to the rate of a $c$-labeled transition from $\mathtt{f}(s_1,\ldots,s_n)$ if the *apparent* $a$-rates (see Eqn. (3)) of the $s_i$ match the corresponding $r_{ai}$; its contribution depends on $W$ and on *conditional probabilities* of a selection $s_{i_j} \xrightarrow{b_j} u_j$ of transitions from the $s_i$. Formally:

**Definition 2.** Every SGSOS specification $\Lambda$ *induces* a rated transition system $(T_\Sigma 0, A, \rho)$, with the rate function $\rho$ defined by induction on the first argument as follows. For a term $s = \mathtt{f}(s_1,\ldots,s_n) \in T_\Sigma 0$, assume that $\rho(s_i \xrightarrow{a} u)$ has been defined for all $i = 1,\ldots,n$, all $a \in A$ and all $u \in T_\Sigma 0$; then, for any $c \in A$ and $t \in T_\Sigma 0$, define $\rho(s \xrightarrow{c} t)$ as below.

Let $\Lambda_c \subseteq \Lambda$ be the set of all those rules with $\mathtt{f}$ and $c$ in the conclusion that are triggered by the tuple of apparent rates $v_{ai} = \rho_a(s_i)$ – cf. (3). Note that $\Lambda_c$ is *finite*. To calculate the value of $\rho(s \xrightarrow{c} t)$, look at each rule $L \in \Lambda_c$ in turn and check whether there exists a substitution $\sigma : \Xi \to T_\Sigma 0$ such that $\sigma\mathtt{t} = t$ and $\sigma\mathtt{x}_i = s_i$ for $i = 1,\ldots,n$. Note that although many such $\sigma$ may exist, their values on each $\mathtt{y}_j$ coincide, since all $\mathtt{y}_j$ appear in $\mathtt{t}$. If $\sigma$ exists, calculate the *contribution* $\gamma_L \in \mathbb{R}^+_0$ of $L$ to $\rho(s \xrightarrow{c} t)$ according to the formula:

$$\gamma_L = W \cdot \prod_{j=1}^{k} \frac{\rho(s_{i_j} \xrightarrow{b_j} \sigma\mathtt{y}_j)}{\rho_{b_j}(s_{i_j})}$$

where $W$, $k$, $i_j$, $b_j$ and $\mathtt{y}_j$ are determined by the shape of rule $L$ – cf. rule format (10). Note that the quotient is well defined since $\rho_{b_j}(s_{i_j}) = r_{b_j i_j} > 0$. If *no* suitable $\sigma$ exist, take $\gamma_L = 0$. Finally, define $\rho(s \xrightarrow{c} t) = \sum_{L \in \Lambda_c} \gamma_L$; the sum exists since $\Lambda_c$ is finite.

**Notation 1.** If in a rule as in (10), for some $\mathtt{x}_i$ and $a \in D_i$ there is exactly one $j$ for which $i_j = i$ and $b_j = a$, instead of the two premises $\mathtt{x}_i \xrightarrow{a@r_{ai}}$ and $\mathtt{x}_i \xrightarrow{a} \mathtt{y}_j$ we shall write simply $\mathtt{x}_i \xrightarrow{a@r_{ai}} \mathtt{y}_j$. Note that, unlike in the frameworks recalled in §2.2, such a premise does *not* require that a transition $\mathtt{x}_i \xrightarrow{a} \mathtt{y}_j$ has rate $r_{ai}$; instead, $r_{ai}$ refers to the *apparent* rate $\rho_a(\mathtt{x}_i)$. To avoid this confusion, @ is used in (10) instead of a comma.

It turns out that SGSOS specifications correspond to distributive laws (8) for the behaviour functor used for modeling stochastic systems:

**Theorem 2.** For all signatures $\Sigma$ and label sets $A$, every SGSOS specification $\Lambda$ for $\Sigma$ and $A$ determines a distributive law $\lambda : \Sigma(\mathrm{Id} \times (\mathcal{R}_\omega)^A) \Longrightarrow (\mathcal{R}_\omega T_\Sigma)^A$ such that $h_\lambda : T_\Sigma 0 \to (\mathcal{R}_\omega T_\Sigma 0)^A$ defined as in (9) coincides with the RTS induced by $\Lambda$. Moreover, every such distributive law $\Lambda$ is defined by an SGSOS specification.

**Corollary 1.** Stochastic bisimilarity on RTSs induced by SGSOS specifications is always a congruence. (*Proof:* Combine Theorems 2 and 1 with Propositions 1 and 2.)

Although technically more involved, the correspondence between SGSOS and RTSs is a perfect match for that for GSOS and LTSs, and lifts the benefits of congruence formats to the equally more involved semantics of stochastic models. In §5 below we shall illustrate that the format affords expressiveness, conciseness and elegance.

## 5 Examples of SGSOS

To illustrate the form of SGSOS specifications, we now present a few simple examples, including operators present in stochastic $\pi$-calculus or in PEPA, as well as some other operators of potential interest.

**Example 1 (atomic actions).** A basic ingredient of most process calculi is prefixing composition with atomic actions. To model stochastic systems, these actions are equipped with basic rates. For the simplest nontrivial example of SGSOS, fix a set $A$ of labels and consider a language with syntax defined by the grammar:

$$P ::= \mathtt{nil} \mid (a, r).P$$

where $a$ ranges over $A$ and $r$ over $\mathbb{R}^+$. The semantics of $\mathtt{nil}$ is defined by the empty set of rules, and the semantics of a unary operator $(a, r).\_$ is defined by a single rule:

$$\frac{}{(a, r).\mathtt{x} \xrightarrow{a@r} \mathtt{x}}$$

Thus, according to Definition 2, the process $P = (a, 2).(b, 3).\mathtt{nil}$ can make a unique transition $P \xrightarrow{a,2} (b, 3).\mathtt{nil}$ in the transition system induced by the rules.

**Example 2 (stochastic choice).** Consider an extension of the language from Example 1 with a binary operator $P + P$ with semantics defined by rules:

$$\frac{\mathtt{x}_1 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1+\mathtt{x}_2 \xrightarrow{a@r} \mathtt{y}} \qquad \frac{\mathtt{x}_2 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1+\mathtt{x}_2 \xrightarrow{a@r} \mathtt{y}}$$

(note the use of Notation 1), for each $a \in A$ and $r \in \mathbb{R}^+$. Note that this is a well-defined SGSOS specification. Although it contains uncountably many rules, for every $a \in A$ exactly two rules are triggered by every tuple of apparent rates. The rules define + to be the stochastic choice operator, as present e.g. in PEPA and stochastic $\pi$-calculus. In particular, according to Definition 2, in the stochastic transition system induced by the rules, the process $P = ((a,2).\mathtt{nil} + (a,2).(b,1).\mathtt{nil}) + (c,3).\mathtt{nil}$ can make three transitions $P \xrightarrow{a,2} \mathtt{nil}$, $P \xrightarrow{a,2} (b,1).\mathtt{nil}$ and $P \xrightarrow{c,3} \mathtt{nil}$. Note, however, that the process $Q = (a,2).\mathtt{nil} + (a,3).\mathtt{nil}$ can only make one transition $Q \xrightarrow{a,5} \mathtt{nil}$. In particular, processes $(a,2).\mathtt{nil}+(a,3).\mathtt{nil}$ and $(a,5).\mathtt{nil}$ are not only stochastic bisimilar, but can actually make exactly the same outgoing transitions.

We remark that when compared to the existing literature, in all our examples the expected semantics of the operators arises naturally from intuitive and elementary specifications, witness of the flexibility of the SGSOS format.

**Example 3 (PEPA-style synchronisation).** Extend the language from Example 2 with a binary synchronisation operator $\underset{L}{\bowtie}$ for each $L \subseteq A$, with semantics defined by a family of rules:

$$\frac{\mathtt{x}_1 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{a@r} \mathtt{y} \underset{L}{\bowtie} \mathtt{x}_2} \qquad \frac{\mathtt{x}_2 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{a@r} \mathtt{x}_1 \underset{L}{\bowtie} \mathtt{y}} \tag{11}$$

$$\frac{\mathtt{x}_1 \xrightarrow{b@r_1} \mathtt{y}_1 \quad \mathtt{x}_2 \xrightarrow{b@r_2} \mathtt{y}_2}{\mathtt{x}_1 \underset{L}{\bowtie} \mathtt{x}_2 \xrightarrow{b@W} \mathtt{y}_1 \underset{L}{\bowtie} \mathtt{y}_2} \tag{12}$$

for each $a \in A \setminus L$, $b \in L$, and $r, r_1, r_2, W \in \mathbb{R}^+$ such that $W = \min(r_1, r_2)$. It is not difficult to see that this, according to Definition 2, is the synchronisation operator of PEPA where the minimal rate law (5) is used. As an example, consider processes:

$$P = (a,1).P_1 + (a,3).P_2 \qquad Q = (a,2).Q_1 \tag{13}$$

where $P_1 \neq P_2$. Then the process $P \underset{\{b\}}{\bowtie} Q$, where $b \neq a$, can make the transitions:

$$P \underset{\{b\}}{\bowtie} Q \xrightarrow{a,1} P_1 \underset{\{b\}}{\bowtie} Q \qquad P \underset{\{b\}}{\bowtie} Q \xrightarrow{a,3} P_2 \underset{\{b\}}{\bowtie} Q \qquad P \underset{\{b\}}{\bowtie} Q \xrightarrow{a,2} P \underset{\{b\}}{\bowtie} Q_1.$$

On the other hand, the outgoing transitions from $P \underset{\{a\}}{\bowtie} Q$ are:

$$P \underset{\{a\}}{\bowtie} Q \xrightarrow{a,\frac{1}{2}} P_1 \underset{\{a\}}{\bowtie} Q_1 \qquad P \underset{\{a\}}{\bowtie} Q \xrightarrow{a,\frac{3}{2}} P_2 \underset{\{a\}}{\bowtie} Q_1.$$

**Example 4 (CCS-style communication).** Similarly, one can extend the language from Example 2 with a CCS-style communication operator. Assume $A = A_0 \cup \{\bar{a} \mid a \in$

$A_0\} \cup \{\tau\}$ (denote $\bar{\bar{a}} = a$) and extend the language with a single binary operator $\|$, with semantics defined by rules:

$$\frac{\mathtt{x}_1 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1 \parallel \mathtt{x}_2 \xrightarrow{a@r} \mathtt{y} \parallel \mathtt{x}_2} \qquad \frac{\mathtt{x}_2 \xrightarrow{a@r} \mathtt{y}}{\mathtt{x}_1 \parallel \mathtt{x}_2 \xrightarrow{a@r} \mathtt{x}_1 \parallel \mathtt{y}} \tag{14}$$

$$\frac{\mathtt{x}_1 \xrightarrow{a@r_1} \mathtt{y}_1 \quad \mathtt{x}_2 \xrightarrow{\bar{a}@r_2} \mathtt{y}_2}{\mathtt{x}_1 \parallel \mathtt{x}_2 \xrightarrow{\tau@W} \mathtt{y}_1 \parallel \mathtt{y}_2} \tag{15}$$

for each $a \in A$ and for each $r, r_1, r_2, W \in \mathbb{R}^+$ such that $W = \min(r_1, r_2)$. This, according to Definition 2, is the communication operator of the original definition of stochastic $\pi$-calculus [20], with the minimal rate law (5) used. For example, consider processes $P$, $Q$ as in (13). The process $P \parallel Q$ can make the following transitions:

$$P \parallel Q \xrightarrow{a,1} P_1 \parallel Q \qquad P \parallel Q \xrightarrow{a,3} P_2 \parallel Q \qquad P \parallel Q \xrightarrow{\bar{a},2} P \parallel Q_1$$
$$P \parallel Q \xrightarrow{\tau,\frac{1}{2}} P_1 \parallel Q_1 \qquad P \parallel Q \xrightarrow{\tau,\frac{3}{2}} P_2 \parallel Q_1.$$

Alternatively, one could use the same rules with $W = r_1 \cdot r_2$. This would correspond to the semantics of parallel composition in the biological stochastic $\pi$-calculus [22], with the mass action law (6) used. For example, the process $P \parallel Q$ above can then make the following transitions:

$$P \parallel Q \xrightarrow{a,1} P_1 \parallel Q \qquad P \parallel Q \xrightarrow{a,3} P_2 \parallel Q \qquad P \parallel Q \xrightarrow{\bar{a},2} P \parallel Q_1$$
$$P \parallel Q \xrightarrow{\tau,2} P_1 \parallel Q_1 \qquad P \parallel Q \xrightarrow{\tau,6} P_2 \parallel Q_1.$$

**Example 5.** Several non-standard, yet meaningful stochastic operators can be defined within the SGSOS format. For example, consider unary "*catalyst*" and "*inhibitor*" operators $\mathtt{cat}_a$ and $\mathtt{inh}_a$ for each $a \in A$, which influence rates of process transitions; they can be seen as stochastic counterparts of the restriction operator of CCS. Their semantics is defined by the rules:

$$\frac{\mathtt{x} \xrightarrow{a@r} \mathtt{y}}{\mathtt{cat}_a(\mathtt{x}) \xrightarrow{a@2r} \mathtt{cat}_a(\mathtt{y})} \qquad \frac{\mathtt{x} \xrightarrow{a@r} \mathtt{y}}{\mathtt{inh}_a(\mathtt{x}) \xrightarrow{a@r/2} \mathtt{inh}_a(\mathtt{y})}$$

$$\frac{\mathtt{x} \xrightarrow{b@r} \mathtt{y}}{\mathtt{cat}_a(\mathtt{x}) \xrightarrow{b@r} \mathtt{cat}_a(\mathtt{y})} \qquad \frac{\mathtt{x} \xrightarrow{b@r} \mathtt{y}}{\mathtt{inh}_a(\mathtt{x}) \xrightarrow{b@r} \mathtt{inh}_a(\mathtt{y})}$$

for each $r \in \mathbb{R}^+$ and $a, b \in A$ such that $b \neq a$. For example, in the derived stochastic transition system we find the transition $\mathtt{cat}_a((a,2).\mathtt{nil}) \xrightarrow{a,4} \mathtt{cat}_a(\mathtt{nil})$. Since the above rules conform to the SGSOS format, it is immediate that operators $\mathtt{cat}_a$ and $\mathtt{inh}_a$ preserve stochastic bisimilarity.

Another example is a binary operator !! of "unfair race parallel composition," which only allows transitions from processes with higher apparent rates than their competitors. Formally, its semantics is defined by rules

$$\frac{\mathtt{x}_1 \xrightarrow{a@r_1} \mathtt{y} \quad \mathtt{x}_2 \xrightarrow{a@r_2}}{\mathtt{x}_1 !! \mathtt{x}_2 \xrightarrow{a@r_1} \mathtt{y} !! \mathtt{x}_2} \qquad \frac{\mathtt{x}_1 \xrightarrow{a@r_2} \quad \mathtt{x}_2 \xrightarrow{a@r_1} \mathtt{y}}{\mathtt{x}_1 !! \mathtt{x}_2 \xrightarrow{a@r_1} \mathtt{x}_1 !! \mathtt{y}}$$

for each $a \in A$ and $r_1, r_2 \in \mathbb{R}_0^+$ such that $r_1 > r_2$. For example, the process $P = ((a,2).Q) \,!!\, ((a,3).T)$ has only one outgoing transition $P \xrightarrow{a,3} ((a,2).Q) \,!!\, T$. Again, stochastic bisimilarity is immediately compositional with respect to !!. This example illustrates the fact that in the semantics of SGSOS operators, apparent rates (3) of subprocesses can be tested, compared and used in an arbitrary fashion. This is in contrast with formats defined for probabilistic systems [2,16], where probabilities of transitions can be used in a very restricted manner. Note however that in SGSOS rates of *single* transitions of subprocesses cannot be used entirely freely. For example, it is not possible to write SGSOS semantics of a hypothetical unary operator $\texttt{even}(\_)$ that would propagate transitions with even rates and suppress those with odd rates. Indeed, such an operator would not preserve stochastic bisimilarity. However, one can define an SGSOS operator that propagates only transitions with labels whose apparent rates are even.

## 6 Associative Parallel Composition for Stochastic Systems

In this section we address an issue in the original design of the stochastic $\pi$-calculus [20], which to our knowledge has not yet been addressed in the literature. Namely, if the minimal rate law (5) is used in the definition (7), then the CCS-style communication operator $\|$ is not associative up to stochastic bisimilarity. Indeed, consider processes

$$\begin{array}{ll} P_1 = (a,r).\texttt{nil} & P_2 = (\bar{a},r).\texttt{nil} \\ Q_1 = (P_1 \parallel P_1) \parallel P_2 & Q_2 = P_1 \parallel (P_1 \parallel P_2). \end{array}$$

Note that $r_a(P_1) = r$, $r_a(P_1 \parallel P_1) = 2r$, and $r_{\bar{a}}(P_2) = r_{\bar{a}}(P_1 \parallel P_2) = r$. This means that, in the derived proved-transitions

$$\begin{array}{l} Q_1 \xrightarrow{\langle \|_1(a,r),(\bar{a},r)\rangle, R_1} (\texttt{nil} \parallel P_1) \parallel \texttt{nil} \\ Q_2 \xrightarrow{\langle (a,r),\|_2(\bar{a},r)\rangle, R_2} \texttt{nil} \parallel (P_1 \parallel \texttt{nil}), \end{array}$$

one has $R_1 = \min(2r, r) \cdot \frac{r}{2r} \cdot \frac{r}{r} = \frac{r}{2}$ and $R_2 = \min(r, r) \cdot \frac{r}{r} \cdot \frac{r}{r} = r$, hence in the resulting RTS, processes $Q_1$ and $Q_2$ do the corresponding $\tau$-transitions respectively with rates $r/2$ and $r$. As a result, they are not stochastic bisimilar. On the other hand, the same operator $\|$ with the rate calculation formula changed to the law of mass action (6), as in [22], is associative. Moreover, CSP-style synchronisation as used in PEPA is associative for both minimal rate and mass action laws.

In the following, we consider parallel composition within the framework of SGSOS and characterise those rate formulas for which CCS-style communication and CSP-style synchronisation operators are associative up to stochastic bisimilarity. It turns out that the CSP-style composition gives much more freedom in the choice of rate formula.

### 6.1 CCS-Style Communication

Consider the language of Example 4, extending those of Examples 1 and 2. Two versions of the language were mentioned there, depending on the choice of the family of rules of type (15) used in the semantics: one where $W = \min(r_1, r_2)$ (the *minimal rate* law) and one where $W = r_1 \cdot r_2$ (the *mass action* law). We will now characterise those

"laws" that give rise to an associative operator ||. More formally, we assume that for each pair $r_1, r_2 \in \mathbb{R}_0^+$ there is exactly one rule of the type (15) in our semantics for each label $a$, and that, moreover, the number $W$ in the conclusion of the rules does not depend on $a$; we can thus treat the $W$'s as a function $W : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$. We then look for those rate functions $W$ for which the operator || is associative up to stochastic bisimilarity.

As the following theorem shows, the choice of $W$ is severely limited: the mass action law is essentially the only choice that makes || associative.

**Theorem 3.** In the situation described above, || is associative up to stochastic bisimilarity if and only if $W(r_1, r_2) = c \cdot r_1 \cdot r_2$ for some constant $c \in \mathbb{R}^+$.

### 6.2 CSP-Style Synchronisation

Consider now the language of Example 3, extending those of Examples 1 and 2. Again, assume that for each pair $r_1, r_2 \in \mathbb{R}_0^+$ there is exactly one rule of the type (12) for each label $a$, and that the number $W$ in the conclusion of the rules does not depend on $a$; thus, as before, we have a function $W : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$. It turns out that, compared to §6.1, one has considerably more freedom in choosing $W$ so that each of the synchronisation operators $\underset{L}{\bowtie}$ is associative:

**Theorem 4.** In the situation described above, each $\underset{L}{\bowtie}$ is associative up to stochastic bisimilarity if and only if $W$ is associative, i.e., $W(r_1, W(r_2, r_3)) = W(W(r_1, r_2), r_3)$ for all $r_1, r_2, r_3 \in \mathbb{R}^+$.

## 7 Conclusions and Future Work

We have defined *SGSOS*, a congruence format for structural operational descriptions of discrete space, continuous time Markov chains. Stochastic bisimilarity is guaranteed to be compositional on languages defined by SGSOS rules. Standard operators of Markovian process algebras, such as prefixing, choice and various forms of synchronization, as well as plenty of non-standard, yet potentially useful operators, are definable in SGSOS. The format arises naturally from the abstract theory of well-behaved operational semantics, based on bialgebras and distributive law.

SGSOS is similar to formats for reactive probabilistic systems developed in [2,16]. Apart from syntactic sugar, the most important difference is the treatment of apparent rates, absent in the probabilistic setting. Rates of single transitions (and their conditional probabilities) are treated in SGSOS just as probabilities of transitions are in [2,16]. In [16], additional complication is necessary to cater for *generative* probabilistic systems. The notions of reactive and generative RTSs coincide, and the additional complexity is not needed in SGSOS.

This is only an initial study of a theory of well-behaved stochastic operational semantics, and several research directions are left open. Look-ahead premises are not allowed in SGSOS, unlike in the probabilistic formats of [16]. Recursive definitions or the name-binding features of stochastic $\pi$-calculus are not currently supported; to treat the latter correctly, one should combine SGSOS with techniques from [10]. Also, non-Markovian processes are not treated here, as a coalgebraic treatment of them is missing. Process algebra for continuous-space Markov chains [9] is another possible direction.

## References


1. Aceto, L., Fokkink, W.J., Verhoef, C.: Structural operational semantics. In: Bergstra, J.A., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 197–292. Elsevier, Amsterdam (2002)
2. Bartels, F.: On Generalised Coinduction and Probabilistic Specification Formats. PhD dissertation, CWI, Amsterdam (2004)
3. Bernardo, M., Gorrieri, R.: A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Theor. Comp. Sci. 202(1–2), 1–54 (1998)
4. Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. Journal of the ACM 42, 232–268 (1995)
5. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the ACM 31, 560–599 (1995)
6. Calder, M., Gilmore, S., Hillston, J.: Automatically deriving ODEs from process algebra models of signalling pathways. In: Procs. CMSB 2005, pp. 204–215 (2005)
7. de Vink, E.P., Rutten, J.J.M.M.: Bisimulation for probabilistic transition systems: A coalgebraic approach. Theoretical Computer Science 221(1–2), 271–293 (1999)
8. Degano, P., Priami, C.: Enhanced operational semantics. ACM Comput. Surv. 28(2), 352–354 (1996)
9. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labelled Markov processes. Information and Computation 179, 163–193 (2002)
10. Fiore, M., Staton, S.: A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In: Proc. LICS 2006, pp. 49–58. IEEE Computer Society Press, Los Alamitos (2006)
11. Götz, N., Herzog, U., Rettelbach, M.: Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In: Performance/SIGMETRICS Tutorials, pp. 121–146 (1993)
12. Hermanns, H., Herzog, U., Katoen, J.-P.: Process algebra for performance evaluation. Theoretical Computer Science 274(1–2), 43–87 (2002)
13. Hillston, J.: On the nature of synchronisation. In: Procs. PAPM 1994, pp. 51–70 (1994)
14. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
15. Hillston, J.: Process algebras for quantitative analysis. In: Procs. LiCS 2005, pp. 239–248. IEEE Computer Society Press, Los Alamitos (2005)
16. Lanotte, R., Tini, S.: Probabilistic bisimulation as a congruence. ACM Trans. Comp. Logic (to appear, 2008)
17. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94, 1–28 (1991)
18. Milner, R.: A calculus of communicating systems. Journal of the ACM (1980)
19. Plotkin, G.D.: A structural approach to operational semantics. DAIMI Report FN-19, Computer Science Department, Aarhus University (1981)
20. Priami, C.: Stochastic $\pi$-calculus. Computer Journal 38(7), 578–589 (1995)
21. Priami, C.: Language-based performance prediction for distributed and mobile systems. Information and Computation 175(2), 119–145 (2002)
22. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the $\pi$-calculus process algebra. In: Proc. Pacific Symp. Biocomp. (2001)
23. Rutten, J.J.M.M.: Universal coalgebra: A theory of systems. Theoretical Computer Science 249, 3–80 (2000)
24. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: Proc. LICS 1997, pp. 280–291. IEEE Computer Society Press, Los Alamitos (1997)

# Compositional Methods for Information-Hiding*

Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi

INRIA and LIX, École Polytechnique, Palaiseau, France
{braun,kostas,catuscia}@lix.polytechnique.fr

**Abstract.** Protocols for information-hiding often use randomized primitives to obfuscate the link between the observables and the information to be protected. The degree of protection provided by a protocol can be expressed in terms of the probability of error associated to the inference of the secret information.

We consider a probabilistic process calculus approach to the specification of such protocols, and we study how the operators affect the probability of error. In particular, we characterize constructs that have the property of not decreasing the degree of protection, and that can therefore be considered safe in the modular construction of protocols.

As a case study, we apply these techniques to the Dining Cryptographers, and we are able to derive a generalization of Chaum's strong anonymity result.

## 1 Introduction

During the last decade, internet activities have become an important part of many people's lives. As the number of these activities increases, there is a growing amount of personal information about the users that is stored in electronic form and that is usually transferred using public electronic means. This makes it feasible and often easy to collect, transfer and process a huge amount of information about a person. As a consequence, the need for mechanisms to protect the user's privacy is compelling.

We can categorize privacy properties based on the nature of the hidden information. *Data protection* usually refers to confidential data like the credit card number. *Anonymity*, on the other hand, concerns the identity of the user who performed a certain action. *Unlinkability* refers to the link between the information and the user, and *unobservability* regards the actions of a user.

Information-hiding protocols aim at ensuring a privacy property during an electronic transaction. For example, the voting protocol Foo 92 ([1]) allows a user to cast a vote without revealing the link between the voter and the vote. The anonymity protocol Crowds ([2]) allows a user to send a message on a public network without revealing the identity of the sender. These kinds of protocols often use *randomization* to introduce *noise*, thus limiting the inference power of a malicious observer.

### 1.1 Information Theory

Recently it has been observed that at an abstract level information-hiding protocols can be viewed as *channels* in the information-theoretic sense. A channel consists of a set of

* This work has been partially supported by the INRIA DREI Équipe Associée PRINTEMPS and by the INRIA ARC project ProNoBiS.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 443–457, 2008.

input values $\mathcal{S}$, a set of output values $\mathcal{O}$ (the observables) and a transition matrix which gives the conditional probability $p(o|s)$ of producing $o$ as the output when $s$ is the input. In the case of privacy preserving protocols, $\mathcal{S}$ contains the secret information that we want to protect and $\mathcal{O}$ the facts that the attacker can observe. This framework allows us to apply concepts from information theory to reason about the knowledge that the attacker can gain about the input by observing the output of the protocol.

In the field of information flow and non-interference there have been various works [3,4,5,6,7] in which the *high information* and the *low information* are seen as the input and output respectively of a (noisy) channel. Non-interference is formalized in this setting as the converse of channel capacity.

Channel capacity has been also used in relation to anonymity in [8,9]. These works propose a method to create covert communication by means of non-perfect anonymity.

A related line of work is [10,11], where the main idea is to express the lack of (probabilistic) information in terms of entropy.

A different information-theoretic approach is taken in [12]. In this paper, the authors define as information leakage the difference between the a priori accuracy of the guess of the attacker, and the a posteriori one, after the attacker has made his observation. The accuracy of the guess is defined as the Kullback-Leibler distance between the *belief* (which is a weight attributed by the attacker to each input hypothesis) and the true distribution on the hypotheses.

### 1.2 Hypothesis Testing

In information-hiding systems the attacker finds himself in the following scenario: he cannot directly detect the information of interest, namely the actual value of the random variable $S \in \mathcal{S}$, but he can discover the value of another random variable $O \in \mathcal{O}$ which depends on $S$ according to a known conditional distribution. The attempt to infer $S$ from $O$ is called *hypothesis testing* (the "hypothesis" to be validated is the actual value of $S$), and it has been widely investigated in statistics. One of the most used approaches to this problem is the Bayesian method, which consists in assuming known the a priori probability distribution of the hypotheses, and deriving from that (and from the matrix of the conditional probabilities) the a posteriori distribution after a certain fact has been observed. It is well known that the best strategy for the adversary is to apply the MAP (Maximum Aposteriori Probability) criterion, which, as the name says, dictates that one should choose the hypothesis with the maximum a posteriori probability for the given observation. "Best" means that this strategy induces the smallest probability of error in the guess of the hypothesis. The probability of error, in this case, is also called *Bayes risk*. In [13], we proposed to define the *degree of protection* provided by a protocol as the Bayes risk associated to the matrix.

A major problem with the Bayesian method is that the a priori distribution is not always known. This is particularly true in security applications. In some cases, it may be possible to approximate the a priori distribution by statistical inference, but in most cases, especially when the input information changes over time, it may not. Thus other methods need to be considered, which do not depend on the a priori distribution. One such method is the one based on the so-called *Maximum Likelihood* criterion.

### 1.3 Contribution

In this paper we consider both the scenario in which the input distribution is known, in which case we consider the Bayes risk, and the one in which we have no information on the input distribution, or it changes over time. In this second scenario, we consider as degree of protection the probability of error associated to the Maximum Likelihood rule, averaged on all possible input distributions. It turns out that such average is equal to the value of the probability of error on the point of uniform distribution, which is much easier to compute.

Next, we consider a probabilistic process algebra for the specification of information-hiding protocols, and we investigate which constructs in the language can be used safely in the sense that by applying them to a term, the degree of protection provided by the term does not decrease. This provides a criterion to build specifications in a compositional way, while preserving the degree of protection.

We apply these compositional methods to the example of the Dining Cryptographers, and we are able to strengthen the strong anonymity result by Chaum. Namely we show that we can have strong anonymity even if some coins are unfair, provided that there is a spanning tree of fair ones. This result is obtained by adding processes representing coins to the specification and using the fact that this can be done with a safe construct.

The proofs are omitted for lack of space. They can be found in the report version of this paper, available on line at `http://www.lix.polytechnique.fr/~catuscia/papers/Anonymity/Compositional/report.pdf`.

### 1.4 Plan of the Paper

In the next section we recall some basic notions. Section 3 introduces the language $\text{CCS}_p$. Section 4 shows how to model protocols and process terms as channels. Section 5 discusses hypothesis testing and presents some properties of the probability of error. Section 6 characterizes the constructs of $\text{CCS}_p$ which are safe. Section 7 applies previous results to find a new property of the Dining Cryptographers. Section 8 concludes.

## 2 Preliminaries

In this section we recall some basic notions of probability theory and probabilistic automata ([14,15]).

A *discrete probability measure* over a set $X$ is a function $\mu : 2^X \mapsto [0,1]$ such that $\mu(X) = 1$ and $\mu(\cup_i X_i) = \sum_i \mu(X_i)$ where $X_i$ is a countable family of pairwise disjoint subsets of $X$. We denote the set of all discrete probability measures over $X$ by $Disc(X)$. For $x \in X$, we denote by $\delta(x)$ (the *Dirac measure* on $x$) the probability measure that assigns probability 1 to $\{x\}$. If $\{c_i\}_i$ are convex coefficients, and $\{\mu_i\}_i$ are probability measures, we will denote by $\sum_i c_i\mu_i$ the probability measure defined as $(\sum_i c_i\mu_i)(Y) = \sum_i c_i\mu_i(Y)$.

A *probabilistic automaton* $\mathcal{M}$ is a tuple $(St, T_{init}, Act, \mathcal{T})$ where $St$ is a set of states, $T_{init} \in St$ is the *initial state*, $Act$ is a set of actions and $\mathcal{T} \subseteq St \times Act \times Disc(St)$ is a *transition relation*. Intuitively, if $(T, a, \mu) \in \mathcal{T}$ then there is a transition from the state $T$

performing the action $a$ and leading to a distribution $\mu$ over the states of the automaton. (We use $T$ for states because later in the paper states will be process terms, and $S$ will be used for certain sequences of actions). We also write $T \xrightarrow{a} \mu$ if $(T, a, \mu) \in \mathcal{T}$. The idea is that the choice of transition among the available ones in $\mathcal{T}$ is performed non-deterministically, and the choice of the target state among the ones allowed by $\mu$ (i.e. those states $T'$ such that $\mu(T') > 0$) is performed probabilistically. A probabilistic automaton $\mathcal{M}$ is *fully probabilistic* if from each state of $\mathcal{M}$ there is at most one transition available.

An *execution fragment* $\phi$ of a probabilistic automaton is a (possibly infinite) sequence $T_0 a_1 T_1 a_2 T_2 \ldots$ of alternating states and actions, such that for each $i$ there is a transition $(T_i, a_{i+1}, \mu_i) \in \mathcal{T}$ and $\mu_i(T_{i+1}) > 0$. We will use $fst(\phi), lst(\phi)$ to denote the first and last state of a finite execution fragment $\phi$ respectively. An *execution* (or *history*) is an execution fragment such that $fst(\phi) = T_{init}$. An execution $\phi$ is maximal if it is infinite or there is no transition from $lst(\phi)$ in $\mathcal{T}$. We denote by $exec^*(\mathcal{M})$ the set of all the finite non-maximal executions of $\mathcal{M}$, and by $exec(\mathcal{M})$ the set of all the executions of $\mathcal{M}$.

A *scheduler* of a probabilistic automaton $\mathcal{M} = (St, T_{init}, Act, \mathcal{T})$ is a function

$$\zeta : exec^*(\mathcal{M}) \to \mathcal{T}$$

such that $\zeta(\phi) = (T, a, \mu) \in \mathcal{T}$ implies that $T = lst(\phi)$. The idea is that a scheduler selects a transition among the ones available in $\mathcal{T}$ and it can base its decision on the history of the execution. The *execution tree* of $\mathcal{M}$ relative to the scheduler $\zeta$, denoted by $etree(\mathcal{M}, \zeta)$, is a fully probabilistic automaton $\mathcal{M}' = (St', T_{init}, Act, \mathcal{T}')$ such that $St' \subseteq exec(\mathcal{M})$, and $(\phi, a, \mu') \in \mathcal{T}'$ if and only if $\zeta(\phi) = (lst(\phi), a, \mu)$ for some $\mu$, and $\mu'(\phi a T) = \mu(T)$. Intuitively, $etree(M, \zeta)$ is produced by unfolding the executions of $\mathcal{M}$ and resolving all nondeterministic choices using $\zeta$. Note that $etree(\mathcal{M}, \zeta)$ is a fully probabilistic automaton.

Given a fully probabilistic automaton $\mathcal{M}$ we can define a probability space on the set $exec(\mathcal{M})$ of executions of $\mathcal{M}$ (see [14] for more details). Similarly, given a probabilistic automaton $\mathcal{M}$ and a scheduler $\zeta$ for $\mathcal{M}$, we can define a probability space on the set of traces of $\mathcal{M}$ by using the same construction on $etree(\mathcal{M}, \zeta)$, which is a fully probabilistic automaton.

## 3 CCS with Internal Probabilistic Choice

We consider an extension of standard CCS ([16]) obtained by adding internal probabilistic choice. The resulting calculus $CCS_p$ can be seen as a simplified version of the probabilistic $\pi$-calculus presented in [17,18] and it is similar to the one considered in [19]. Like in those calculi, computations have both a probabilistic and a nondeterministic nature. The main conceptual novelty is a distinction between *observable* and *secret* actions, introduced for the purpose of specifying information-hiding protocols.

We assume a countable set $Act$ of actions $a$, and we assume that it is partitioned into a set $Sec$ of *secret actions* $s$, a set $Obs$ of *observable actions* $o$, and the silent action $\tau$. For each $s \in Sec$ we assume a complementary action $\overline{s} \in Sec$ such that $\overline{\overline{s}} = s$, and

**Table 1.** The semantics of $\mathrm{CCS}_p$

$$\text{PROB}\ \frac{}{\sum_i p_i T_i \xrightarrow{\tau} \sum_i p_i \delta(T_i)} \qquad \text{ACT}\ \frac{j \in I}{\lfloor\!+\!\rfloor_I a_i.T_i \xrightarrow{a_j} \delta(T_j)}$$

$$\text{PAR1}\ \frac{T_1 \xrightarrow{a} \mu}{T_1 \mid T_2 \xrightarrow{a} \mu \mid T_2} \qquad \text{PAR2}\ \frac{T_2 \xrightarrow{a} \mu}{T_1 \mid T_2 \xrightarrow{a} T_1 \mid \mu} \qquad \text{REP}\ \frac{T \mid !T \xrightarrow{a} \mu}{!T \xrightarrow{a} \mu \mid !T}$$

$$\text{COM}\ \frac{T_1 \xrightarrow{a} \delta(T_1') \quad T_2 \xrightarrow{\overline{a}} \delta(T_2')}{T_1 \mid T_2 \xrightarrow{\tau} \delta(T_1' \mid T_2')} \qquad \text{RES}\ \frac{T \xrightarrow{b} \mu \qquad \phi \neq a, \overline{a}}{(\nu a)T \xrightarrow{b} (\nu a)\mu}$$

the same for $Obs$. The silent action $\tau$ does not have a complementary action, so the notation $\overline{a}$ will imply that $a \in Sec$ or $a \in Obs$.

The syntax of $\mathrm{CCS}_p$ is the following:

| | | |
|---|---|---|
| $T$ | ::= | *process term* |
| | $\sum_i p_i T_i$ | *probabilistic choice* |
| $\mid$ | $\lfloor\!+\!\rfloor_i s_i.T_i$ | *secret choice* $(s_i \in Sec)$ |
| $\mid$ | $\lfloor\!+\!\rfloor_i r_i.T_i$ | *nondeterministic choice* $(r_i \in Obs \cup \{\tau\})$ |
| $\mid$ | $T \mid T$ | *parallel composition* |
| $\mid$ | $(\nu a)\, T$ | *restriction* |
| $\mid$ | $!T$ | *replication* |

All the summations in the syntax are finite. We will use the notation $T_1 \oplus_p T_2$ to represent a binary probabilistic choice $\sum_i p_i T_i$ with $p_1 = p$ and $p_2 = 1 - p$. Similarly we will use $a_1.T_1 \lfloor\!+\!\rfloor a_2.T_2$ to represent a binary secret or nondeterministic choice.

The semantics of a given $\mathrm{CCS}_p$ term is a probabilistic automaton whose states are process terms, whose initial state is the given term, and whose transitions are those derivable from the rules in Table 1. We will use the notations $(T, a, \mu)$ and $T \xrightarrow{a} \mu$ interchangeably. We denote by $\mu \mid T$ the measure $\mu'$ such that $\mu'(T' \mid T) = \mu(T')$ for all processes $T'$ and $\mu'(T'') = 0$ if $T''$ is not of the form $T' \mid T$, and similarly for $T \mid \mu$. Furthermore we denote by $(\nu a)\mu$ the measure $\mu'$ such that $\mu'((\nu a)T) = \mu(T)$, and $\mu'(T') = 0$ if $T'$ is not of the form $(\nu a)T$.

Note that in the produced probabilistic automaton, all transitions to non-Dirac measures are silent. Note also that a probabilistic term generates exactly one (probabilistic) transition.

A transition of the form $T \xrightarrow{a} \delta(T')$, i.e. a transition having for target a Dirac measure, corresponds to a transition of a non-probabilistic automaton (a standard labeled

transition system). Thus, all the rules of CCS$_p$ specialize to the ones of CCS except from PROB. The latter models the internal probabilistic choice: a silent $\tau$ transition is available from the sum to a measure containing all of its operands, with the corresponding probabilities.

A secret choice $\boxplus_i s_i.T_i$ produces the same transitions as the nondeterministic term $\boxplus_i r_i.T_i$, except for the labels.

The distinction between the two kind of labels influences the notion of scheduler for CCS$_p$: the secret actions are assumed to be *inputs* of the system, so a secret choice (with different guards) is determined by the input. The scheduler has to resolve only the residual nondeterminism.

In the following, we use the notation $X \rightharpoonup Y$ to represent the partial functions from $X$ to $Y$, and $\phi_{|Sec}$ represents the projection of $\phi$ on $Sec$.

**Definition 1.** *Let $T$ be a process in CCS$_p$ and $\mathcal{M}$ be the probabilistic automaton generated by $T$. A scheduler is a function $\zeta : Sec^* \to exec^* \rightharpoonup \mathcal{T}$ such that:*

*(i) if $s = s_1 s_2 \dots s_n$ and $\phi_{|Sec} = s_1 s_2 \dots s_m$ with $m \leq n$, and*
*(ii) there exists a transition $(lst(\phi), a, \mu)$ such that, if $a \in Sec$ then $a = s_{m+1}$*

*then $\zeta(s)(\phi)$ is defined, and it is one of such transitions. We will write $\zeta_s(\phi)$ for $\zeta(s)(\phi)$.*

Note that this definition of scheduler is different from the one used in probabilistic automaton, where the scheduler can decide to stop, even if a transition is allowed. Here the scheduler must proceed whenever a transition is allowed (provided that if it is labeled by a secret, that secret is the next one in the input string $s$).

We now adapt the definition of *execution tree* from the notion found in probabilistic automata. In our case, the execution tree depends not only on the scheduler, but also on the input.

**Definition 2.** *Let $\mathcal{M} = (St, T, Act, \mathcal{T})$ be the probabilistic automaton generated by a CCS$_p$ process $T$, where $St$ is the set of processes reachable from $T$. Given an input $s$ and a scheduler $\zeta$, the* execution tree *of $T$ for $s$ and $\zeta$, denoted by $etree(T, s, \zeta)$, is a fully probabilistic automaton $\mathcal{M}' = (St', T, Act, \mathcal{T}')$ such that $St' \subseteq exec(\mathcal{M})$, and $(\phi, a, \mu') \in \mathcal{T}'$ if and only if $\zeta_s(\phi) = (lst(\phi), a, \mu)$ for some $\mu$, and $\mu'(\phi a T) = \mu(T)$.*

## 4 Modeling Protocols for Information-Hiding

We propose here an abstract model for information-hiding protocols, and we show how to represent this model in CCS$_p$. An extended example is presented in Section 7.

### 4.1 Protocols as Channels

We view protocols as *channels* in the information-theoretic sense [20]. The secret information that the protocol is trying to conceal constitutes the input of the channel, and the observables constitute the outputs. The set of the possible inputs and that of the possible outputs will be denoted by $\mathcal{S}$ and $\mathcal{O}$ respectively. We assume that $\mathcal{S}$ and $\mathcal{O}$ are

of finite cardinality $m$ and $n$ respectively. We also assume a discrete probability distribution over the inputs, which we will denote by $\vec{\pi} = (\pi_{s_1}, \pi_{s_2}, \ldots, \pi_{s_m})$, where $\pi_s$ is the probability of the input $s$.

To fit the model of the channel, we assume that at each run, the protocol is given exactly one secret $s_i$ to conceal. This is not a restriction, because the $s_i$'s can be complex information like sequences of keys or tuples of individual data. During the run, the protocol may use randomized operations to increase the level of uncertainty about the secrets and obfuscate the link with the observables. It may also have internal interactions between internal components, or other forms of nondeterministic behavior, but let us rule out this possibility for the moment, and consider a purely probabilistic protocol. We also assume there is exactly one output from each run of the protocol, and again, this is not a restrictive assumption because the elements of $\mathcal{O}$ can be structured data.

Given an input $s$, a run of the protocol will produce each $o \in \mathcal{O}$ with a certain probability $p(o|s)$ which depends on $s$ and on the randomized operations performed by the protocol. Note that $p(o|s)$ depends only on the probability distributions on the mechanisms of the protocol, and not on the input distribution. The probabilities $p(o|s)$, for $s \in \mathcal{S}$ and $o \in \mathcal{O}$, constitute a $m \times n$ array $M$ which is called the *matrix* of the channel, where the rows are indexed by the elements of $\mathcal{S}$ and the columns are indexed by the elements of $\mathcal{O}$. We will use the notation $(\mathcal{S}, \mathcal{O}, M)$ to represent the channel.

Note that the input distribution $\vec{\pi}$ and the probabilities $p(o|s)$ determine a distribution on the output. We will represent by $p(o)$ the probability of $o \in \mathcal{O}$. Thus both the input and the output can be considered *random variables*. We will denote these random variables by $S$ and $O$.

If the protocol contains some forms of nondeterminism, like internal components giving rise to different interleaving and interactions, then the behavior of the protocol, and in particular the output, will depend on the scheduling policy. We can reduce this case to previous (purely probabilistic) scenario by assuming a scheduler $\zeta$ which resolves the nondeterminism entirely. Of course, the conditional probabilities, and therefore the matrix, will depend on $\zeta$, too. We will express this dependency by using the notation $M_\zeta$.

### 4.2 Process Terms as Channels

A given CCS$_p$ term $T$ can be regarded as a protocol in which the input is constituted by sequences of secret actions, and the output by sequences of observable actions. We assume that only a finite set of such sequences is relevant. This is certainly true if the term is terminating, which is usually the case in security protocols where each session is supposed to terminate in finite time.

Thus the set $S$ could be, for example, the set of all sequences of secret actions up to a certain length (for example, the maximal length of executions) and analogously $O$ could be the set of all sequences of observable actions up to a certain length. To be more general, we will just assume $\mathcal{S} \subseteq_{fin} Sec^*$ and $\mathcal{O} \subseteq_{fin} Obs^*$.

**Definition 3.** *Given a term $T$ and a scheduler $\zeta : \mathcal{S} \to exec^* \to \mathcal{T}$, the matrix $M_\zeta(T)$ associated to $T$ under $\zeta$ is defined as the matrix such that, for each $s \in \mathcal{S}$ and $o \in \mathcal{O}$,*

*$p(o|s)$ is the probability of the set of the maximal executions in $etree(T, s, \zeta)$ whose projection in $Obs$ is $o$.*

## 5 Inferring the Secrets from the Observables

In this section we discuss possible methods by which an adversary can try to infer the secrets from the observables, and consider the corresponding probability of error, that is, the probability that the adversary draws the wrong conclusion. We regard the probability of error as a representative of the degree of protection provided by the protocol, and we study its properties with respect to the associated matrix.

We start by defining the notion of *decision function*, which represents the guess the adversary makes about the secrets, for each observable. This is a well-known concept, particularly in the field of *hypothesis testing*, where the purpose is to try to discover the valid hypothesis from the observed facts, knowing the probabilistic relation between the possible hypotheses and their consequences. In our scenario, the hypotheses are the secrets.

**Definition 4.** *A decision function for a channel $(\mathcal{S}, \mathcal{O}, M)$ is any function $f : \mathcal{O} \to \mathcal{S}$.*

Given a channel $(\mathcal{S}, \mathcal{O}, M)$, an input distribution $\vec{\pi}$, and a decision function $f$ , the *probability of error* $\mathcal{P}(f, M, \vec{\pi})$ is the average probability of guessing the wrong hypothesis by using $f$, weighted on the probability of the observable (see for instance [20]). The probability that, given $o$, $s$ is the wrong hypothesis is $1 - p(s|o)$ (with a slight abuse of notation, we use $p(\cdot|\cdot)$ to represent also the probability of the input given the output). Hence we have:

**Definition 5 (Probability of error, [20]).** $\mathcal{P}(f, M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} p(o)p(f(o)|o).$

Given a channel $(\mathcal{S}, \mathcal{O}, M)$, the best decision function that the adversary can use, namely the one that minimizes the probability of error, is the one associated to the so-called MAP rule, which prescribes choosing the hypothesis $s$ which has *Maximum Aposteriori Probability* (for a given $o \in \mathcal{O}$), namely the $s$ for which $p(s|o)$ is maximum. The fact that the MAP rule represent the 'best bet' of the adversary is rather intuitive, and well known in the literature. We refer to [20] for a formal proof.

The MAP rule is used in the so-called *Bayesian approach* to hypothesis testing, and the corresponding probability of error is also known as *Bayes risk*. We will denote it by $\mathcal{P}_{MAP}(M, \vec{\pi})$. The following characterization is an immediate consequence of Definition 5 and of the Bayes theorem $p(s|o) = p(o|s)\pi_s/p(o)$.

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} \max_{s}(p(o|s)\pi_s)$$

It is natural then to define the degree of protection associated to a process term as the infimum probability of error that we can obtain from this term under every compatible scheduler (in a given class).

In the following, we assume the class of schedulers $\mathcal{A}$ to be the set of all the schedulers compatible with the given input $\mathcal{S}$.

It turns out that the infimum probability of error on $\mathcal{A}$ is actually a minimum:

**Proposition 1.** *For every $CCS_p$ process $T$ we have*

$$\inf_{\zeta \in \mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T), \vec{\pi}) \;=\; \min_{\zeta \in \mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T), \vec{\pi})$$

Thanks to previous proposition, we can define the degree of protection provided by a protocols in terms of the minimum probability of error.

**Definition 6.** *Given a $CCS_p$ process $T$, the protection $Pt_{MAP}(T)$ provided by $T$, in the Bayesian approach, is given by*

$$Pt_{MAP}(T, \vec{\pi}) \;=\; \min_{\zeta \in \mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T), \vec{\pi}).$$

The problem with the MAP rule is that it assumes that the input distribution is known to the adversary. This is often not the case, so it is natural to try to approximate it with some other rule. One such rule is the so-called ML rule, which prescribes the choice of the $s$ which has *Maximum Likelihood* (for a given $o \in \mathcal{O}$), namely the $s$ for which $p(o|s)$ is maximum. The name comes from the fact that $p(o|s)$ is called the *likelihood* of $s$ given $o$. We will denote the corresponding probability of error by $\mathcal{P}_{ML}(M, \vec{\pi})$. The following characterization is an immediate consequence of Definition 5 and of the Bayes Theorem.

$$\mathcal{P}_{ML}(M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} \max_s (p(o|s))\pi_s$$

It has been shown (see for instance [21]) that under certain conditions on the matrix, the ML rule approximates indeed the MAP rule, in the sense that by repeating the protocol the adversary can make the probability of error arbitrarily close to 0, with either rule.

We could now define the degree of protection provided by a term $T$ under the ML rule as the minimum $\mathcal{P}_{ML}(M_\zeta(T), \vec{\pi})$, but it does not seem reasonable to give a definition that depends on the input distribution, since the main reason to apply a non-Bayesian approach is that indeed we do not know the input distribution. Instead, we define the degree of protection associated to a process term as the *average* probability of error with respect to all possible distributions $\vec{\pi}$:

**Definition 7.** *Given a $CCS_p$ process $T$, the protection $Pt_{ML}(T)$ provided by $T$, in the Maximum Likelihood approach, is given by*

$$Pt_{ML}(T) \;=\; \min_{\zeta \in \mathcal{A}} \; (m-1)! \int_{\vec{\pi}} \mathcal{P}_{ML}(M_\zeta(T), \vec{\pi}) \; d\vec{\pi}$$

In the above definition, $(m-1)!$ represents a normalization function: $\frac{1}{(m-1)!}$ is the hyper-volume of the domain of all possible distributions $\vec{\pi}$ on $\mathcal{S}$, namely the $(m-1)$-dimensional space of points $\vec{\pi}$ such that $0 \leq \pi_s \leq 1$ and $0 \leq \sum_{s \in S} \pi_s = 1$ (where $m$ is the cardinality of $\mathcal{S}$).

Fortunately, it turns out that this definition is equivalent to a much simpler one: the average value of the probability of error, under the Maximum Likelihood rule, can be obtained simply by computing $\mathcal{P}_{ML}$ on the uniform distribution $\vec{\pi}_u = (\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m})$.

**Theorem 1**

$$Pt_{ML}(T) \;=\; \min_{\zeta \in \mathcal{A}} \; \mathcal{P}_{ML}(M_\zeta(T), \vec{\pi}_u)$$

The next corollary follows immediately from Theorem 1 and from the definitions of $\mathcal{P}_{MAP}$ and $\mathcal{P}_{ML}$.

**Corollary 1**

$$Pt_{ML}(T) \;=\; \min_{\zeta \in \mathcal{A}} \; \mathcal{P}_{MAP}(M_\zeta(T), \vec{\pi}_u)$$

We conclude this section with some properties of $\mathcal{P}_{MAP}$. Note that the same properties hold also for $\mathcal{P}_{ML}$ on the uniform distribution, because $\mathcal{P}_{ML}(M, \vec{\pi}_u) = \mathcal{P}_{MAP}(M, \vec{\pi}_u)$.

The next proposition shows that the probabilities of error are *concave* functions with respect to the space of matrices.

**Proposition 2.** *Consider a family of channels* $\{(\mathcal{S}, \mathcal{O}, M_i)\}_{i \in I}$*, and a family* $\{c_i\}_{i \in I}$ *of convex coefficients, namely* $0 \leq c_i \leq 1$ *for all* $i \in I$*, and* $\sum_{i \in I} c_i \;=\; 1$*. Then:*

$$\mathcal{P}_{MAP}(\sum_{i \in I} c_i \, M_i, \vec{\pi}) \;\geq\; \sum_{i \in I} c_i \, \mathcal{P}_{MAP}(M_i, \vec{\pi})$$

**Corollary 2.** *Consider a family of channels* $\{(\mathcal{S}, \mathcal{O}, M_i)\}_{i \in I}$*, and a family* $\{c_i\}_{i \in I}$ *of convex coefficients. Then:*

$$\mathcal{P}_{MAP}(\sum_{i \in I} c_i \, M_i, \vec{\pi}) \;\geq\; \min_{i \in I} \; \mathcal{P}_{MAP}(M_i, \vec{\pi})$$

The next proposition shows that if we transform the observables, and collapse the columns corresponding to observables which have become the same after the transformation, the probability of error does not decrease.

**Proposition 3.** *Consider a channel* $(\mathcal{S}, \mathcal{O}, M)$*, where* $M$ *has conditional probabilities* $p(o|s)$*, and a transformation of the observables* $f : \mathcal{O} \to \mathcal{O}'$*. Let* $M'$ *be the matrix whose conditional probabilities are* $p'(o'|s) = \sum_{f(o)=o'} p(o|s)$ *and consider the new channel* $(\mathcal{S}, \mathcal{O}', M')$*. Then:*

$$\mathcal{P}_{MAP}(M', \vec{\pi}) \;\geq\; \mathcal{P}_{MAP}(M, \vec{\pi})$$

The following propositions are from the literature.

**Proposition 4 ([21]).** *Given* $\mathcal{S}$*,* $\mathcal{O}$*, let* $M$ *be a matrix indexed on* $\mathcal{S}$*,* $\mathcal{O}$ *such that all the rows of* $M$ *are equal, namely* $p(o|s) = p(o|s')$ *for all* $o \in \mathcal{O}$*,* $s, s' \in \mathcal{S}$*. Then,*

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \;=\; 1 - \max_{s} \; \pi_s$$

*Furthermore* $\mathcal{P}_{MAP}(M, \vec{\pi})$ *is the maximum probability of error, i.e. for every other matrix* $M'$ *indexed on* $\mathcal{S}$*,* $\mathcal{O}$ *we have:*

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \;\geq\; \mathcal{P}_{MAP}(M', \vec{\pi}).$$

**Proposition 5 ([22]).** *Given a channel $(\mathcal{S}, \mathcal{O}, M)$, the rows of $M$ are equal (and hence the probability of error is maximum) if and only if $p(s|o) = \pi_s$ for all $s \in \mathcal{S}$, $o \in \mathcal{O}$.*

The condition $p(s|o) = \pi_s$ means that the observation does not give any additional information concerning the hypothesis. In other words, the *a posteriori* probability of $s$ coincides with its *a priori* probability. The property $p(s|o) = \pi_s$ for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$ was used as a definition of (strong) anonymity by Chaum [23] and was called *conditional anonymity* by Halpern and O'Neill [24].

## 6 Safe Constructs

In this section we investigate constructs of the language $\text{CCS}_p$ which are *safe* with respect to the protection of the secrets.

We start by giving some conditions that will allow us to ensure the safety of the parallel and the restriction operators.

**Definition 8.** *Consider process terms $T_1, T_2$, and observables $o_1, o_2, \ldots, o_k$ such that*

*(i) $T_1$ does not contain any secret action, and*
*(ii) the observable actions of $T_1$ are included in $o_1, o_2, \ldots, o_k$.*

*Then we say that $T_1$ and $o_1, o_2, \ldots, o_k$ are safe with respect to $T_2$.*

The following theorem states our main results for $Pt_{MAP}$. Note that they are also valid for $Pt_{ML}$, because $Pt_{ML}(T) = Pt_{MAP}(T, \vec{\pi}_u)$.

**Theorem 2.** *The probabilistic choice, the nondeterministic choice, and a restricted form of parallel composition are safe constructs, namely, for every input probability $\pi$, and any terms $T_1, T_2, \ldots, T_h$, we have*

$$(1) \qquad Pt_{MAP}(\sum_i p_i\, T_i, \vec{\pi}) \;\geq\; \sum_i p_i\, Pt_{MAP}(T_i, \vec{\pi}) \;\geq\; \min_i\, Pt_{MAP}(T_i, \vec{\pi})$$

$$(2) \qquad Pt_{MAP}(\boxed{+}_i\, o_i.T_i, \vec{\pi}) \;=\; \min_i Pt_{MAP}(T_i, \vec{\pi})$$

$$(3) \quad Pt_{MAP}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)) \;\geq\; Pt_{MAP}(T_2, \vec{\pi})$$

*if $T_1$ and $o_1, o_2, \ldots, o_k$ are safe w.r.t. $T_2$*

Unfortunately the safety property does not hold for the secret choice. The following is a counterexample.

**Example 1.** Let $Sec = \{s_1, s_2\}$ and assume that $\mathcal{S}$ does not contain the empty sequence. Let $T = o_1.0 \boxed{+} o_2.0$. Then $Pt_{MAP}(T, \vec{\pi})$ is maximum (i.e. $Pt_{MAP}(T, \vec{\pi}) = 1 - \max \vec{\pi}$) because for every sequence $s \in \mathcal{S}$ we have $p(o_1|s) = p(o_2|s)$. Let $T' = s_1.T \boxed{+} s_2.T$. We can now define a scheduler such that, if the secret starts with $s_1$, it selects $o_1$, and if the secret starts with $s_2$, it selects $o_2$. Hence, under this scheduler, $p(o_1|s_1 s) = p(o_2|s_2 s) = 1$ while $p(o_1|s_2 s) = p(o_2|s_1 s) = 0$. Therefore $Pt_{MAP}(T', \vec{\pi}) = 1 - p_1 - p_2$ where $p_1$ and $p_2$ are the maximum probabilities of the secrets of the form $s_1 s$ and $s_2 s$, respectively. Note now that either $\max \vec{\pi} = p_1$ or

$\max \vec{\pi} = p_2$ because of the assumption that $\mathcal{S}$ does not contain the empty sequence. Let $\vec{\pi}$ be such that both $p_1$ and $p_2$ are positive. Then $1 - p_1 - p_2 < 1 - \max \vec{\pi}$, hence $Pt_{MAP}(T', \vec{\pi}) < Pt_{MAP}(T, \vec{\pi})$.

The reason why we need the condition *(i)* in Definition 8 for the parallel operator is analogous to the case of secret choice. The following is a counterexample.

**Example 2.** Let $Sec$ and $\mathcal{S}$ be as in Example 1. Define $T_1 = s_1.0 \boxplus s_2.0$ and $T_2 = o_1.0 \boxplus o_2.0$. Clearly, $Pt_{MAP}(T_2, \vec{\pi}) = 1 - \max \vec{\pi}$. Consider now the term $T_1 \mid T_2$ and define a scheduler that first executes an action $s$ in $T_1$ and then, if $s$ is $s_1$, it selects $o_1$, while if $s$ is $s_2$, it selects $o_2$. The rest proceeds like in Example 1, where $T' = T_1 \mid T_2$ and $T = T2$.

The reason why we need the condition *(ii)* in Definition 8 is that without it the parallel operator may create different interleavings, thus increasing the possibility of an adversary discovering the secrets. The following is a counterexample.

**Example 3.** Let $Sec$ and $\mathcal{S}$ be as in Example 1. Define $T_1 = o.0$ and $T_2 = s_1.(o_1.0 \oplus_{.5} o_2.0) \boxplus s_2.(o_1.0 \oplus_{.5} o_2.0)$. It is easy to see that $Pt_{MAP}(T_2, \vec{\pi}) = 1 - \max \vec{\pi}$. Consider the term $T_1 \mid T_2$ and define a scheduler that first executes an action $s$ in $T_2$ and then, if $s$ is $s_1$, it selects first $T_1$ and then the continuation of $T_2$, while if $s$ is $s_2$, it selects first the continuation of $T_2$ and then $T_1$. Hence, under this scheduler, $p(oo_1|s_1s) = p(oo_2|s_1s) = .5$ and also $p(o_1o|s_2s) = p(o_2o|s_2s) = .5$ while $p(oo_1|s_2s) = p(oo_2|s_2s) = 0$ and $p(o_1o|s_1s) = p(o_2o|s_1s) = 0$. Therefore we have that $Pt_{MAP}(T, \vec{\pi}) = 1 - p_1 - p_2$ where $p_1$ and $p_2$ are the maximum probabilities of the secrets of the form $s_1s$ and $s_2s$, respectively. Following the same reasoning as in example 1, we have that for certain $\vec{\pi}$, $Pt_{MAP}(T', \vec{\pi}) < Pt_{MAP}(T, \vec{\pi})$.

## 7 A Case Study: The Dining Cryptographers

In this section we consider the Dining Cryptographers (DC) protocol proposed by Chaum in [23], we show how to describe it in CCS$_p$, and we apply the results of previous section to obtain a generalization of Chaum's strong anonymity result.

In its most general formulation, the DC consists of a multigraph where one of the nodes (cryptographers) may be secretly designated to pay for the dinner. The cryptographers would like to find out whether there is a payer or not, but without either discovering the identity of the payer, nor revealing it to an external observer. The problem can be solved as follows: we put on each edge a probabilistic coin, which can give either 0 or 1. The coins get tossed, and each cryptographer computes the binary sum of all (the results of) the adjacent coins. Furthermore, it adds 1 if it is designated to be the payer. Finally, all the cryptographers declare their result.

It is easy to see that this protocol solves the problem of figuring out the existence of a payer: the binary sum of all declarations is 1 if and only if there is a payer, because all the coins get counted twice, so their contribution to the total sum is 0.

The property we are interested in, however, is the anonymity of the system. Chaum proved that the DC is strongly anonymous if all the coins are fair, i.e. they give 0 and 1 with equal probability, and the multigraph is connected, namely there is a path between

**Table 2.** The dining cryptographers protocol expressed in $\mathrm{CCS}_p$

$$
\begin{aligned}
Crypt_i &= c_{i,i_1}(x_1) \,.\, \ldots \,.\, c_{i,i_k}(x_k) \,.\, pay_i(x) \,.\, \bar{d}_i\langle x_1 + \ldots + x_k + x\rangle \\
Coin_h &= \bar{c}_{\ell,h}\langle 0\rangle \,.\, \bar{c}_{r,h}\langle 0\rangle.0 \;\oplus_{p_h}\; \bar{c}_{\ell,h}\langle 1\rangle \,.\, \bar{c}_{r,h}\langle 1\rangle.0 \\
Collect &= d_1(y_1) \,.\, d_2(y_2) \,.\, \ldots \,.\, d_n(y_n) \,.\, \overline{out}\langle y_1, y_2, \ldots, y_n\rangle \\
DC &= (\nu\vec{c})(\nu\vec{d})(\textstyle\prod_i Crypt_i \mid \prod_h Coin_h \mid Collect)
\end{aligned}
$$

each pair of nodes. To state formally the property, let us denote by $s$ the secret identity of the payer, and by $o$ the collection of the declarations of the cryptographers.

**Theorem 3 ([23]).** *If the multigraph is connected, and the coins are fair, then DC is strongly anonymous, namely for every $s$ and $o$, $p(s|o) = \pi_s$ holds.*

We are now going to show how to express the DC in $\mathrm{CCS}_p$. We start by introducing a notation for value-passing in $\mathrm{CCS}_p$, following standard lines.

$$
\begin{aligned}
Input \quad & c(x).T = \textstyle\biguplus_v c_v.T[v/x] \\
Output \quad & \bar{c}\langle v\rangle = \bar{c}_v
\end{aligned}
$$

The protocol can be described as the parallel composition of the cryptographers processes $Crypt_i$, of the coin processes $Coin_h$, and of a process *Collect* whose purpose is to collect all the declarations of the cryptographers, and output them in the form of a tuple. See Table 2. In this protocol, the secret actions are $pay_i$. All the others are observable actions.

Each coin communicates with two cryptographers. $c_{i,h}$ represents the communication channel between $Coin_h$ and $Crypt_i$ if $h$ is indeed the index of a coin, otherwise it represents a communication channel "with the environment". We will call the latter *external*. In the original definition of the DC there are no external channels, we have added them to prove a generalization of Chaum's result. They could be interpreted as a way for the environment to influence the computation of the cryptographers and hence test the system, for the purpose of discovering the secret.

We are now ready to state our generalization of Chaum's result.

**Theorem 4.** *A DC is strongly anonymous if it has a spanning tree consisting of fair coins only.*

*Proof.* Consider the term $DC$ in Table 2. Remove all the coins that do not belong to the spanning tree, and the corresponding restriction operators. Let $T$ be the process term obtained this way. Let $\mathcal{A}$ be the class of schedulers which select the value 0 for all the external channels. This situation corresponds to the original formulation of Chaum and so we can apply Chaum's result (Theorem 3) and Proposition 5 to conclude that all the rows of the matrix $M$ are the same and hence, by Proposition 4, $\mathcal{P}_{MAP}(M, \vec{\pi}) = 1 - \max_i \pi_i$.

Consider now one of the removed coins, $h$, and assume, without loss of generality, that $c_{\ell,h}(x)$, $c_{r,h}(x)$ are the first actions in the definitions of $Crypt_\ell$ and $Crypt_r$. Consider the class of schedulers $\mathcal{B}$ that selects value 1 for $x$ in these actions. The matrix

$M'$ that we obtain is isomorphic to $M$: the only difference is that each column $o$ is now mapped to a column $o + w$, where $w$ is a tuple that has 1 in the $\ell$ and $r$ positions, and 0 in all other positions, and $+$ represents the componentwise binary sum. Since this map is a bijection, we can apply Proposition 3 in both directions and derive that $\mathcal{P}_{MAP}(M', \vec{\pi}) = 1 - \max_i \pi_i$.

We can conclude, therefore, that $Pt_{MAP}(T, \vec{\pi}) = 1 - \max_i \pi_i$ in the class of schedulers $\mathcal{A} \cup \mathcal{B}$.

By repeating the same reasoning on each of the removed coins, we can conclude that $Pt_{MAP}(T, \vec{\pi}) = 1 - \max_i \pi_i$ for any scheduler $\zeta$ of $T$.

Consider now the term $T' = (\nu c_{\ell,h} c_{r,h})(Coin_h \mid T)$ obtained from $T$ by adding back the coin $h$. By applying Theorem 2 we can deduce that $Pt_{MAP}(T', \vec{\pi}) \geq Pt_{MAP}(T, \vec{\pi})$. By repeating this reasoning, we can add back all the coins, one by one, and obtain the original $DC$. Hence we can conclude that $Pt_{MAP}(DC, \vec{\pi}) \geq Pt_{MAP}(T, \vec{\pi}) = 1 - \max_i \pi_i$ and, since $Pt_{MAP}(T, \vec{\pi})$ is maximum, we have $Pt_{MAP}(DC, \vec{\pi}) = 1 - \max_i \pi_i$, which concludes the proof.

Interestingly, also the other direction of Theorem 4 holds. We report this result for completeness, however we have proved it by using traditional methods, not by applying the compositional methods of Section 6.

**Theorem 5.** *A DC is strongly anonymous only if it has a spanning tree consisting of fair coins only.*

## 8 Conclusion and Future Work

In this paper we have investigated the properties of the probability of error associated to a given information-hiding protocol, and we have investigated CCS$_p$constructs that are safe, i.e. that are guaranteed not to decrease the protection of the protocol. Then we have applied these results to strengthen a result of Chaum: the dining cryptographers are strongly anonymous if and only if they have a spanning tree of fair coins.

In the future, we would like to extend our results to other constructs of the language. This is not possible in the present setting, as the examples after Theorem 2 show. The problem is related to the scheduler: the standard notion of scheduler is too powerful and can leak secrets, by depending on the secret choices that have been made in the past. All the examples after Theorem 2 are based on this kind of problem. In [25], we have studied the problem and we came out with a language-based solution to restrict the power of the scheduler. We are planning to investigate whether such approach could be exploited here to guarantee the safety of more constructs.

## References

1. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
2. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for Web transactions. ACM Transactions on Information and System Security 1, 66–92 (1998)
3. McLean, J.: Security models and information flow. In: Proc. of SSP, pp. 180–189. IEEE, Los Alamitos (1990)

4. Gray III, J.W.: Toward a mathematical foundation for information flow security. In: Proc. of SSP 1991, pp. 21–35. IEEE, Los Alamitos (1991)
5. Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. In: Proc. of QAPL 2001. ENTCS, pp. 238–251. Elsevier Science B.V., Amsterdam (2001)
6. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. In: Proc. of QAPL 2004. ENTCS, vol. 112, pp. 149–166. Elsevier Science B.V., Amsterdam (2005)
7. Lowe, G.: Quantifying information flow. In: Proc. of CSFW 2002, pp. 18–31. IEEE Computer Society Press, Los Alamitos (2002)
8. Moskowitz, I.S., Newman, R.E., Crepeau, D.P., Miller, A.R.: Covert channels and anonymizing networks. In: Jajodia, S., Samarati, P., Syverson, P.F. (eds.) WPES, pp. 79–88. ACM, New York (2003)
9. Moskowitz, I.S., Newman, R.E., Syverson, P.F.: Quasi-anonymous channels. In: IASTED CNIS, pp. 126–131 (2003)
10. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 41–53. Springer, Heidelberg (2003)
11. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 54–68. Springer, Heidelberg (2003)
12. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Belief in information flow. Journal of Computer Security (to appear, 2008)
13. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Probability of error in information-hiding protocols. In: Proc. of CSF, pp. 341–354. IEEE, Los Alamitos (2007)
14. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Tech. Rep. MIT/LCS/TR-676, PhD thesis, MIT (1995)
15. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2, 250–273 (1995)
16. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice-Hall, Englewood Cliffs (1989)
17. Herescu, O.M., Palamidessi, C.: Probabilistic asynchronous $\pi$-calculus. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 146–160. Springer, Heidelberg (2000)
18. Palamidessi, C., Herescu, O.M.: A randomized encoding of the $\pi$-calculus with mixed choice. Theoretical Computer Science 335, 373–404 (2005)
19. Deng, Y., Palamidessi, C., Pang, J.: Compositional reasoning for probabilistic finite-state behaviors. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 309–337. Springer, Heidelberg (2005)
20. Cover, T.M., Thomas, J.A.: Elements of Information Theory. John Wiley & Sons, Inc., Chichester (1991)
21. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Information and Computation (to appear, 2007)
22. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
23. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
24. Halpern, J.Y., O'Neill, K.R.: Anonymity and information hiding in multiagent systems. Journal of Computer Security 13, 483–512 (2005)
25. Chatzikokolakis, K., Palamidessi, C.: Making random choices invisible to the scheduler. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR. LNCS, vol. 4703, pp. 42–58. Springer, Heidelberg (2007)

# Products of Message Sequence Charts*

Philippe Darondeau, Blaise Genest, and Loïc Hélouët

IRISA, campus de Beaulieu, F-35042 Rennes Cedex

**Abstract.** An effective way to assemble partial views of a distributed system is to compute their product. Given two Message Sequence Graphs, we address the problem of computing a Message Sequence Graph that generates the product of their languages, when possible. Since all MSCs generated by a Message Sequence Graph $G$ may be run within fixed bounds on the message channels (that is, $G$ is existentially bounded), a subproblem is to decide whether the considered product is existentially bounded. We show that this question is undecidable, but turns co-NP-complete in the restricted case where all synchronizations belong to the same process. This is the first positive result on the decision of existential boundedness. We propose sufficient conditions under which a Message Sequence Graph representing the product can be constructed.

## 1 Introduction

Scenario languages, and in particular Message Sequence Charts (MSCs) have met a considerable interest over the last decade in both academia and industry. MSCs allow for the compact description of distributed systems executions, and their visual aspect made them popular in the engineering community. Our experience with industry (France-Telecom) showed us that MSCs are most often used there together with extensions such as optional parts (that is choice) and (weak) concatenation, while iteration is left implicit. (Compositional) Message Sequence Graphs ((C)MSC-graphs) is the academic framework in which choice, weak concatenation and iteration of MSCs are formalized. For a recent survey of Message Sequence Graphs, we refer the reader to [6,9]. A challenging problem is to automatically implement MSC-languages (that is, sets of MSCs) given by (C)MSC-graphs. Apart from the restricted case of Local Choice (C)MSC-graphs [8,7], this problem has received no satisfactory solution, since either deadlocks arise from the implementation [14,4], or implementation may exhibit unspecified behaviors [2]. A further challenge is to help designing (C)MSC-graphs for complex systems, while keeping analysis and implementability decidable. Systems often result from assembling modules, reflecting different aspects. A possible way to help the modular modeling of systems into (C)MSC-graphs is thus to provide a product operator. A first attempt in this direction is [10], where the amalgamation allows the designer to merge 2 nodes of 2 MSG-graphs but not their paths. We feel that a more flexible operation, defined on MSC languages and therefore independent from MSC block decompositions, is needed.

* Work supported by France Telecom R&D (CRE CO2) and ANR project DOTS.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 458–473, 2008.

Shuffling the linearizations of the languages of two (C)MSC-graphs is not the right product. On the one hand, such shuffling kills *existential bounds* [11], i.e., there is no upper bound on the size of the message channels within which all MSCs in the shuffled language *can* be run. Existential bounds are an important feature of (safe C)MSC-graphs which allow their analysis. On the other hand, two states of the (C)MSC-graphs (one for each module) may represent incompatible aspects. Hence, one needs some synchronization to control the product operation, in order to avoid incompatibilities and non existentially bounded behaviors. Control may be introduced with synchronization points: one module waits at a synchronization point until the other module reaches a compatible synchronization point, and then both can proceed. Synchronizations may be defined either per process or per state of the (C)MSC-graphs. State oriented synchronization conflicts with weak concatenation since it means that all processes of the *same* module pass simultaneously the synchronization barrier, which diverges strongly from the semantics of (C)MSC-graphs. Second, it harms implementability, since state-synchronized products of implementable (C)MSC-graphs may not be implementable. We therefore choose to define synchronizations per process, by means of shared local events identified by names common to both MSCs. Formally, we define thus a mixed product of MSCs that amounts to shuffling their respective events on each process, simultaneously and independently, except for the shared events that are not interleaved but coalesced. One appealing property of this definition of product is that the product of two implementable (C)MSC-graphs is also implementable (albeit with possible deadlocks), since it suffices to take the product of the implementations processwise, coalescing shared events.

In order to be represented as a (safe C)MSC-graph, an MSC language needs to be existentially bounded. So far, no algorithm is known to check the existential boundedness of an MSC language in a non-trivial case (e.g., existential boundedness is undecidable even for deterministic deadlock-free Communicating Finite State Machines, see *http://perso.crans.org/˜genest/GKM07.pdf*). This is the challenging problem studied in this paper. We show that checking existential boundedness of the product of two (safe C)MSC-graphs is in general undecidable, as one expects. Surprisingly, if all shared events (synchronizations) belong to the same process, then this question becomes decidable. Once a product is known to be existentially bounded, results [12,4] on representative linearizations can be used. Namely, languages of MSCs defined by the globally cooperative subclass of safe CMSC-graphs have regular sets of linear representatives, where the regular representations can be computed from the CMSC-graphs and conversely. Thus, given two globally cooperative CMSC-graphs such that their product is existentially bounded, this product can be represented with a globally cooperative CMSC-graph. The authors of [4] ignore the contents of messages in the definition of MSCs. We consider messages with contents, and adapt the FIFO requirement of [4] to both weak ([2,13]) and strong FIFO ([1]). We recast the correspondence established in [4] into these different frameworks, and compare the complexity and decidability of these two semantics.

The paper is organized as follows. Section 2 recalls the background of MSCs and MSC-graphs. Section 3 introduces the product of MSC-languages. Section 4 recalls the definition of existential channel bounds for MSC-languages. It is shown in Sections 5 and 6 that one can, in general, not check the existential boundedness of the product of two existentially bounded MSC-languages, whereas this problem is co-NP-complete (weak FIFO) or PSPACE (strong FIFO) when the synchronizations are attached to a single process. Section 7 defines for that special case an operation of product on CMSC-graphs. Many proofs are skipped or only briefly sketched by lack of space, but they are available in the full version of the paper, available at *http://perso.crans.org/˜genest/DGH07.pdf*.

## 2 Background

To begin with, we recall the usual definition of *compositional Message Sequence Charts* (*CMSCs* for short), which describe executions of communication protocols, and of CMSC-graphs, which are generators of CMSC sets. Let $\mathcal{P}$, $\mathcal{M}$, and $\mathcal{A}$ be fixed finite sets of *processes*, *messages* and *actions*, respectively. Processes may perform *send* events $\mathcal{S}$, *receive* events $\mathcal{R}$ and *internal* events $\mathcal{I}$. That is, the set of types of events of an MSC is $\mathcal{E} = \mathcal{S} \cup \mathcal{R} \cup \mathcal{I}$ where $\mathcal{S} = \{p\,!\,q\,(m) \,|\, p,q \in \mathcal{P}, p \neq q, m \in \mathcal{M}\}$,$\mathcal{R} = \{p\,?\,q\,(m) \,|\, p,q \in \mathcal{P}, p \neq q, m \in \mathcal{M}\}$, and $\mathcal{I} = \{p\,(a) \,|\, p \in \mathcal{P}, a \in \mathcal{A}\}$. For each $p \in \mathcal{P}$, we let $\mathcal{E}_p = \mathcal{S}_p \cup \mathcal{R}_p \cup \mathcal{I}_p$ where $\mathcal{S}_p$, $\mathcal{R}_p$, and $\mathcal{I}_p$ are the restrictions of $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{I}$, respectively, to the considered process $p$ (*e.g.*, $p\,?\,q\,(m) \in \mathcal{S}_p$). We define now MSCs over $\mathcal{E}$.

**Definition 1.** *A compositional Message Sequence Chart $M$ is a tuple $M = (E, \lambda, \mu, (<_p)_{p\in\mathcal{P}})$ where*

- *$E$ is a finite set of events, with types $\lambda(e)$ given by a labeling map $\lambda : E \to \mathcal{E}$,*
- *for each $p \in \mathcal{P}$, $<_p$ is a total order on $E_p = \lambda^{-1}(\mathcal{E}_p)$,*
- *$\mu : E \to E$ is a partially defined, injective mapping,*
- *if $\mu(e_1) = e_2$ then $\lambda(e_1) = p\,!\,q\,(m)$ and $\lambda(e_2) = q\,?\,p\,(m)$ for some $p, q$ and $m$,*
- *[**weak FIFO**] if $e_1 <_p e_1'$, $\lambda(e_1) = \lambda(e_1') = p\,!\,q\,(m)$ and $\mu(e_1')$ is defined, then $\mu(e_1) <_q \mu(e_1')$ (in particular, $\mu(e_1)$ is defined).*
- *the union $<$ of $\cup_{p\in\mathcal{P}} <_p$ and $\cup_{e\in E} \{(e, \mu(e))\}$ is an acyclic relation.*
- *$M$ is an MSC if the partial map $\mu$ is a bijection between $\lambda^{-1}(\mathcal{S})$ and $\lambda^{-1}(\mathcal{R})$.*

Def. 1 extends the original definition of [5] (see also [4]) by considering messages with non trivial contents. There are then two alternatives to the FIFO condition. *Strong FIFO* requires that $e_1 <_p e_1'$, $\lambda(e_1) = p\,!\,q\,(m)$, $\lambda(e_1') = p\,!\,q\,(m')$ and $\mu(e_1')$ defined entail $\mu(e_1) <_q \mu(e_1')$, i.e., there is a single channel from $p$ to $q$. The *weak FIFO* requirement used in Def. 1 means that there are as many FIFO channels from $p$ to $q$ as there are types of events $p\,!\,q\,(m)$. In general, there are undecidable problems in the strong FIFO semantics, as weak realizability [1], which are decidable in the weak FIFO semantics [13]. Anyway, all (un)decidability results established in this paper hold for both FIFO semantics, even though complexity depends on the semantics used.

Given a CMSC $X = (E, \lambda, \mu, (<_p)_{p\in\mathcal{P}})$, let $\leq_X$ be the reflexive and transitive closure of the relation $<$ from Def. 1. A *linear extension* of $X$ is an enumeration of $E$ compatible with $\leq_X$. A *linearization* of $X$ is the image of a linear extension of $X$ under the map $\lambda : E \to \mathcal{E}$ (hence it is a word of $\mathcal{E}^*$). Let $\mathcal{L}in(X)$ denote the set of linearizations of $X$. For a set $\mathcal{X}$ of CMSCs, let $\mathcal{L}in(\mathcal{X})$ denote the union of $\mathcal{L}in(X)$ for all $X \in \mathcal{X}$. Linearizations can be defined more abstractly as follows:

**Definition 2.** *Let $\mathcal{L}in \subseteq \mathcal{E}^*$ be the set of all words $w$ such that for all $p, q$ and $m$, the number of occurrences $q\,?\,p\,(m)$ is at most equal to the number of occurrences $p\,!\,q\,(m)$ in every prefix $v$ of $w$, and both numbers are equal for $v = w$. In the strong FIFO setting, we furthermore require the equality of contents of the $i$-th emission from $p$ to $q$ and of the $i$-th reception on $q$ from $p$.*

Any linearization $w$ of an *MSC* belongs to $\mathcal{L}in$ (it may not be the case for a CMSC). Conversely, because of weak or strong FIFO, a word $w = \epsilon_1 \dots \epsilon_n \in \mathcal{L}in$ is the linearization of a *unique* MSC, $Msc(w) = (\{1, \dots, n\}, \lambda, \mu, (<_p))$, with:

– $\lambda(i) = \epsilon_i$ and $i <_p j$ if $i < j$ and $\epsilon_i, \epsilon_j \in \mathcal{E}_p$,
– $\mu(i) = j$ if the letter $\epsilon_i = p\,!\,q\,(m)$ occurs $k$ times in $\epsilon_1 \dots \epsilon_i$ and the letter $\epsilon_j = q\,?\,p\,(m)$ occurs $k$ times in $\epsilon_1 \dots \epsilon_j$ for some $p, q, m, k$.

**Definition 3.** *Two words $w, w' \in \mathcal{L}in$ are equivalent (notation $w \equiv w'$) if $Msc(w)$ and $Msc(w')$ are isomorphic. For any language $\mathcal{L} \subseteq \mathcal{L}in$, we write $[\mathcal{L}] = \{w \mid w \equiv w', w' \in \mathcal{L}\}$. A language $\mathcal{L} \subseteq \mathcal{L}in(\mathcal{X})$ is a* representative set *for a set $\mathcal{X}$ if $\mathcal{L} \cap \mathcal{L}in(X) \neq \emptyset$ for all $X \in \mathcal{X}$, or equivalently, if $[\mathcal{L}] = \mathcal{L}in(\mathcal{X})$.*

We deduce the following properties. For any MSC $X$, $\mathcal{L}in(X)$ is an equivalence class in $\mathcal{L}in$. For any MSC $X$ and for any $w \in \mathcal{L}in$, $w \in \mathcal{L}in(X)$ if and only if $X$ is isomorphic to $Msc(w)$. A similar property does not hold for arbitrary CMSCs. For instance, $(p\,!\,q\,(m))\,(q\,?\,p\,(m))\,(q\,?\,p\,(m))$ belongs to $\mathcal{L}in(X)$ for two different CMSCs $X$, where the emission is matched by $\mu$ either with the first or with the second reception.

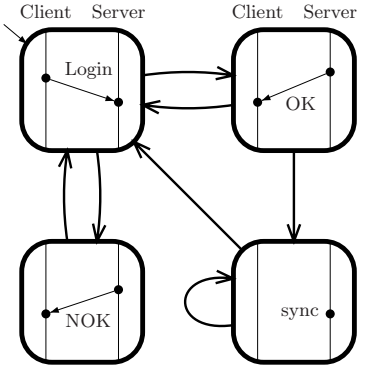



**Fig. 1.** Identification Scenario $G_1$

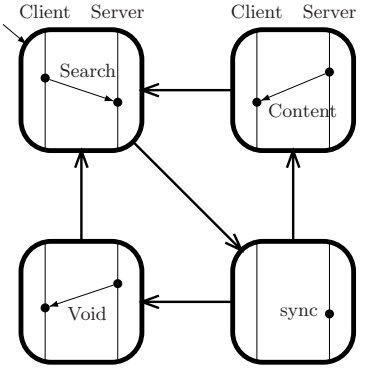



**Fig. 2.** Searching Scenario $G_2$

We define the concatenation $X_1 \cdot X_2$ of two CMSCs $X_i = (E^i, \lambda^i, \mu^i, (<^i_p)_{p\in\mathcal{P}})$ as the set of CMSCs $X = (E^1 \uplus E^2, \lambda^1 \uplus \lambda^2, \mu, (<_p)_{p\in\mathcal{P}})$ such that:

– $\mu \cap (E^i \times E^i) = \mu^i$ and $<_p \cap (E^i \times E^i) = <^i_p$ for $i \in \{1,2\}$ and $p \in \mathcal{P}$,
– $e \in E^2$ and $e \leq_X e'$ entail $e' \in E^2$ for all $e, e' \in E^1 \uplus E^2$.

We let $\mathcal{X}_1 \cdot \mathcal{X}_2$ be the union of $X_1 \cdot X_2$ for all $X_i \in \mathcal{X}_i$, $i \in \{1,2\}$. We can now give a description of sets of MSCs with rational operations.

**Definition 4.** *A CMSC-graph is a tuple $G = (V, \rightarrow, \Lambda, V^0, V^f)$ where $(V, \rightarrow)$ is a finite graph, $V^0, V^f \subseteq V$ are the subsets of initial and final vertices, respectively, and $\Lambda$ maps each vertex $v$ to a CMSC $\Lambda(v)$. We define $\mathcal{L}(G)$ as the set of all* MSCs *in $\Lambda(v_0) \cdot \Lambda(v_1) \cdot \ldots \cdot \Lambda(v_n)$ where $v_0, v_1 \ldots, v_n$ is a path in $G$ from some initial vertex $v_0 \in V^0$ to some final vertex $v_n \in V^f$. The CMSC-graph $G$ is* safe *if any such set $\Lambda(v_0) \cdot \ldots \cdot \Lambda(v_n)$ contains at least one MSC.*

Intuitively, the semantics of CMSC-graphs is defined using the concatenation of the CMSCs labeling the vertices met along the paths in these graphs. Notice that $\Lambda(v_0) \cdot \ldots \cdot \Lambda(v_n)$ may contain an arbitrary number of CMSCs, but at most one of these CMSCs is an MSC. An example of a non-safe CMSC-graph is $G = (V, \rightarrow, \Lambda, \{v_0\}, \{v_f\})$ where $V = \{v_0, v_f\}$, $v_0 \rightarrow v_f$, the CMSC $\Lambda(v_0)$ has a single event labeled with $q\,?\,p\,(m)$, and the CMSC $\Lambda(v_f)$ has a single event labeled with $p\,!\,q\,(m)$. Indeed the two events cannot be matched by $\mu$ in $\Lambda(v_0) \cdot \Lambda(v_f)$. Notice that this is a XCMSG [12]. The reason why we do not allow XCMSGs is that safe XCMSGs are not necessarily existentially bounded, hence the Mazurkiewicz trace coding needed for the results of [4] that we use for Theorem 3 fails. Fig. 1 and 2 show two (C)MSC-graphs. Their nodes are labeled with MSCs. Concatenating *OK* and the local event *sync* gives an MSC with 3 events. The reception of *OK* and the event *sync* are unordered (in $G_1$). On the contrary, the event *sync* and the reception of *Void* are ordered (in $G_2$).

A safe CMSC-graph $G$ may always be expanded into a safe *atomic* CMSC-graph $G'$, that is a graph in which each node is labeled with a single event, such that $\mathcal{L}(G) = \mathcal{L}(G')$. In the following, every safe CMSC-graph is assumed to be atomic. The expansion yields, by the way, a regular representative set for $\mathcal{L}(G)$.

## 3 Product of MSC-Languages

In order to master the complexity of distributed system descriptions, it is desirable to have at one's disposal a composition operation that allows us to weave different aspects of a system. When system aspects are CMSC-graphs with disjoint sets of processes, the concatenation of their MSC-languages can be used to this effect. Else, some parallel composition with synchronization capabilities is needed. We propose here to shuffle the events of the two MSC-graphs per process, except for the common events that serve to the synchronization. We require that all common events are internal events. Formally, what we define is an extension of the *mixed product* of words. The intersection with a regular

language could be used in place of the synchronizations to control the shuffle, but this would not change significantly the results of this paper. However, synchronizing on messages could change the results, as we can encode shared events using shared messages, but not the other way around.

First, we recall the definition of the *mixed product* $L_1 \parallel L_2$ of two languages $L_1, L_2$ of words (see [3]), defined on two alphabets $\Sigma_1, \Sigma_2$ not necessarily disjoint. Let $\Sigma = \Sigma_1 \cup \Sigma_2$. For $i = 1, 2$ let $\pi_i : \Sigma^* \to \Sigma_i^*$ be the unique monoid morphism such that $\pi_i(\sigma) = \sigma$ for $\sigma \in \Sigma_i$ and $\pi_i(\sigma) = \varepsilon$, otherwise. Then $L_1 \parallel L_2 = \{w \mid \pi_i(w) \in L_i, i = \{1, 2\}\}$ is the set of all words $w \in \Sigma^*$ with respective projections $\pi_i(w)$ in $L_i$; e.g., $\{ab\} \parallel \{cad\} = \{cabd, cadb\}$ ($a$ is the synchronizing action).

**Definition 5.** *For $i = \{1, 2\}$, let $\mathcal{X}_i$ be an MSC-language over some $\mathcal{E}^i$, such that $x \in \mathcal{E}^1 \cap \mathcal{E}^2$ implies $x = p(a)$ for some $p, a$. The* mixed product $\mathcal{X}_1 \parallel \mathcal{X}_2$ *is $Msc\,(\,(\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2)) \cap \mathcal{L}in\,)$ and it is an MSC-language over $\mathcal{E}^1 \cup \mathcal{E}^2$.*

The mixed product operation serves to compose the languages of two CMSC-graphs that share only internal events, as is the case for the CMSC-graphs $G_1, G_2$ of Fig. 1,2. The synchronization *sync* ensures that in any MSC in $\mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$, the server never answers a search request from the client unless the client is logged in. Thus, synchronizations serve to avoid mixing incompatible fragments of the two CMSC-graphs. When a set $\mathcal{X}$ is a singleton $\mathcal{X} = \{X\}$, we abusively write $X \parallel \mathcal{Y}$ instead of $\{X\} \parallel \mathcal{Y}$. Note that even though $X_1$ and $X_2$ are MSCs, $X_1 \parallel X_2$ may contain more than one MSC. Under weak FIFO semantics, mixing all linearizations pairwise yields all and only linearizations of a product of MSCs. However, the product of two linearizations of strong FIFO MSCs may contain words that are not linearizations of strong FIFO MSCs. Intersecting with $\mathcal{L}in$ allows us to keep only linearizations of (strong FIFO) MSCs.

**Proposition 1.** $\mathcal{L}in(\mathcal{X}_1 \parallel \mathcal{X}_2) = (\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2)) \cap \mathcal{L}in.$

**Lemma 1.** *$(\mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)) \cap \mathcal{L}in$ is closed under $\equiv$ (see Def. 3).*

However, $\{X_1\} \parallel \{X_2\}$ may be larger than $Msc\,(w_1 \parallel w_2)$ for fixed representations $w_1 \in \mathcal{L}in(X_1)$ and $w_2 \in \mathcal{L}in(X_2)$. This situation is illustrated with

$$
\begin{aligned}
w_1 &= (p\,!\,q\,(m_1))\,(q\,?\,p\,(m_1))\,(p\,!\,q\,(m_1))\,(q\,?\,p\,(m_1)),\\
w_1' &= (p\,!\,q\,(m_1))^2\,(q\,?\,p\,(m_1))^2,\\
w_2 &= (q\,!\,p\,(m_2))\,(p\,?\,q\,(m_2))\,(q\,!\,p\,(m_2))\,(p\,?\,q\,(m_2)),\\
w_2' &= (q\,!\,p\,(m_2))^2\,(p\,?\,q\,(m_2))^2,\\
w_3 &= (p\,!\,q\,(m_1))^2\,(q\,!\,p\,(m_2))^2\,(p\,?\,q\,(m_2))^2\,(q\,?\,p\,(m_1))^2.
\end{aligned}
$$

and $X_1 = Msc(w_1) = Msc(w_1')$, $X_2 = Msc(w_2) = Msc(w_2')$, $X_3 = Msc(w_3)$. There is no synchronization. Now $X_3 \in Msc(w_1' \parallel w_2')$, but $X_3 \notin Msc(w_1 \parallel w_2)$. This observation shows that products must be handled with care. Indeed, an advantage of CMSC-graphs is to represent large sets of linearizations with small subsets of representatives. However, $w_1$ is a representative for $X_1$, $w_2$ is for $X_2$, but $w_1 \parallel w_2$ is not a set of representatives for $X_1 \parallel X_2$.

## 4 Bounds for MSCs and Products

We review in this section ways of classifying CMSC-graphs based on bounds for communication channels, and we examine how these bounds behave under product of CMSC-languages. We focus on MSC-languages with *regular* representative sets. As indicated earlier, a regular representative set for the language of a safe CMSC-graph $G$ may be obtained by expanding $G$ into an atomic CMSC-graph $G'$. As observed in [12], it follows from a pumping lemma that whenever $\mathcal{L} \subseteq \mathcal{L}in$ is a regular representative set for some $\mathcal{X}$, the words in $\mathcal{L}$ are uniformly $B$-bounded, for some $B > 0$, as defined hereafter. First, the definition of a channel depends on the semantics. In the weak FIFO setting, a channel is a triple $p, q \in \mathcal{P}, m \in \mathcal{M}$, and $p!q(m)$ is an emission ($q?p(m)$ is a reception) on this channel. In the strong FIFO setting, a channel is a pair $p, q \in \mathcal{P}$, and $p!q(m)$ is an emission ($q?p(m')$ is a reception) on this channel for any $m, m' \in \mathcal{M}$. A word $w \in \mathcal{E}^*$ is *$B$-bounded* if, for any prefix $v$ of $w$ and any channel $c$, the number of emissions on $c$ in $v$ exceeds the number of receptions on $c$ in $v$ by at most $B$.

A MSC $X$ is *$\forall$-$B$-bounded* if every linearization $w \in \mathcal{L}in(X)$ is $B$-bounded. A MSC $X$ is *$\exists$-$B$-bounded* if some linearization $w \in \mathcal{L}in(X)$ is $B$-bounded. A set of MSCs $\mathcal{X}$ is *$\exists$-$B$-bounded* if all MSCs $X \in \mathcal{X}$ are $\exists$-$B$-bounded; $\mathcal{X}$ is *existentially bounded* if it is $\exists$-$B$-bounded for some $B$. Let $\mathcal{L}in^B(\mathcal{X})$ denote the set of $B$-bounded words $w$ in $\mathcal{L}in(\mathcal{X})$. Clearly, any $\mathcal{X}$ with a regular representative set is existentially $B$-bounded for some $B$, but it may not be $\forall$-$B$-bounded for any $B$. Conversely, when an MSC-language $\mathcal{X}$ is $\exists$-$B$-bounded, $\mathcal{L}in^B(\mathcal{X})$ is a representative set for $\mathcal{X}$, but it is not necessarily a regular language.

**Proposition 2.** $\mathcal{L}in^B(\mathcal{X}_1 \parallel \mathcal{X}_2) = (\mathcal{L}in^B(\mathcal{X}_1) \parallel \mathcal{L}in^B(\mathcal{X}_2)) \cap \mathcal{L}in^B$.

The above result shows that the mixed product behaves nicely with respect to bounded linearizations. If $\mathcal{X}_1$ and $\mathcal{X}_2$ are $\forall$-$B$-bounded, then $\mathcal{L}in(\mathcal{X}_i) = \mathcal{L}in^B(\mathcal{X}_i)$, and using Prop. 1, their product is also $\forall$-$B$-bounded. However, it may occur that both $\mathcal{X}_1$ and $\mathcal{X}_2$ are $\exists$-$B$-bounded but their mixed product is not existentially bounded. For instance, for all $j$, let $X_1^j$ be the MSC with $j$ messages $m_1$ from $p$ to $q$ and $X_2^j$ be the MSC with $j$ messages $m_2$ from $q$ to $p$. All these MSCs are $\exists$-1-bounded since $(p!q(m_1)q?p(m_1))^j \in \mathcal{L}in(X_1^j)$ is 1-bounded. Define $\mathcal{X}_1 = \{X_1^j \mid j > 0\}$ and $\mathcal{X}_2 = \{X_2^j \mid j > 0\}$, thus $\mathcal{X}_1$, $\mathcal{X}_2$ are $\exists$-1-bounded, but $\mathcal{X}_1 \parallel \mathcal{X}_2$ is not $\exists$-$B$-bounded for any $B$ since $Msc(p!q(m_1)^B(q!p(m_2)p?q(m_2))^Bq?p(m_1)^B) \in \mathcal{X}_1 \parallel \mathcal{X}_2$, but it is not $\exists$-$(B-1)$-bounded.

**Definition 6.** *Given an MSC $X = (E, \lambda, \mu, (<_p)_{p\in\mathcal{P}})$ and a non-negative integer $B$, let $Rev_B$ be the binary relation on $E$ such that $e\, Rev_B\, e'$ if and only if, for some channel $c$, $e$ is the $i$-th reception on channel $c$ and $e'$ is the $i+B$-th emission on channel $c$. We also define $Rev_{\geq B} = \cup_{B' \geq B} Rev_{B'}$.*

**Proposition 3 (lemma 2 in [11]).** *A MSC $X$ is $\exists$-$B$-bounded if and only if the relation $< \cup Rev_B$ is acyclic, if and only if the relation $< \cup Rev_{\geq B}$ is acyclic.*

If $X$ is $\exists$-$B$-bounded then $X$ is $\exists$-$B'$-bounded for all $B' \geq B$, because $Rev_{B'}$ is included in the least order relation containing $Rev_B$ and $\bigcup_{p\in\mathcal{P}} <_p$. For instance, in $Msc(p!q(m_1)^B\,(q!p(m_2)\,p?q(m_2))^B\,q?p(m_1)^B)$ let $(a_i, b_i)$ denote the $i$-th pair of events $(p!q(m_1), q?p(m_1))$ and $(c_i, d_i)$ the $i$-th pair of events $(q!p(m_2), p?q(m_2))$, then $a_B <_p d_1\, Rev_{(B-1)}\, c_B <_q b_1\, Rev_{(B-1)}\, a_B$ is a cycle.

## 5 Monitored Product of MSC-Languages

It is important to analyze formally MSC-languages, since following paths in MSC-graphs does not help grasping all the generated scenarios. Most often, in decidable cases [7,16], the analysis of an MSC-language $\mathcal{X}$ amounts to check either the membership of a given MSC $X$, or whether $\mathcal{L}in(\mathcal{X})$ has an empty intersection with a regular language $L$ (representing the complement of a desired property). In the case of a product language $\mathcal{X}_1 \parallel \mathcal{X}_2$, membership can be checked using the projections, since $X \in \mathcal{X}_1 \parallel \mathcal{X}_2$ if and only if $\pi_i(X) \in \mathcal{X}_i$ for $i = 1, 2$. However, in order to analyse regular properties of $\mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$, one often needs computing a safe CMSC-graph $G$ such that $\mathcal{L}(G) = \mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$. In particular, one needs an existential bound $B$ for the product. Unfortunately, the theorem below shows that one cannot decide whether such $G$ exists when $G_1$ and $G_2$ share events on two processes or more.

**Theorem 1.** *Let $G_1, G_2$ be two (safe C)MSC-graphs. It is undecidable whether $\mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$ is existentially bounded, in both weak and strong FIFO semantics.*

*Proof.* We show that the Post correspondence problem may be reduced to the above decision problem. Given two finite lists of words $u_1, \ldots, u_n$ and $w_1, \ldots, w_n$ on some alphabet $\Sigma$ with at least two symbols, the problem is to decide whether $u_{i_1} u_{i_2} \ldots u_{i_k} = w_{i_1} w_{i_2} \ldots w_{i_k}$ for some non-empty sequence of indices $i_1 \cdots i_k$. This problem is known to be undecidable for $n > 7$. Given an instance of the Post correspondence problem, *i.e.*, two lists of words $u_1, \ldots, u_n$ and $w_1, \ldots, w_n$ on $\Sigma$, consider the two MSC-graphs $G_1 = (V, \rightarrow, \Lambda_1, V^0, V^f)$ and $G_2 = (V, \rightarrow, \Lambda_2, V^0, V^f)$, with the same underlying graph $(V, \rightarrow, V^0, V^f)$, constructed as follows ($G_1$ is partially shown in Fig. 3).

Define $V = \{v_0, v_1, \ldots, v_n, v_{n+1}\}$ with $V^0 = \{v_0\}$ and $V^f = \{v_{n+1}\}$. Let $v_0 \rightarrow v_i$, $v_i \rightarrow v_j$, and $v_i \rightarrow v_{n+1}$ for all $i, j \in \{1, \ldots, n\}$ (where possibly $i = j$). Finally let $v_{n+1} \rightarrow v_{n+1}$.

For each $v \in V$, $\Lambda_1(v)$ is a finite MSC over $\mathcal{P}_1 = \{p, q\}$, $\mathcal{A}_1 = \{1, \ldots, n\} \cup \Sigma$, $\mathcal{M}_1 = \{m_1, m_1'\}$. Actions $i \in \{1, \ldots, n\}$ represent indices of pairs of words $(u_i, v_i)$ and they occur on process $p$. Actions $\sigma \in \Sigma$ represent letters of words $u_i$ and they occur on process $q$. Let $\Lambda_1(v_0)$ be the empty MSC. For $i \in \{1, \ldots, n\}$, let $\Lambda_1(v_i)$ be the MSC with $p!q(m_1)$ followed by $p(i)$ on process $p$ and with $q?p(m_1)$ followed by the sequence $q(\sigma_{i,1})\, q(\sigma_{i,2}) \ldots q(\sigma_{i,l_i})$, representing $u_i = \sigma_{i,1}\, \sigma_{i,2} \ldots \sigma_{i,l_i}$, on process $q$. Finally let $\Lambda_1(v_{n+1})$ be the MSC with the events $p!q(m_1')$ and $q?p(m_1')$ on processes $p$ and $q$, respectively.

For each $v \in V$, $\Lambda_2(v)$ is a finite MSC over $\mathcal{P}_2 = \{p, r, q\}$, $\mathcal{A}_2 = \{1, \ldots, n\} \cup \Sigma$, $\mathcal{M}_2 = \{m_2, m_2'', m_2'\}$. For $i = 0, \ldots, n$, $\Lambda_2(v_i)$ is defined alike $\Lambda_1(v_i)$ but now

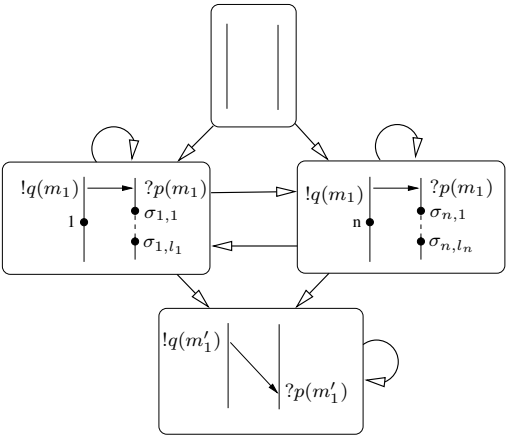



**Fig. 3.**

replacing the message $p!q(m_1), q?p(m_1)$ with two messages $p!r(m_2), r?p(m_2)$, $r!q(m_2''), q?r(m_2'')$ and $u_i$ with $w_i$. $\Lambda_2(v_{n+1})$ is the MSC with the events $p?q(m_2')$ and $q!p(m_2')$ on processes $p$ and $q$, respectively.

For $i = 1, 2$ let $\mathcal{X}_i = \mathcal{L}(G_i)$, then $\mathcal{L}in^1(\mathcal{X}_i)$ is a regular representative set for $\mathcal{X}_i$. If the Post correspondence problem has no solution, then $\mathcal{X}_1 \parallel \mathcal{X}_2$ is empty, hence it is existentially bounded. In the converse case, $\mathcal{X}_1 \parallel \mathcal{X}_2$ contains for all $B$ some MSC including a crossing of $B$ messages $m_1'$ by $B$ messages $m_2'$, hence it is not existentially bounded. □

The proof of Theorem 1 is inspired by the proof that $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset$ is undecidable for generic MSC-graphs $G_1, G_2$ [15]. Theorem 1 motivates the introduction of a *monitor* process *mp* and a *monitored product* in which all synchronizations are (internal) events located on the monitor process. The set of synchronizations is denoted by $\mathcal{SE}$. The *monitored product* $\mathcal{X}_1 \parallel_{\mathrm{mp}} \mathcal{X}_2$ of sets $\mathcal{X}_1$ and $\mathcal{X}_2$ on monitor process $mp \in \mathcal{P}$ is defined only if $\mathcal{SE} \subseteq \{mp(a) \mid a \in \mathcal{A}\}$. In that case, we set $\mathcal{X}_1 \parallel_{\mathrm{mp}} \mathcal{X}_2 = \mathcal{X}_1 \parallel \mathcal{X}_2$. For instance, in the monitored product $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ of the CMSC-graphs of Fig. 1 and Fig. 2, we can choose $mp = server$ and $\mathcal{SE} = \{mp(sync)\}$. The adequacy of the monitored product to weave aspects of a distributed system is confirmed by the following theorem, which holds for both strong and weak FIFO semantics. We conjecture that the problem is PSPACE-complete in the strong FIFO case.

**Theorem 2.** *Given two safe CMSC-graphs $G_1, G_2$, one can decide whether the monitored product of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ is $\exists$-bounded. The problem is co-NP-complete and in PSPACE for weak and strong FIFO semantics respectively.*

The next section sketches a proof for this theorem. Notice that the proof is trivial in the case where $G_1, G_2$ have disjoint sets of processes except for *mp*. Then, $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is existentially bounded (with the bound given by the maximum of the minimal existential bounds of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$).

## 6 Checking Existential Boundedness

We prove Theorem 2 in two stages. First, we show that if the monitored product $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is existentially bounded, then this property holds for a 'small' bound with respect to the size of $G_1$ and $G_2$.

**Proposition 4.** *Given two safe CMSC-graphs $G_1$ and $G_2$, the MSC-language $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is existentially bounded if and only if it is existentially $B^w$-bounded (resp. $B^s$-bounded) for weak (resp. strong) FIFO semantics, where $B^w = 2K_1B'$, $B^s = 2K_2K_3B'$, and $K_1, K_2, B'$ (resp. $K_3$) are polynomial (resp. exponential) in the size of $P, G_1, G_2$.*

Then we show that one can check whether the monitored product of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ is $\exists$-$B$-bounded, using the bounds $B^w, B^s$ of Prop. 4. Notice that $B^s$ written in binary is of size polynomial in $|G_1| + |G_2|$.

**Proposition 5.** *Given two safe CMSC-graphs $G_1, G_2$ and an integer $B$, it is co-NP-complete (resp. PSPACE) to decide whether $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is $\exists$-$B$-bounded, for weak (resp. strong) FIFO semantics. The PSPACE result holds also when $B$ is written in binary.*

★ *Graph representation of monitored products*

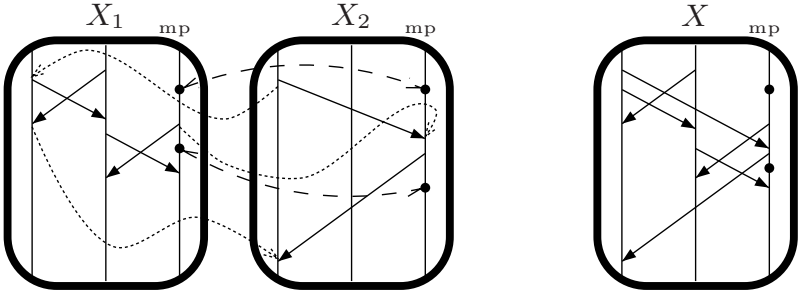



**Fig. 4.** $X \in X^1 \parallel_{\mathrm{mp}} X^2$ and the corresponding relations $\rightarrow^1 \cup \rightarrow^2, \leftrightarrow$

These two results are obtained using special representations for MSCs constructed by monitored product. Let $X \in X_1 \parallel_{\mathrm{mp}} X_2$ then $\exists w \in \mathcal{L}in$: $X = Msc(w)$ and $\pi_i(w) = w_i \in \mathcal{L}in(X_i)$. The MSC $X$ is determined up to isomorphism by its projections on processes, because of FIFO. More precisely, for each $p \in \mathcal{P}$, $\pi_p(w) \in \pi_p(w_1) \parallel \pi_p(w_2)$. Moreover, for $p = mp$, $\pi_p(w_1)$ and $\pi_p(w_2)$ have the same projection on $\mathcal{SE}$. Therefore the projection $(E_p, <_p)$ of $X$ on each process $p$ may be seen as an interleaving of $(E_p^1, <_p^1)$ and $(E_p^2, <_p^2)$ where the synchronized pairs of events $e_1 \in E_{mp}^1$ and $e_2 \in E_{mp}^2$ with labels in $\mathcal{SE}$ are coalesced. Let $\longleftrightarrow \subseteq E_{mp}^1 \times E_{mp}^2$ be the relation comprising synchronized pairs of events. For each $p \in \mathcal{P}$, let $\rightarrow_p^1 \subseteq E_p^2 \times E_p^1$ (resp. $\rightarrow_p^2 \subseteq E_p^1 \times E_p^2$) be the relation comprising ordered pairs of events $e_2\, e_1$ (resp. $e_1\, e_2$) switching from $E_p^2$ to $E_p^1$

(resp. $E_p^1$ to $E_p^2$) in the interleaved sequence $(E_p, <_p)$. The MSC $X$ may now be represented by the juxtaposition of $X_1$ and $X_2$ interlinked with $\longleftrightarrow$ and with the relations $\rightarrow_p^1$ and $\rightarrow_p^2$ for all $p \in \mathcal{P}$. The result is a *graph*, that we denote $X_{1\|_{\mathrm{mp}}2}$, with set of nodes $E^1 \cup E^2$. Conversely, any acyclic graph connecting $X_1$ and $X_2$ with relations $\rightarrow_p^i$ and $\longleftrightarrow$ represents a non-empty set of weak FIFO MSCs $\mathcal{X}$. We say that the transitive closure $<_{\|_{\mathrm{mp}}}$ of $<_i^p$, $\rightarrow_p^i$ and $\longleftrightarrow$ is *compatible with strong FIFO* if there do not exist two messages $(s, r), (s', r')$ on the same channel $c$ such that $s < s'$ and $r' < r$. There may be several such MSCs if for some $p$ the relation $\rightarrow_p^1 \cup \rightarrow_p^2 \cup <_p^1 \cup <_p^2$ is not a total order on $E_p$. Otherwise, the original MSC $X$ may be reconstructed from $X_{1\|_{\mathrm{mp}}2}$ as follows: $E$ is the quotient of $E^1 \cup E^2$ by the equivalence relation $\longleftrightarrow$ and $<=<_{\|_{\mathrm{mp}}}|_E$. For an illustration, see Fig. 4 where the edges of the graph represent the relations $<_p^i$, $\mu_i$, $\longleftrightarrow$ (dashed) and $\rightarrow_p^i$ (dotted). The graph is compatible with strong FIFO. A unique MSC $X$ can be reconstructed from it, depicted on the right of the figure. More formally, we can state the following lemma:

**Lemma 2.** *Let $G_1$ and $G_2$ be safe and atomic CMSC-graphs and $B$ an integer. Then $\mathcal{L}(G_1) \|_{\mathrm{mp}} \mathcal{L}(G_2)$ is $\exists$-$B$-bounded if and only if, for any synchronized pair of MSCs $X_1 \in \mathcal{L}(G_1)$ and $X_2 \in \mathcal{L}(G_2)$ with respective sets of events $E^1$ and $E^2$, there is no subset $\{e_1, \ldots, e_n\} \subseteq E^1 \cup E^2$ with at most two events in $E_p^1 \cup E_p^2$ for each process $p \in \mathcal{P}$ such that:*

1. *for all $j$, $(e_j, e_{(j+1) \bmod n})$ belongs to one of the relations $<^i$, $Rev_B$, or $E_p^i \times E_p^{3-i}$ for $i = 1$ or $2$ and $p \in \mathcal{P}$,*
2. *there is no proper cycle in $\{e_1, \ldots, e_n\}$ w.r.t. the transitive closure $<_{\|_{\mathrm{mp}}}$ of the relation $<^1 \cup <^2 \cup \longleftrightarrow \cup \rightarrow$ where $\longleftrightarrow$ is the synchronizing relation among coalesced events, and $e \rightarrow e'$ if $e = e_j \in E_p^i$ and $e' = e_{(j+1) \bmod n} \in E_p^{3-i}$ for some $j \in \{1, \ldots n\}$, $i \in \{1, 2\}$ and $p \in \mathcal{P}$,*
3. *in the strong FIFO case, $<_{\|_{\mathrm{mp}}}$ is compatible with strong FIFO.*

The proofs of Prop. 4 and 5 are based on synchronized paths and Lemma 3. A *synchronized path* $\alpha\beta_1 \cdots \beta_n\gamma$ of $G_1, G_2$ is a sequence of pairs of paths $\alpha = (\alpha^1, \alpha^2), \beta_i = (\beta_i^1, \beta_i^2), \gamma = (\gamma^1, \gamma^2)$, where $\alpha^k\beta_1^k \cdots \beta_n^k\gamma^k$ is a path of $G_k$, $\pi_{\mathcal{SE}}(\alpha^1) = \pi_{\mathcal{SE}}(\alpha^2)$, $\pi_{\mathcal{SE}}(\beta_i^1) = \pi_{\mathcal{SE}}(\beta_i^2)$ and $\pi_{\mathcal{SE}}(\gamma^1) = \pi_{\mathcal{SE}}(\gamma^2)$. Furthermore, $\beta_i^k$ is a loop of $G_k$ for all $i, k$. Lemma 3 claims that if $n$ is sufficiently large, there exists a synchronized loop of $\rho_2$ which has no contribution to the ordering between events in $\rho_1$ and $\rho_2$. This loop can thus be removed or iterated without compromising acyclicity, and is compatible with strong FIFO if needed.

**Lemma 3.** *Let $G_1$ and $G_2$ be safe and atomic CMSC-graphs, $K$ be an integer and $(\alpha^1, \alpha^2)(\beta_1^1, \beta_1^2) \cdots (\beta_K^1, \beta_K^2)(\gamma^1, \gamma^2)$ be a synchronized path of $G_1, G_2$. Let $\rightarrow$ be a partial order on a set $E$ of $n \leq 2|\mathcal{P}|$ events of $\alpha^1 \cup \alpha^2 \cup \gamma^1 \cup \gamma^2$ compatible with the order of the synchronized path. For all $j \geq i \geq 1, \ell \geq 0$, we denote by $<_{i,j}^{\ell}$ the relation on $(\alpha^1, \alpha^2)(\beta_1^1, \beta_1^2) \cdots [(\beta_i^1, \beta_i^2) \cdots (\beta_j^1, \beta_j^2)]^{\ell} \cdots (\beta_K^1, \beta_K^2)(\gamma^1, \gamma^2)$ generated by the synchronizations and the relation $\rightarrow$.*

- *For all* $i, j, i', j'$, $<^1_{i,j} = <^1_{i',j'}$, *denoted* $<$, *and this relation is a partial order.*
- *Let* $K_1, K_2, K_3$ *be the constants of Prop. 4.*
- *If* $K > K_1$, *then there exists* $i$ *such that for all* $x, y \in \alpha^1 \cup \alpha^2 \cup \gamma^1 \cup \gamma^2$ *and* $l \geq 0$, *we have* $x <^\ell_{i,i} y$ *iff* $x < y$ *(in particular,* $<^\ell_{i,i}$ *is a partial order).*
- *If* $K > K_2 K_3$ *and* $<$ *is compatible with strong FIFO, then there exist* $i, j$ *such that* $<^\ell_{i,j}$ *is an order compatible with strong FIFO, for all* $l \geq 0$.

★ *General outline of the proof for Prop. 4*

Let $X \in \mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$, thus $X$ may be represented in product form by $X_{1\parallel_{\mathrm{mp}}2} = (X_1, X_2, \longleftrightarrow, (\rightarrow^i_p)^{i=1,2}_{p\in\mathcal{P}})$. Suppose that $X$ is not $\exists$-$B$-bounded for some $B = 2KB'$. By Prop. 3, $< \cup Rev_{\geq B}$ has a cycle in $X$.We have $Rev^1_{\geq B} \cup Rev^2_{\geq B} \subseteq Rev_{\geq B} \subseteq Rev^1_{\geq B/2} \cup Rev^2_{\geq B/2}$. Therefore, the union of $\longleftrightarrow$ and the relations $<^i$, $Rev^i_{\geq KB'}$, and $\rightarrow^i_p$ for $i = 1, 2$ has a cycle $e_1\, e_2 \ldots e_m$ with $e_j \neq e_k$ for $j \neq k$. We let $e_{m+1} = e_1$. One can assume that $e_1\, e_2 \ldots e_m$ contains no synchronization event with shared label and at most two events on each process $p$ (Lemma 5.5 in [4]), hence $m \leq 2\,|\mathcal{P}|$. Furthermore, there is at least one pair of events $(e_j, e_{j+1})$ in $Rev^i_{B_j}$, w.l.o.g. $e_1\, Rev^1_{B_1}\, e_2$, with $B_1 \geq KB'$. Notice that $(e_1 \cdots e_m)$ is also a cycle for the union of $\longleftrightarrow$, $<^i$, $Rev^i_{\geq KB'}$, and $\rightarrow^i_p \cap (e_j, e_{j+1})_{j\leq m}$ for $i = 1, 2$, that is we need to consider only a linear number of pairs in $\rightarrow^i_p$. We construct MSCs $X'_1 \in \mathcal{L}(G_1), X'_2 \in \mathcal{L}(G_2)$ embedding $X_1, X_2$ via $\phi : X_i \hookrightarrow X'_i$ such that $\phi(e_1)\, \phi(e_2) \ldots \phi(e_m)$ is a cycle for $\phi(Rev_{\geq KB'}) \cup <_{X'}$, where $(X', <_{X'})$ is the oriented graph obtained by connecting $X'_1$ and $X'_2$ with $\longleftrightarrow$ and $\phi(\rightarrow^i_p) \cap (\phi(e_j), \phi(e_{j+1}))_{j\leq m}$. More precisely, $X'_1, X'_2$ are such that $\phi(e_1)\, Rev^1_{\geq 2B_1+1}\, \phi(e_2)$ and $e_j\, Rev^i_{B_j}\, e_{j+1} \Rightarrow \phi(e_j)\, Rev^i_{\geq B_j}\, \phi(e_{j+1})$ for $j \neq 1$ and $B_j \geq KB'$. As soon as $<_{X'}$ is a partial order (compatible with strong FIFO if needed), Prop. 4 follows by induction and by applying Lemma 2.

★ *General outline of the proof for Prop. 5*

In order to conclude that $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is not $\exists$-$B$-bounded, one should search for MSCs $X_1 \in \mathcal{L}(G_1)$, $X_2 \in \mathcal{L}(G_2)$, and $X \in (X_1 \parallel_{\mathrm{mp}} X_2)$ such that $<_X \cup Rev_B$ contains a cycle. In the weak FIFO setting, we use a small model property. Assume that the product of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ is not existentially $B^w$ bounded. We apply Lemma 2 to obtain a synchronized pair of paths $\rho_1, \rho_2$ of $G_1, G_2$, with a set $E$ of at most $2|\mathcal{P}|$ events, and a relation $\rightarrow \in E \times E$ which creates a cycle with $<^i \cup Rev^i_{B^w}$. By contradiction, assume that the minimal size of such a synchronized path $(\rho_1, \rho_2)$ (that is its number of transitions) is larger than $((4|\mathcal{P}|B^w + 1)K_1 B'$, then it contains $(4|\mathcal{P}|B^w + 1)K_1$ synchronized pairs of loops. Applying Lemma 3 with $\ell = 0$, we know that there are $4|\mathcal{P}|B^w + 1$ loops which can be individually deleted without changing the order on $E$. There are at most $2|\mathcal{P}|B^w$ messages which can affect the $Rev^i_{B^w}$ relation, hence $4|\mathcal{P}|B^w$ loops which contain some emission or reception of such messages. Therefore, one synchronized pair of loops can be deleted without changing the order on $E$ nor the $Rev^i_{B^w}$ relations, which contradicts the minimality of $\rho_1, \rho_2$. To obtain

a co-NP algorithm, it suffices to guess a path of $G_1$ and a path of $G_2$ of size polynomial, to guess $2|\mathcal{P}|$ events, and to check in polynomial time that there is no cycle in $<^1 \cup <^2 \cup \leftrightarrow \cup \rightarrow$, whereas there is a cycle in $<^1 \cup <^2 \cup \leftrightarrow \cup \rightarrow \cup Rev^i_{B^w}$. Notice that we cannot do the same in the strong FIFO setting, since the exponential bound $B^s$ would lead to a co-NEXPTIME algorithm. Instead, we construct a finite automaton, whose language is empty iff the product is existentially $B^s$ bounded. Each state can be described in polynomial space w.r.t. $|G_1|, |G_2|$ and $||B^s|| = \log_2(B^s)$ written in binary.

★ *General outline of the co-NP-completeness reduction*

We prove the co-NP hardness of the problem of deciding either the existential-boundedness or the existential-$B$-boundedness of the product of languages of two MSC-graphs. We do not use the contents of the messages, hence the reduction holds for both weak and strong FIFO semantics. Let $\phi$ be a 3-CNF-SAT instance, with $n$ variables and $m$ clauses. This formula is true iff for each clause, one can choose a literal of the clause to be true, and no conflict occurs on a variable (one cannot choose a literal and its opposite being true). Let $B > m + 1$. We build two MSC-graphs $G_1$ and $G_2$ on processes $\{p, q, r, p_i, p'_i \mid 1 \leq i \leq n\}$ such that $G_1 \parallel_{\text{mp}} G_2$ is $\exists$-$B$-bounded iff $\phi$ is non satisfiable. We let $mp = p$.

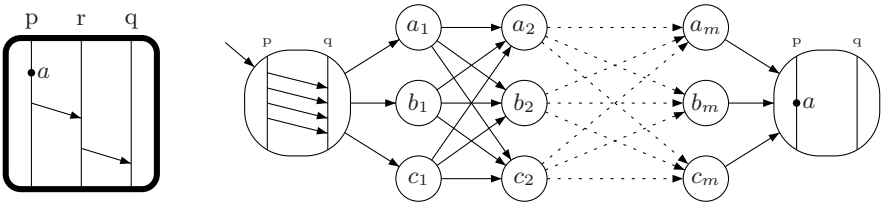



**Fig. 5.** MSC $M_1$ and MSC-graph $G_2$

Graph $G_1$ is made of one node, both initial and final. The node is labeled by MSC $M_1$, which is a synchronization action $a$ on process $p$, then a message from $p$ to $r$, then a message from $r$ to $q$. For graph $G_2$, the initial node is labeled with $B + 1$ messages from $p$ to $q$. Then $G_2$ has a succession of $m$ choices between three nodes $a_i, b_i, c_i$, $i \leq m$. Then the final node of $G_2$ is labeled by the synchronization event $a$ on process $p$. Informally, the $m$ choices correspond to the $m$ clauses, and $a_i, b_i, c_i$ correspond to the choice of the first, second and third literal true in the $i$-th clause. That is, if the first literal in the $i$-th clause is $v_j$, then $a_i$ is labeled by a message from $q$ to $p_j$ and a message from $p'_j$ to $p$. If the first literal in the $i$-th clause is $\neg v_j$, then $a_i$ is labeled by a message from $q$ to $p'_j$ and a message from $p_j$ to $p$. Any MSC from $G_2$ corresponds to some choice of literal true in each of the clauses and vice versa. Now, a conflict occurs on one variable iff the receptions on $q$ from $p$ (in $G_2$) are before the synchronization event $a$, iff for the corresponding MSC $M_2$ of $G_2$, all MSCs in $M_2 \parallel_{\text{mp}} M_1$ are $\exists$-$B$-bounded.

## 7 CMSC-Graph Representation of a Monitored Product

In the case where $\mathcal{L}(G_1) \parallel_{\text{mp}} \mathcal{L}(G_2)$ is $\exists$-bounded, one may wish to compute a safe CMSC-graph representation of this MSC-language, which can be input to existing tools for analyzing MSC-graphs (MSCan, SOFAT...). For this purpose, we use the results from [4], where a syntax-semantics correspondence is established between *globally cooperative* CMSC-graphs [7], and MSC-languages $\mathcal{X}$ with regular representative sets $\mathcal{L}in^B(\mathcal{X})$ for some $B > 0$.

**Definition 7.** *$G = (V, \rightarrow, \Lambda, V^0, V^f)$ is a* globally cooperative *CMSC-graph if*

– *$G$ is a safe CMSC-graph, and*
– *for any circuit $v_1 \dots v_n$ in $G$, all CMSCs in the set $\Lambda(v_1) \cdot \ldots \cdot \Lambda(v_n)$ have connected communication graphs.*

*The communication graph induced by $X = (E, \lambda, \mu, (<_p)_{p\in\mathcal{P}})$ is the undirected graph $(Q, E)$ with the set of vertices $Q = \{p \in \mathcal{P} \mid (\exists e \in E)\ \ \lambda(e) \in \mathcal{S}_p \cup \mathcal{R}_p\}$ and with the set of edges $E = \{\{p, q\} \mid (\exists e_1, e_2 \in E)\,(\exists m \in \mathcal{M})\ \lambda(e_1) = p\,!\,q\,(m) \wedge \lambda(e_2) = q\,?\,p\,(m)\}$.*

Notice that the MSC-graph from Fig. 3 is globally cooperative. Thus, boundedness of the product of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ stays undecidable even when both $G_1, G_2$ are globally cooperative (Theorem 1). Quite remarkably, $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset$ is decidable as soon as $G_1$ or $G_2$ is globally cooperative [7].

**Theorem 3.** *Let $\mathcal{X}$ be a set of MSCs. The following are equivalent:*

– *$\mathcal{X} = \mathcal{L}(G)$ for some globally cooperative CMSC-graph $G$,*
– *$\mathcal{L}in^B(\mathcal{X})$ is a regular representative set for $\mathcal{X}$ for sufficiently large $B > 0$. Moreover, $B$ and a finite automaton recognizing $\mathcal{L}in^B(\mathcal{X})$ can be computed effectively from $G$. Conversely, $G$ can be computed effectively from $\mathcal{L}in^B(\mathcal{X})$.*

The statement of Theorem 3 is the same as (a fragment of) the main theorem of [4]. However, we consider in this paper messages with contents, while [4] does not. Instead of proving Theorem 3 from scratch, we derive it from [4]. The strong FIFO case comes directly from the proof of [4]. For weak FIFO, we use a translation from sets of weak FIFO MSCs to sets of FIFO MSCs with exactly one (type of) message $m$ (hence they embed in weak FIFO MSCs). In few words, the translation adds as many processes as types of messages per channel, and it preserves the existential boundedness of sets of MSCs, although the bound $B$ may grow to $3B$. Once this translation is defined, the proof of Theorem 3 is almost immediate.

Now let $G_1, G_2$ be two globally cooperative CMSC-graphs. If $\mathcal{L}(G_1) \parallel_{\text{mp}} \mathcal{L}(G_2)$ is $\exists$-bounded, then this MSC-language is $\exists$-$B$-bounded, for $B \in \{B^s, B^w\}$ as defined in Prop. 4. Therefore, $\mathcal{L}in^B(\mathcal{L}(G_1) \parallel_{\text{mp}} \mathcal{L}(G_2))$ is a representative set for $\mathcal{L}(G_1) \parallel_{\text{mp}} \mathcal{L}(G_2)$. By Prop. 2, $\mathcal{L}in^B(\mathcal{L}(G_1) \parallel_{\text{mp}} \mathcal{L}(G_2)) = \mathcal{L}in^B(\mathcal{L}(G_1)) \parallel_{\text{mp}} \mathcal{L}in^B(\mathcal{L}(G_2)) \cap \mathcal{L}in^B$. Since both $G_1, G_2$ are globally cooperative, we get that both $\mathcal{L}in^B(\mathcal{L}(G_1))$ and $\mathcal{L}in^B(\mathcal{L}(G_2))$ are regular and effectively computable. Since the shuffle of regular language is regular, we get the following.

**Theorem 4.** *Let $G_1, G_2$ be two globally cooperative CMSC-graphs such that $\mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$ is $\exists$-bounded. Then one can effectively compute a globally cooperative CMSC-graph $G$ with $\mathcal{L}(G) = \mathcal{L}(G_1) \parallel_{\mathrm{mp}} \mathcal{L}(G_2)$. Moreover, $G$ is of size at most exponential and doubly exponential in the size of $|G_1|, |G_2|$, respectively with weak and strong FIFO.*

## 8 Conclusion

We presented a framework to work with the controlled products of distributed components, granted that synchronizations are operated on a single monitor process, and components are given as globally cooperative CMSC-graphs. Namely, one can test whether the monitored product of components can be represented as a globally cooperative CMSC-graph. In that case, a complete analysis of the product system can be performed with existing tools. We analyze the problem in both weak and strong FIFO contexts. Weak FIFO enjoys a better complexity, while strong FIFO allows us to use *non-synchronized* actions with common names on different components (it suffices to rename the actions according to components, perform the product, and then rename the actions back). A direction for future work is to propose guidelines and tools for modeling product systems with one monitor process.

## References

1. Alur, R., Etessami, K., Yannakakis, M.: Realizability and Verification of MSC Graphs. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 797–808. Springer, Heidelberg (2001)
2. Caillaud, B., Darondeau, P., Hélouët, L., Lesventes, G.: HMSCs as Partial Specifications.. with PNs as Completions. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 125–152. Springer, Heidelberg (2001)
3. Duboc, C.: Mixed Product and Asynchronous Automata. Theoretical Computer Science 48(3), 183–199 (1986)
4. Genest, B., Kuske, D., Muscholl, A.: A Kleene Theorem and Model Checking for a Class of Communicating Automata. Inf. Comput. 204(6), 920–956 (2006)
5. Gunter, E., Muscholl, A., Peled, D.: Compositional Message Sequence Charts. STTT 5(1), 78–89, (2003); In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 496–511. Springer, Heidelberg (2001)
6. Genest, B., Muscholl, A., Peled, D.: Message Sequence Charts. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 537–558. Springer, Heidelberg (2004)
7. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state High-level MSCs: Model-checking and Realizability. JCSS 72(4), 617–647 (2006); Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.): ICALP 2002. LNCS, vol. 2380, pp. 617–647. Springer, Heidelberg (2002)
8. Hélouët, L., Jard, C.: Conditions for synthesis of communicating automata from HMSCs. In: FMICS 2000, pp. 203–224 (2000)
9. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M.A., Thiagarajan, P.S.: A theory of regular MSC languages. Inf. Comput. 202(1), 1–38 (2005)

10. Klein, J., Caillaud, B., Hélouët, L.: Merging scenarios. In: FMICS 2004, pp. 209–226 (2004)
11. Lohrey, M., Muscholl, A.: Bounded MSC communication. Inf. Comput. 189(2), 160–181 (2004)
12. Madhusudan, P., Meenakshi, B.: Beyond Message Sequence Graphs. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 256–267. Springer, Heidelberg (2001)
13. Morin, R.: Recognizable Sets of Message Sequence Charts. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 523–534. Springer, Heidelberg (2002)
14. Mukund, M., Kumar, K.N., Sohoni, M.A.: Bounded time-stamping in message-passing systems. TCS 290(1), 221–239 (2003)
15. Muscholl, A., Peled, D., Su, Z.: Deciding properties of Message Sequence Charts. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 226–242. Springer, Heidelberg (1998)
16. Muscholl, A., Peled, D.: Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999)

# What Else Is Decidable about Integer Arrays?*

Peter Habermehl[1], Radu Iosif[2], and Tomáš Vojnar[3]

[1] LSV, ENS Cachan, CNRS, INRIA; 61 av. du Président Wilson, F-94230 Cachan, France
and LIAFA, University Paris 7, Case 7014, 75205 Paris Cedex 13
haberm@liafa.jussieu.fr
[2] VERIMAG,CNRS, 2 av. de Vignate, F-38610 Gières, France
iosif@imag.fr
[3] FIT BUT, Božetěchova 2, CZ-61266, Brno, Czech Republic
vojnar@fit.vutbr.cz

**Abstract.** We introduce a new decidable logic for reasoning about infinite arrays of integers. The logic is in the $\exists^*\forall^*$ first-order fragment and allows (1) Presburger constraints on existentially quantified variables, (2) difference constraints as well as periodicity constraints on universally quantified indices, and (3) difference constraints on values. In particular, using our logic, one can express constraints on consecutive elements of arrays (e.g., $\forall i \,.\, 0 \leq i < n \rightarrow a[i+1] = a[i] - 1$) as well as periodic facts (e.g., $\forall i \,.\, i \equiv_2 0 \rightarrow a[i] = 0$). The decision procedure follows the automata-theoretic approach: we translate formulae into a special class of Büchi counter automata such that any model of a formula corresponds to an accepting run of an automaton, and vice versa. The emptiness problem for this class of counter automata is shown to be decidable as a consequence of earlier results on counter automata with a flat control structure and transitions based on difference constraints.

## 1 Introduction

Arrays are a fundamental data structure in computer science. They are used in all modern imperative programming languages. To verify software which manipulates arrays, it is essential to have a sufficiently powerful logic, which can express meaningful program properties, arising as verification conditions within, e.g., inductive invariant checking, or verification of pre- and post-conditions. In order to have an automatic decision procedure for the program verification problems, one needs a decidable logic.

In this paper, we develop a logic of arrays indexed by integer numbers, and having integers as values. To be as general as possible, and also to avoid having to deal explicitly with expressions containing out-of-bounds array accesses, we interpret formulae over both-ways infinite arrays. Bounded arrays can then be conveniently expressed in the logic by restricting indices to be within given bounds.

Properties that are typically of interest about arrays in a program are (existentially quantified) boolean combinations of formulae of the form $\forall \mathbf{i}.G \rightarrow V$ where $G$ is a *guard expression* containing constraints over the universally quantified index variables $\mathbf{i}$

* The work was supported by the French Ministry of Research (RNTL project AVERILES), the Czech Grant Agency (projects 102/07/0322, 102/05/H050), the Czech-French Barrande project 2-06-27, and the Czech Ministry of Education by project MSM 0021630528.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 474–489, 2008.

(which often range in between some existentially quantified bounds), and $V$ is a *value expression* containing constraints over array values. Based on examples, we identified two types of array properties which seem to appear quite often in programs: (1) properties relating consecutive elements of an array, e.g., $\forall i \,.\, l_1 \leq i < l_2 \rightarrow a[i+1] = a[i] - 1$, which states the fact that each value of $a$ between two bounds $l_1$ and $l_2$ is less than its predecessor by one, (2) properties stating periodic facts, e.g., $\forall i \,.\, i \equiv_2 0 \rightarrow a[i] = 0$, stating that all even elements of an array $a$ are equal to 0.

Without specific syntactic restrictions, a logic with such an expressive power can be easily shown to be undecidable as one can encode histories of computations of a 2-counter machine [13] as models of a formula over arrays. From this reduction, one can derive two restrictions leading to decidability. The first restriction forbids references to $a[i]$ and $a[i+1]$ in the same formula, which is considered in the work of Bradley, Manna, and Sipma [5]. The second restriction, considered in this paper, allows only array formulae $\forall \mathbf{i}.G \rightarrow V$ in which $V$ does not contain disjunctions. We have chosen the second option, mainly to retain the possibility of relating consecutive arrays elements, i.e., $a[i]$ and $a[i+1]$, which appears to be important for expressing properties of programs.

We introduce a new logic **LIA** (Logic on Integer Arrays) in the $\exists^*\forall^*$ first-order fragment. **LIA** is essentially the set of existentially quantified boolean combinations of (1) array formulae of the form $\forall \mathbf{i} \,.\, \varphi(\mathbf{k},\mathbf{i}) \rightarrow \psi(\mathbf{k},\mathbf{i},\mathbf{a})$ where $\mathbf{i}$ is a set of index variables and $\mathbf{a}$ (resp. $\mathbf{k}$) is a set of existentially quantified array (resp. *array-bound*) variables, $\varphi$ is a formula on index variables with difference as well as periodicity constraints on variables $\mathbf{i}$ wrt. the array-bounds $\mathbf{k}$, and $\psi$ is a difference constraint on array terms, and (2) Presburger arithmetic formulae on array-bound variables. In [8], we give an example program showing the usefulness of this logic to express verification conditions.

We prove decidability of the logic **LIA** using the classical idea of the connection between logic and automata [18]: from a formula $\varphi$ of the logic, we build an automaton $A_\varphi$ such that $\varphi$ is satisfiable if and only if the language of $A_\varphi$ is not empty. Decidability of the logic then follows from decidability of the emptiness problem for the class of automata that is deployed. To this end, we define a new class of counter automata, called FBCA (bi-infinite Flat Büchi Counter Automata). These are counter automata running to infinity in both left and right directions, equipped with a Büchi acceptance condition. For an arbitrary formula $\varphi$ of **LIA**, we give the construction of the FBCA $A_\varphi$ whose runs correspond to models of $\varphi$: the value of the counter $x_a$ at a given point $i$ in an execution of $A_\varphi$ corresponds to the value of $a[i]$ in a model of $\varphi$. We prove decidability of **LIA** by showing that the emptiness problem for FBCA is decidable by extending known results [6,4] on flat counter automata with difference bound constraints.

**Related work.** In the seminal paper [12], the read and write functions from/to arrays and their logical axioms were introduced. A decision procedure for the quantifier-free fragment of the theory of arrays was presented in [10]. Since then, various decidable logics on arrays have been considered—e.g., [17,11,9,16,1,7]. These logics include working with various predicates (reasoning about sortedness, permutations, etc.) and in terms of various arithmetic (usually Presburger) constraints on array indices and/or values of array entries. However, unlike our logic, most of these works consider quantifier free formulae. In these cases, nested array reads (like $a[a[i]]$) are allowed, which is not the case in our logic.

In [5], an interesting logic within the $\exists^*\forall^*$ fragment is developed. Unlike our decision procedure based on automata theory, the decision procedure of [5] is based on the fact that the universal quantification can be replaced by a finite conjunction. The result is parameterised in the sense of allowing an arbitrary decision procedure to be used for the data stored in arrays. However, compared to our results, [5] does not allow modulo constraints (allowing to speak about periodicity in the array values), general difference constraints on *universally* quantified indices (only $i - j \leq 0$ is allowed), nor reasoning about array entries at a fixed distance (i.e., reasoning about $a[i]$ and $a[i+k]$ for a constant $k$ and a universally quantified index $i$). The authors of [5] give also interesting undecidability results for extensions of their logic. For example, they show that relating adjacent array values ($a[i]$ and $a[i+1]$), or having nested reads, leads to undecidability.

A restricted form of universal quantification within $\exists^*\forall^*$ formulae is also allowed in [2], where decidability is obtained based on a small model property. Unlike [5] and our work, [2] allows a hierarchy-restricted form of array nesting. However, similar to the restrictions presented above, neither modulo constraints on indices nor reasoning about array entries at a fixed distance are allowed. A similar restriction not allowing to express properties of consecutive elements of arrays then appears also in [3] where a quite general $\exists^*\forall^*$ logic on multisets of elements with associated data values is considered.

**Remark.** For space reasons, all proofs are deferred to [8].

## 2 Counter Automata

Given a formula $\varphi$, we denote by $FV(\varphi)$ the set of its free variables. If we denote a formula as $\varphi(x_1,...,x_n)$, we assume $FV(\varphi) \subseteq \{x_1,...,x_n\}$. For $\varphi(x_1,...,x_n)$, we denote by $\varphi[t/x_i]$, $1 \leq i \leq n$, the formula in which each occurrence of $x_i$ is replaced by a term $t$. Given a formula $\varphi$, we denote by $\models \varphi$ the fact that $\varphi$ is logically valid, i.e., it holds in every structure corresponding to its signature. By $\sigma : \mathbb{Z} \to \mathbb{Z}$, $\sigma(n) = n+1$, we denote the successor function on integers. In the following, we work with two sets of arithmetic formulae: difference bound matrices and Presburger arithmetic.

A *difference bound matrix* (DBM) formula is a conjunction of inequalities of the form $x - y \leq c$, $x \leq c$, or $x \geq c$ where $c \in \mathbb{Z}$ is a constant. If there is no constraint between $x$ and $y$, we may explicitly write $x - y \leq \infty$. In the following, $\mathbb{Z}^\infty$ denotes $\mathbb{Z} \cup \{\infty\}$. Let $\mathbf{z} = \{z_1, \ldots, z_n\}$ be a designated set of variables, called *parameters*. A *parametric DBM* formula is a conjunction of a DBM formula with atomic propositions of the forms $x \leq f(\mathbf{z})$ or $x \geq f(\mathbf{z})$ where $f$ is a linear combination of parameters, i.e., $f = a_0 + \sum_{i=1}^{n} a_i z_i$ for some $a_i \in \mathbb{Z}$, $0 \leq i \leq n$.

A *Presburger arithmetic* (PA) formula is a disjunction of conjunctions of either linear constraints of the form $\sum_{i=1}^{n} a_i x_i + b \geq 0$ or modulo constraints $\sum_{i=1}^{n} a_i x_i + b \equiv c \mod d$ where $a_i, b, c, d \in \mathbb{Z}$, $c \geq 0$ and $d > 0$, are constants. It is well-known that every formula of the arithmetic of integers with addition $\langle \mathbb{Z}, \geq, +, 0, 1 \rangle$ can be written in this form due to quantifier elimination [15]. Clearly, every DBM formula is also in PA.

A *counter automaton* (CA) is a tuple $A = \langle \mathbf{x}, Q, \rightarrow \rangle$ where $\mathbf{x}$ is a finite set of counters ranging over $\mathbb{Z}$, $Q$ a finite set of control states, and $\rightarrow$ a transition relation given by rules $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$ where $\varphi$ is an arithmetic formula relating current values of counters $\mathbf{x}$ to

their future values $\mathbf{x}'$. A *configuration* of a CA $A$ is a pair $(q,\nu)$ where $q \in Q$ is a control state, and $\nu : \mathbf{x} \to \mathbb{Z}$ is a valuation of the counters in $\mathbf{x}$. For a configuration $c = (q,\nu)$, we designate by $val(c) = \nu$ the valuation of the counters in $c$. A configuration $(q',\nu')$ is an *immediate successor* of $(q,\nu)$ if and only if $A$ has a transition rule $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$ such that $\models \varphi(\nu(\mathbf{x}),\nu'(\mathbf{x}'))$. A configuration $c$ is a *successor* of another configuration $c'$ iff there exists a sequence of configurations $c = c_0 c_1 \dots c_n = c'$ such that, for all $0 \leq i < n$, $c_{i+1}$ is an immediate successor of $c_i$. Given two control states $q, q' \in Q$, a run of $A$ from $q$ to $q'$ is a finite sequence of configurations $c_0 c_1 \dots c_n$ with $c_0 = (q,\nu)$, $c_n = (q',\nu')$ for some valuations $\nu,\nu' : \mathbf{x} \to \mathbb{Z}$, and $c_{i+1}$ is an immediate successor of $c_i$ for all $0 \leq i < n$.

Let $S$ be a set. A *bi-infinite sequence* of $S$ is a function $\beta : \mathbb{Z} \to S$.[1] We denote by ${}^{\omega}S^{\omega}$ the set of all bi-infinite sequences over $S$. A *bi-infinite Büchi counter automaton* (BCA) is a tuple $A = \langle \mathbf{x}, Q, L, R, \to \rangle$ where $\mathbf{x}$ is a finite set of counters, $Q$ is a finite set of control states, $L, R \subseteq Q$ are the left-accepting and right-accepting states, and $\to$ is a transition relation defined in the same way as for counter automata.

A *run* of a BCA $A$ is a bi-infinite sequence of configurations $\dots c_{-2} c_{-1} c_0 c_1 c_2 \dots$ such that, for all $i \in \mathbb{Z}$, $c_{i+1}$ is an immediate successor of $c_i$. A run $r$ is *left-accepting* iff there exists a state $q \in L$ and an infinite decreasing sequence of integers $\dots < i_2 < i_1 < 0$ such that, for all $j \in \mathbb{N}$, we have $r(i_j) = (q,\nu_j)$ for some valuations $\nu_j$ of the counters of $A$. Symmetrically, a run is *right-accepting* iff there exists a state $q \in R$ and an infinite increasing sequence of integers $0 < i_0 < i_1 < i_2 < \dots$ such that, for all $j \in \mathbb{N}$, we have $r(i_j) = (q,\nu_j)$ for some valuations $\nu_j$ of the counters of $A$. A run is *accepting* iff it is both left- and right-accepting. The set of all accepting runs of $A$ is denoted as $\mathcal{R}(A)$. If $r \in \mathcal{R}(A)$ is a run of $A$, we define as $val(r) = \dots val(r(-1)) val(r(0)) val(r(1)) \dots$ the bi-infinite sequence of valuations in $r$, and we let $\mathcal{V}(A) = \{val(r) \mid r \in \mathcal{R}(A)\}$.

**Lemma 1.** *For any BCA $A$, we have $r \in \mathcal{R}(A)$ if and only if $r \circ \sigma \in \mathcal{R}(A)$.*

A *control path* in a CA (or BCA) $A$ is a finite sequence $q_0 q_1 \dots q_n$ of control states such that, for all $0 \leq i < n$, there exists a transition rule $q_i \xrightarrow{\varphi_i} q_{i+1}$. A *cycle* is a control path starting and ending in the same control state. An *elementary cycle* is a cycle in which each state appears only once, except the first one that appears twice. A CA (or BCA) is said to be *flat* iff each control state belongs to at most one elementary cycle.

**Decidability and Closure Properties of FBCA.** We consider in the following the class of bi-infinite Büchi counter automata which are flat, whose elementary cycles are labelled with parametric DBM formulae, and the remaining transitions are labelled with PA formulae. Moreover, each transition constraint enforces the values of parameters to remain constant. We call this class FBCA. We prove that the emptiness problem for FBCA is decidable using results of [6,4] and their extensions that can be found in [8].

**Lemma 2.** *The emptiness problem is decidable for the class of FBCA.*

[1] In the early literature [14], a bi-infinite sequence is defined as the equivalence class of all compositions $\beta \circ \sigma^n \circ \sigma^{-m}$ for arbitrary $n, m \in \mathbb{N}$. This is because a bi-infinite sequence remains the same if shifted left or right. For simplicity, we formally distinguish here the bi-infinite sequences $\beta$, $\beta \circ \sigma^n$, and $\beta \circ \sigma^{-n}$ for $n > 0$.

The FBCA class is also effectively closed under union and intersection. However, before proceeding, we need to elucidate the meaning of these operations for CA (BCA). For a valuation $\nu : \mathbf{x} \to \mathbb{Z}$, if $\mathbf{z} \subseteq \mathbf{x}$ is a subset of the counters in $\mathbf{x}$, let $\nu\downarrow_{\mathbf{z}}$ denote the restriction of $\nu$ to the domain $\mathbf{z}$. For some subset $\mathbf{z} \subset \mathbf{x}$ of the counters of $A$ and $s \in \mathcal{V}(A)$, we define the restriction operator on sequences $s\downarrow_{\mathbf{z}} = \ldots val(s(-1))\downarrow_{\mathbf{z}} val(s(0))\downarrow_{\mathbf{z}} val(s(1))\downarrow_{\mathbf{z}} \ldots$, and $\mathcal{V}(A)\downarrow_{\mathbf{z}} = \{s\downarrow_{\mathbf{z}} \mid s \in \mathcal{V}(A)\}$. Symmetrically, for $\mathbf{z} \supset \mathbf{x}$, we define the extension operator on sequences $\mathcal{V}(A)\uparrow_{\mathbf{z}} = \{v \in {}^{\omega}(\mathbf{z} \mapsto \mathbb{Z})^{\omega} \mid v\downarrow_{\mathbf{x}} \in \mathcal{V}(A)\}$.

A class of counter automata is said to be *closed* under union and intersection if there exist operations $\uplus$ and $\otimes$ such that, for any two FBCA $A_i = \langle \mathbf{x_i}, Q_i, L_i, R_i, \rightarrow_i \rangle$, $i = 1,2$, we have that $\mathcal{V}(A_1 \uplus A_2) = \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cup \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$ and $\mathcal{V}(A_1 \otimes A_2) = \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$, respectively. The class is said to be *effectively* closed under union and intersection if these operators are effectively computable.

**Proposition 1.** *Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ be a FBCA. Let $A^c = \langle \mathbf{x}, Q, L^c, R^c, \rightarrow \rangle$ be the FBCA such that (1) for all $q \in L$ and $q' \in Q$, $q'$ belongs to the same elementary cycle as $q$ iff $q' \in L^c$, (2) for all $q \in R$ and $q' \in Q$, $q'$ belongs to the same elementary cycle as $q$ iff $q' \in R^c$. Then we have that $\mathcal{R}(A) = \mathcal{R}(A^c)$.*

Assuming w.l.o.g. that $Q_1 \cap Q_2 \neq \emptyset$, the union is defined as $A_1 \uplus A_2 = \langle \mathbf{x_1} \cup \mathbf{x_2}, Q_1 \cup Q_2, L_1 \cup L_2, R_1 \cup R_2, \rightarrow_1 \cup \rightarrow_2 \rangle$. The product is defined as $A_1 \otimes A_2 = \langle \mathbf{x_1} \cup \mathbf{x_2}, Q_1 \times Q_2, L_1^c \times L_2^c, R_1^c \times R_2^c, \rightarrow \rangle$ where $\rightarrow$ is as follows: $(q_1, q_1') \xrightarrow{\varphi_1 \wedge \varphi_2} (q_2, q_2')$ iff $q_1 \xrightarrow{\varphi_1} q_2$ is a transition rule of $A_1$ and $q_1' \xrightarrow{\varphi_2} q_2'$ is a transition rule of $A_2$. Here, $L_i^c$ and $R_i^c$ denote the extended left-accepting and right-accepting sets of $A_i$ from Proposition 1 for $i = 1,2$.

**Lemma 3.** *The class of FBCA is effectively closed under union and intersection.*

## 3 A Logic for Integer Arrays

In this section we define the Logic of Integer Arrays (**LIA**) that we use to specify properties of programs handling arrays of integers.

**Syntax.** We consider three types of variables. The *array-bound variables* $(k,l)$ appear within the so-called array-bound terms. These terms can be used to define intervals of indices and also as static references inside arrays. The *index* $(i,j)$ and *array* $(a,b)$ *variables* are used to build array terms. Fig. 1 shows the syntax of the logic **LIA**. We use the symbol $\top$ to denote the boolean value *true*. In the following, we will use $f \leq i \leq g$ instead of $f \leq i \wedge i \leq g$, $i < f$ instead of $i \leq f - 1$, and $i = f$ instead of $f \leq i \leq f$. Intuitively, our logic is the set of existentially quantified boolean combinations of:

1. Array formulae of the form $\forall \mathbf{i} \,.\, \varphi(\mathbf{k}, \mathbf{i}) \rightarrow \psi(\mathbf{k}, \mathbf{i}, \mathbf{a})$ where $\mathbf{k}$ is a set of array-bound variables, $\mathbf{i}$ is a set of index variables, $\mathbf{a}$ is a set of array variables, $\varphi$ is an arithmetic formula on index variables, and $\psi$ is an arithmetic formula on array terms. In particular, $\psi$ is a DBM formula, and $\varphi$ is composed of atomic propositions of the form either $f \leq i$, $i \leq f$, $i - j \leq n$, or $i \equiv_s t$ where $f$ is a linear combination of array-bound variables, $n \in \mathbb{Z}$, and $0 \leq t < s$. Both $\mathbf{k}$ and $\mathbf{a}$ variables are free in the array formulae, but they can be existentially quantified at the top-most level.
2. PA formulae on array-bound variables.

| | | | |
|---|---|---|---|
| $n, m, s, t \ldots$ | $\in$ | $\mathbb{Z}$ | constants $(0 \le t < s)$ |
| $k, l, \ldots$ | $\in$ | $BVar$ | array-bound variables |
| $i, j, \ldots$ | $\in$ | $IVar$ | index variables |
| $a, b, \ldots$ | $\in$ | $AVar$ | array variables |
| $B$ | $:=$ | $n \mid k \mid B+B \mid B-B$ | array-bound terms |
| $I$ | $:=$ | $i \mid I+n$ | index terms |
| $A$ | $:=$ | $a[I] \mid a[B]$ | array terms |
| $G$ | $:=$ | $B \le I \mid I \le B \mid I-I \le n \mid I \equiv_s t \mid G \vee G \mid G \wedge G$ | guard expressions |
| $V$ | $:=$ | $A \le B \mid B \le A \mid A-A \le n \mid V \wedge V$ | value expressions |
| $C$ | $:=$ | $B \le n \mid B \equiv_s t$ | array-bound constraints |
| $P$ | $:=$ | $\top \to V \mid G \to V \mid \forall i \,.\, P$ | array properties |
| $U$ | $:=$ | $P \mid C \mid \neg U \mid U \vee U \mid U \wedge U$ | universal formulae |
| $F$ | $:=$ | $U \mid \exists k \,.\, F \mid \exists a \,.\, F$ | **LIA** formulae |

**Fig. 1.** Syntax of the logic **LIA**

**Examples.** To accustom the reader with the logic, we consider several properties of interest that can be stated about arrays. For instance, a strictly increasing ordering of an array $a$ up to a certain bound is defined as $\exists k\, \forall i \,.\, 0 \le i < k \to a[i] - a[i+1] \le -1$. The fact that the first $k$ elements of an array $a$ are below the first $l$ elements of an array $b$ at distance 5 is defined as $\exists k, l\, \forall i, j \,.\, 0 \le i < k \wedge 0 \le j < l \to a[i] - b[j] \le -5$. Equality of two arrays up to a certain bound can be expressed as $\exists n \forall i \,.\, 0 \le i < n \to a[i] = b[i]$. The use of modulo constraints as guards for indices allows one to express periodic facts, e.g., $\forall i, j \,.\, i \equiv_2 0 \wedge j \equiv_2 1 \to a[i] \le a[j]$ meaning that any value at some even position is less than or equal to any value at some odd position in $a$. In [8], we show that to prove the correctness of an array merging program, such properties are needed.

**Semantics.** The logic **LIA** is interpreted on *both-ways infinite arrays*. This allows us to conveniently deal with out-of-bound reference situations common in programs handling arrays. One can prevent and/or check for out-of-bound references by introducing explicit existentially quantified array-bound variables for array variables. Let $\varphi(\mathbf{k}, \mathbf{a})$ be any **LIA** formula. A *valuation* is a pair of partial functions[2] $\langle \iota, \mu \rangle$ with $\iota : BVar \cup IVar \to \mathbb{Z}_\perp$ associating an integer value with every free integer variable and $\mu : AVar \to {}^{\omega}\mathbb{Z}^{\omega}_{\perp}$ associating a bi-infinite sequence of integers with every array symbol $a \in \mathbf{a}$. The valuation $\iota$ is extended in the standard way to array-bound terms ($\iota(B)$) and index terms ($\iota(I)$). By $I_{\iota,\mu}(A)$, we denote the value of the array term $A$ given by the valuation $\langle \iota, \mu \rangle$. The semantics of a formula $\varphi$ is defined in terms of the forcing relation $\models$ as follows:

$$
\begin{array}{rrcl}
 & \langle \iota, \mu \rangle \models A \le B & \iff & I_{\iota,\mu}(A) \le \iota(B) \\
I_{\iota,\mu}(a[I]) = \mu(a)(\iota(I)) & \langle \iota, \mu \rangle \models A_1 - A_2 \le n & \iff & I_{\iota,\mu}(A_1) - I_{\iota,\mu}(A_2) \le n \\
I_{\iota,\mu}(a[B]) = \mu(a)(\iota(B)) & \langle \iota, \mu \rangle \models \forall i \,.\, G \to V & \iff & \forall\, n \in \mathbb{Z} \,.\, \langle \iota[i \leftarrow n], \mu \rangle \models G \to V \\
 & \langle \iota, \mu \rangle \models \exists a \,.\, \psi & \iff & \exists\, \beta \in {}^{\omega}\mathbb{Z}^{\omega} \,.\, \langle \iota, \mu[a \leftarrow \beta] \rangle \models \psi
\end{array}
$$

For space reasons, we do not give here a full definition. However, the missing rules are standard in first-order arithmetic. A *model* of $\varphi(\mathbf{k}, \mathbf{a})$ is a valuation $\langle \iota, \mu \rangle$ such that

[2] The symbol $\perp$ is used to denote that a partial function is undefined at a given point.

the formula obtained by interpreting each variable $k \in \mathbf{k}$ as $\iota(k)$ and each array variable $a \in \mathbf{a}$ as $\mu(a)$ is logically valid: $\langle \iota, \mu \rangle \models \varphi$. We define $[\![\varphi]\!] = \{\langle \iota, \mu \rangle \mid \langle \iota, \mu \rangle \models \varphi\}$. A formula is *satisfiable* if and only if $[\![\varphi]\!] \neq \emptyset$.

**An Undecidability Result.** The reason behind the restriction that array terms may not occur within disjunctions in value expressions (cf. Fig. 1) is that, without it, the logic becomes undecidable. The essence of the proof is that an array formula $\forall \mathbf{i}.G \rightarrow V_1 \vee \ldots \vee V_n$, for $n > 1$, corresponds to $n$ nested loops in a counter automaton. Undecidability is shown by reduction from the halting problem for 2-counter machines [13].

**Lemma 4.** *The logic obtained by extending* **LIA** *with disjunctions within the value expressions is undecidable.*

Note that having more than one nested loop is a necessary condition for undecidability of 2-counter machines since a flat 2-counter machine would trivially fall into the class of decidable counter machines from [6,4].

## 4 Decidability of the Satisfiability Problem

The idea behind our method for deciding the satisfiability problem for **LIA** is that, for any formula of **LIA**, there exists an FBCA $A_\varphi$ such that $\varphi$ has a model if and only if $A_\varphi$ has an accepting run. More precisely, each array variable in $\varphi$ has a corresponding counter in $A_\varphi$, and given any model of $\varphi$ that associates integer values to all array entries, $A_\varphi$ has a run such that the values of the counters at different points of the run match the values of the array entries at corresponding indices in the model. Since, by Lemma 2, the emptiness problem is decidable for FBCA, this leads to decidability of **LIA**.

In order to build an automaton from a **LIA** formula, we first normalise it into an existentially quantified positive boolean combination of simple array property formulae (cf. Fig. 1). Second, each such array property formula is translated into an FBCA. The final automaton $A_\varphi$ is defined recursively on the structure of the normalised formula with the $\uplus$ and $\otimes$ operators being the counterparts for the $\vee$ and $\wedge$ connectives, respectively.

### 4.1 Normalisation of Formulae

The goal of this step is to transform any formula written using the syntax of Figure 1 into a formula of the following normal form:

$$\exists \mathbf{k} \exists \mathbf{a} \,.\, \bigvee_c \Big( \bigwedge_d \phi_{cd}(\mathbf{a}, \mathbf{k}) \Big) \wedge \theta_c(\mathbf{k}) \tag{NF}$$

where $\mathbf{a}$ is a set of array variables, $\mathbf{k}$ is a set of integer variables, and

- $\theta_d$ is a conjunction of terms of the forms (i) $g(\mathbf{k}) \geq 0$ or (ii) $g(\mathbf{k}) \equiv_s t$ with $g$ being a linear combination of the variables in $\mathbf{k}$ and $0 \leq t < s$,
- $\phi_{cd}$ is a formula of the following forms for $\sim \in \{\leq, \geq\}, m \in \mathbb{N}, 0 \leq t < s, 0 \leq v < u$, $q \in \mathbb{Z}$, and $f_k, g_l, f_k^1, g_l^1, f_k^2, g_l^2$ being linear combinations of array-bound variables:

$$\forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \;\wedge\; \bigwedge_{l=1}^{L} i \leq g_l \;\wedge\; i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \tag{F1}$$

The (F1) formulae bind all values of $a$ in some interval by some linear combination $h$ of variables in $\mathbf{k}$.

$$\forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \wedge \bigwedge_{l=1}^{L} i \leq g_l \wedge i \equiv_s t \rightarrow a[i] - b[i+p] \sim q \tag{F2}$$

Here, $p \in \mathbb{Z}$. The (F2) formulae relate all values of $a$ and $b$ in the same interval such that the distance between the indices of $a$ and $b$, respectively, is constant.

$$\forall i,j \,.\, \textstyle\bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i - j \leq p \wedge i \equiv_s t \wedge j \equiv_u v \rightarrow a[i] - b[j] \sim q \tag{F3}$$

Here, $p \in \mathbb{Z}^\infty$. The (F3) formulae relate all values of $a$ with all values of $b$ within two (possibly equal) intervals. The case when $p = \infty$ corresponds to the situation when no constraint $i - j \leq p$ with $p \in \mathbb{Z}$ is used.

**Lemma 5.** *A formula of* **LIA** *can be equivalently written in the form (NF).*

In the following, we refer to the *matrix* of $\varphi$ as to the formula obtained by forgetting the existential quantifier prefix from the (NF) form of $\varphi$.

### 4.2 Formulae and Constraint Graphs

In [6,4], the set of runs of a flat counter automaton is represented by an unbounded constraint graph. Here, we view the models of a formula as a constraint graph both left- and right-infinite. These constraint graphs are then seen as executions of FBCA, relating in this way models of formulae to runs of automata.

Let $\varphi(\mathbf{k}, \mathbf{a})$ be a formula of type (F1)-(F3), and $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ a valuation of its array-bound variables $\mathbf{k}$. For the rest of this section, we fix the valuation $\iota$, and we denote by $\varphi_\iota$ the formula obtained from $\varphi$ by replacing each occurrence of $k \in \mathbf{k}$ by the value $\iota(k)$.

The formula $\varphi_\iota$ can thus be represented by a weighted directed graph $G_{\iota,\varphi}$ in which each node $(a,n)$ represents the array entry $a[n]$ for some $a \in \mathbf{a}$ and $n \in \mathbb{Z}$, and there is a path of weight $w$ between nodes $(a,n)$ and $(b,m)$ iff the constraint $a[n] - b[m] \leq w$ is implied by $\varphi_\iota$. In the next section, we will show that these graphs are in a one-to-one correspondence with the accepting runs of an FBCA.

In order to build the constraint graph of a formula, one needs to pay attention to the following issue. Consider, e.g., the formula $\forall i,j . i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$. The constraint graph of this formula needs to have a path of weight 5 between, e.g., $a[0]$ and $b[1]$, $a[0]$ and $b[3]$, $a[0]$ and $b[5]$, etc. As one can easily notice, the span of such paths is potentially unbounded. Since we would like this graph to represent a computation of a flat counter automaton, it is essential to define it as a sequence composed of (a possibly unbounded number of) repetitions of a finite number of (finite) sub-graphs (see, e.g., Fig. 2(a) or Fig. 2(b)). To this end, we introduce intermediary nodes which are connected between themselves with 0 arcs such that, for each non-local constraint of the form $a[n] - b[m] \leq w$ where $|n - m|$ can be arbitrarily large, there exists exactly one path of weight $w$ through these nodes. E.g., in Fig. 2(a), there is a path

$(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \ldots \xrightarrow{0} (t_\varphi,1) \xrightarrow{0} (b,1)$ for the constraint $a[0]-b[1] \leq 5$, another path $(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \ldots \xrightarrow{0} (t_\varphi,3) \xrightarrow{0} (b,3)$ for the constraint $a[0]-b[3] \leq 5$, etc.

Formally, the constraint graph $G_{\iota,\varphi} = \langle V, E \rangle$ of a formula $\varphi$ of type (F1)-(F3) is defined as follows: The set of vertices is $V = (\mathcal{A} \cup \mathcal{T} \cup \{\zeta\}) \times \mathbb{Z}$. Here, $\mathcal{A} = \{a\}$ for (F1) formulae, and $\mathcal{A} = \{a,b\}$ for (F2)-(F3) formulae, with $a$ or $a,b$ being the arrays that appear in $\varphi$ of type (F1) or (F2)-(F3), respectively. Next, $\mathcal{T} = \emptyset$ for (F1)-(F2) formulae, and $\mathcal{T} = \{t_\varphi\}$ for (F3) formulae where $t_\varphi$ is a unique auxiliary symbol (track) associated with each formula $\varphi$ of type (F3). Finally, $\zeta$ is a special shared symbol (zero track). The set of edges $E$ is defined based on the type (F1)-(F3) of $\varphi$. For space reasons, we give here only the definitions for formulae of type (F3), which are the most interesting. Formulae (F1) and (F2) are treated in [8]. In general, for all types of formulae, we have:

$$E \supset \{(\zeta,k) \xrightarrow{0} (\zeta,k+1) \mid k \in \mathbb{Z}\} \cup \{(\zeta,k+1) \xrightarrow{0} (\zeta,k) \mid k \in \mathbb{Z}\}$$

i.e., the value of the zero track stays constant.

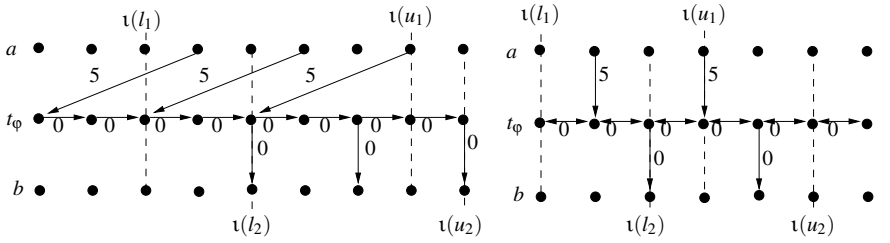
**Constraint graphs for (F3) formulae.** Let $\varphi$ be the formula below where $0 \leq s < t$, $0 \leq u < v$, $p \in \mathbb{Z}^\infty$, $q \in \mathbb{Z}$, and $f_k^1$, $g_l^1$, $f_k^2$, $g_l^2$ are linear combinations of array-bound variables:

$$\forall i,j \,.\, \underbrace{\bigwedge_{k=1}^{\kappa_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{\iota_1} i \leq g_l^1 \wedge i \equiv_s t}_{\phi^1} \wedge \underbrace{\bigwedge_{k=1}^{\kappa_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{\iota_2} j \leq g_l^2 \wedge j \equiv_u v}_{\phi^2} \wedge i - j \leq p \rightarrow a[i] - b[j] \sim q$$

Let $\phi^1(i,\mathbf{k})$ and $\phi^2(j,\mathbf{k})$ be the subformulae defining the ranges of $i$ and $j$, respectively, and $\mathcal{P}_\iota^1 = \{n \in \mathbb{Z} \mid \models \phi_\iota^1[n/i]\}$ and $\mathcal{P}_\iota^2 = \{n \in \mathbb{Z} \mid \models \phi_\iota^2[n/j]\}$ be these ranges under the valuation $\iota$. Let $T_\leq = \{(t_\varphi,k) \xrightarrow{0} (t_\varphi,k+1) \mid k \in \mathbb{Z} \wedge \exists n \in \mathcal{P}_\iota^1 \; \exists m \in \mathcal{P}_\iota^2 \,.\, n-m \leq p\}$ and $T_\geq = \{(t_\varphi,k) \xrightarrow{0} (t_\varphi,k-1) \mid k \in \mathbb{Z} \wedge \exists n \in \mathcal{P}_\iota^1 \; \exists m \in \mathcal{P}_\iota^2 \,.\, n-m \geq p\}$. Note that $T_\leq$ and $T_\geq$ are empty if the precondition of $\varphi$ is not satisfiable. The set of edges $E$ is defined by the following case split:

1. If $p < \infty$, we consider two cases based on the direction of $a[i]-b[j] \sim q$:
   (a) for $a[i]-b[j] \leq q$, we have (Fig. 2(a)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi,k-p) \mid k \in \mathcal{P}_\iota^1\} \cup \{(t_\varphi,k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}_\iota^2\} \cup T_\leq$
   (b) for $a[i]-b[j] \geq q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi,k+p) \mid k \in \mathcal{P}_\iota^2\} \cup \{(t_\varphi,k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}_\iota^1\} \cup T_\geq$
2. If $p = \infty$, we consider again two cases based on the direction of $a[i]-b[j] \sim q$:
   (a) for $a[i]-b[j] \leq q$, we have (Fig. 2(b)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi,k) \mid k \in \mathcal{P}_\iota^1\} \cup \{(t_\varphi,k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}_\iota^2\} \cup T_\leq \cup T_\geq$
   (b) for $a[i]-b[j] \geq q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi,k) \mid k \in \mathcal{P}_\iota^2\} \cup \{(t_\varphi,k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}_\iota^1\} \cup T_\leq \cup T_\geq$

Nothing else is in $E$.




(a) $\forall i, j. l_1 \le i \le u_1 \wedge l_2 \le j \le u_2 \wedge i - j \le 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \le 5$

(b) $\forall i, j. l_1 \le i \le u_1 \wedge l_2 \le j \le u_2 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \le 5$

**Fig. 2.** Examples of constraint graphs for (F3) formulae

**Relating constraint graphs and models of formulae.** We can now prove a correspondence between constraint graphs and models of formulae of the forms (F1)-(F3). Namely, it is the fact that if the vertices of a constraint graph for a formula $\varphi$ can be labelled in a consistent way, then from the labelling, one can extract a model for $\varphi$, and vice versa. This formalises correctness of the construction for constraint graphs using the additional tracks.

Let $\varphi(\mathbf{k}, \mathbf{a})$ be a formula of the forms (F1)-(F3), $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ a valuation of the array-bound variables in $\varphi$, and $G_{\iota,\varphi} = (V, E)$ its corresponding constraint graph. A *labelling* $Lab : V \rightarrow \mathbb{Z}$ of $G_{\iota,\varphi}$ is called *consistent* if and only if (1) for all edges $v_1 \xrightarrow{k} v_2 \in E$, we have $Lab(v_1) - Lab(v_2) \le k$ and (2) $Lab((\zeta, n)) = 0$ for all $n \in \mathbb{Z}$.

**Lemma 6.** *Let $\varphi(\mathbf{k}, \mathbf{a})$ be a formula of the form (F1)-(F3). Then, for all valuations $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ and $\mu : \mathbf{a} \rightarrow {}^{\omega}\mathbb{Z}^{\omega}$, we have that $\langle \iota, \mu \rangle \models \varphi$ if and only if there exists a consistent labelling Lab of $G_{\iota,\varphi}$ such that $\mu(a)(i) = Lab((a, i))$ for all $a \in \mathbf{a}$ and $i \in \mathbb{Z}$.*

### 4.3 From Formulae to Counter Automata

In this section, we describe the construction of an FBCA $A_\varphi$ corresponding to a formula $\varphi$ such that (1) each run of $A_\varphi$ corresponds to a model of $\varphi$, and (2) for each model of $\varphi$, $A_\varphi$ has at least one corresponding run. In this way, we effectively reduce the satisfiability problem for **LIA** to the emptiness problem for FBCA.

The construction of FBCA is by induction on the structure of the formulae. For the rest of this section, let $\varphi$ be a formula, $\mathbf{k}$ the set of array-bound variables in $\varphi$, and $\mathbf{a}$ the set of array variables in $\varphi$, i.e., $FV(\varphi) = \mathbf{k} \cup \mathbf{a}$. Suppose that $\varphi$ is the matrix of a formula in the normal form (NF), i.e., $\varphi : \bigvee_{i \in I} \theta_i(\mathbf{k}) \wedge \bigwedge_{j \in J} \psi_{ij}(\mathbf{k}, \mathbf{a})$ where $\theta_i$ are PA constraints and $\psi_{ij}$ are formulae of types (F1)-(F3). The automaton $A_\varphi$ is defined as $\biguplus_{i \in I} A_{\theta_i} \otimes \bigotimes_{j \in J} A_{\psi_{ij}}$ where $\uplus$ and $\otimes$ are the union and intersection operators on FBCA. The construction of counter automata $A_{\psi_{ij}}$ for the formulae $\psi_{ij}$ of type (F1)-(F3) relies on the definition of the constraint graphs in Section 4.2. Namely, each accepting run of $A_{\psi_{ij}}$ gives a consistent valuation of the constraint graph of $\psi_{ij}$.

**Counter Automata Templates.** To simplify the definition of counter automata, we note that each constraint graph for the basic formulae of type (F1)-(F3) is composed

of *horizontal*, *vertical*, and *diagonal* edges, which are defined in roughly the same way for all types of formulae (cf. Section 4.2). We take advantage of this fact and we start by defining three types of counter automata *templates*, which are subsequently used to define the counter automata for the basic formulae.[3] More precisely, the automata for (F1)-(F3) formulae will be defined as $\otimes$-products of particular instances of the automata templates for the horizontal, vertical, and diagonal edges of the appropriate constraint graphs. In the following definitions, we assume the existence of a special counter $x_\tau$ (tick) incremented by each transition rule, i.e., we suppose that the constraint $x'_\tau = x_\tau + 1$ is implicitly in conjunction with each formula labelling a transition rule. Intuitively, the role of the $x_\tau$ counter is to synchronise all automata composed by the $\otimes$-product on a common current position.

*The template for the horizontal edges.* Let $a$ be an array symbol, $dir \in \{\texttt{left}, \texttt{right}, \texttt{bi}\}$ be a *direction* parameter, and $\phi$ be a formula on array-bound variables. Let $\mathbf{x_k}$ be the set $\{x_k \mid k \in FV(\phi)\}$. We define the template $H(a, dir, \phi) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

- $\mathbf{x} = \{x_a\} \cup \mathbf{x_k}$. These counters will have the same names in all instances of $H$.
- $Q = \{q_L, q_R, p_L, p_R\}$. The control states are required to have fresh names in every instance of $H$. $L = \{q_L, p_L\}$ and $R = \{q_R, p_R\}$.
- $q_L \xrightarrow{\xi} q_L$, $q_R \xrightarrow{\xi} q_R$, $q_L \xrightarrow{\phi(\mathbf{x_k}) \wedge \xi} q_R$, $p_L \xrightarrow{\top} p_L$, $p_R \xrightarrow{\top} p_R$, and $p_L \xrightarrow{\neg\phi(\mathbf{x_k})} p_R$.

In the above, $\phi(\mathbf{x_k})$ is the formula obtained by replacing each occurrence of an array-bound variable $k \in FV(\phi)$ by its corresponding counter $x_k$. The formula $\xi(x_a, x'_a)$ is $x_a - x'_a \leq 0$ if $dir = \texttt{right}$, $x'_a - x_a \leq 0$ if $dir = \texttt{left}$, and $x'_a = x_a$ if $dir = \texttt{bi}$. Moreover, for each transition rule, we assume the conjunction $\bigwedge_{k \in FV(\phi)} x'_k = x_k$ to be added implicitly to the labelling formula, i.e., the value of an $x_k$ counter stays constant throughout a run.

If the formula $\phi$ holds for a given valuation of the parameters $\mathbf{x_k}$, then any accepting run of (any instance of) $H$ visits $q_L$ infinitely often on the left and $q_R$ infinitely often on the right. Otherwise, if $\phi$ does not hold for the given valuation of $\mathbf{x_k}$, the instance automata have a run that goes infinitely often through $p_L$ on the left and through $p_R$ on the right. In this case, the automata do not impose any constraints on $x_a$.

*The template for the diagonal edges.* Let $a, b$ be array symbols, $q \in \mathbb{Z}$, $p, s \in \mathbb{N}^+$, $t \in [0, s-1]$, and $dir \in \{\texttt{left}, \texttt{right}\}$ be a direction parameter. In the following, we refer to the sets $\mathbb{L} = \{l_1, \ldots, l_K\}$ and $\mathbb{U} = \{u_1, \ldots, u_L\}$ of lower and upper bounds, respectively, where $l_i$ and $u_j$ are linear combinations of array-bound variables. Let $\mathbf{x_k} = \{x_k \mid k \in \bigcup_{i=1}^{K} FV(l_i) \cup \bigcup_{j=1}^{L} FV(u_j)\}$. Further, we assume that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ and we deal with the case of $\mathbb{L} \cup \mathbb{U} = \emptyset$ later on. We define the template $D(a, b, p, q, s, t, \mathbb{L}, \mathbb{U}, dir) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

- $\mathbf{x} = \{x_a, x_b\} \cup \mathbf{x_k} \cup \{x_i \mid 1 \leq i < p\}$. The counters $x_a, x_b$, and $\mathbf{x_k}$ will have the same names in all instances of $D$. On the other hand, the counters $x_i$, $1 \leq i < p$, will have fresh names in every instance of $D$. The $x_i$ counters are used for splitting

[3] By a *template*, we mean a class of counter automata which all share the same structure.

diagonal edges that span over more than one position into series of diagonal edges connecting only adjacent positions.[4]

- $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \le i < s\} \cup \{q_i^j \mid 0 \le j < s,\ j+1 \le i < j+p\}$. The control states are required to have fresh names in every instance of $D$. Let $L = \{q_L\} \cup \{q_i \mid 0 \le i < s\}$ and $R = \{q_R\} \cup \{q_i \mid 0 \le i < s\}$.
- $q_L \xrightarrow{\top} q_L$, $q_R \xrightarrow{\top} q_R$, and $q_L \xrightarrow{\neg(\exists i\ .\ \bigwedge_{l\in\mathbb{L}} i \ge l(\mathbf{x_k}) \wedge \bigwedge_{u\in\mathbb{U}} i \le u(\mathbf{x_k}) \wedge i \equiv_s t)} q_R$.
- $q_L \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \ge l(\mathbf{x_k})-1 \wedge (\bigvee_{l\in\mathbb{L}} x_\tau = l(\mathbf{x_k})-1) \wedge x_\tau+1 \equiv_s i} q_i$ for all $0 \le i < s$.
- $q_i \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \ge l(\mathbf{x_k}) \wedge \bigwedge_{u\in\mathbb{U}} x_\tau < u(\mathbf{x_k}) \wedge \xi_i[x_a/x_0, x_b/x_p]} q_{(i+1) \bmod s}$ for all $0 \le i < s$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \wedge x_\tau \equiv_s i \wedge \xi_i[x_a/x_0, x_b/x_p]} q_{i+1}^i$ for all $0 \le i < s$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \wedge x_\tau \equiv_s i \wedge \xi_i[x_a/x_0, x_b/x_p]} q_R$ for all $0 \le i < s$ if $p = 1$.
- $q_i^j \xrightarrow{\xi_i[x_a/x_0, x_b/x_p]} q_{i+1}^j$ for all $0 \le j < s$, $j < i < j+p-1$.
- $q_{j+p-1}^j \xrightarrow{\xi_i[x_a/x_0, x_b/x_p]} q_R$ for all $0 \le j < s$ if $p > 1$.

In the above, $l(\mathbf{x_k})$ and $u(\mathbf{x_k})$ denote the expressions $l$ and $u$ in which each occurrence of an array-bound variable $k$ is replaced by its corresponding parameter $x_k$. As before, for each transition rule, we assume the conjunction $\bigwedge_{k\in FV(\phi)} x_k' = x_k$ to be added implicitly to the labelling formula, i.e., we require that the value of an $x_k$ counter stays constant throughout the run. The formulae $\xi_i$ are defined as follows:

- if $dir = \texttt{right}$, $\xi_i = \bigwedge_{k\in K_i} x_k - x'_{k+1} \le \alpha_k$ for $K_i = \{k \mid 0 \le k < p,\ i \equiv_s k+t\}$, $\alpha_0 = q$, and $\alpha_k = 0$, $k > 0$,
- if $dir = \texttt{left}$, $\xi_i = \bigwedge_{k\in K_i} x'_{k-1} - x_k \le \alpha_k$, $K_i = \{k \mid 1 \le k \le p,\ k+i \equiv_s t\}$, $\alpha_1 = q$, and $\alpha_k = 0$, $k > 1$.

Finally, for the case $\mathbb{L} = \mathbb{U} = \emptyset$, we define any instance of $D(a,b,p,q,s,t,\emptyset,\emptyset,dir)$ to be $A_1 \otimes A_2$ where $A_1$ is an instance of $D(a,b,p,q,s,t,\emptyset,\{0\},dir)$ and $A_2$ is an instance of $D(a,b,p,q,s,t,\{0\},\emptyset,dir)$.

The construction can be understood by considering an accepting run of (any instance of) $D$. Let us consider the case in which there exists a value $i$ in between the bounds that satisfies also the modulo constraint. If this is not the case, there will be an accepting run that takes the transition $q_L \xrightarrow{\neg(\exists i\ .\ \bigwedge_{l\in\mathbb{L}} i \ge l(\mathbf{x_k}) \wedge \bigwedge_{u\in\mathbb{U}} i \le u(\mathbf{x_k}) \wedge i \equiv_s t)} q_R$ exactly once.

Since the run is accepting, it must visit a state from $L$ infinitely often on the left, and a state from $R$ infinitely often on the right. There are three cases: (1) $\mathbb{L} \neq \emptyset$ and $\mathbb{U} \neq \emptyset$, (2) $\mathbb{L} = \emptyset$ and $\mathbb{U} \neq \emptyset$, and (3) $\mathbb{L} \neq \emptyset$ and $\mathbb{U} = \emptyset$. In the case (1), a bi-infinite run will visit $q_L$ infinitely often on the left and $q_R$ infinitely often on the right. Notice that the run

[4] For instance, the constraint $a[i] - b[i+3] \le 5$ can be split to $a[i] - x_1[i+1] \le 5$, $x_1[i+1] - x_2[i+2] \le 0$, and $x_2[i+2] - b[i+3] \le 0$. The constraints for array values of neighbouring indices can then be conveniently expressed by using the current and future values of the appropriate counters (e.g., for our example constraint, $x_a - x'_1 \le 5$, $x_1 - x'_2 \le 0$, and $x_2 - x'_b \le 0$, which of course appear on subsequent transitions of the appropriate FBCA).

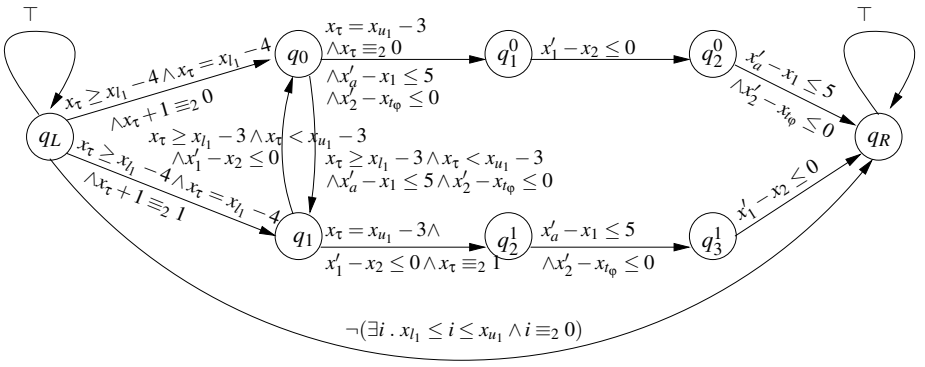



**Fig. 3.** The FBCA for the diagonal edges in the formula $\varphi : \forall i, j . l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$ from Fig. 2(a) obtained as $D(a, t_\varphi, 3, 5, 2, 0-3, \{l_1 - 3\}, \{u_1 - 3\}, \texttt{left})$. To understand the formula $\xi_0$ on the transition from $q_0$ to $q_1$, note that the constraint $i \equiv_s k + t$ in the definition of the set $K_0$ instantiates to $0 \equiv_2 k - 3$, and hence $K_0 = \{1, 3\}$. A similar reasoning applies for the other transitions.

cannot visit the loop $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ infinitely often due to the presence of both lower and upper bounds on $x_\tau$. In the case (2), the run cannot take any of the transitions $q_L \rightarrow q_i$, $0 \leq i < s$, due to the emptiness of $\mathbb{L}$, which makes the guard unsatisfiable. Hence, the only possibility for an accepting bi-infinite run is to visit the states $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ infinitely often on the left. Due to the presence of the upper bound on $x_\tau$, the run cannot stay forever inside this loop and must exit via one of the $q_i \rightarrow q_{i+1}^i$ (or $q_i \rightarrow q_R$ for $p = 1$) transitions, getting trapped into $q_R$ on the right. Case (3) is symmetric to (2).

Note that, in all cases, due to the modulo tests on $x_\tau$ in the entry and exit of the main loop $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ on any accepting run, whenever a state $q_i$, $0 \leq i < s$, is visited, the value of the $x_\tau$ counter must equal $i$ modulo $s$. Note also that the role of the $q_i^j$ states is to describe constraints corresponding to edges that start inside the given interval bounds and lead above its upper bound (or vice versa). The number of such edges is bounded. We do not use the same construction at the beginning of the interval as the templates are applied such that none of the edges represented goes below the lower bounds.

*Template for the vertical edges.* Let $a, b$ be array symbols, $q$ a linear combination of array-bound variables, $p, s \in \mathbb{N}^+$, and $t \in [0, s-1]$. We again refer to the sets $\mathbb{L} = \{l_1, \ldots, l_K\}$ and $\mathbb{U} = \{u_1, \ldots, u_L\}$ of lower and upper bounds, respectively, where $l_i$ and $u_j$ are linear combinations of array-bound variables. Also, let $\mathbf{x_k} = \{x_k \mid k \in \bigcup_{i=1}^{K} FV(l_i) \cup \bigcup_{j=1}^{L} FV(u_j)\}$. Further, we assume that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ and we deal with the case of $\mathbb{L} \cup \mathbb{U} = \emptyset$ later on. We define the template $V(a, b, p, q, s, t, \mathbb{L}, \mathbb{U}) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

- $\mathbf{x} = \{x_a, x_b\} \cup \mathbf{x_k}$. The counters $x_a, x_b$, $\mathbf{x_k}$ have the same names in all instances of $V$.
- $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\}$. The control states are required to have fresh names in every instance of $V$. $L = \{q_L\} \cup \{q_i \mid 0 \leq i < s\}$ and $R = \{q_R\} \cup \{q_i \mid 0 \leq i \leq s\}$.
- $q_L \xrightarrow{\top} q_L$, $q_R \xrightarrow{\top} q_R$, and $q_L \xrightarrow{\neg(\exists i \,.\, \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k}) \,\wedge\, i \equiv_s t)} q_R$.

- $q_L \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k})-1 \,\wedge\, \bigvee_{l\in\mathbb{L}} x_\tau+1=l(\mathbf{x_k}) \,\wedge\, x_\tau+1\equiv_s i} q_i$, $0 \leq i < s$.
- $q_i \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u\in\mathbb{U}} x_\tau < u(\mathbf{x_k}) \,\wedge\, x_a - x_b \leq q(\mathbf{x_k})} q_{(i+1) \mod s}$, $0 \leq i < s$ and $i \equiv_s t$.
- $q_i \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u\in\mathbb{U}} x_\tau < u(\mathbf{x_k})} q_{(i+1) \mod s}$, $0 \leq i < s$ and $i \not\equiv_s t$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \,\wedge\, x_\tau\equiv_s i \,\wedge\, x_a - x_b \leq q(\mathbf{x_k})} q_R$, $0 \leq i < s$ and $i \equiv_s t$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \,\wedge\, x_\tau\equiv_s i} q_R$, $0 \leq i < s$ and $i \not\equiv_s t$.

In the above, $l(\mathbf{x_k})$, $u(\mathbf{x_k})$, and $q(\mathbf{x_k})$ denote the expressions $l$, $u$, and $q$ where each occurrence of an array-bound variable $k$ is replaced by the parameter $x_k$. As before, we assume that for each transition rule the conjunction $\bigwedge_{k\in FV(\phi)} x'_k = x_k$ is added implicitly to the labelling formula, i.e., the value of an $x_k$ counter stays constant throughout the run. Finally, if $\mathbb{L} = \mathbb{U} = \emptyset$, we define any instance of $V(a,b,p,q,s,t,\emptyset,\emptyset)$ as $A_1 \otimes A_2$ where $A_1$ is an instance of $V(a,b,p,q,s,t,\emptyset,\{0\})$ and $A_2$ is an instance of $V(a,b,p,q,s,t,\{0\},\emptyset)$. The intuition behind the construction of $V$ is similar to the one of $D$.

### 4.4 Counter Automata for Basic Formulae

We are now ready to define the construction of FBCA for the basic formulae. This is done by composing instances of templates using the $\otimes$ operator for intersection (cf. Section 2). For space reasons, we only give here the construction of the FBCA for (F3) formulae. The formulae of type (F1), (F2), and PA constraints on array-bound variables are treated analogously in [8]. Let $\varphi$ be an (F3)-type formula

$$\forall i,j\,.\; \underbrace{\bigwedge_{k=1}^{\kappa_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{\kappa_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i-j \leq p \wedge i \equiv_s t \wedge j \equiv_u v}_{\phi} \rightarrow a[i] - b[j] \sim q$$

where $0 \leq s < t$ and $0 \leq u < v$. Let $\mathbb{L}_i = \{f_1^i, \ldots, f_{K_i}^i\}$ and $\mathbb{U}_i = \{g_1^i, \ldots, g_{L_i}^i\}$ for $i = 1,2$, respectively. By $\phi$, we denote the precondition of $\varphi$. The automaton $A_\varphi$ is defined as $A_\varphi = A_1 \otimes A_2 \otimes A_3$ where $A_1, A_2, A_3$ are instantiated according to Table 1.

### 4.5 Assembling Automata for Entire Normalised Formulae

Given a formula $\varphi(\mathbf{k},\mathbf{a})$ which is a positive boolean combination of formulae of types (F1)-(F3) and PA constraints on the array-bound variables $\mathbf{k}$, let $A_\varphi$ be the automaton defined inductively on the structure of $\varphi$ as follows:

- if $\varphi$ is of type (F1)-(F3), or a PA constraint on $\mathbf{k}$, then $A_\varphi$ is as in Section 4.4,
- if $\varphi = \psi_1 \wedge \psi_2$, then $A_\varphi = A_{\psi_1} \otimes A_{\psi_2}$,
- if $\varphi = \psi_1 \vee \psi_2$, then $A_\varphi = A_{\psi_1} \uplus A_{\psi_2}$.

Let $r \in \mathcal{R}(A_\varphi)$ be an accepting run of $A_\varphi$ and $\delta(r) = val(r(0))(x_\tau)$ be the value of the $x_\tau$ (tick) counter at position 0 on $r$. We denote by $\eta(r) = r \circ \sigma^{-\delta(r)}$ the *centered run* obtained from $r$ by shifting it such that the value of $x_\tau$ at position 0 is also 0. By Lemma 1, $r$ is an accepting run of $A_\varphi$ if and only if $\eta(r)$ is. Notice that $r$ induces the

**Table 1.** The instantiation table for (F3) formulae. Note that in some lines, we shift the original bounds appearing in the formula in order to be able to re-use the prepared templates that do not explicitly deal with edges leaving from within the given bounds and going below the lower bound. Due to the way the templates are constructed, the shifting preserves the semantics of the formula—instead of edges going below the lower bound of a certain interval, we obtain the same edges just going above the upper bound of the shifted interval, which our templates are prepared for. Given a set of integers $S$ and an integer $p$, we use the notation $S+p$ for $\{s+p \mid s \in S\}$.

| p | $\sim$ | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|---|
| $\infty$ | $\leq$ | $V(a,t_\varphi,q,s,t,\mathbb{L}_1,\mathbb{U}_1)$ | $H(t_\varphi,\mathtt{bi},\exists i,j.\phi)$ | $V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$ |
| $\infty$ | $\geq$ | $V(b,t_\varphi,-q,u,v,\mathbb{L}_2,\mathbb{U}_2)$ | $H(t_\varphi,\mathtt{bi},\exists i,j.\phi)$ | $V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$ |
| 0 | $\leq$ | $V(a,t_\varphi,q,s,t,\mathbb{L}_1,\mathbb{U}_1)$ | $H(t_\varphi,\mathtt{right},\exists i,j.\phi)$ | $V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$ |
| 0 | $\geq$ | $V(b,t_\varphi,-q,u,v,\mathbb{L}_2,\mathbb{U}_2)$ | $H(t_\varphi,\mathtt{left},\exists i,j.\phi)$ | $V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$ |
| $>0$ | $\leq$ | $D(a,t_\varphi,p,q,s,t-p,\mathbb{L}_1-p,\mathbb{U}_1-p,\mathtt{left})$ | $H(t_\varphi,\mathtt{right},\exists i,j.\phi)$ | $V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$ |
| $>0$ | $\geq$ | $D(b,t_\varphi,p,-q,u,v,\mathbb{L}_2,\mathbb{U}_2,\mathtt{right})$ | $H(t_\varphi,\mathtt{left},\exists i,j.\phi)$ | $V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$ |
| $<0$ | $\leq$ | $D(a,t_\varphi,-p,q,s,t,\mathbb{L}_1,\mathbb{U}_1,\mathtt{right})$ | $H(t_\varphi,\mathtt{right},\exists i,j.\phi)$ | $V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$ |
| $<0$ | $\geq$ | $D(b,t_\varphi,-p,-q,u,v+p,\mathbb{L}_2+p,\mathbb{U}_2+p,\mathtt{left})$ | $H(t_\varphi,\mathtt{left},\exists i,j.\phi)$ | $V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$ |

following valuations on $\mathbf{k}$ and $\mathbf{a}$, respectively: $\iota_r(k) = val(\eta(r)(0))(x_k)$ for all $k \in \mathbf{k}$, and $\mu_r(a)(i) = val(\eta(r)(i))(x_a)$ for all $a \in \mathbf{a}$ and $i \in \mathbb{Z}$.

For an arbitrary valuation $\nu \in \mathcal{V}(A_\varphi)$, there exists $r \in \mathcal{R}(A_\varphi)$ such that $\nu = val(r)$. Let $M_\varphi(\nu) = \langle \iota_r, \mu_r \rangle$ be the valuation of the free variables in $\varphi$ that corresponds to $r$. One can see now that $M_\varphi$ defines a function $M_\varphi : \mathcal{V}(A_\varphi) \to (\mathbf{k} \mapsto \mathbb{Z}) \times (\mathbf{a} \mapsto {}^{\omega}\mathbb{Z}^{\omega})$.[5]

**Theorem 1.** *Let $\varphi(\mathbf{k},\mathbf{a})$ be a positive boolean combination of formulae of types (F1)-(F3) and PA constraints on the array-bound variables $\mathbf{k}$, and $A_\varphi$ be the automaton defined in the previous. Then, $M_\varphi(\mathcal{V}(A_\varphi)) = [\![\varphi]\!]$.*

The proof is by induction on the structure of $\varphi$. For the base case, we use the correspondence between models and constraint graphs of formulae (F1)-(F3) (Lemma 6). The inductive step follows as a consequence of the fact that the class of FBCA is closed under union and intersection (Lemma 3). The main result of the paper is the following:

**Corollary 1.** *The logic* **LIA** *is decidable.*

The proof of Corollary 1 uses the normalisation step (cf. Lemma 5) to rewrite any formula of **LIA** into the form (NF) and applies Theorem 1 to the matrix of the formula (i.e., the formula obtained by skipping the existential quantifier prefix).

## 5 Conclusions and Future Work

We have presented a new decidable logic for reasoning about properties of programs with integer arrays. This logic allows one to relate adjacent array values as well as to

[5] By definition, for each $\nu \in \mathcal{V}(A_\varphi)$, there exist valuations $\iota_r$ and $\mu_r$, so $M_\varphi$ is defined for all $\nu \in \mathcal{V}(A_\varphi)$. Let $r_1, r_2 \in \mathcal{R}(A_\varphi)$ be two runs such that $val(r_1) = val(r_2) = \nu$. We have $\delta(r_1) = \delta(r_2)$, therefore $\eta(r_1) = \eta(r_2)$, which leads to $\iota_{r_1} = \iota_{r_2}$ and $\mu_{r_1} = \mu_{r_2}$.

express periodic facts relating all values situated at equidistant positions. We have established decidability of this logic by following the automata-theoretic approach. To this end, we have defined a new class of Büchi automata with counters, for which emptiness is decidable. We translate each formula into a corresponding automaton of this kind and transform deciding satisfiability of the formula to deciding emptiness of the automaton.

Future work will include the study of the complexity of our decision procedure and its implementation. We furthermore plan to develop invariant generation methods in order to give automatic correctness proofs for programs with integer arrays.

## References


1. Armando, A., Ranise, S., Rusinowitch, M.: Uniform Derivation of Decision Procedures by Superposition. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, p. 2001. Springer, Heidelberg (2001)
2. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized Verification with Automatically Computed Inductive Assertions. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, Springer, Heidelberg (2001)
3. Bouajjani, A., Jurski, Y., Sighireanu, M.: A Generic Framework for Reasoning About Dynamic Networks of Infinite-State Processes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, Springer, Heidelberg (2007)
4. Bozga, M., Iosif, R., Lakhnech, Y.: Flat Parametric Counter Automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, Springer, Heidelberg (2006)
5. Bradley, A.R., Manna, Z., Sipma, H.B.: What 's Decidable About Arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, Springer, Heidelberg (2005)
6. Comon, H., Jurski, Y.: Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, Springer, Heidelberg (1998)
7. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decision Procedures for Extensions of the Theory of Arrays. Annals of Mathematics and Artificial Intelligence 50 (2007)
8. Habermehl, P., Iosif, R., Vojnar, T.: What else is decidable about integer arrays? Technical Report TR-2007-8, Verimag (2007)
9. Jaffar, J.: Presburger Arithmetic with Array Segments. Inform. Proc. Letters 12 (1981)
10. King, J.: A Program Verifier. PhD thesis, Carnegie Mellon University (1969)
11. Mateti, P.: A Decision Procedure for the Correctness of a Class of Programs. Journal of the ACM 28(2) (1980)
12. McCarthy, J.: Towards a Mathematical Science of Computation. In: IFIP Congress (1962)
13. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Inc., Englewood Cliffs (1967)
14. Nivat, M., Perrin, D.: Ensembles reconnaissables de mots biinfinis. Canad. J. Math. 38, 513–537 (1986)
15. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves, Warsaw, Poland, pp. 92–101 (1929)
16. Stump, A., Barrett, C.W., Dill, D.L., Levitt, J.R.: A Decision Procedure for an Extensional Theory of Arrays. In: Proc. of LICS 2001 (2001)
17. Suzuki, N., Jefferson, D.: Verification Decidability of Presburger Array Programs. Journal of the ACM 27(1) (1980)
18. Thomas, W.: Automata on Infinite Objects. In: Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, Elsevier, Amsterdam (1990)

# Model Checking Freeze LTL over One-Counter Automata*

Stéphane Demri[1], Ranko Lazić[3], and Arnaud Sangnier[1,2]

[1] LSV, ENS Cachan, CNRS, INRIA
[2] EDF R&D
[3] Department of Computer Science, University of Warwick, UK

**Abstract.** We study complexity issues related to the model-checking problem for LTL with registers (a.k.a. freeze LTL) over one-counter automata. We consider several classes of one-counter automata (mainly deterministic vs. nondeterministic) and several syntactic fragments (restriction on the number of registers and on the use of propositional variables for control locations). The logic has the ability to store a counter value and to test it later against the current counter value. By introducing a non-trivial abstraction on counter values, we show that model checking LTL with registers over deterministic one-counter automata is PSPACE-complete with infinite accepting runs. By constrast, we prove that model checking LTL with registers over nondeterministic one-counter automata is $\Sigma_1^1$-complete [resp. $\Sigma_1^0$-complete] in the infinitary [resp. finitary] case even if only one register is used and with no propositional variable. This makes a difference with the facts that several verification problems for one-counter automata are known to be decidable with relatively low complexity, and that finitary satisfiability for LTL with a unique register is decidable. Our results pave the way for model-checking LTL with registers over other classes of operational models, such as reversal-bounded counter machines and deterministic pushdown systems.

## 1 Introduction

*Logics for data words and trees.* Data words are sequences in which each position is labelled by a letter from a finite alphabet and by another letter from an infinite alphabet (the datum). This fundamental and simple model captures the timed words accepted by timed automata [1], and its extension to trees is useful to model XML documents with values, see e.g. [4,15]. In order to really speak about data, known logical formalisms for data words/trees contain a mechanism that stores a value and tests it later against other values, see e.g. [5,9]. This is a powerful feature shared by other memoryful temporal logics [18,16]. However, the satisfiability problem for these logics becomes easily undecidable even when stored data can be tested only for equality. For instance, first-order logic for data words restricted to three individual variables is undecidable [5], whereas LTL with registers (also known as freeze LTL) restricted to a single register is undecidable over infinite data words [9]. By contrast, decidable fragments of the satisfiability problems have been found in [5,10,19] either by imposing syntactic restrictions

* Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 490–504, 2008.

(bound the number of registers, constrain the polarity of temporal formulae, etc.) or by considering subclasses of data words (finiteness for example). Similar phenomena occur with metric temporal logics and timed words [22,23]. A key point for all these logical formalisms is the ability to store a value from an infinite alphabet, which is a feature also present in models of register automata, see e.g [7,21,25]. However, the storing mechanism has a long tradition (apart from its ubiquity in programming languages) since it appeared for instance in real-time logics [2] (the data are time values) and in so-called hybrid logics (the data are node addresses), see an early undecidability result with reference pointers in [13]. Meaningful restrictions for hybrid logics can also lead to decidable fragments, see e.g. [24].

*Our motivations.* In this paper, our main motivation is to analyze the effects of adding a binding mechanism with registers to specify runs of operational models such as pushdown systems and counter automata. The registers are simple means to compare data values at different points of the execution. Indeed, runs can be naturally viewed as data words: for example, the finite alphabet is the set of locations and the infinite alphabet is the set of data values (natural numbers, stacks, etc.). To do so, we enrich an ubiquitous logical formalism for model-checking techniques, namely linear-time temporal logic LTL, with registers. Even though this was the initial motivation to introduce LTL with registers in [10], most decision problems considered in [10,19,9] are essentially oriented towards satisfiability. In this paper, we focus on the following type of model-checking problem: given a set of runs generated by an operational model, more precisely by a one-counter automaton, and a formula from LTL with registers, is there a run satisfying the given formula? In our context, it will become clear that the extension with two counters is undecidable. It is not difficult to show that this model-checking problem differs from those considered in [19,10] and are of a different nature from those for hybrid logics investigated in [12,27]. However, since two consecutive counter values in a run are ruled by the set of transitions, constraints on data that are helpful to get fine-tuned undecidability proofs for satisfiability problems in [10,9] may not be allowed on runs. This is precisely what we want to understand in this work. Like in [6], LTL with registers makes sense to specify and reason about configurations of operational models, precisely counter systems.

*Our contribution.* We study complexity issues related to the model-checking problem for LTL with registers over one-counter automata that are simple operational models but our undecidability results can be obviously lifted to pushdown systems when registers store the stack value. Moreover, in order to determine borderlines for decidability, we also present results for deterministic one-counter models that are less powerful but remain interesting when they are viewed as a means to specify an infinite path on which model checking is performed, see analogous issues in [20].

We consider several classes of one-counter automata (deterministic and nondeterministic) and several fragments by restricting the use of registers or the use of letters from the finite alphabet. Moreover, we distinguish finite accepting runs from infinite ones as data words. Unlike several results from [22,23,9,19], the decidability status of the model checking does not depend on the fact that we consider finite data words instead of infinite ones. In this paper, we present the following results.

– Model checking LTL with registers over deterministic one-counter automata is PSPACE-complete (see Sect. 3.2). PSPACE-hardness is established by reducing QBF and it also holds when no letters from the finite alphabet are used in formulae. When the number of registers is bounded, the problem can be solved in polynomial time. In order to get these complexity upper bounds, we introduce an abstraction on counter values even though the counter values may not be bounded along the unique run of the deterministic automata. This makes a substantial difference with [20] in which no data values are considered, but still our problem amounts to model checking a path specified by a deterministic one-counter automaton.
– Model checking LTL with registers over nondeterministic one-counter automata restricted to a unique register and without alphabet is $\Sigma_1^1$-complete in the infinitary case by reducing the recurrence problem for Minsky machines (see Sect. 4). In the finitary case, the problem is shown $\Sigma_1^0$-complete by reducing the halting problem for Minsky machines. These results are quite surprising since several verification problems for one-counter automata are known to be decidable with relatively low complexity [14,26]. Moreover, finitary satisfiability for LTL with one register is decidable [9] even though with nonprimitive complexity.

Because of lack of space, omitted proofs can be found in [11].

## 2 Preliminaries

### 2.1 One-Counter Automaton

Let us recall standard definitions and notations about our operational models. A one-counter automaton is a tuple $\mathcal{A} = \langle Q, q_I, \delta, F\rangle$ where $Q$ is a finite set of locations, $q_I \in Q$ is the initial location, $F \subseteq Q$ is the set of accepting locations and $\delta \subseteq Q \times L \times Q$ is the transition relation over the instruction set $L = \{\texttt{inc}, \texttt{dec}, \texttt{ifzero}\}$. A counter valuation $v$ is an element of $\mathbb{N}$ and a configuration of $\mathcal{A}$ is a pair in $Q \times \mathbb{N}$. The initial configuration is the pair $\langle q_I, 0\rangle$. As usual, a one-counter automaton $\mathcal{A}$ induces a (possibly infinite) transition system $\langle Q \times \mathbb{N}, \rightarrow\rangle$ such that $\langle q, n\rangle \rightarrow \langle q', n'\rangle$ iff one of the conditions below holds true: (1) $\langle q, \texttt{inc}, q'\rangle \in \delta$ and $n' = n+1$, (2) $\langle q, \texttt{dec}, q'\rangle \in \delta$ and $n' = n-1$ (and $n' \in \mathbb{N}$), (3) $\langle q, \texttt{ifzero}, q'\rangle \in \delta$ and $n = n' = 0$. A finite [resp. infinite] *run* $\rho$ is a finite [resp. infinite] sequence $\rho = \langle q_0, n_0\rangle \rightarrow \langle q_1, n_1\rangle \rightarrow \cdots$ where $\langle q_0, n_0\rangle$ is the initial configuration. A finite run is *accepting* iff it ends with an accepting location. An infinite run $\rho$ is accepting iff it contains an accepting location infinitely often (Büchi acceptance condition).

A one-counter automaton $\mathcal{A}$ is *deterministic* whenever it corresponds to a deterministic one-counter Minsky machine: for every location $q$, either $\mathcal{A}$ has a unique transition from $q$ incrementing the counter, or $\mathcal{A}$ has exactly two transitions from $q$, one with instruction `ifzero` and the other one with instruction `dec`, or $\mathcal{A}$ has no transition from $q$ (not present in original deterministic Minsky machines). In the transition system induced by any deterministic one-counter automaton, each configuration has at most one successor. One-counter automata in full generality are understood as *nondeterministic* one-counter automata.

### 2.2 LTL over Data Words

Formulae of the logic $\mathrm{LTL}^{\downarrow,\Sigma}$ where $\Sigma$ is a finite alphabet are defined as follows:

$$\phi ::= a \mid \uparrow_r \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathtt{U} \phi \mid \mathtt{X}\phi \mid \downarrow_r \phi$$

where $a \in \Sigma$ and $r$ ranges over $\mathbb{N} \setminus \{0\}$. We write $\mathrm{LTL}^{\downarrow}$ to denote LTL with registers for some unspecified finite alphabet. An occurrence of $\uparrow_r$ within the scope of some freeze quantifier $\downarrow_r$ is bound by it; otherwise it is free. A sentence is a formula with no free occurrence of any $\uparrow_r$. Given a natural number $n > 0$, we write $\mathrm{LTL}_n^{\downarrow,\Sigma}$ to denote the restriction of $\mathrm{LTL}^{\downarrow,\Sigma}$ to registers in $\{1, \ldots, n\}$. Models of $\mathrm{LTL}^{\downarrow,\Sigma}$ are *data words*. A data word $\sigma$ over a finite alphabet $\Sigma$ is a non-empty word in $\Sigma^{<\omega}$ or $\Sigma^{\omega}$, together with an equivalence relation $\sim^{\sigma}$ on word indices. We write $|\sigma|$ for the length of the data word, $\sigma(i)$ for its letters where $0 \leq i < |\sigma|$.

A *register valuation* $v$ for a data word $\sigma$ is a finite partial map from $\mathbb{N} \setminus \{0\}$ to the indices of $\sigma$. Whenever $v(r)$ is undefined, the formula $\uparrow_r$ is interpreted as false. The satisfaction relation $\models$ is defined as follows (Boolean clauses are omitted).

$$\begin{aligned}
\sigma, i \models_v a &\overset{\text{def}}{\Leftrightarrow} \sigma(i) = a \\
\sigma, i \models_v \uparrow_r &\overset{\text{def}}{\Leftrightarrow} r \in \mathrm{dom}(v) \text{ and } v(r) \sim^{\sigma} i \\
\sigma, i \models_v \mathtt{X}\phi &\overset{\text{def}}{\Leftrightarrow} i+1 < |\sigma| \text{ and } \sigma, i+1 \models_v \phi \\
\sigma, i \models_v \phi_1 \mathtt{U} \phi_2 &\overset{\text{def}}{\Leftrightarrow} \text{for some } j \geq i,\ \sigma, j \models_v \phi_2 \text{ and for all } i \leq j' < j,\ \sigma, j' \models_v \phi_2 \\
\sigma, i \models_v \downarrow_r \phi &\overset{\text{def}}{\Leftrightarrow} \sigma, i \models_{v[r \mapsto i]} \phi
\end{aligned}$$

$v[r \mapsto i]$ denotes the register valuation equal to $v$ except that the register $r$ is mapped to the position $i$. In the sequel, we omit the subscript "$v$" in $\models_v$ when sentences are involved. We use the standard abbreviations for the temporal operators ($\mathtt{G}$, $\mathtt{F}$, ...) and for the Boolean operators and constants ($\vee$,$\Rightarrow$, $\top$, $\bot$, ...). The infinitary [resp. finitary] satisfiability problem for LTL with registers, noted $\omega$-SAT-LTL$^{\downarrow}$ [resp. $f$-SAT-LTL$^{\downarrow}$], is defined as follows: given a finite alphabet $\Sigma$ and a formula $\phi$ in $\mathrm{LTL}^{\downarrow,\Sigma}$, is there an infinite [resp. a finite] data word $\sigma$ such that $\sigma, 0 \models \phi$?

**Theorem 1.** *[9] $\omega$-SAT-LTL$^{\downarrow}$ restricted to one register is $\Pi_1^0$-complete and $f$-SAT-LTL$^{\downarrow}$ restricted to one register is decidable with non-primitive recursive complexity.*

Given a one-counter automaton $\mathcal{A} = \langle Q, q_I, \delta, F \rangle$, finite [resp. infinite] accepting runs of $\mathcal{A}$ can be viewed as finite [resp. infinite] data words over the alphabet $Q$. Indeed, given a run $\rho$, the equivalence relation $\sim^{\rho}$ is defined as follows: $i \sim^{\rho} j$ iff the counter value at the $i$th position of $\rho$ is equal to the counter value at the $j$th position of $\rho$. In order to ease the presentation, in the sequel we store in registers counter values, which is an equivalent way to proceed by slightly adapting the semantics for $\uparrow_r$ and $\downarrow_r$, and the values stored in registers (data).

The finitary [resp. infinitary] (existential) model-checking problem over one-counter automata for LTL with registers, noted $\mathrm{MC}^{<\omega}$ [resp. $\mathrm{MC}^{\omega}$] is defined as follows: given a one-counter automaton $\mathcal{A} = \langle Q, q_I, \delta, F \rangle$ and a sentence $\phi$ in $\mathrm{LTL}^{\downarrow,Q}$, is there a finite [resp. infinite] accepting run $\rho$ of $\mathcal{A}$ such that $\rho, 0 \models \phi$? If the answer is "yes", we write

$\mathcal{A} \models^{<\omega} \phi$ [resp. $\mathcal{A} \models^{\omega} \phi$ ]. In this existential version of model checking, this problem can be viewed as a variant of satisfiability in which satisfaction of a formula can be only witnessed within a specific class of data words, namely the accepting runs of the automata. Results for the universal version of model checking will follow easily from those for the existential version.

We write $\mathrm{MC}^{\alpha}_{n}$ to denote the restriction of $\mathrm{MC}^{\alpha}$ to formulae with at most $n$ registers. Very often, it makes sense that only counter values are known but not the current location of a configuration, which can be understood as an internal information about the system. We write $\mathrm{PureMC}^{\alpha}_{n}$ to denote the restriction of $\mathrm{MC}^{\alpha}_{n}$ (its "pure" version) to formulae with atomic formulae only of the form $\uparrow_r$.

*Example 1.* Here are properties that can be stated in $\mathrm{LTL}^{\downarrow,Q}_{2}$ along a run.

- "There is a suffix such that all the counter values are different": $\mathtt{FG}(\downarrow_1 \mathtt{XG}\neg \uparrow_1)$.
- "Whenever location $q$ is reached with current counter value $n$ and next current counter value $m$, if there is a next occurrence of $q$, the two consecutive counter values are also $n$ and $m$": $\mathtt{G}(q \Rightarrow \downarrow_1 \mathtt{X} \downarrow_2 \mathtt{XG}(q \Rightarrow \uparrow_1 \wedge \mathtt{X} \uparrow_2))$.

We show how to get rid of propositional variables by reducing the model-checking problem over one-counter automata to its pure version.

**Lemma 2 (Purification).** *Given a one-counter automaton $\mathcal{A}$ and a sentence $\phi$ in $\mathrm{LTL}^{\downarrow,Q}_{n}$, one can compute in logarithmic space in $|\mathcal{A}| + |\phi|$ a one-counter automaton $\mathcal{A}_P$ and $\phi_P$ in $\mathrm{LTL}^{\downarrow,\emptyset}_{max(n,1)}$ such that $\mathcal{A} \models^{<\omega} \phi$ [resp. $\mathcal{A} \models^{\omega} \phi$ ] iff $\mathcal{A}_P \models^{<\omega} \phi_P$ [resp. $\mathcal{A}_P \models^{\omega} \phi_P$]. Moreover, $\mathcal{A}$ is deterministic iff $\mathcal{A}_P$ is deterministic.*

The idea of the proof is simply to identify locations with patterns about the changes of the unique counter that can be expressed in $\mathrm{LTL}^{\downarrow,\emptyset}$.

*Proof.* Let $\mathcal{A} = \langle Q, q_I, \delta, F\rangle$ with $Q = \{q_1, ..., q_n\}$ and $\phi$ in $\mathrm{LTL}^{\downarrow,Q}$. In order to define $\mathcal{A}_P$, we identify locations with patterns about the changes of the unique counter. For each location $q_i$ in $Q$ we associate the new sequence of transitions described in Fig. 1 and $q_i \xrightarrow{\mathtt{a}} q_j \in \delta$ iff $q_i^F \xrightarrow{\mathtt{a}} q_j \in \delta'$. In the sequence of picks numbered from 0 to $n+1$, the only pick of height 2 is one numbered $i$. In order to identify the beginning of the first pick of height 3 we introduce formulae in $\mathrm{LTL}^{\downarrow,\emptyset}_{1}$: $\varphi_{\neg\frac{3}{7}}$ expresses that "among the 7 next counter values (including the current counter value), there are no 3 equal values" and $\varphi_{0\sim 6}$ expresses that "the current counter value is equal to the counter value at the 6th next position". We write LOC to denote the formula $\varphi_{\neg\frac{3}{7}} \wedge \varphi_{0\sim 6}$. By a simple case analysis, one can check that for $k \geq 0$, in the run of $\mathcal{A}_P$, LOC holds true iff the current location is in $Q$. We pose $\phi_i = \mathtt{X}^{6+2(i-1)} \downarrow_1 \mathtt{X}^2 \neg \uparrow_1$ for $1 \leq i \leq n$. One can check that for $k \geq 0$, in the run of $\mathcal{A}_P$ LOC $\wedge\, \phi_i$ holds true iff the current location is $q_i$. $\phi_P$ is equal to $\mathrm{T}(\phi)$ with the map T that is homomorphic for Boolean operators and $\downarrow_r$, and its restriction to $\uparrow_r$ is identity. The rest of the inductive definition is as follows.

$$\mathrm{T}(q_i) = \phi_i;\ \mathrm{T}(\mathtt{X}\phi) = \mathtt{X}^{10+2(n+1)+1}\mathrm{T}(\phi);\ \mathrm{T}(\phi \mathtt{U} \phi') = \big(\mathrm{LOC} \Rightarrow \mathrm{T}(\phi)\big)\mathtt{U}\big(\mathrm{LOC} \wedge \mathrm{T}(\phi')\big)$$

We remark that $\phi$ and $\phi_P$ have the same amount of registers unless $\phi$ has no register. □

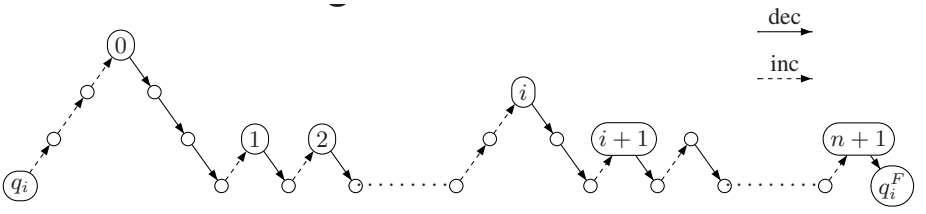


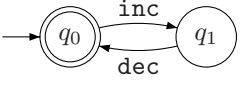

**Fig. 1.** Encoding $q_i$ by a pattern made of $n+2$ increasing picks of length $10+2(n+1)$

## 3 Model Checking Deterministic One-Counter Automata

In this section, we show that $\mathrm{MC}^{\omega}$ restricted to deterministic one-counter automata is PSPACE-complete and the same restriction for $\mathrm{MC}^{<\omega}$ is in EXPSPACE. First, we show PSPACE-hardness.

**Proposition 3.** $\mathrm{PureMC}^{<\omega}$ *and* $\mathrm{PureMC}^{\omega}$ *restricted to deterministic one-counter automata are* PSPACE*-hard problems.*

*Proof.* Consider a QBF instance $\phi = \forall \mathrm{p}_1\ \exists \mathrm{p}_2\ \cdots\ \forall \mathrm{p}_{2N-1}\ \exists\ \mathrm{p}_{2N}\ \Psi(\mathrm{p}_1, ..., \mathrm{p}_{2N})$ where $\mathrm{p}_1$,...,$\mathrm{p}_{2N}$ are propositional variables and $\Psi(\mathrm{p}_1, \ldots, \mathrm{p}_{2N})$ is a quantifier-free propositional formula built over $\mathrm{p}_1, \ldots, \mathrm{p}_{2N}$. The fixed deterministic one-counter automaton $\mathcal{A}$ below generates the sequence of counter values $(01)^{\omega}$.




Let $\psi$ be the formula in $\mathrm{LTL}^{\downarrow,\emptyset}$ defined from the family $\psi_1, \ldots, \psi_{2N+1}$ of formulae with $\psi = \downarrow_{2N+1} \psi_1 \colon \psi_{2N+1} = \Psi[\mathrm{p}_i \leftarrow (\uparrow_i \Leftrightarrow \uparrow_{2N+1})]$ and for $i \in \{1, ..., N\}$, $\psi_{2i} = \mathtt{F}(\downarrow_{2i} \psi_{2i+1})$ and $\psi_{2i-1} = \mathtt{G}(\downarrow_{2i-1} \psi_{2i})$. One can show that $\phi$ is satisfiable iff $\mathcal{A}_{\phi} \models_{\omega} \psi$. For $\mathrm{PureMC}^{<\omega}$, one can enforce the sequence of counter values from the accepting run to be $(01)^{2N}0$ and then use $\mathtt{X}$ to define the $\psi_i$s. □

Observe that in the reduction, we use an unbounded number of registers (see Theorem 12) but a fixed deterministic one-counter automaton.

### 3.1 Properties on Runs for Deterministic Automata

Any deterministic one-counter automaton $\mathcal{A}$ has at most one infinite run, possibly with an infinite amount of counter values. If this run is not accepting, i.e. no accepting location is repeated infinitely, then for no formula $\phi$, we have $\mathcal{A} \models^{\omega} \phi$. We show below that we can decide in polynomial-time whether $\mathcal{A}$ has accepting runs either finite or infinite. Moreover, we shall show that the infinite unique run has some regularity.

Let $\rho^{\omega}_{\mathcal{A}}$ be the unique run (if it exists) of the deterministic one-counter automaton $\mathcal{A}$ represented by the following sequence of configurations $\langle q_0, n_0\rangle\ \langle q_1, n_1\rangle\ \langle q_2, n_2\rangle \ldots$

**Lemma 4.** *Let $\mathcal{A}$ be a deterministic one-counter automaton with an infinite run. There are $K_1, K_2, K_3$ such that $K_1 + K_2 \leq |Q|^3$, $K_3 \leq |Q|$ and for every $i \geq K_1$, $\langle q_{i+K_2}, n_{i+K_2}\rangle = \langle q_i, n_i + K_3\rangle$.*

Hence, the run $\rho^\omega_{\mathcal{A}}$ can be encoded by its first $K_1 + K_2$ configurations. $\rho^\omega_{\mathcal{A}}$ has a simple structure: it is composed of a polynomial-size prefix $\langle q_0, n_0\rangle \cdots \langle q_{K_1-1}, n_{K_1-1}\rangle$ followed by the polynomial-size loop $\langle q_{K_1}, n_{K_1}\rangle \cdots \langle q_{K_1+K_2-1}, n_{K_1+K_2-1}\rangle$ repeated infinitely often. The effect of applying the loop consists in adding $K_3$ to every counter value. Testing whether $\mathcal{A}$ has an infinite run or $\rho^\omega_{\mathcal{A}}$ is accepting amounts to check whether there is an accepting location in the loop, which can be done in cubic time in $|Q|$. In the rest of this section, we assume that $\rho^\omega_{\mathcal{A}}$ is accepting. Similarly, testing whether $\mathcal{A}$ has a finite accepting run amounts to check whether an accepting location occurs in the prefix or in the loop.

When $K_3 = 0$ and $\mathcal{A}$ has an infinite run, $\rho^\omega_{\mathcal{A}}$ is precisely

$$\langle q_0, n_0\rangle \cdots \langle q_{K_1-1}, n_{K_1-1}\rangle(\langle q_{K_1}, n_{K_1}\rangle \cdots \langle q_{K_1+K_2-1}, n_{K_1+K2-1}\rangle)^\omega.$$

It is then possible to apply a polynomial-space labelling algorithm à la CTL for model checking $\mathrm{LTL}^{\downarrow,Q}$ formulae on $\mathcal{A}$. However, one needs to take care of register valuations, which explains why unlike the polynomial-time algorithm for model checking ultimately periodic models on LTL formulae (see e.g., [20]), model checking restricted to deterministic automata with $K_3 = 0$ is still PSPACE-hard.

### 3.2 A PSPACE Symbolic Model-Checking Algorithm

In this section, we provide decision procedures for solving $\mathrm{MC}^{<\omega}$ and $\mathrm{MC}^{\omega}$ restricted to deterministic one-counter automata. Let us introduce some notations. Let $\rho^\omega_{\mathcal{A}} = \langle q_0, n_0\rangle \langle q_1, n_1\rangle \langle q_2, n_2\rangle \ldots$ be the unique run of the deterministic one-counter automaton $\mathcal{A}$ and $\phi$ be a sentence with $N \geq 1$ registers. Let $i \geq 0$ be a position in $\rho^\omega_{\mathcal{A}}$ and $m$ be a register value in $\mathbb{N}$. We write $\mathrm{pos}_{\mathcal{A}}(i, m)$ to denote the following (possibly infinite) set of offsets: $\mathrm{pos}_{\mathcal{A}}(i, m) = \{j \in \mathbb{N} : m = n_{i+j}\}$. The values $m$ should be understood as register values when evaluation of subformulae is done at position $i$. In general, the set $\{\mathrm{pos}_{\mathcal{A}}(i, m) \subseteq \mathbb{N} : i, m \in \mathbb{N}\}$ can be infinite but if we restrict ourselves to $m$ in $\{n_0, \ldots, n_i\}$ then it is not anymore the case. After all, this is a reasonable assumption when $m$ is intended to be a value stored in a register. Before showing this property, we establish that whenever $K_3 > 0$, two positions with identical counter values are separated by a distance that is bounded by a polynomial in $|Q|$.

**Lemma 5.** *Suppose $K_3 > 0$. For all $i \leq j$, (I) $n_i = n_j$ and $i < K_1$ imply $(j - i) \leq K_1 + K_1K_2$, (II) $n_i = n_j$ and $i \geq K_1$ imply $(j - i) \leq K_2^2$.*

**Lemma 6.** *$\{\mathrm{pos}_{\mathcal{A}}(i, m) : i \in \mathbb{N},\ m \in \{n_0, \ldots, n_i\}\}$ is finite and its cardinality is polynomial in $|Q|$.*

We write $\mathrm{REGVALUES}_{\mathcal{A}}$ to denote the above finite set with polynomial cardinality. Observe that even though the set of counter values occurring in $\rho^\omega_{\mathcal{A}}$ may be infinite (exactly when $K_3 > 0$) we can represent symbolically each register value $v(r)$ at a

position $i$ by a concise representation for $\mathrm{pos}_{\mathcal{A}}(i, v(r))$. One consequence of the proof of Lemma 6 is that $|\mathrm{REGVALUES}_{\mathcal{A}}|$ is bounded by $(1 + K_1 + K_2^2)^2 + K_2 \times (1 + K_1 + K_2^2)$.

We define below the equivalence relation $\equiv$ between positions of $\rho^{\omega}_{\mathcal{A}}$: $i \equiv i'$ iff $q_i = q'_i$, and for all $\alpha, \beta \geq 0$, ($n_{i+\alpha} = n_{i+\beta}$ iff $n_{i'+\alpha} = n_{i'+\beta}$) and ($q_{i+\alpha} = q_{i+\beta}$ iff $q_{i'+\alpha} = q_{i'+\beta}$). Typically, $i$ and $i'$ are equivalent whenever the path starting at position $i$ is isomorphic to the path starting at position $i'$. It is easy to see that $\equiv$ has at most $K_1 + K_2$ equivalence classes since $i \equiv_{K_2} i'$ and $i, i' \geq K_1$ imply $i \equiv i'$ (here $\equiv_{K_2}$ is the congruence relation). We extend $\equiv$ to pairs composed of positions and register valuations. Given positions $i, i' \in \mathbb{N}$ and register valuations $v, v'$ such that $\mathrm{ran}(v) \subseteq \{n_0, \ldots, n_i\}$ and $\mathrm{ran}(v') \subseteq \{n_0, \ldots, n_{i'}\}$, $\langle i, v\rangle \equiv \langle i', v'\rangle$ iff (1) $i \equiv i'$ and (2) for all $\alpha \geq 0$ and registers $r \in \{1, \ldots, N\}$, $n_{i+\alpha} = v(r)$ iff $n_{i'+\alpha} = v'(r)$. Again, $\equiv$ is an equivalence relation. A pair $\langle i, v\rangle$ is called a *context*.

Condition (2) on the definition of $\equiv$ is equivalent to: for every register $r \in \{1, \ldots, N\}$, $\mathrm{pos}_{\mathcal{A}}(i, v(r)) = \mathrm{pos}_{\mathcal{A}}(i', v'(r))$. Consequently,

**Lemma 7.** *There are polynomials $P_1$ and $P_2$ such that the number of equivalence classes for $\equiv$ on contexts $\langle i, v\rangle$ is bounded by $P_1(|Q|) \times [P_2(|Q|)]^N$ ($N$ is the number of registers).*

The bound is exactly $(K_1 + K_2) \times [(1 + K_1 + K_2^2)^2 + K_2 \times (K_2^2 + K_1 + 1)]^N$, where $(1+K_1+K_2^2)^2+K_2\times(K_2^2+K_1+1)$ is the cardinal of $\mathrm{REGVALUES}_{\mathcal{A}}$ and $K_1+K_2$ is the number of equivalent positions w.r.t. to $\equiv$.

**Lemma 8.** *If $\langle i, v\rangle \equiv \langle i', v'\rangle$, then (I) for all $j > 0$, $\langle i + j, v\rangle \equiv \langle i' + j, v'\rangle$ and (II) for every formula $\psi \in \mathrm{LTL}^{\downarrow,Q}_N$, $\rho^{\omega}_{\mathcal{A}}, i \models_v \psi$ iff $\rho^{\omega}_{\mathcal{A}}, i' \models_{v'} \psi$.*

Lemma 8(I) is by an easy verification (recurrence on $j$) whereas Lemma 8(II) is by structural induction on $\psi$.

### 3.3 Abstraction and Complexity Issues

We have seen that the equivalence relation $\equiv$ on contexts has finite index. We present below a means to represent symbolically an equivalence class. In the case $K_3 = 0$, a *symbolic context* is a pair $\langle i, pos\rangle$ where $i \in \{0, \ldots, K_1+K_2-1\}$ and $pos$ is a symbolic register valuation of the form $\{1, \ldots, N\} \to \{n_0, \ldots, n_{K_1+K_2-1}\}$. A context $\langle i, v\rangle$ is represented by the symbolic context $\langle i', pos\rangle$ where

- $i < K_1$ implies $i' = i$ otherwise $i'$ is the unique element of $\{K_1, \ldots, K_1+K_2-1\}$ such that $i \equiv_{K_2} i'$,
- for $r \in \{1, \ldots, N\}$, $pos(r) = v(r)$. Observe that $v(r) \in \{n_0, \ldots, n_{K_1+K_2-1}\}$ and $pos(r)$ can be encoded with $\mathcal{O}(log(|Q|))$ bits.

When $K_3 > 0$, the definition of a symbolic context is modified for the second component only since the set of counter values along the run is infinite. A *symbolic context* remains a pair $\langle i, pos\rangle$ but $i \in \{0, \ldots, K_1 + K_2 - 1\}$ and $pos$ is a symbolic register valuation of the form $\{1, \ldots, N\} \to \mathcal{P}(\{0, \ldots, K_1 + K_1K_2\}) \cup \mathcal{P}(\{0, \ldots, K_2^2\})$. Moreover, when $i < K_1$, $pos(r) \subseteq \{0, \ldots, K_1 + K_1K_2\}$, otherwise $pos(r) \subseteq \{0, \ldots, K_2^2\}$.

Indeed, from Lemma 5, whenever $K_3 > 0$, for all $i \in \mathbb{N}$ and $m \in \{n_0, \ldots, n_i\}$, if $i < K_1$, then $\mathrm{pos}_{\mathcal{A}}(i,m) \subseteq \{0, \ldots, K_1 + K_1K_2\}$ otherwise $\mathrm{pos}_{\mathcal{A}}(i,m) \subseteq \{0, \ldots, K_2^2\}$. A context $\langle i, v\rangle$ is represented by the symbolic context $\langle i', pos\rangle$ where $i'$ is defined as above and for $r \in \{1, \ldots, N\}$, $pos(r) = \mathrm{pos}_{\mathcal{A}}(i, v(r))$.

Each value $pos(r)$ can be encoded with a polynomial amount of bits in $|Q|$. One can compute in polynomial time in $|Q|$ the range of any symbolic register valuation (whether $K_3 = 0$ or not) thanks to Lemma 6. When we bound the number of registers, the number of symbolic contexts occurring in $\rho^{\omega}_{\mathcal{A}}$ is polynomial in $|Q|$ and they can be computed in polynomial time.

Given a context $\langle i, v\rangle$, we write $[\langle i, v\rangle]$ to denote its corresponding symbolic context (w.r.t. $\mathcal{A}$). Symbolic contexts correspond to the equivalence classes of $\equiv$:

**Lemma 9.** *Let $\langle i, v\rangle$ and $\langle i', v'\rangle$ be contexts. Then $[\langle i, v\rangle] = [\langle i', v'\rangle]$ iff $\langle i, v\rangle \equiv \langle i', v'\rangle$.*

Let us define a map $next$ that takes as argument a symbolic context $\langle i, pos\rangle$ and returns the symbolic context obtained at the next step. This is a well-defined function because taking two contexts that are $\equiv$-equivalent, moving one step forward leads to two new contexts that are also $\equiv$-equivalent (see Lemma 10 below). The map $next$ is defined as follows : $next(\langle i, pos\rangle) = \langle i', pos'\rangle$ where

- if $i < K_1 + K_2 - 1$ then $i' = i + 1$, otherwise $i' = K_1$.
- if $K_3 > 0$, then for $r \in \{1, \ldots, N\}$, $pos'(r) = \{\alpha - 1 : \alpha \in pos(r), \alpha > 0\}$,
- if $K_3 = 0$, then $pos' = pos$.

**Lemma 10.** *Let $\langle i, v\rangle$ be a context with $\mathrm{ran}(v) \subseteq \{n_0, \ldots, n_i\}$. Then $next([\langle i, v\rangle]) = [\langle i + 1, v\rangle]$.*

Below, we solve the model-checking problem by following an automata-based approach [30]. We consider alternating word automata with Büchi acceptance condition on $\omega$-words, see e.g. [29]: every infinite branch of accepting runs has an accepting state repeated infinitely often. Let $\phi$ be a formula in NNF built over disjunction $\vee$ and the release operator $\mathtt{R}$ (dual of $\mathtt{U}$). Observe that $\mathtt{X}$ and $\downarrow_r$ are self-dual. We build an alternating automaton $\mathcal{A}_\phi$ that can be viewed as the product between the run of $\mathcal{A}$ and the automaton for $\phi$. The synchronization mode between these two components takes into account the presence of registers. When $K_3 > 0$ and $\mathcal{A}$ has an accepting run (which can be checked in PTIME), let $\mathcal{A}_\phi = \langle \Sigma, S, s_0, \delta, F\rangle$ be defined as follows:

- $\Sigma = \{a\}$ and $S$ is the set of states of the form $\langle\langle i, pos\rangle, \psi\rangle$ where $\langle i, pos\rangle$ is a symbolic context and $\psi$ is a subformula of $\phi$.
- the initial state is $s_0 = \langle\langle 0, pos_0\rangle, \phi\rangle$ where $pos_0$ is the symbolic register valuation representing the zero register valuation and $F$ is the set of accepting states whose outermost connective of the second component is not until.
- Here is the transition function $\delta$ (obvious dual clauses are omitted):
  - $\delta(\langle\langle i, pos\rangle, q\rangle, a) = \top$ if $q = q_i$, otherwise $\delta(\langle\langle i, pos\rangle, q\rangle, a) = \bot$,
  - $\delta(\langle\langle i, pos\rangle, \neg \uparrow_r\rangle, a) = \bot$ if $0 \in pos(r)$, otherwise $\delta(\langle\langle i, pos\rangle, \neg \uparrow_r\rangle, a) = \top$,
  - $\delta(\langle\langle i, pos\rangle, \psi \wedge \psi'\rangle, a) = \delta(\langle\langle i, pos\rangle, \psi\rangle, a) \wedge \delta(\langle\langle i, pos\rangle, \psi'\rangle, a)$,
  - $\delta(\langle\langle i, pos\rangle, \mathtt{X}\psi\rangle, a) = \langle next(\langle i, pos\rangle), \psi\rangle$,

- $\delta(\langle\langle i, pos\rangle, \downarrow_r \psi\rangle, a) = \delta(\langle\langle i, pos[r \leftarrow \mathrm{pos}_{\mathcal{A}}(i, n_i)]\rangle, \psi\rangle, a)$,
- $\delta(\langle\langle i, pos\rangle, \psi \mathtt{U} \psi'\rangle, a) =$
  $\delta(\langle\langle i, pos\rangle, \psi'\rangle, a) \vee (\delta(\langle\langle i, pos\rangle, \psi\rangle, a) \wedge \langle next(\langle i, pos\rangle), \psi \mathtt{U} \psi'\rangle)$.

When $K_3 = 0$, the clauses for $\downarrow_r$ and $\uparrow_r$ can be easily adapted. We write $\mathcal{A}_\phi^{\langle\langle i,pos\rangle,\psi\rangle}$ to denote the automaton defined from $\mathcal{A}_\phi$ with initial location $\langle\langle i, pos\rangle, \psi\rangle$.

**Lemma 11.** *Let $i \in \mathbb{N}$ and $v$ be a register valuation with range $\{n_0, \ldots, n_i\}$. For every subformula $\psi$ of $\phi$, $\rho^\omega_{\mathcal{A}}, i \models_v \psi$ iff $\mathcal{A}_\phi^{\langle[\langle i,v\rangle],\psi\rangle}$ accepts an infinite run.*

The proof (by structural induction) is a variant of the one for LTL and uses Lemma 8 and 10. This will allow us to characterize precisely the complexity of model checking.

**Theorem 12.** $\mathrm{MC}^\omega$ *restricted to deterministic one-counter automata is* PSPACE*-complete and its restriction to $n \geq 1$ registers is in* PTIME.

*Proof.* $\mathcal{A}_\phi$ is an hesitant alternating word automata over a 1-letter alphabet with each set $S_j$ of the partition being a set of states with identical subformulae. By [17, Theorem 5.6], the nonemptiness problem for hesitant alternating word automata over a 1-letter alphabet can be solved in space $\mathcal{O}(m\ log^2 n)$ where $n$ is the number of states and $m$ is the number of elements in the partition of the set of states. In order to obtain the PSPACE upper bound, it is sufficient to check that the on-the-fly version of the algorithm given in the proof of [17, Theorem 5.6] can be performed (computation of the transition function on demand). This is possible partly because in $\mathcal{A}_\phi$, $m$ is linear in $|\phi|$, $n$ is exponential in $|\phi|$, for each state $s$, $\delta(s, a)$ can be built in polynomial-time in $|\phi|$ and testing if a state is accepting can be done in linear time in $|\phi|$. Moreover, each state in $\mathcal{A}_\phi$ can be encoded in polynomial space in $|\mathcal{A}| + |\phi|$.

When the number of registers is fixed, $\mathcal{A}_\phi$ has a polynomial number of states and since the nonemptiness problem for weak alternating word automata over a 1-letter alphabet can be solved in linear time [3], we get the PTIME upper bound. □

For the finitary case, we cannot invoke the result in [3] because the length of the word is a distinguishing factor.

**Corollary 13.** $\mathrm{MC}^{<\omega}$ *restricted to deterministic one-counter automata is in* EXPSPACE.

The proof consists in designing an alternating word automata on $\omega$-words with a two-letter alphabet on the lines of the previous construction. However, the second letter marks the end of the word so that all the branches detect the end of the word in a synchronous way. The recognized $\omega$-words are among $a^* \cdot b \cdot a^\omega$. Then, we invoke the quadratic space upper bound for the nonemptiness of alternating automata [28], which provides the EXPSPACE upper bound since $\mathcal{A}_\phi$ is of exponential size in $|\phi|$ and $\mathcal{A}_\phi$ can be built in polynomial space in $|\phi|$.

## 4 Model Checking Nondeterministic One-Counter Automata

In this section, we show that several model-checking problems over nondeterministic one-counter automata are undecidable by reducing decision problems for Minsky machines. Undecidability is preserved even in presence of a unique register. This is quite surprising since $f$-SAT-LTL$^\downarrow$ restricted to one register is decidable [9].

In order to illustrate the significance of the following results, it is worth recalling that the halting problem for Minsky machines with incrementing errors is reducible to finitary satisfiability for LTL with one register [9]. We show below that, if we have existential model checking of one-counter automata instead of satisfiability, then we can use one-counter automata to refine the reduction in [9] so that runs with incrementing errors are excluded. More precisely, in the reduction in [9], we were not able to exclude incrementing errors because the logic is too weak to express that, for every decrement, the datum labelling it was seen before (remember that we have no past operators). Now, the one-counter automata are used to ensure that such faulty decrements cannot occur.

**Theorem 14.** $\mathrm{MC}_1^{<\omega}$ *is* $\Sigma_1^0$*-complete.*

*Proof.* The $\Sigma_1^0$ upper bound is by an easy verification since the existence of a finite run (encoded in $\mathbb{N}$) verifying an $\mathrm{LTL}_1^{\downarrow,Q}$ formula (encoded in first-order arithmetic) can be encoded by a $\Sigma_1^0$ formula. So, let us reduce the halting problem for two-counter automata to $\mathrm{MC}_1^{<\omega}$. Let $\mathcal{A} = \langle Q, q_I, \delta, F\rangle$ be a two-counter automaton: the set of instructions $L$ is $\{\mathtt{inc}, \mathtt{dec}, \mathtt{ifzero}\} \times \{1, 2\}$. We build a one-counter automaton $\mathcal{B} = \langle Q', q'_I, \delta', F'\rangle$ and a sentence $\phi$ in $\mathrm{LTL}_1^{\downarrow,Q'}$ such that $\mathcal{A}$ reaches an accepting location iff $\mathcal{B} \models^{<\omega} \phi$.

For each run in $\mathcal{A}$ $\begin{pmatrix} q_I \\ c_1^0 = 0 \\ c_2^0 = 0 \end{pmatrix} \xrightarrow{\mathtt{inst}^0} \begin{pmatrix} q^1 \\ c_1^1 \\ c_2^1 \end{pmatrix} \xrightarrow{\mathtt{inst}^1} \cdots \begin{pmatrix} q^N \\ c_1^N \\ c_2^N \end{pmatrix}$ where $\mathtt{inst}^{\mathtt{i}}$s are instructions, we associate the run in $\mathcal{B}$ below

$$\begin{pmatrix} q_I \\ 0 \end{pmatrix} \xrightarrow{\star} \begin{pmatrix} \langle q_I, \mathtt{inst}^0, q^1\rangle \\ n^1 \end{pmatrix} \xrightarrow{\star} \begin{pmatrix} \langle q^1, \mathtt{inst}^1, q^2\rangle \\ n^2 \end{pmatrix} \cdots \begin{pmatrix} \langle q^{N-1}, \mathtt{inst}^{\mathtt{N-1}}, q^N\rangle \\ n^N \end{pmatrix}$$

where $\xrightarrow{\star}$ hides steps for updating the counter according to the constraints described below. During these steps, auxiliary locations are used and there are of two types: locations that increment or decrement the counter in order to reach an adequate data value ($\mathrm{busyup}_{t,t'}$ and $\mathrm{busydown}_{t,t'}$ where $t, t'$ are transitions) and intermediate locations to perform $\epsilon$-transitions. The data values in the run of $\mathcal{B}$ are governed by the rules below:

(ii) after any configuration labelled by $\langle q, \mathtt{inc}, c, q'\rangle$ (incrementation of the counter $c$), there is no configuration labelled by some $\langle q_1, \mathtt{inc}, c', q'_1\rangle$ with the same counter value,

(iii) after any configuration labelled by $\langle q, \mathtt{inc}, c, q'\rangle$, there is at most one configuration labelled by some $\langle q_1, \mathtt{dec}, c, q'_1\rangle$ with the same counter value (there are more incrementations than decrementations),

(iv) after any configuration labelled by $\langle q, \mathtt{inc}, c, q'\rangle$, there is no configuration labelled by some $\langle q_1, \mathtt{dec}, c', q'_1\rangle$ with the same counter value and $c \neq c'$,

(v) after any configuration labelled by $\langle q, \mathtt{inc}, c, q'\rangle$, there is no configuration labelled by $\langle q_1, \mathtt{ifzero}, c, q'_1\rangle$ followed by a configuration labelled by some $\langle q_1, \mathtt{dec}, c, q'_1\rangle$ with the same counter value as $\langle q, \mathtt{inc}, c, q'\rangle$,

(vi) after any configuration labelled by $\langle q, \mathtt{inc}, c, q'\rangle$ for which there is no subsequent configuration labelled by $\langle q_1, \mathtt{dec}, c, q'_1\rangle$ with the same counter value, there is also no $\langle q_2, \mathtt{ifzero}, c, q'_2\rangle$,

Now, let us define $\mathcal{B}$. We shall partly encode in its control graph the satisfaction of these conditions. For instance, two successive incrementation transitions in $\mathcal{A}$, leads to an incrementation in $\mathcal{B}$ since we enforce that the counter value is fresh in $\mathcal{B}$ iff its letter is some $\langle -, \texttt{inc}, -, - \rangle$ (incrementation instructions). When we write $q \xrightarrow{\top} q'$ we mean $q \xrightarrow{\texttt{inc}} auxi_{q,q'} \xrightarrow{\texttt{dec}} q'$ for an auxiliary location $auxi_{q,q'}$.

- $Q'$ is equal to $\delta \uplus (\{q_I\} \cup \{\text{busydown}_{t,t'}, \text{busyup}_{t,t'} : t, t' \in \delta\})$ plus some unspecified auxiliary locations,
- $F' = \{\langle q, l, c, q' \rangle \in \delta : q \in F\} \cup (\{q_I\} \cap F)$ and $q'_I = q_I$,
- The relation $\delta'$ contains the following transitions:
  - For $\langle q_I, \texttt{inc}, c, q \rangle \in \delta$, add $q'_I \xrightarrow{\texttt{inc}} \langle q_I, \texttt{inc}, c, q \rangle$ to $\delta'$;
  - For $t = \langle q_I, \texttt{ifzero}, c, q \rangle \in \delta$, add $q'_I \xrightarrow{\top} t$ to $\delta'$;
  - For every transition $t = \langle q, \texttt{inc}, c, q' \rangle \in \delta$,
    1. if $t' = \langle q', \texttt{inc}, c', q'' \rangle \in \delta$, then add $t \xrightarrow{\texttt{inc}} t'$ to $\delta'$,
    2. if $t' = \langle q', \texttt{ifzero}, c', q'' \rangle \in \delta$ with $c' \neq c$ or $t' = \langle q', \texttt{dec}, c, q'' \rangle \in \delta$, then add $t \xrightarrow{\top} t'$ to $\delta'$,
    3. if $t' = \langle q', \texttt{dec}, c', q'' \rangle \in \delta$ with $c' \neq c$, then add $t \xrightarrow{\top} \text{busydown}_{t,t'}$, $\text{busydown}_{t,t'} \xrightarrow{\texttt{dec}} \text{busydown}_{t,t'}$, and $\text{busydown}_{t,t'} \xrightarrow{\texttt{dec}} t'$ to $\delta'$ (decrement the counter until it reaches a value for a previous incrementation),
  - For every transition $t = \langle q, l, c, q' \rangle \in \delta$ with $l \in \{\texttt{dec}, \texttt{ifzero}\}$,
    1. if $t' = \langle q', \texttt{inc}, c', q'' \rangle \in \delta$, then add $t \xrightarrow{\top} \text{busyup}_{t,t'}$, $\text{busyup}_{t,t'} \xrightarrow{\texttt{inc}} \text{busyup}_{t,t'}$, and $\text{busyup}_{t,t'} \xrightarrow{\texttt{inc}} t'$ to $\delta'$ (increment the counter until it reaches a new value),
    2. if $t' = \langle q', \texttt{ifzero}, c', q'' \rangle \in \delta$ then add $t \xrightarrow{\top} t'$ to $\delta'$,
    3. if $t' = \langle q', \texttt{dec}, c', q" \rangle \in \delta$, then add to $\delta'$ the transitions from Figure 2. Observe that this is the only case for which we do not know whether the counter increases or not.

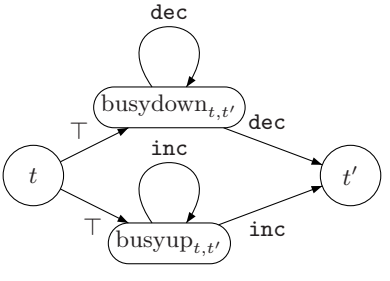



**Fig. 2.** Transitions in $\delta'$

In runs of $\mathcal{B}$, we are only interested in positions with letters in $\delta$. The control graph of $\mathcal{B}$ guarantees that the succession of transitions in $\mathcal{A}$ is valid assuming that we ignore the intermediate (auxiliary or busy) configurations

The formula $\phi$ is the conjunction of the following requirements: (ii)-(vi) plus

(i) some configuration in $F'$ is visited,

(vii) after any configuration labelled by $t = \langle q, \mathtt{inc}, c, q' \rangle$, there is no configuration labelled by some $\mathrm{busyup}_{t,t'}$ with the same counter value and such that the next configuration has the same label unless there is some configuration labelled by some $\langle q_1, \mathtt{inc}, c, q_1' \rangle$ in between,

$$\mathtt{G}(t \Rightarrow \downarrow_1 \neg(\neg(\bigvee_{\langle -,\mathtt{inc},-,-\rangle} \langle -, \mathtt{inc}, -, -\rangle)\mathtt{U}\bigvee_{t'}(\mathrm{busyup}_{t,t'} \wedge \uparrow_1 \wedge \mathtt{X}\, \mathrm{busyup}_{t,t'})))$$

(viii) after any configuration labelled by $t = \langle q, \mathtt{inc}, c, q' \rangle$, there is no configuration labelled by some $\langle q_1, \mathtt{dec}, c, q_1' \rangle$ with a different counter value unless there is some configuration labelled by some $\langle q_2, \mathtt{inc}, c, q_2' \rangle$ in between,

$$\mathtt{G}(t \Rightarrow \downarrow_1 \neg(\neg(\bigvee_{\langle -,\mathtt{inc},-,-\rangle} \langle -, \mathtt{inc}, -, -\rangle)\mathtt{U}(\bigvee_{\langle -,\mathtt{dec},c,-\rangle} (\langle -, \mathtt{dec}, c, -\rangle \wedge \neg \uparrow_1))))$$

It is easy to check that each condition in (i)-(viii) can be expressed in $\mathrm{LTL}_1^{\downarrow,Q'}$ (some examples are indeed provided above). Now consider any run of $\mathcal{B}$ which satisfies (ii)-(viii). The key achievement of the definitions of $\mathcal{B}$ and $\phi$ is that, for every position in the run, the counter value is fresh iff either its letter is some $\langle q, \mathtt{inc}, c, q' \rangle$ or the letter is not in $\delta \cup \{q_I\}$. For any counter $c \in \{1, 2\}$, we can define its value as the number of $\langle q, \mathtt{inc}, c, q' \rangle$ letters for which a latter letter $\langle q_1, \mathtt{dec}, c, q_1' \rangle$ with the same value of the counter $\mathcal{B}$ has not yet occurred. Observe that the conditions (vii), (viii) and the control graph of $\mathcal{B}$ induce a stack discipline for the counter values of configurations with labels of the form either $\langle -, \mathtt{inc}, c, - \rangle$ and $\langle -, \mathtt{dec}, c, - \rangle$. This guarantees that no configuration labelled by $\langle -, \mathtt{dec}, c, - \rangle$ has a new counter value.

For any run of $\mathcal{B}$ which satisfies (ii)-(viii), we can thus extract a valid run of $\mathcal{A}$. Conversely, any valid run of $\mathcal{A}$ can be encoded in the same way as a run of $\mathcal{B}$ which satisfies (ii)-(viii). The latter is done by inserting auxiliary letters as required to reach appropriate values of the counter of $\mathcal{B}$. □

**Theorem 15.** $\mathrm{MC}_1^{\omega}$ *is* $\Sigma_1^1$*-complete.*

The proof is similar to the proof of Theorem 14 except that instead of reducing the halting problem for Minsky machines, we reduce the recurrence problem for nondeterministic Minsky machines that is known to be $\Sigma_1^1$-hard [2]. The $\Sigma_1^1$ upper bound is by an easy verification since an accepting run can be viewed as a function $f : \mathbb{N} \to \mathbb{N}$ and then checking that it satisfies an $\mathrm{LTL}_1^{\downarrow,Q}$ formula can be expressed in first-order arithmetic. Another consequence of the Purification Lemma is the result below.

**Theorem 16.** $\mathrm{PureMC}_1^{<\omega}$ *is* $\Sigma_1^0$*-complete and* $\mathrm{PureMC}_1^{\omega}$ *is* $\Sigma_1^1$*-complete.*

The above-mentioned undecidability holds true even if we restrict ourselves to one-counter automata for which there are no transitions with identical instructions going from the same location. A one-counter automaton $\mathcal{A}$ is *weakly deterministic* whenever for every location $q$, if $\langle q, l, q' \rangle, \langle q, l', q'' \rangle \in \delta$, we have $l = l'$ implies $q' = q''$. The transition systems induced by these automata are not necessarily deterministic.

**Theorem 17.** $\mathrm{PureMC}_1^{<\omega}$ *[resp.* $\mathrm{PureMC}_1^{\omega}$*] restricted to weakly deterministic one-counter automata is* $\Sigma_1^0$*-complete [resp.* $\Sigma_1^1$*-complete].*

The proof uses the Purification Lemma and provides reductions from the model-checking problems to their restrictions to weakly deterministic automata.

## 5 Conclusion

We have shown that model checking $\mathrm{LTL}^{\downarrow}$ over one-counter automata is undecidable, which contrasts with the decidability of many verification problems for one-counter automata [14,26]. For instance, we have shown that model checking nondeterministic one-counter automata over $\mathrm{LTL}^{\downarrow}$ restricted to a unique register and without alphabet is already $\Sigma^1_1$-complete in the infinitary case. On the decidability side, a suitable abstraction has been introduced to establish the PSPACE upper bound for model checking $\mathrm{LTL}^{\downarrow}$ over deterministic one-counter automata in the infinitary case.

Viewing runs as data words is an idea that can be pushed further. For instance, the decidability status of model checking $\mathrm{LTL}^{\downarrow}$ over the class of reversal-bounded counter automata [8] remains open. Hence, our results pave the way for model checking $\mathrm{LTL}^{\downarrow}$ over other classes of operational models that are known to admit powerful techniques for solving verification tasks. Finally, among the specific problems left open by this paper, we wish to mention the complexity of model-checking deterministic one-counter automata with $\mathrm{LTL}^{\downarrow}$ in the finitary case (the complexity is however known in the infinitary case). Finitary nonemptiness problem for 1-letter hesitant alternating word automata also faces the difficulty to determine the end of the word (synchronization is needed), see e.g. [17].

*Acknowledgement.* We would like to thank Philippe Schnoebelen for suggesting simplifications in the proofs of Lemma 2 and Proposition 3.

## References

1. Alur, R., Dill, D.: A theory of timed automata. TCS 126, 183–235 (1994)
2. Alur, R., Henzinger, T.: A really temporal logic. In: FOCS 1989, pp. 164–169. IEEE, Los Alamitos (1989)
3. Bernholtz, O., Vardi, M., Wolper, P.: An automata-theoretic approach to branching-time model checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 142–155. Springer, Heidelberg (1994)
4. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: PODS 2006, pp. 10–19 (2006)
5. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS 2006, pp. 7–16. IEEE, Los Alamitos (2006)
6. Bouajjani, A., Jurski, Y., Sighireanu, M.: A generic framework for reasoning about dynamic networks of infinite-state processes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 690–705. Springer, Heidelberg (2007)
7. Bouyer, P., Petit, A., Thérien, D.: An algebraic approach to data languages and timed languages. I & C 182(2), 137–162 (2003)
8. Dang, Z., Ibarra, O., Pietro, P.S.: Liveness verification of reversal-bounded multicounter machines with a free counter. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 132–143. Springer, Heidelberg (2001)

9. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. In: LICS 2006, pp. 17–26. IEEE, Los Alamitos (2006)
10. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint LTL: Decidability and complexity. I & C 205(1), 2–24 (2007)
11. Demri, S., Lazić, R., Sangnier, A.: Model checking freeze LTL over one-counter automata. Research report, Laboratoire Spécification et Vérification, ENS Cachan (2008)
12. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). Journal of Applied Logic 4(3), 279–304 (2006)
13. Goranko, V.: Hierarchies of modal and temporal logics with references pointers. Journal of Logic, Language, and Information 5, 1–24 (1996)
14. Jančar, P., Kučera, A., Moller, F., Sawa, Z.: DP lower bounds for equivalence-checking and model-checking of one-counter automata. I & C 188(1), 1–19 (2004)
15. Jurdziński, M., Lazić, R.: Alternation-free modal mu-calculus for data trees. In: LICS 2007, pp. 131–140 (2007)
16. Kupferman, O., Vardi, M.: Memoryful Branching-Time Logic. In: LICS 2006, pp. 265–274. IEEE, Los Alamitos (2006)
17. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. JACM 47(2), 312–360 (2000)
18. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: LICS 2002, pp. 383–392. IEEE, Los Alamitos (2002)
19. Lazić, R.: Safely freezing LTL. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 381–392. Springer, Heidelberg (2006)
20. Markey, N., Schnoebelen, P.: Model checking a path. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 251–261. Springer, Heidelberg (2003)
21. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. TOCL 5(3), 403–435 (2004)
22. Ouaknine, J., Worrell, J.: On Metric Temporal Logic and faulty Turing machines. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 217–230. Springer, Heidelberg (2006)
23. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Logical Methods in Computer Science 3(1:8), 1–27 (2007)
24. Schwentick, T., Weber, V.: Bounded-variable fragments of hybrid logics. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 561–572. Springer, Heidelberg (2007)
25. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
26. Serre, O.: Parity games played on transition graphs of one-counter processes. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 337–351. Springer, Heidelberg (2006)
27. ten Cate, B., Franceschet, M.: On the complexity of hybrid logics with binders. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 339–354. Springer, Heidelberg (2005)
28. Vardi, M.: Alternating automata and program verification. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 471–485. Springer, Heidelberg (1995)
29. Vardi, M.: Alternating automata: unifying truth and validity checking for temporal logics. In: McCune, W. (ed.) CADE 1997. LNCS, vol. 1249, pp. 191–206. Springer, Heidelberg (1997)
30. Vardi, M., Wolper, P.: Automata-theoretic techniques for modal logics of programs. JCSS 32, 183–221 (1986)

# Author Index